Отчёт

по Лабораторной работе

«Параллельный вариант алгоритма «Решето Эратосфена»

Преподаватель: Кулябов Дмитрий Сергеевич

Выполнил: Кремер Илья

Группа: НК-401

Москва, РУДН – 2011

Оглавление

Постановка задачи и её математическое решение

Условие задачи можно сформулировать следующим образом: «Найти все простые числа до заданного n включительно». Требуется также написать демонстрационную программу на Fortran, эффективно использующую технологии параллельного программирования.

Решето Эратосфена – это один из самых известных алгоритмов нахождения всех простых чисел до заданного n. Он имеет следующую схему:

- Выписываются все числа от 2 до n.
- Берётся первое простое число (это будет 2) и вычёркиваются все числа, ему кратные
- Берётся следующее простое число (это будет 3) и вычёркиваются все числа, ему кратные
- И так до тех пор, пока это возможно
- В итоге в последовательности остаются только простые числа

Этот алгоритм получил название «решето», потому что этот процесс напоминает некое просеивание чисел. Рассмотрим ход работы алгоритма на примере n = 26:

2 3 4 5 6 7 8 9 10 11 12 13 14
15 16 17 18 19 20 21 22 23 24 25 26

2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~
15 ~~16~~ 17 ~~18~~ 19 ~~20~~ 21 ~~22~~ 23 ~~24~~ 25 ~~26~~

После первого прохода мы вычеркнули все чётные числа, оставив только двойку. Дальше возьмём 3 (для краткости вычеркнутые числа удалены из след. записи) и пройдёмся по всем числам, кратным тройке:

2 **3** 5 7 ~~9~~ 11 13 ~~15~~ 17 19 ~~21~~ 23 25

Следующее действия – взять пятёрку и пройтись по всем числам, кратным 5:

2 **3** **5** 7 11 13 17 19 23 ~~25~~

Вышла последовательность:

2 **3** **5** **7** **11** **13** **17** **19** **23**

в которой все числа – простые. Дальше по описанному алгоритму мы можем взять 7 и вычеркнуть все числа кратные семи, взять 11 и вычеркнуть все числа кратные 11-ти и т.д., но таких чисел больше конечно уже не встретится. Перестать отсеивать числа необходимо, когда взятое простое число оказалось больше, чем целая часть корня из n. В данном случае, последнее число, которое мы рассмотрели, было 5; число 7 можно не брать, т.к. оно больше корня из 26.

Программное решение

Изложенный выше алгоритм программно можно представить следующим образом:

Пусть `primes` – массив элементов булевого типа, изначально заполненный значениями `true`. Начнём идти по нему в цикле от `2` до корня из `n`, и, встречая `true`, вычёркивать из него (выставлять `primes[i] = false`) все числа с этим шагом до `n`.

В результате получится массив, у которого элементы `true` указывают на то, что индекс этого элемента – простое число. То есть для предыдущего примера (`n = 26`), элементы `primes[2]`, `primes[3]`, `5`, `7`, `11`, `13`, `17`, `19` и `23` будут иметь значение `true`, а остальные – `false`. Имея такой массив, нетрудно вывести все индексы в файл, список или на экран в качестве конечного ответа.

Схематично изобразим такой алгоритм:

```
// до этого момента все primes[i] равны true
for (int i = 2; i * i <= n; i++) {
    if (primes[i]) {
        for (int j = i * i; j <= n; j += i) {
            // начиная с первого кратного числа и до n
            primes[j] = false;
        }
    }
}

for (int i = 2; i <= n; i++) {
    if (primes[i]) {
        // число i - простое
    }
}
```

Распараллеливание

Очевидно, что итерации по `i` не зависят друг от друга, из-за чего сразу же возникает желание распараллелить этот цикл, использую автоматические средства вроде тех, что реализованы в OpenMP. Однако добавить строчку `#pragma omp for` прямо над строчкой цикла не получится, OMP-расширение компилятора укажет на ошибку в строчке `for (int i = 2 ...)`, т.к. оно должно без каких-либо подсчётов узнать, сколько итераций содержит цикл. Условие `i * i <= n` мешает ему это сделать.

