МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМ. М.В. ЛОМОНОСОВА

Механико-математический факультет

Курсовая работа

Студент 3 курса: Нагорных Я.В.
Научный руководитель: Богачев К.Ю.

Москва
2017

Содержание

Введение

Печать большив массивов чисел всегда занимает много времени. Кроме того, у печати данных мало ресурсов для ускорения.

Печать чисел с плавающей запятой также является проблемой, так как само значение числа и его экспоненту нельзя обрабатывать независимо.

Стандартный подход недостаточно точен и в некоторых случаях дает неверные результаты. Кроме того использование функций стандартных библиотек (`printf`, `sprintf`) достаточно затратно по времени.

Цели работы:

1. Ускорить печать больших массивов;

2. Использовать быстрые алгоритмы печати целых чисел и чисел с плавающей точкой.

1 Проблемы и способы их решения

Как уже было сказано, у печати массивов мало ресурсов для ускорения. Также проблемой является и то, что печать данных файл должна быть строго последовательной, поэтому нельзя "простым" образом использовать распараллеливание.

Однако, известно что большую часть времени занимает преобразование типа `int` или `double` в буффер типа `const char *` непосредственно для печати. Именно это можно и распараллелить, используя многопоточное программирование. Непосредственно печать в сам файл упирается в возможности диска. Ее ускорить нельзя.

Кроме того, можно заменить стандартный алгоритм преобразования числа в строку, на более быстрые. Мы будем использовать алгоритм Grisu2, о котором будет рассказано позже.

2 Описание алгоритма

2.1 Используемые структуры и классы

Структура `writer_chunk`. В ней находится элемент класса `writer_file`, строковый буффер (готовый для печати) и его порядковый номер (`chunk_id`). Кроме того, хранится флаг, является ли этот `writer_chunk` последним.

Класс `writer_file`. Он организовывает правильную печать в файл.

Структура `printer_chunk`. Этот тип состоит из лямбда-функии, которая должна обработать определенный фрагмент массива чисел, и элемента типа `writer_chunk`, который должна вернуть функция.

Класс `mutex_wait_queue`**.** Это реализация *блокирующей очереди*, или *мьютексной очереди*. Под ней понимается очередь со следующим свойством: когда поток пытается прочитать что-то из пустой очереди, то он блокируется, до тех пор, пока какой-нибудь другой поток не положит в нее элемент. У этой очереди есть следующие методы:

- `dequeue` – достает верхний элемент из очереди, если очередь непустая. Иначе, поток, вызвавший этот метод блокируется. Также можно передать время блокировки, по истечении которого, поток разблокируется и вернется ни с чем;
- `dequeue_all` – аналогично `dequeue`, но достает все элементы, находящиеся в очереди, и складывает в указатель вектор из них;
- `enqueue` – складывает элемент в конец очереди.

Класс `parallel_writer`**.** Он хранит в себе поток `m_writer`, вектор потоков `m_printer`. Поток `m_writer` будет заниматься печатью в файл. Потоки `m_printers` занимаются тем, что конвертируют элементы типа `printer_chunk` (числа) в элементы типа `writer_chunk` (строки). Помимо потоков и их количества этот класс хранит две блокирующие очереди `m_print_queue` и `m_write_queue`, состоящие из `printer_chunk` и `writer_chunk` соответственно. Зачем нужны такие очереди будет сказано позже.

2.2 Распределение задач

Управляющий (главный) поток будет складывать элементы типа `printer_chunk` в очередь `m_print_queue`. Потоки `m_printers` будут доставать из этой очереди `printer_chunk`-и на обработку. Они должны конвертировать числа в буфферы, готовые для печати. Эти готовые буфферы `writer_chunk` они складывают в другую очередь `m_write_queue`. `parallel_writer` Поток `m_writer` должен забирать готовые буфферы из этой очереди и печатать их в правильном порядке в файл.

Схематично работа потоков показана на Рисунке 1.

2.3 Описание Grisu2

В статье [1] описан алгоритм Grisu и его улучшения, также доказана их точно. Опишем кратко эти алгоритмы.

Идея алгоритма. Предполагается, не умаляя общности, что у числа с плавающей точкой v отрицательный показатель. Тогда это число можно выразить как $v = \dfrac{f_v}{2^{-e_v}}$, где f_v – мантисса, а e_v – экспонента. Десятичные цифры v могут быть вычислены путем нахождения десятичного показателя t, для которого $1 \leqslant \dfrac{f_v \times 10^t}{2^{-e_v}} < 10$.

Первая цифра является целой частью этой дроби. Последующие цифры вычисляются путем повторного использования оставшейся дроби: нужно умножить числитель на 10 и взять целую часть от вновь полученной дроби.

