УДК 004.41

Путинцев И.А., Варданян А.Э., Простов Д.И., Романенко М.М., Красильников И.В.

# РАЗРАБОТКА БИБЛИОТЕКИ ПАРАЛЛЕЛЬНОЙ СОРТИРОВКИ С ПРИМЕНЕНИЕМ НИЗКОУРОВНЕВОГО ПРОГРАММИРОВАНИЯ

**Путинцев Илья Александрович**, студент 1 курса магистратуры факультета информационных технологий и управления;
**Варданян Андраник Эдуардович**, студент 1 курса магистратуры факультета информационных технологий и управления;
**Простов Дмитрий Иванович**, студент 1 курса магистратуры факультета информационных технологий и управления;
**Романенко Михаил Михайлович**, студент 1 курса магистратуры факультета инженерной химии;
**Красильников Игорь Владимирович**, к.т.н., доцент, доцент кафедры информационных компьютерных технологий, e-mail: muctr@list.ru;
Российский химико-технологический университет им. Д.И. Менделеева, Москва, Россия
125480, Москва, ул. Героев Панфиловцев, д. 20

*Параллельные вычисления уже давно являются одними из самых важных направлений в современной науке и в растущей системе больших данных. В данной работе исследована целесообразность использования языка низкого уровня для построения библиотеки с универсальным интерфейсом, исследована работа процессора в направлении параллельной сортировки, создана библиотека с универсальным интерфейсом, позволяющим использовать любой из методов сортировки в параллельном режиме, и проведены эксперименты эффективности применяемых подходов.*

***Ключевые слова:*** *низкоуровневое программирование, ассемблер, параллельные вычисления, сортировка.*

## THE DEVELOPMENT OF A LIBRARY OF PARALLEL SORTING USING LOW-LEVEL PROGRAMMING

Putintsev I.A., Vardanyan A.E., Prostov D.I., Romanenko M.M., Krasilnikov I.V.
D. Mendeleev University of Chemical Technology of Russia, Moscow, Russia

*Parallel computing has long been one of the most important areas in modern science and in the growing big data system. In this paper we investigate the feasibility of using low level language for building a library with universal interface, investigated the work of the CPU in the direction parallel sorting, a library with universal interface posevnaya to use any of the filtering techniques in parallel and carried out experimentation the effectiveness of approaches.*

***Keywords:*** *low-level programming, assembler, parallel computing, sorting.*

**Введение**

Параллельные вычисления находятся в ранге приоритетных направлений уже достаточно давно. Ещё с 50-х годов прошлого столетия данной теме уделялось значительное внимание, и выделялись огромные средства на развитие именно этой части науки. Однако, несмотря на все старания научных работников в течение больше полувека громадного скачка в направлении одновременных вычислений сделать так и не удалось. Сегодня, как и в XX веке, параллельные вычисления остаются «перспективным направлением». И как многие сейчас шутят, параллельные вычисления «так и будут всегда оставаться перспективным направлением», потому что работа в данном векторе науки является одним из наиболее сложных творческих процессов. Однако, эта тема на сегодняшний день достойна внимания гораздо больше, чем прежде. Изменение закона Мура и развитие работы в системе больших данных требуют обработки информации именно с применением распараллеливания. Данная работа посвящена созданию параллельной библиотеки для сортировки больших данных с минимальной затратой времени.

Так как сегодня наиболее популярной операционной системой для компьютеров является линейка продуктов под названием *Windows* от американского производителя *Microsoft*, то и разработка библиотеки велась под данную систему. Разработанная библиотека является динамической (*DLL*). Динамическая библиотека – сборник подпрограмм, позволяющий программисту подключение данного ресурса в процессе выполнения основной программы. Динамическую библиотеку также можно при жёстко ограниченных ресурсах отключить и выгрузить из памяти, что позволит задействовать ресурсы компьютера максимально эффективно.

**Метод «Быстрой сортировки»**

Для выполнения сортировки выбран метод «Быстрой сортировки». Метод быстрой сортировки основывается на перестановке элементов, расположенных максимально далеко друг от друга, разбивая заданный массив на две части относительно основного (опорного) элемента последовательности данных. При выполнении метода данные располагаются по разные стороны от

опорного значения и продолжают перестанавливаются до тех пор, пока массив не будет полностью отсортирован.

При создании библиотеки использовался ассемблер по причине его близости к оборудованию и максимально точному и быстрому выполнению инструкций.