Но так или иначе перейдём сразу к кодам на языке Fortran, т.к. по заданию программа должна быть реализована на нём. Нам придётся воспользоваться циклом `do-while`, для того, чтобы перевести с Си строчку с циклом по `i`:

```
i=2
do while (i * i <= maxnum)
    if (primes(i)) then
        do j = i * i, maxnum, i
            primes(j) = .FALSE.
        enddo
    endif
    i=i+1
enddo
```

И тогда мы сразу получим код, так же синтаксически не поддающийся распараллеливанию из-за `do-while`. Решение очень простое: введём переменную `limit` равную целой части от корня из `maxnum`. В самом начале говорилось «идти по нему в цикле от `2` до корня из `n`», фактически это оно и будет, важно только заметить, что там должно стоять округление не `round` или `ceil`, а `floor`. Возьмём число `maxnum = 15` в качестве примера. `limit = nint(sqrt(15.0))` даст нам `4`, хотя по условию i^2 `<= 15`, `i` не должно превосходить `3`. Программа не станет выдавать некорректный результат, но это будет добавлять ненужные итерации.

```
limit = int(sqrt(maxnum + 0.0))
do i = 2, limit
    if (primes(i)) then
        do j = i * i, maxnum, i
            primes(j) = .FALSE.
        enddo
    endif
enddo
```

Полученный код легко распараллеливается директивой `omp do`:

```
!$omp parallel shared(primes) private(i, j, limit)
    limit = int(sqrt(maxnum + 0.0))
!$omp do schedule(static, 1)
    do i = 2, limit
        if (primes(i)) then
            do j = i * i, maxnum, i
                primes(j) = .FALSE.
            enddo
        endif
    enddo
!$omp end parallel
```

Поскольку одновременного обращения к элементам массива `primes` быть не может, мы не боимся сделать его `shared`[1]. Это сразу убирает задачу объединения кусков массива в один после прогона цикла. Переменную `limit` мы также добавляем в список `shared`, чтобы максимально предотвратить гонку данных, хотя трудно переоценить крохотный шанс её возникновения в данном случае. Отметим, что ситуация с `limit` – хороший пример для использования директивы `firstprivate`, однако оставим представленный вариант с тем, чтобы элементарно сократить длину первой строчки.

Эффективность

Ряд запусков на современном ПК[2] показал, что при больших `maxnum` (десятки миллионов) наиболее выигрышным для любого количества процессов оказывается распределение `schedule(dynamic, 1)`. Любые отличные от `1` значения `chunk` (особенно большие) сразу начинают снижать скорость. Такая же ситуация просматривается и со статичным распределением итераций.

Любопытно, что очень незначительный – около 10% (если сравнивать 1 и 8 процессов) прирост производительности даёт автоматическое `auto` и при любом количестве процессов наиболее эффективный даёт автоматический `runtime`; его использование практически неотличимо от `schedule(dynamic)` на максимальном (соответствующем количеству логических ядер) количестве процессов, но в конечном варианте программы был выбран всё-таки этот тип, т.к. на промежуточных значениях количества нитей он немного (10% разницы) выходил вперёд.

---

[1] Специально оставленный в исходном коде комментарий демонстрирует это утверждение.
[2] Процессор: Intel i7 860, компиляция: gfortran -fopenmp (из MinGW), ОС: Windows 7.

Нитей	static	dynamic	dynamic, 64	runtime
1	6.00	6.00	6.00	6.00
2	5.15	3.25	3.50	3.15
4	3.00	2.20	3.30	2.15
8	2.55	2.05	3.70	2.05

Таблица, приведённая выше показывает время выполнения алгоритма для в секундах для `maxnum = 100.000.000`. Для заметно более меньших чисел, напр. `200.000` общая тенденция сохраняется.

Комментарии к прикреплённой программе

Разберёмся сначала с общей структурой программы. Выполнение алгоритма происходит в теле (`program LAB2_PRIMES`) , там объявляются все необходимые переменные, названия которых соответствуют названиям, используемым в этой работе. Следует только пояснить нетронутые в отчёте:

```
integer num_threads
double precision start
character(20) argv
```

Первая – конечно же, количество нитей. Это значение, как и `maxnum` вводится с клавиатуры. Вторая служит для замера времени: после того, как «процедура отсева» будет выполнена, мы вычтем из `omp_get_wtime()` это значение, узнав, таким образом, затраченное время. Третья переменная нужна для записи аргумента командной строки.

