Параллельные и распределенные вычисления (подборка материалов)

Алексей Толстиков, Николай Александрович Лиходед

2 сентября 2021 г.

Содержание

1 Исследование влияния размера матриц и блоков на время реализации блочного алгоритма перемножения матриц на многоядерном CPU

Необходимый для выполнения работы теоретический материал и алгоритмы имеются в файле «Блочное ММ».

Работать над лабораторной можно в кооперации двух студентов.

Для получения максимально информативных результатов в этой лабораторной работе квадратный массив нужно представлять линейным.[1]

Задание

1. Программно реализовать (`C` или `C++`, `OpenMP`) алгоритмы точечного и блочного перемножения матриц (достаточно перемножать квадратные матрицы). В блочном варианте не допускать повторяющихся одних и тех же вычислений.

2. Экспериментально исследовать влияние на время реализации алгоритмов:

   - размеров матриц и блоков (случай $r = 1$ и $r = n$ — обычный точечный алгоритм);
   - выбора цикла $dopar$ (внешний, внутренний), который служит для образования потоков вычислений;

3. Сравнить время реализации точечных алгоритмов с временем реализации блочных алгоритмов. Сравнивать как последовательные, так и параллельные версии программ.

Результаты экспериментов представить в виде графиков и таблиц.

Для заполнения матриц A и B использовать случайные числа (целые или вещественные) из диапазона от -100 до 100. Для получения случайных чисел использовать библиотечную функцию rand(), подключив заголовочный файл `stdlib.h`, или функции из заголовочного файла `random.h (C++11 и выше)`. Для вычислений выбрать параметры:

- N_1, N_2, … — размеры матриц (рассмотреть не менее двух наборов размеров матриц: небольшие размеры (до 500, если матрица квадратная) и размеры побольше (например, от 1500 до 2000), рукомендуется рассматривать квадратные матрицы;
- отдельно обратить внимание на матрицы со сторой равной степени двойки;
- r — размер блоков, рассмотреть несколько случаев: единицы (1, 2, 5), десятки (10, 15, 20, 30, 50), сотни (100, 200, 500, n), другие размеры блока по собственному усмотрению;
- уточнить оптимальное значение r разбив необходимый интервал более подробно, например, это мог бы быть интевал $50..100$, тогда можно в нем выбра значения с шагом 5.

`OpenMP` поддерживается большинством компиляторов, поэтому для использования достаточно выставить соответствующие флаги компилятора:

```
gcc -fopenmp
Intel   -openmp (Linux, MacOSX), -Qopenmp (Windows)
Microsoft   -openmp (настройки проекта в Visual Studio)
```

[1] В данном случае вместо квадратной матрицы $n \times n$ используется линейный массив из n^2 элементов. Первые n элементов соответствуют первой строке матрицы, далее n элементов – второй строке матрицы, последние n элементов – последней строке матрицы.

`Microsoft Visual C++ 2005 и выше` поддерживает `OpenMP 2.0` в редакциях Professional и Team System, 2010 — в редакциях Professional, Premium и Ultimate, 2012+ — во всех редакциях.

Для включения поддержки `OpenMP` в Visual Studio нужно в свойствах проекта в категории «Configuration Properties» ⟶ «C/C++» ⟶ «Language» выставить опцию «Open MP support» в значение «Yes». Это нужно проделать отдельно для Debug и Release конфигураций (или сразу выбрать конфигурацию Active).

Внимание: если не включить поддержку `OpenMP`, то программа скомпилируется без ошибок, но будет работать последовательно.

Содержание работы должно включать следующие пункты.

1. Входные данные: выбранные параметры.
2. Листинг программы.
3. Скрипт запуска всего эксперимента и получения лога или полного отчета сразу.
4. Выходные данные: графики и (или) таблицы (Excel или iPython предпочтительно). **Запуск в Release сборке.**
5. Проверку корректности работы алгоритмов.

Особенности задания в 2021 году

Вы можете попробовать использовать `go`, `Rust`, `java`, `numpy.array` в Python или других языках.