Библиотека параллельной сортировки ориентирована на максимально быстрое выполнение любого из методов, выбранных программистом для решения его задачи с обработкой данных. На данный момент в продукте реализованы два метода сортировки: метод быстрой сортировки и метод сортировки вставками. Оба метода являются очень эффективными программными алгоритмами. Все операции, производимые с элементами массива, производятся непосредственно с участием регистров процессора в основной степени. Сам массив располагается в оперативной памяти компьютера. Процессор использует регистры для сравнения значений элементов и по результатам полученных вычислений решает, менять ли выбранные ячейки или оставить на местах.

В поисках наиболее эффективного подхода была проверена эффективность распараллеливания отдельных участков кода непосредственно выбранного метода сортировки. В частности, было исследовано, насколько возможно сделать одновременное выполнение циклических участков. То есть, изучалось, насколько будет целесообразно во время выполнения цикла в одном потоке запускать другой цикл в следующем витке.

Такой метод распараллеливания привёл к очень сложному и запутанному процессу выполнения библиотеки, хотя подразумевалось, что он будет наиболее быстрым из всех. Также сложностей добавило ещё и то, что при работе такого рода один виток, выполнив цикл инструкций, вставал в очередь и ждал выполнения от «коллеги по цеху»; дальше между значениями, полученными в результате исполнения циклов, происходило сравнение, решалось, как поступать с элементом, и снова запускались потоки. В случае решения задачи подобными методами исходный код становился очень сложным для понимания, что приводило к множеству ошибок в алгоритме. Также в этом подходе очень много времени уходило на ожидание одного витка другим, что делало всю задачу распараллеливания малоэффективным занятием. После произведенных исследований и проведения анализа результатов используемого подхода, было принято решение сделать выполнение потоков самостоятельным не относительно участка кода, а относительно используемых участков памяти.

**Метод сортировки «Раскидка»**

Было решено добавить ещё один метод сортировки массива под названием «Раскидка». В результате выполнения этого метода решается наиболее важная задача параллельных вычислений – гонка потоков.

В «Раскидке» происходит распределение значений элементов по двум независимым участкам. Для корректной работы потоков следует сделать для каждого из них свой диапазон адресов памяти. Первый поток берёт на исполнение первую половину до середины, второй поток берёт от середины и до конца. Такой подход позволяет работать каждому из витков процесса максимально быстро и независимо. Также при использовании этого метода почти полностью исключается необходимость ожидания одного потока другим. Они работают в том темпе, который позволяет им на данный момент использовать система. Ожидание присутствует только уже когда работа одним из витков уже полностью выполнена, а второй все ещё занят, поэтому первый ждёт завершения работы второго лишь для вывода окончательных результатов пользователю [1].

Самый важный момент в работе «Раскидки» – выбор правильного среднего элемента данных. Если выбрать просто тот, что находится посередине, то может оказаться что распределение произойдет относительно очень маленького или очень большого значения, это приведёт к тому, что вся сортировка будет выполняться лишь одним витком, второй будет просто запущен без дела и по сути он будет просто занимать ресурсы вычислительной машины. Для решения этой задачи был разработан новый подход к выполнению параллельности. Он заключается в том, что средний элемент массива каждый раз при выполнении вычисляется. То есть, он берётся в зависимости от значений предоставленных данных и является «плавающим» для каждого раза сортировки.

Для вычисления среднего элемента библиотекой происходит полный анализ всех значений предоставленных данных и относительно его наибольшего и наименьшего значения считается значение среднего элемента. Формула для нахождения значения среднего элемента имеет вид:

$$M = \frac{H - L}{2}, \quad (1)$$

где $M$ – значение среднего элемента;

$H$ – значение наибольшего элемента;

$L$ – значение наименьшего элемента.

В результате таких вычислений было получено значение среднего элемента для конкретного случая сортировки. С каждой новой сортировкой следует производить подобные действия заново. Получив значение среднего элемента, метод распределяет все элементы на основе сравнения со средним. Те элементы, что являются меньше, отправляются в левую часть, те элементы, что больше или равны, переносятся в правую. Принцип работы метода «Раскидка» библиотеки параллельной сортировки приводится на рисунке 1.

