МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования Национальный исследовательский Нижегородский государственный университет им. Н. И. Лобачевского

Институт информационных технологий, математики и механики

Кафедра дифференциальных уравнений, математического и численного анализа

Направление подготовки
02.04.02. Фундаментальная информатика и информационные технологии

Направленность образовательной программы
магистерская программа «Компьютерная графика и моделирование живых и технических систем»

Отчёт
Отчет по лабораторной работе

на тему:

«Промахи кэша»

Квалификация (степень)
магистр

Форма обучения
очная

Выполнил:
студент гр. 381706 – 2м
Бабаев Иван Владимирович

Нижний Новгород
2018

Оглавление

Введение

Кэш микропроцессора — кэш (сверхоперативная память), используемый микропроцессором компьютера для уменьшения среднего времени доступа к компьютерной памяти. Является одним из верхних уровней иерархии памяти. Кэш использует небольшую, очень быструю память (обычно типа SRAM), которая хранит копии часто используемых данных из основной памяти. Если большая часть запросов в память будет обрабатываться кэшем, средняя задержка обращения к памяти будет приближаться к задержкам работы кэша. Данные между кэшем и памятью передаются блоками фиксированного размера, также называемые линиями кэша (англ. cache line) или блоками кэша. Большинство современных микропроцессоров для компьютеров и серверов имеют как минимум три независимых кэша: кэш инструкций для ускорения загрузки машинного кода, кэш данных для ускорения чтения и записи данных и буфер ассоциативной трансляции (TLB) для ускорения трансляции виртуальных (логических) адресов в физические, как для инструкций, так и для данных. Кэш данных часто реализуется в виде многоуровневого кэша (L1, L2, L3).

Кэш промахи

Cache Miss (промах кэша) случается, когда запрашиваемые данные отсутствуют в кэше и их нужно подгружать из основного источника.

Виды промахов:

- Промах по чтению из кэша инструкций. Обычно дает очень большую задержку, поскольку процессор не может продолжать исполнение программы (по крайней мере, текущего потока исполнения) и вынужден простаивать в ожидании загрузки инструкции из памяти.
- Промах по чтению из кэша данных. Обычно дает меньшую задержку, поскольку инструкции, не зависящие от запрошенных данных, могут продолжать исполняться, пока запрос обрабатывается в основной памяти. После получения данных из памяти можно продолжать исполнение зависимых инструкций.
- Промах по записи в кэш данных. Обычно дает наименьшую задержку, поскольку запись может быть поставлена в очередь и последующие инструкции практически не ограничены в своих возможностях. Процессор может продолжать свою работу, кроме случаев промаха по записи с полностью заполненной очередью.

Профилирование тестовой программы.

Для демонстрации проблемы кэш-промахов была написана небольшая программа на языке C++ (смотри приложение 1), которая прибавляет ко всем элементам матрицы заданную константу и повторяет заданное число раз (30000).

Профилирование производилось на тестовой машине со следующими характеристиками:

Операционная система: Windows 10

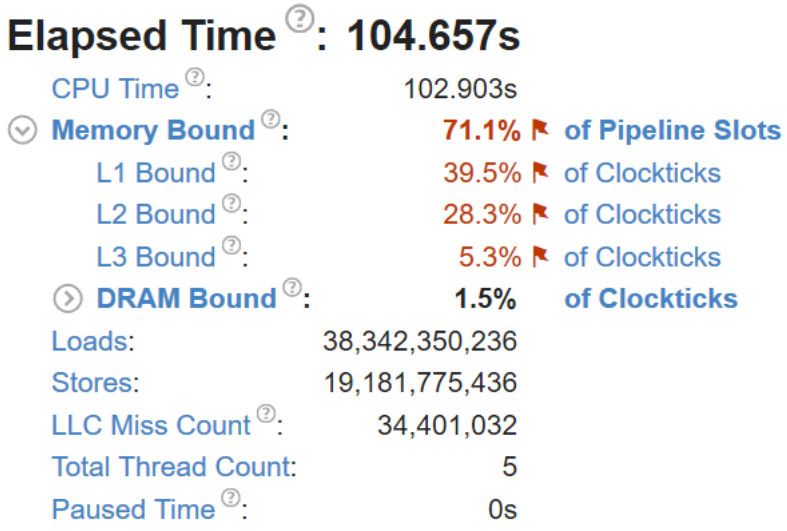
CPU: Intel Core i5-3470, Caches: L1 – 64kb x4, L2- 256Kb x4, L3 – 1024Kb x6.

