Национальный исследовательский ядерный университет
«МИФИ» (Московский Инженерно–Физический Институт)
Кафедра №42 «Криптология и кибербезопасность»

ОТЧЁТ

Лабораторная работа №4:
«Технология OpenMP. Особенности настройки»

Группа Б21-525
Студент Р.Т. Мясников
Преподаватель М.А. Куприяшин

Москва 2023

Оглавление

1. Описание рабочей среды

- Модель процессора:	Intel Core i3-10110U CPU @ 2.10GHz
- Число ядер:	2
- Число потоков:	4
- Архитектура:	x86-64
- ОС:	Linux, дистрибутив Ubuntu v20.04
- RAM объем:	2x8192 MB
- RAM тип:	DDR4
- Среда разработки:	Visual Studio Code
- Компилятор:	gcc v9.4.0
- Версия OpenMP:	201511

2. _OPENMP

При помощи переменной предпроцессора `_OPENMP` определена дата принятия используемого стандарта `OPENMP`.

Вывод программы - `201511`. Дата записывается в формате `yyyymm`, то есть версии `201511` соответствует дата - `ноябрь 2015`.

3. Процессы и потоки

Использованы функции `omp_get_num_procs()` и `omp_get_max_threads()` для определения числа доступных процессов и потоков.

Вывод программы - `num_procs: 4; max_threads: 4`.

4. Dynamic

Опция `dynamic` используется для управления способом распределения и выполнения итераций циклов в параллельных конструкциях for. Когда эта опция используется, итерации цикла динамически распределяются между потоками во время выполнения программы.

Когда цикл запускается с опцией dynamic, итерации разделяются между потоками таким образом, что каждый поток берет одну итерацию из очереди и продолжает выполнение. Это позволяет эффективно распределять неравномерные или неизвестные заранее нагрузки на итерации цикла между потоками.

Использована функция `omp_get_dynamic()`:

Вывод программы - `0`

5. Таймер

Определено разрешение таймера при помощи `omp_get_wtick()`.

Вывод программы - `0.000000001 [c]`

6. Особенности распределения нагрузки

OpenMP поддерживает вложенные параллельные области, что означает возможность создания параллельных конструкций внутри других параллельных участков кода. По умолчанию OpenMP может ограничивать количество потоков во вложенной области параллелизма так, чтобы не создавать излишнюю нагрузку на систему. Стоит учитывать, что использование вложенных параллельных областей может вызвать дополнительные накладные расходы из-за создания и управления дополнительными потоками.

Определены текущие настройки среды при помощи функций `omp_get_nested()` и `omp_get_max_active_levers()`.

`omp_get_max_active_levels()` - возвращает максимальное количество уровней вложенной параллельности, которые разрешены в данной реализации OpenMP на вашей системе.

`omp_get_nested()` - запрашивает состояние флага, разрешающего вложенный параллелизм.

Вывод программы - `nested: 0; max levels: 1`

7. Распределение нагрузки

Опция `schedule` определяет конкретный способ распределения итераций данного цикла:

`STATIC` - блочно-циклическое распределение итераций: первый блок из m итераций выполняет первый поток, второй блок - второй и т.д. до последнего потока, затем распределение снова начинается с первого потока; по умолчанию значение m равно 1;

`DYNAMIC` - динамическое распределение итераций с фиксированным размером блока: сначала все потоки получают порции из m итераций, а затем каждый поток, заканчивающий свою работу, получает следующую порцию из m итераций;

`GUIDED` - динамическое распределение итераций блоками уменьшающегося размера; аналогично распределению `DYNAMIC`, но размер выделяемых блоков все время уменьшается, что в ряде случаев позволяет аккуратнее сбалансировать загрузку потоков;

Получены настройки среды с использованием функции `omp_get_schedule`.

Вывод программы - `the dynamic schedule is applied, with chunks made of 1`

8. Механизм явной блокировки

Разработан пример вычислительного алгоритма, использующий механизм явной блокировки.

Описание алгоритма

Дано два значения: `numeral` и `max_number`. Найти все пары чисел от 1 до `max_number`, последняя цифра НОДа которых ровна `numeral`. Ответ записать в файл.

При записи значений в файл необходимо использовать блокировку, иначе возникнет состояние гонки, при котором потоки будут пытаться изменить общий ресурс одновременно.

Реализация

```
omp_lock_t writelock;
omp_init_lock(&writelock);

#pragma omp parallel for
for (int i = 1; i < max_number; ++i)
{
    for (int j = i; j < max_number; ++j) {
        int a = j;
        int b = i;

        while (a != b) {
            if (a > b) {
                a -= b;
            } else {
                b -= a;
            }
        }

        if (a % 10 == numeral) {
            omp_set_lock(&writelock);
            fprintf(fd, "%d and %d - %d\n", i, j, a);
            omp_unset_lock(&writelock);
        }
    }
}
```

9. Сравнение различных типов разделения нагрузки

Для сравнения различных типов разделения нагрузки использован алгоритм из лабораторной работы 3 - `shell sort`.