Мы используем динамическое выделение памяти под массив `primes`, чтобы позволить человеку самому ввести его размер с клавиатуры.

Перед тем, как открыть параллельную область установим выбранное пользователем количество процессов с помощью функции `omp_set_num_threads()`. При вводе ограничимся 24, т.к. для новейших мощных ПК этого явно предостаточно, а использовать данную программу на СК не имеет смысла, по крайней мере предварительно не переписав её для `integer(kind=8)` или `real`.

Вычислим `limit` так, как об этом говорилось в середине отчёта и тут же начнём параллельно выполнять цикл. Раскомментируйте строчку вывода, чтобы увидеть какие числа «вычёркивает» каждый процесс.

Цикл закончен – вычисляем и тут же выводим время его работы.

Далее нам остаётся только решить, что делать с полученной последовательностью. Прикреплённая программа позволяет выбирать пользователю из 4 вариантов:

- Не выводить ничего
- Выводить числа таблицей шириной в 8 чисел, рассчитанной максимум на 10-значные числа – это позволяет работать со стандартной шириной терминала в 80 символов
- Выводить числа в столбик
- Выводить числа в файлы, максимум по 50.000 чисел на файл

Для того, чтобы выбрать каждый из вариантов следует:

- Запускать с аргументом `-q`
- Запускать без аргументов
- Запускать с аргументом `-column`
- Запускать с аргументом `-file`

Выводом чисел занимается подпрограмма `print_answer`, её аргументы перечислены в комментариях внутри кодов. Она выводит в консоль предупреждение, если вывод в файл может занять несколько продолжительное время. Однако следует учесть, что вывод в файлы в любом случае на порядок быстрее, чем вывод в консоль.

Перед выходом из программы освобождаем память, выделенную под `primes`.

В прикреплённых снимках показана работа программы (тесты в Windows 7, о которых шла речь выше производились отдельно, так что несоответствие скоростям, представленным в таблице закономерно) в обычном режиме и с выводом в файлы.

Снимки экрана

```
kremchik@VirtualBox: ~/Documents/My parallel/labs
File  Edit  View  Search  Terminal  Help
kremchik@VirtualBox:~/Documents/My parallel/labs$ gfortran -fopenmp primes.f90
kremchik@VirtualBox:~/Documents/My parallel/labs$ ./a.out
  Используйте:
    -q       чтобы скрыть вывод ответа
    -column  чтобы вывести числа в столбик
    -file    чтобы выводить числа в файлы

 Нитей:          1
 Докуда считать: 60

 Время вычисления (сек):    0.000096

         2         3         5         7        11        13        17        19

        23        29        31        37        41        43        47        53

        59

 Всего:        17
kremchik@VirtualBox:~/Documents/My parallel/labs$ ./a.out
  Используйте:
    -q       чтобы скрыть вывод ответа
    -column  чтобы вывести числа в столбик
    -file    чтобы выводить числа в файлы

 Нитей:          3
 Докуда считать: 60

 Время вычисления (сек):    0.000253

         2         3         5         7        11        13        17        19

        23        29        31        37        41        43        47        53

        59

 Всего:        17
kremchik@VirtualBox:~/Documents/My parallel/labs$
```

```
kremchik@VirtualBox: ~/Documents/My parallel/labs
File  Edit  View  Search  Terminal  Help
kremchik@VirtualBox:~/Documents/My parallel/labs$ ls
a.out  primes.f90
kremchik@VirtualBox:~/Documents/My parallel/labs$ ./a.out -file
  Используйте:
    -q       чтобы скрыть вывод ответа
    -column  чтобы вывести числа в столбик
    -file    чтобы выводить числа в файлы

 Нитей:          1
 Докуда считать: 1000000

 Время вычисления (сек):    0.067504

 Числа выведены в файл(ы) _prime_numbers_partXX.txt


 Всего:     78498
kremchik@VirtualBox:~/Documents/My parallel/labs$ ls
a.out  _prime_numbers_part000.txt  _prime_numbers_part101.txt  primes.f90
kremchik@VirtualBox:~/Documents/My parallel/labs$ tail _prime_numbers_part101.txt
    999883
    999907
    999917
    999931
    999953
    999959
    999961
    999979
    999983

kremchik@VirtualBox:~/Documents/My parallel/labs$
```