Распределение вычислительной нагрузки в многопроцессорной вычислительной системе

1. О сборке и запуске

Сборка программы основывается на *make*. В ***Makefile*** целью по умолчанию стоит сборка и компиляция исполнительного файла с основным алгоритмом системы (название цели – *main_proc*). Название получаемого файла – ***solution***. Первым аргументом программа получает входной файл (без первого аргумента программа выдаст предустановленную ошибку), второй аргумент указывается опционально, в нем содержится количество процессов, которое пользователь хочет задействовать при решении задачи (по умолчанию = 1). Пример вызова: *./solutuion file_1.xml 10*. Цели *main_proc.o* и *xml_worker.o* собирают объектные файлы из соотв. файлов с кодом. Цель *drawer* собирает и компилирует исполнительный файл, содержащий алгоритм отрисовки диаграммы. Итоговый исполнительный файл – ***picture***. При запуске первым аргументом следует указать соотв. Log-файл (подробнее см. п.4). Последняя цель – *clean*. Она чистит директорию от всех объектных и исполнительных файлов, порожденных всеми иными целями.

2. Об используемом формате входного файла

При разработке решения был выбран следующий формат входного файла: см рис.1

Длина указываемого списка процессоров и программ может быть произвольной, а вот длина списка обменов – уже нет. Его длина должна быть равна (для успешной работы алгоритма) *n_prog*(n_prog-1)/2*, где *n_prog* – длина списка программ. Список обменов задает значение интенсивности обмена для всех пар программ (их как раз *n_prog*(n_prog-1)/2*). Причем первые *(n_prog-1)* значений в нем задают интенсивность обменов первой программы со всеми остальными, следующие *(n_prog-2)* обменов задают интенсивность обмена второй программы со всеми, начиная с третьей, и так далее. Соотв. последний элемент этого списка задает интенсивность обмена последней и предпоследней программы из списка программ. Если файл не будет соотв. этому формату, или же значения *val* в какой-то из секций окажутся неправильными (по условию задачи разных значений может быть небольшое количество), то будет выведено стандартное сообщение об ошибке. Кроме того, мы добавили значение 0 для интенсивности обмена, которое обозначает, что программы не ведут обмен.

```
<?xml version="1.0" encoding="UTF-8"?>
<network>
   <processors>
      <список процессоров в виде:>
      <processor>val</processor>
   </processors>

   <programs>
      <список программ в виде:>
      <program>val</program>

      <links>
         <список обменов в виде:>
         <link>val</link>
      </links>
   </programs>
</network>
```

Рис. 1

3. *Об особенностях работы алгоритма*

Комментарии в программе иногда отсылают к файлу README, большая часть из них относится к структуре входного файла и особенностям вычисления целевой функции нагрузки на сеть, которая непосредственно использует массив интенсивности обменов между программами. В программе массив интенсивности обменов хранится как массив длиной *(n_prog-1)*, его членами являются массивы с числами. Первый из них – длиной *(n_prog-1)*, в котором хранятся первые *(n_prog-1)* значений интенсивностей, последующие *(n_prog – 2)* значений из файла будут храниться во втором члене-массиве. Последний, *(n_prog-1)-ый* член-массив будет содержать одно число, соотв. последнему обмену в списке обменов из файла. Соотв. все вычисления целевой функции проводятся по следующей схеме: (см. рис. 2).

```
for (int i = 0; i < n_prog-1; i++){
   for (int j = i+1; j < n_prog-i-1g; j++){
      <действия с элементом [i][j]>
   }
}
```

Рис. 2

Основной алгоритм реализован в двух функциях: ***find_best*** и ***find_best_parallel*** (см. файл ***main_proc.cpp***) для однопроцессного и многопроцессного вычисления соответственно. В многопроцессном алгоритме процессы синхронизируются и хранят «общие» данные при помощи неименованных каналов. Структура вектора ответов, выдаваемого этими функциями следующая: *n_prog+2* значений, первое – найденное наилучшее значение функции, второе – число итераций, и последующие *n_prog* – вектор распределения программ по процессорам. В случае многопроцессной реализации кол-во итераций по всем процессам суммируется. В результате выводятся все требуемые в условии задачи данные, дополнительно к которым выводится время в миллисекундах.