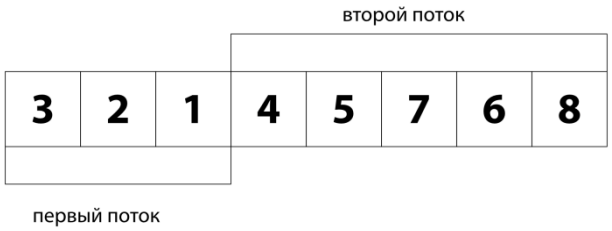


**Рис.1. Шаблон распараллеливания методом «Раскидка» для 2-х потоков**

Вычисления среднего элемента и «Раскидка» в целом являются довольно затратными методами в плане времени, но их выполнение того стоит, так как в дальнейшем при обработке огромных массивов данных приводит к огромному сокращению времени выполнения.

В ходе работы использовался язык программирования Ассемблер, ориентированный на работу с процессорами *Intel* архитектуры x86. Все потоки вызывались при помощи встроенного в систему *Windows API*. Использование потоков из уже готовых функций повысило скорость создания библиотеки, при этом сохранив её универсальность, и скорость выполнения программы осталось на том же высоком уровне [2].

Также при разработке библиотеки важным оставался вопрос нехватки регистров для работы. Так как регистров в процессоре ограниченное количество, то на два витка одновременного выполнения их может и не хватить. Эта задача решилась сама собой при вызове витков операционной системой. Проблемы подобного рода в ОС *Windows* решаются выделением для каждого витка собственного регистра. Такой подход был очень удивительным. Видимо, каждому из вызываемых потоков система предоставляет наиболее быстрый участок памяти для работы, выступающий в роли «виртуального регистра» для каждого витка. В дополнение к регистрам каждый из потоков, как оказалось, получает и свой собственный стек. Это очень удобно при параллельном программировании. Позволяет программисту оставить все ухищрения на манер поворота регистра (имеется в виду использование *rol*/*ror* команд ассемблера) и сосредоточиться на более важных моментах работы.

**Сравнение результатов**

С библиотекой параллельной сортировки были проведены несколько десятков экспериментов для сравнения результатов работы и скорости выполнения. Вычислительных ошибок во время экспериментов выявлено не было. Результаты, содержащие скорость выполнения, сопоставлены с результатами работы сортировки системой *Open MP* и *Windows Thread*. В данной статье приводится время выполнения программ (в миллисекундах) с использованием библиотеки параллельной сортировки при обработке массивов значений из 200 миллионов элементов (рис.2) [3].

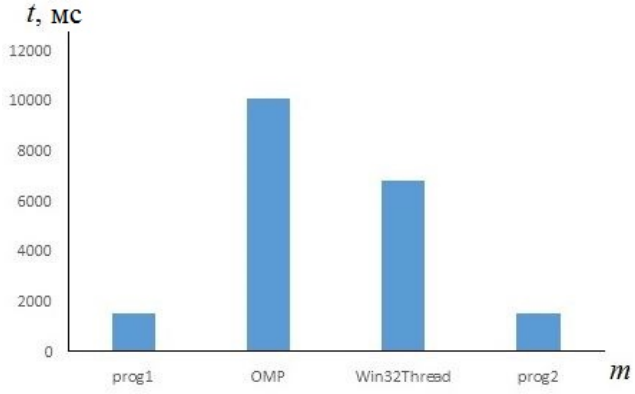


**Рис.2. Результаты выполнения сортировки 200 миллионов элементов компьютером 83 (200 мл. эл. = 785 МБ) Pentium (R) Dual Core 2.5GHz 2.00ГБ (ОЗУ), *t* – время сортировки, *m* – метод сортировки**

На рисунке 2 представлено время выполнения программ с разными подходами: prog1 – разработанная в ходе работы библиотека с одним вызванным витком выполнения, ОМП – программа с использованием *OpenMP*, Win32Thread – сортировка с применением *Windows 32 Thread*, prog2 – разработанная в ходе работы библиотека, в которой использовались 2 внешних витка.

**Список литературы**

1. Левитин А.В. Алгоритмы: введение в разработку и анализ. М.: Вильямс, 2006. 576 с.

2. Таненбаум Э. Архитектура компьютера. СПб.: Питер, 2007. 848 с.

3. Win32 API. Урок 17. Динамические библиотеки. Библиотека Интернет Индустрии I2R.ru [Электронный ресурс]. Режим доступа: http://www.i2r.ru/static/565/out_17611.shtml (дата обращения 12.05.2017).