GPU: NVIDIA GeForce GTX 970, 4058Mb

Оперативная память: 16384Mb DDR3

Профилирование проводилось с помощью инструмента Intel VTune Amplifier Memory Access Tool, который предоставляет набор метрик для выявления проблем, связанных с доступом к памяти.

При запуске анализа работы приложения, Memory Access Tool запускает профилируемое приложение и собирает информацию о его работе. По завершении работы формируется отчет, указывающий критические точки и показывающий общую производительность:



Рассмотрим Memory bound – метрики эффективности операций с памятью:

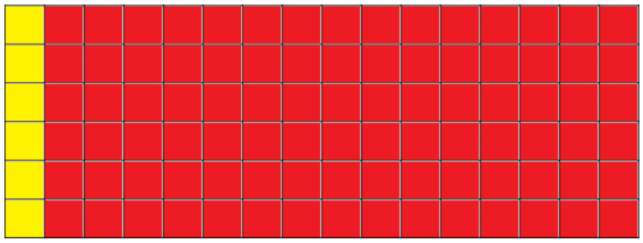
- **L1 Bound** - процент времени, которое процессор ожидал данные с кэша L1.
- **L2 Bound** - процент времени, которое процессор ожидал данные с кэша L2.
- **L3 Bound** – процент времени, которое процессор ожидал данные с кэша L3.
- **DRAM Bound** – процент времени, которое процессор ожидал данные с основной памяти.

В данном случае наблюдается проблема при работе с кэш-памятью. Memory Access Tool указывает места кода, в которых возникает проблема:

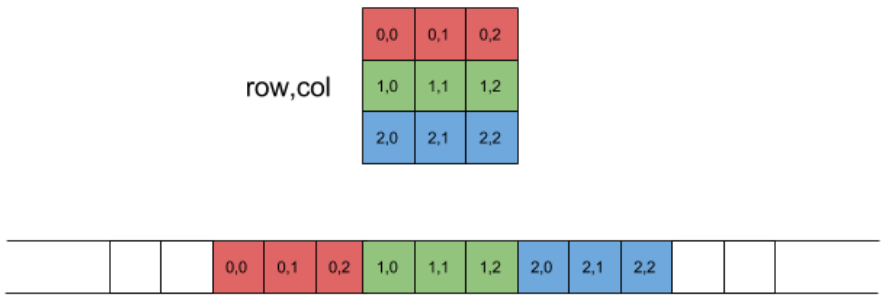
Sou ▲	Source	CPU Time: Total	CPU Time: Self	Memory Bound: Total	Memory Bound: Self			
					L1 Bound	L2 Bound	L3 Bound	DRAM Bound
5								
6	`void AddConstToMatrix(int** x, int n, int m, int c)`							
7	`{`							
8	`    for (int j = 0; j < m; j++)`							
9	`        for (int i = 0; i < n; i++)`							
10	`            x[i][j] = c + x[i][j];`							
11	`}`							
12								
13	`void AddConstToMatrixB(int** x, int n, int m, int c)`							
14	`{`							
15	`    for (int ii = 0; ii < n; ii += 8)`							
16	`        for (int jj = 0; jj < m; jj += 8)`							
17	`            for (int i = ii; i < ii + 8; i++)`							
18	`                for (int j = jj; j < jj + 8; j++)`							
19	`                    x[i][j] = x[i][j] + c;`							
20	`}`							
21								
22	`int main()`							
23	`{`							
24	`    int n = 800;`							
25	`    int m = 800;`							
26	`    int **x;`							
27	`    x = new int *[n];`							
28	`    for (int i = 0; i < n; i++) {`							
29	`        x[i] = new int[m];`							
30	`    }`							
31								
32	`    for (int i = 0; i < n; i++) {`							
33	`        for (int j = 0; j < m; j++) {`							
34	`            x[i][j] = 1-i;`							
35	`        }`							
36	`    }`							
37								
38	`    for (int k = 0; k < 30000; k++)`							
39	`        AddConstToMatrix(x, n, m, 2);`	102.522s	102.522s	71.3%	39.5%	28.4%	5.3%	1.5%
40								
41	`    for (int i = 0; i < n; i++) {`							
42	`        delete[] x[i];`							
43	`    }`							
44	`    delete[] x;`							
45								
46	`    return 0;`							
47	`}`							

```
void AddConstToMatrix(int** x, int n, int m, int c)
{
    for (int j = 0; j < m; j++)
        for (int i = 0; i < n; i++)
            x[i][j] = c + x[i][j];
}
```

