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 для печати вещественных чисел и SSE2 для печати целых чисел, о которых 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 и последовательную печать готовых буферов в файл.

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

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` и 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 Преобразование чисел в строковый тип

В статье [1] описан алгоритм Grisu и его улучшения, также доказана их точность. 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.

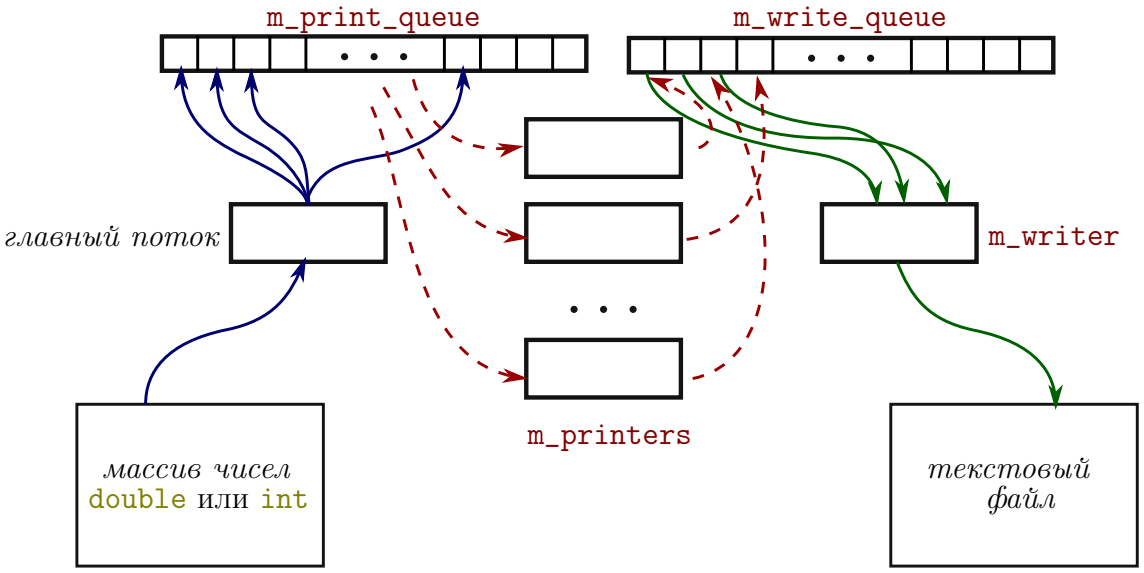

Рисунок 1: Работа потоков.

Идея Grisu состоит в том, чтобы кешировать приблизительные значения $\frac{10^t}{2^{e_t}}$. 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.

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 $x = f_x \times 2^{e_x}$.

Но у Grisu есть недостаток: так число 1 будет напечатано в виде `1000000000000000000e-19`. Поэтому будем использовать Grisu2. Этот алгоритм является усовершенствованием предыдущего и не записывает лишние нули в конец числа. Так если целочисленный тип `diy_fp` содержит более двух дополнительных битов, то эти биты могут использоваться для сокращения выходной строки. В отличие от Grisu, Grisu2 не генерирует полное десятичное представление, а просто вовзвращает цифры (`123`) и соответствующий показатель (`-2`). Затем процедура форматирования объединяет эти данные для получения представления в требуемом формате.

Для преобразования целых чисел используется алгоритм SSE2, о котором по-

дробнее написано в статье [2]. Суть алгоритма заключается в быстром логарифмировании числа по основанию 10.

Кроме того, если в массиве есть n подряд идущих одинаковых чисел x, то будем записывать их как `n*x`. Такая запись может сэкономить память и время работы.

3 Результаты работы и ускорение

Время работы в секундах для массива с разными случайными числами представлено в следующей таблице:

Размер массива	Число потоков				Стандартная печать
	16	12	4	1	
10000000	0.609	0.550	0.880	3.196	4.256
	0.567	0.500	0.841	3.239	4.176
	0.506	0.473	0.802	3.052	4.188
50000000	2.420	2.528	4.044	15.377	22.476
	2.522	2.446	4.273	16.309	21.116
	2.587	2.339	4.179	15.327	20.893
100000000	5.025	4.665	8.276	32.461	41.712
	4.787	4.630	7.970	30.571	41.785
	4.844	4.544	8.078	30.757	41.961

Время работы на массиве с множеством повторяющихся чисел:

Размер массива	Число потоков				Стандартная печать
	16	12	4	1	
10000000	0.318	0.249	0.188	0.652	3.645
	0.334	0.256	0.190	0.629	3.622
	0.307	0.251	0.192	0.661	3.620
50000000	1.657	1.274	0.884	3.183	18.412
	1.505	1.247	0.891	3.167	18.306
	1.522	1.262	0.894	3.175	18.261
100000000	3.105	2.441	1.726	6.306	36.194
	2.983	2.453	1.820	6.329	36.388
	3.246	2.505	1.759	6.339	36.419
500000000	16.194	12.575	8.681	31.505	181.221
	16.414	12.564	8.787	31.291	181.406
	15.815	12.555	8.764	31.690	182.524

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]

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