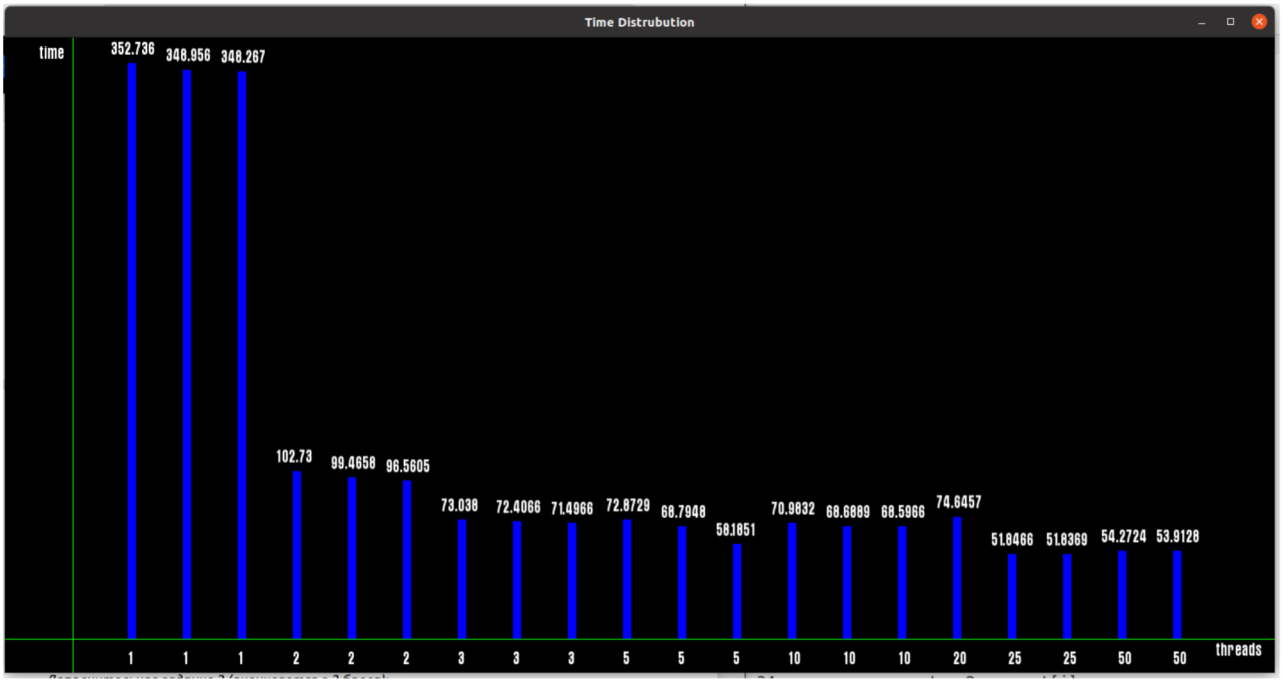
4. *О парсинге входного файла*

Разбор входного файла осуществляется с использованием библиотеки *Xerces* (The Apache Xerces™ Project - xerces.apache.org). Все, что связано с парсингом, помещено в отдельный модуль: ***xml_worker.cpp***. В нем содержится функция ***make_parsing***, которая заполняет переданные ей в качестве параметров по ссылке переменные основной программы значениями из файла. Также эта функция осуществляет контроль соответствия входного файла описанному в п.3 формату: в ней прописаны все стандартные ошибки.

5. *О подсчете времени и отрисовке диаграммы*

Основная программа также отсчитывает время выполнения *основной части алгоритма,* то есть только то время, которое было затрачено на выполнение функций ***find_best*** и ***find_best_parallel***. Подсчет времени осуществляется с помощью функции *clock_gettime*. Также в функции печати (***print_res***) время и количество процессов, которое было задействовано в программе записывается в специальный файл, название которого формируется по правилу: *“log_time_” + <название входного файла>*. Если программа впервые запускается на конкретном входном файле, то соотв. файл будет создан.

После того как пользователь несколько раз запустит программу (хотя бы 1), станет доступна возможность отображения диаграммы по всем проведенным им запускам программы на конкретном файле. Для отрисовки написана отдельная программа – ***drawer.cpp***. Основным инструментом стала библиотека *SFML*. Диаграмма выводится в следующем виде: (см. рис.3)