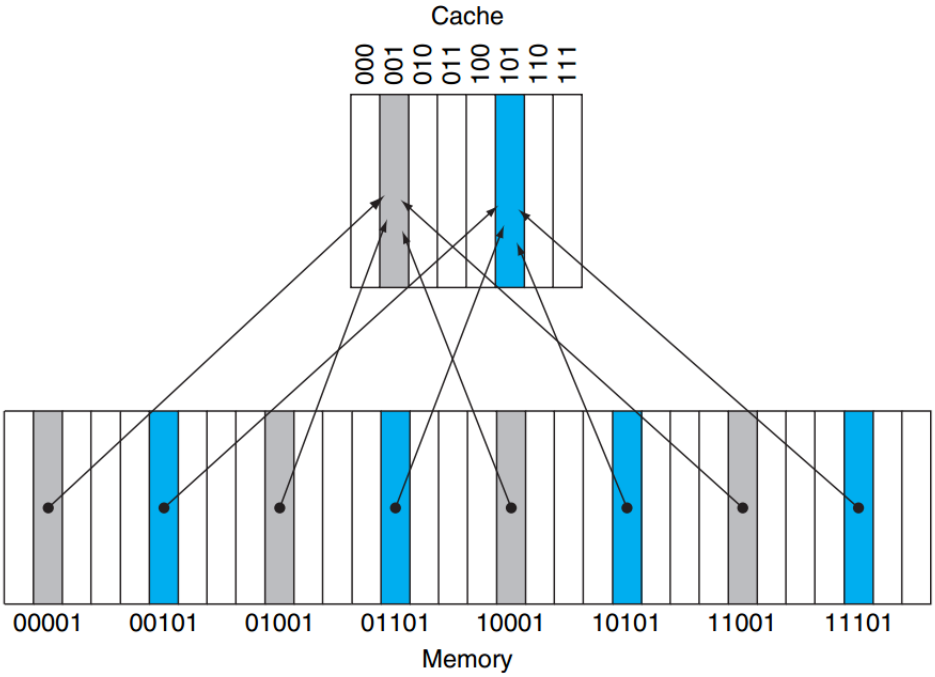
Код должен быть покрыт `unit`-тестами для выбранного языка, где-то для этого есть стандартные инструменты, а где-то нужно скачать и использовать дополнительные библиотеки.

1.1 Эмуляция работы кэша[2]

Необходимо проэмулировать работу k-way полностью ассоциативного кэша размера $cacheSize$ и с размером кэш-линии $lineSize$.

Для выполнения эмуляции будем:

- Обрабатывать адреса памяти один за одним, при этом считая, что каждый раз у нас запрашивается только один байт из памяти по данному адресу.
- Изначально весь кэш пустует.
- Если случается `CACHE_MISS` и необходимо вытеснить одну из линий в кэше, то будем вытеснять самую давно не используемую линию. Т.е. в момент записи линии и в момент `CACHE_HIT` ее внутренний счетчик времени выставляется в текущий момент.
- Для определения номера кэш-линии необходимо разделить $address$ на $lineSize$ без остатка.
- Для определения номера набора в кэше необходимо вычислить остаток от деления номера кэш-линии на количество кэш-линий в одном банке памяти в кэше.



В первой строке входных данных записаны четыре целых числа $cacheSize$, $associativity$, $lineSize$ и n ($2^{15} \le cacheSize \le 2^{24}$, $associativity \in \{4, 6, 8, 12, 16\}$, $lineSize \in \{64, 128\}$, $1 \le n \le 100000$).

Вторая строка содержит n целых чисел, последовательность адресов, к которым выполняется обращение.

Гарантируется:

- $cacheSize = associativity \cdot 2^k$, для некоторого натурального k.
- Все адреса в памяти из диапазона $[0; 2^{30} - 1]$.

Требуется выведите два числа, количество обращений к памяти, которые завершились `CACHE_HIT` и `CACHE_MISS` соответственно.

[2]Задача является дополнительной, желающие будут сдавать решение в систему автоматической проверки.

1.2 Быстрое транспонирование[3]

Требуется выполнить несколько операций транспонирования квадратной подматрицы данной матрицы A. Матрицу A будем представлять линейным массивом d, т.е. $A_{i,j} = d_{i \cdot n + j}$. Индексация строк и столбцов матрицы A начинается с 0.

Так как наша цель быстро выполнять именно операцию транспонирования, то входную матрицу нужно сгенерировать следующим генератором:

```
vector<int> generate_input(int n, int seed) {
    vector<int> d(n * n);
    for (size_t i = 0; i < d.size(); ++i) {
        d[i] = seed;
        seed = ((long long) seed * 197 + 2017) \& 987654;
    }
    return d;
}
```

Кроме этого выводить в результате саму матрицу не требуется, а нужно вывести только результат работы следующей функции:

```
long long get_hash(const vector<int>& d) {
    const long long MOD = 987654321054321LL;
    const long long MUL = 179;

    long long result_value = 0;
    for (size_t i = 0; i < d.size(); ++i)
        result_value = (result_value * MUL + d[i]) & MOD;
    return result_value;
}
```

В первой строке входных данных записаны два целых числа n и $seed$ ($1 \le n, seed \le 10000$). Вторая строка содержит число k ($1 \le k \le 10$) — количество операций транспонирования. Каждая из последующих k строк содержит описание подматрицы, которую нужно транспонировать: i_{min}, j_{min} и $side$ ($0 \le i_{min}, j_{min} < n, 1 \le size \le n - \max i_{min}, j_{min}$).

Требуется вычислить результат вызова функции `get_hash` с параметром d равным результату транспонирования исходной матрицы k раз.

[3]Задача является дополнительной, желающие будут сдавать решение в систему автоматической проверки.