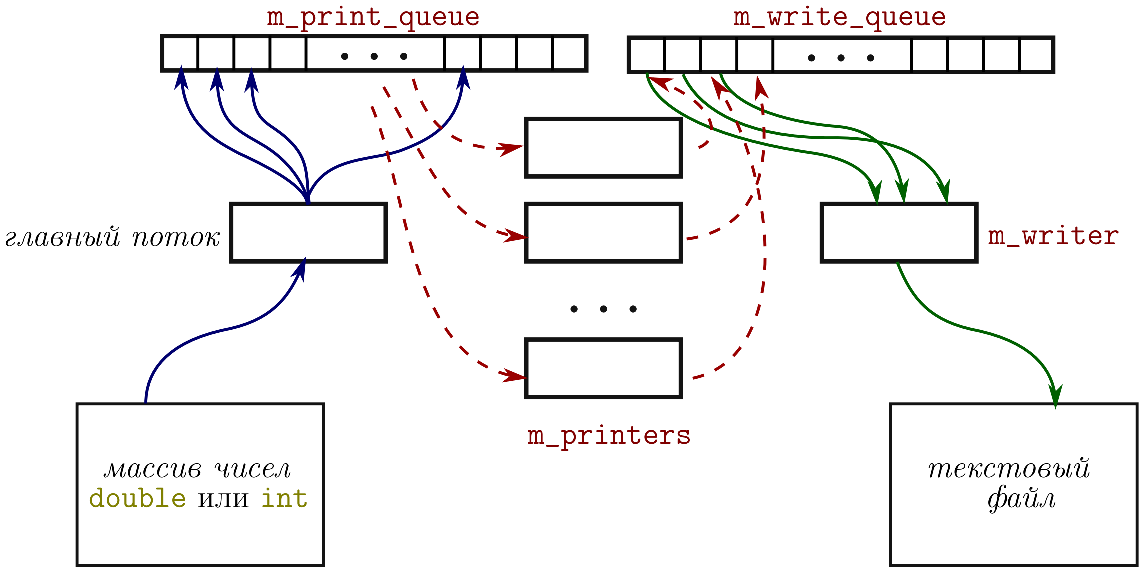

Рисунок 1: Работа потоков.

Идея Grisu состоит в том, чтобы кешировать приблизительные значения $\dfrac{10^t}{2^{e_t}}$. Дорогих операций с большими числами не будет: они заменяются операциями с целыми числами фиксированного размера.

Кэш для всевозможных значений t и e_t может быть дорогостоящим. Из-за этого требования к кеш-памяти в Grisu упрощены. Кэш хранит только нормированные приближения с плавающей точкой всех соответствующих степеней десяти: $\tilde{c}_k := \left[10^k\right]^\star_q$, где q – точность кэшированных чисел. Кэшированные числа сокращают большую часть экспоненты v, так что остается только небольшой показатель.

Процесс генерации цифр использует степени десяти с экспонентой $e_{\tilde{c}_t}$, близкой к e_v. Разница между двумя показателями будет небольшой.

Фактически, Grisu выбирает степени десяти так, что разница лежит в определенном диапазоне. Разные диапазоны дают разные подпрограммы для генерации цифр. Также стоит учесть, что наименьшая разница не всегда является наиболее эффективным выбором.

Определим `diy_fp` для x как беззнаковое целое число f_x, состоящее из q битов, и знакового целого числа e_x неограниченного диапазона. Значение x можно вычислить как $x = f_x \times 2^{e_x}$.

В следующем алгоритме мы параметризуем оставшийся показатель по переменным α и γ. Положим $\gamma \geqslant \alpha + 3$, а затем представим интересные варианты для этих параметров. Для начального обсуждения предположим, что $\alpha := 0$ и $\gamma := 3$.

Реализация. Алгоритм Grisu:

- *Вход:* положительное число с плавающей точкой v точности p.
- *Условие:* точность `diy_fp` удовлетворяет $q \geqslant p + 2$, а кеш степеней десяти

состоит из предварительно вычисленных нормированных округленных diy_fp значений $\tilde{c}_k := \left[10^k\right]_q^\star$

- *Вывод:* строковое представление в основании 10 для V такое, что $[V]_p^\square = v$. То есть V должен быть округлен до v при чтении вновь.

  Шаги алгоритма:

1. *Преобразование:* определим нормированный diy_fp w такой, что $w = v$.
2. *Кэширование степеней:* находим нормированное $\tilde{c}_{-k} = f_c \times 2^{e_c}$ такое, что $\alpha \leqslant e_c + e_w + q \leqslant \gamma$.
3. *Произведение:* пусть $\tilde{D} = f_D \times 2^{e_D} := w \otimes \tilde{c}_{-k}$.
4. *Выход:* определим искомое $V := \tilde{D} \times 10^k$. Вычислим десятичное представление $\tilde{D}$, за которым следует строка "e" и десятичное представление k.

Поскольку значение diy_fp больше, чем значение входного числа, преобразование шага 1 дает точный результат. По определению diy_fp-ы имеют бесконечный диапазон экспоненциальности и показатель степени w, следовательно, достаточно велик для нормирования. Заметим, что показатель e_w удовлетворяет $e_w \leqslant e_v - (q - p)$.

Легко показать, что $\forall i, 0 < \tilde{e}_{c_i} - \tilde{e}_{c_{i-1}} \leqslant 4$, и поскольку кеш неограничен, требуемый $\tilde{c}_{-k}$ должен находиться в кеше. Это является причиной первоначального требования $\gamma \geqslant \alpha + 3$.

Разумеется, бесконечный кеш не нужен. k зависит только от типа номера с плавающей точкой ввода (его диапазон экспоненты), точности diy_fp и пары α и γ.

Результатом Grisu является строка, содержащая десятичное представление $\tilde{D}$, за которым следуют символ "e" и k знаков. Таким образом, он представляет собой число $V := \tilde{D} \times 10^k$. Утверждается, что V дает v при округлении до числа с плавающей точкой с точностью p.

2.4 Описание SSE2

3 Результаты работы и ускорение

4 Заключение

Приложение

Список литературы

[1] Florian Loitsch. Printing Floating-Point Numbers Quickly and Accurately with Integers, 2004.

[2] Wojciech Mula. SSE: conversion integers to decimal representation, 2011.

[3] https://github.com/miloyip/itoa-benchmark/blob/master/readme.md

[4] Богачев К. Ю.. Основы параллельного программирования. – М.: Бином. Лаборатория знаний, 2010.