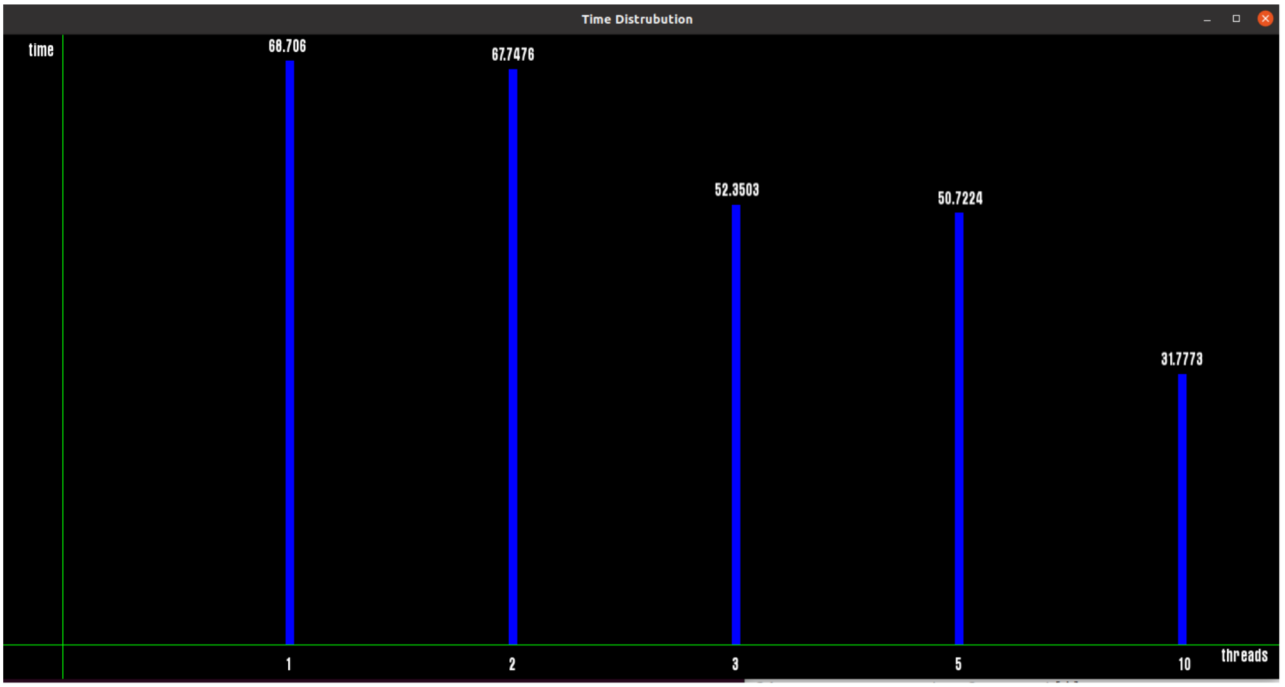


Рис.3

Высота столбика пропорциональна времени выполнения соответствующего запуска программы. В задании сказано провести исследование в процентах. Однако, если запусков программы на одном и том же количестве процессов было несколько, не совсем понятно относительно какого запуска программы считать процент, поэтому был выбран

изображенный на двух примерах из рис.3 формат отрисовки (с абсолютным временем в миллисекундах). Отрисовку в процентах можно сделать, поменяв буквально пару строк кода. Столбики располагаются равноудаленно друг от друга по горизонтали, под столбиком подписано число потоков(они отсортированы слева направо в порядке возрастания), участвовавших в вычислении соотв. результата, над столбиком написано время, которое было занято выполнением основного алгоритма соотв. запуска программы.

Для того, чтобы скомпилировать файл с кодом для отрисовки, как было сказано в п.1 в *Makefile* есть цель *drawer*. Получаемый исполнительный файл – ***picture***. При его запуске первым аргументом следует указать соотв. файл *log_time_ + <имя некого входного файла>.* Пример вызова: *./picture log_time_file_1.xml*

6. О наборе тестов

Для демонстрации корректной работы программы к ней прикладывается набор тестов. Есть основные тесты. Они названы следующим образом: *test_<число процессоров>_1_<№ теста>.xml*. В них содержатся тестовые наборы данных, соответствующие условиям из пункта 5.2. задания. На рис. 3 приведены диаграммы, построенные по двум из них. В каких-то из них решение находится, в каких-то – нет. Есть тест *test_demo_ideal.xml*, который моделирует ситуацию, когда находится нулевое значение целевой функции. Также есть 6 тестов, названных *test_err_<№ теста>*, которые демонстрируют корректную отработку ошибок. Все тесты прошли проверку под *valgrind*, однако при формировании файлов для диаграмм эти запуски в итоге не использовались, так как *valgrind* сильно увеличивает время работы программы.