Результаты измерений

Следующие таблицы содержат полученные в результате эксперимента данные: среднее время работы для различного числа потоков и конфигураций массивов.

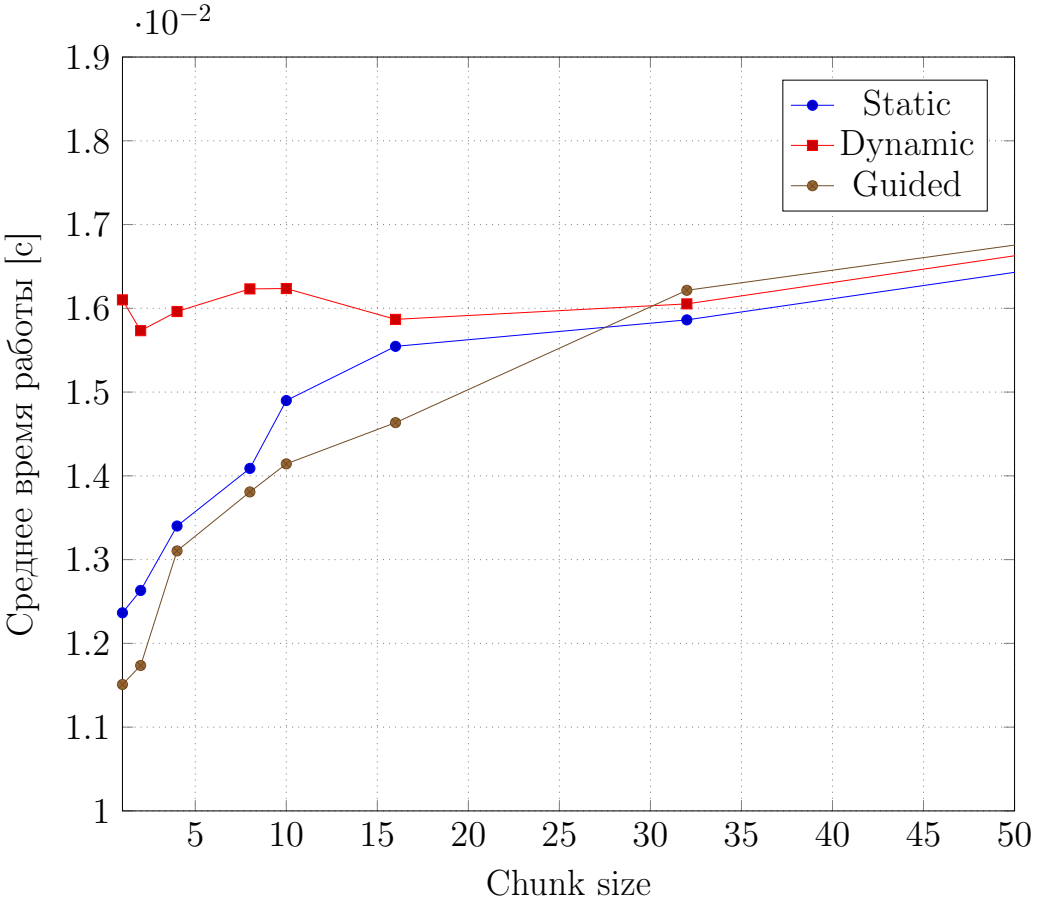
chunk	static
1	0.012365
2	0.012632
4	0.013401
8	0.014089
10	0.014899
16	0.015546
32	0.015862
100	0.018006
500	0.020852

chunk	dynamic
1	0.016101
2	0.015733
4	0.015962
8	0.016232
10	0.016236
16	0.015868
32	0.016053
100	0.018222
500	0.020596

chunk	guided
1	0.011509
2	0.011736
4	0.013104
8	0.013808
10	0.014143
16	0.014635
32	0.016215
100	0.018255
500	0.020820

Графики

Зависимость времени работы от размеров фрагментов



Результаты

Анализируя результаты эксперимента, можно обратить внимание на следующее:

- наибольшая `скорость` была получена при использовании `guided` при `1 chunk size`: `0.011509 [c]`;
- `dinamic` на всех значениях `chunk size` показывала близкие результаты времени;
- для значений `chunk size` 32 и больше значения времени для всех трёх типов разделения нагрузки близки;
- высокие значения времени `dynamic` относительно `static` и `guided` при малых `chunk size` обусловлены тем, что в реализованном алгоритме нагрузка на все потоки распределена равномерна, из-за чего нагладные расходы `dinamic` излишни;

10. Заключение

В ходе лабораторной работы были изучены основные настройки среды `openMP`: определена версия стандарта, определено число процессов и потоков, выяснено назначение опции `dynamic`, определено разрешение таймера, уточнены особенности работы со вложенными параллельными областям, уточнены особенности распределения нагрузки, разработан пример вычислительного алгоритма (использующего механизм явных блокировок), для алгоритма из работы 3 повторены вычислительные эксперименты для разных типов разделения нагрузки и размеров фрагмента-разработан алгоритм.

11. Приложение

Код программы расположен на github
Запуск программы: `./run`