В данном случае проблема возникает в функции добавления константы ко всем элементам матрицы. А именно, в размещении памяти. Если получать доступ к кэшу в строковом порядке, то будет использоваться вся память кэша. Если идти по столбцам, то кэш закончится прежде, чем память сможет быть повторно используема.



Двумерный массив представляется в памяти в виде:



Поэтому, если идти по столбцам, то промежутки между данными будут велики и память кэша быстро закончится. Размер входных данных (размер матрицы): 800 * 800 * sizeof(int) = 2500Kb.

Исправим наш пример:

```
void AddConstToMatrix(int** x, int n, int m, int c)
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            x[i][j] = c + x[i][j];
}
```

Запустим Memory Access Tool:

Elapsed Time: 11.614s

CPU Time:	10.948s	
Memory Bound:	**1.4%**	**of Pipeline Slots**
L1 Bound:	0.8%	of Clockticks
L2 Bound:	0.0%	of Clockticks
L3 Bound:	1.1%	of Clockticks
DRAM Bound:	**0.0%**	**of Clockticks**
DRAM Bandwidth Bound:	0.0%	of Elapsed Time
Loads:	37,174,315,196	
Stores:	18,594,157,808	
LLC Miss Count:	0	
Total Thread Count:	4	
Paused Time:	0s	

Обращение по строкам дало значительный прирост производительности и сделало процент ожидания данных с L3 достаточно малым, и обнулило ожидание данных с кэша L2.

Заключение

Тестовая программа показала, что неверное обращение к элементам массива плохо сказывается на чтении\записи данных и влечет потерю производительности из-за долгого чтения. Правильное чтение\запись ускорили работу программы в 9 раз и обнулили промахи кэша L2, практически обнулили промахи кэша L3.

Литература

[1] Intel® VTune™ Amplifier, https://software.intel.com/en-us/vtune-amplifier-help

[2] Department of Computer Science at University of Verona, Cache performance presentation, http://www.di.univr.it/documenti/OccorrenzaIns/matdid/matdid566638.pdf

[3] HOW FAST CAN YOU GO – OPTIMIZING MEMORY CACHE PERFORMANCE CLAIRE CATES DISTINGUISHED DEVELOPER CLAIRE.CATES@SAS.COM, https://www.cmg.org/wp-content/uploads/2015/10/memory_cache.pdf

[4] Why software developers should care about CPU caches, https://medium.com/software-design/why-software-developers-should-care-about-cpu-caches-8da04355bb8a

Приложение 1

Код программы:

```
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <memory>

void AddConstToMatrix(int** x, int n, int m, int c)
{
		for (int i = 0; i < n; i++)
			for (int j = 0; j < m; j++)
				x[i][j] = c + x[i][j];
}

int main()
{
	int n = 800;
	int m = 800;
	int **x;
	x = new int *[n];
	for (int i = 0; i < n; i++) {
		x[i] = new int[m];
	}

	for (int i = 0; i < n; i++) {
		for (int j = 0; j < m; j++) {
			x[i][j] = i;
		}
	}

	for (int k = 0; k < 30000; k++)
		AddConstToMatrix(x, n, m, 2);

	for (int i = 0; i < n; i++) {
		delete[] x[i];
	}
	delete[] x;

	return 0;
}
```