Университет ИТМО

Кафедра вычислительной техники

Отчет по прохождению практики

Студента

P3311 группы

Морозова С.Д.

Руководитель
Соснин В.В.

Санкт-Петербург

2016

Содержание

1 Введение

Тема прохождения практики — параллельные вычисления. Цель задания — сравнить различные функции в языке C, которые можно использовать для измерения времени работы параллельных программ.

Однако требования руководителя практики таковы, что перед тем как приступить к выполнению основного задания нужно ознакомиться с системой компьютерной вёрстки TEX(LATEX), которая должна использоваться для написания отчёта, и ознакомиться с системой контроля версий Git, с последующим созданием учетной записи на сайте GitHub или анагичном.

2 Система компьютерной верстки TEX(LATEX)

2.1 Краткое описание

TEX — система компьютерной вёрстки с формулами, разработанная американским профессором информатики Дональдом Кнутом. Название происходит от греческого слова $\tau\varepsilon\chi\upsilon\eta$ — «искусство», «мастерство», поэтому последняя буква читается как русская Х. Хотя TeX является системой набора и верстки, развитые возможности макроязыка TeX делают его Тьюринг-полным языком программирования.

TEX работает с боксами (box) и клеем (glue). Бокс — двумерный объект прямоугольной формы, характеризуется тремя величинами (высота, ширина, глубина). Элементарные боксы — это буквы, которые объединяются в боксы-слова, которые в свою очередь сливаются в боксы-строчки, боксы-абзацы и т.д.

Между боксами располагается клей, который имеет некоторую ширину по умолчанию и степени увеличения/уменьшения этой ширины. Объединяясь в бокс более высокого порядка, боксы могут шевелиться, но после того как найдено оптимальное решение, это состояние закрепляется, и полученный бокс выступает как единое целое.

Инетересный факт. На версии 3.0 дизайн был заморожен, поэтому в новых версиях не будет добавления новой функциональности, только исправление ошибок. Версия TEX'а ассимтотически приближается к числу π. Это факт говорит о том, что последняя версия 3.14159265 (январь 2014) является крайне стабильной и возможны лишь мелькие исправления. Дональд Кнут заявил, что последнее обновление (сделанное после его смерти) сменит номер версии на π, и с этого момента все ошибки станут особенностями.

LATEX — созданный Лесли Лэмпортом набор макрорасширений (или макропакет) системы компьютерной вёрстки TEX, который облегчает набор сложных документов. Стоит отметить, что как и любой другой макропа-

кет[1] LaTeX не может расширить возможности TeX (все, что можно сделать в одном пакете можно сделать и в любом другом). Пакет позволяет автоматизировать многие задачи набора текста и подготовки статей, включая набор текста на нескольких языках, нумерацию разделов и формул, размещение иллюстраций и таблиц на странице, ведение библиографии и др. Все это делает LaTeX крайне удобным инструментом для написания научных статей, диссертаций и т.п..

2.2 Сравнение LaTeX и MS Word

В качестве сравнения — перечислим плюсы и минусы LaTeX перед MS Word(а так же всеми его аналогами).
Плюсы LaTeX:

- Кроссплатформенность
- Язык международного обмена по математике и физике (большинство научных издательств принимают тексты в печать только в этом формате)

Минусы LaTeX:

- Не является системой типа WYSIWYG [2]
- При серьезных отклонениях от стандартных стилей документов требуется достаточно сложное программирование

То есть, выбирая между LaTeX и MS Word, стоит обратить внимание на то,какой текст вы собираетесь печатать, насколько нестандартный будет стиль текста, на его примерный объем. В некоторый случаях достаточно использовать MS Word, в других — использование LaTeX может заметно упростить работу.

[1] Так же существуют Plain TeX, AMS-TeX, AMS-LaTeX и т.д.

[2] What You See Is What You Get(Что видишь, то и получишь). Стоит отметить, что существуют дистрибутивы TeX в которых есть попытки реализовать WYSIWYG. Например платный дистрибутив BaKoMa TeX + текстовый редактор BaKoMa TeX Word.

2.3 Выбор инструмента редактирования

В ходе изучения всех возможных вариантов работа с LaTeX для создания данного отчета, была выбрана программа Textmaker [3].
Выбор Textmaker'а обусловлен следующими его особенностями:

- Автоматическая подсветка синтаксиса
- Функция автодополнения команд LaTeX
- Сокрытие блоков кода (Code folding)
- Быстрая навигация по структуре документа
- Указание на строку с ошибкой, для быстрой отладки
- Интегрированный просмотр PDF

[3] Оффициальный сай Textmaker: http://www.xm1math.net/texmaker/

3 Системы контроля версий

3.1 Краткое описание

Система контроля версий (СКВ) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов.

СКВ широко используются при разработке программного обеспечения, для хранения кодов разрабатываемых программ. Однако данные системы подходят не только программистам. Художники, которые хотят сохранять каждое изображение/эксиз своей работы, писатели пишущие книги или научные статьи, бухгалтеры, которые хранять разные версии отчетов и т.д., все они могут использовать СКВ для достижения своих целей.

Иначе говоря СКВ можно применять в любых областях в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов.

3.2 Git

Git — созданная Линусом Торвальдсом, распределенная система контроля версий.

3.2.1 Особенности

Одной из основных особенностей Git состоит в способе хранения данных. В принципе, большинство других систем хранит информацию как список изменений (патчей) для файлов. Эти системы (CVS, Subversion, Perforce, Bazaar и другие) относятся к хранимым данным как к набору файлов и изменений, сделанных для каждого из этих файлов во времени, как показано на Рис. 1

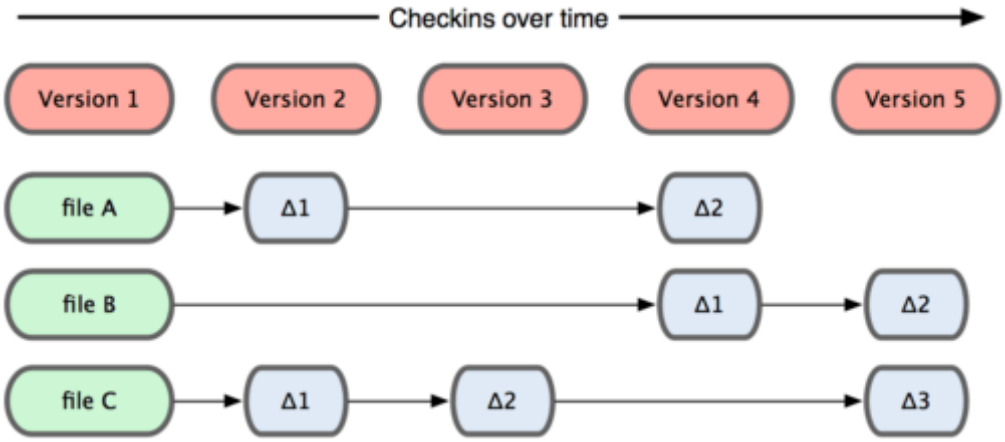


Рис. 1: Другие системы хранят данные как изменения к базовой версии для каждого файла.

Git не хранит свои данные в таком виде. Вместо этого Git считает хранимые данные набором слепков небольшой файловой системы. Каждый раз, когда вы фиксируете текущую версию проекта, Git, по сути, сохраняет слепок того, как выглядят все файлы проекта на текущий момент. Ради эффективности, если файл не менялся, Git не сохраняет файл снова, а делает ссылку на ранее сохранённый файл. То, как Git подходит к хранению данных, похоже на Рис. 2

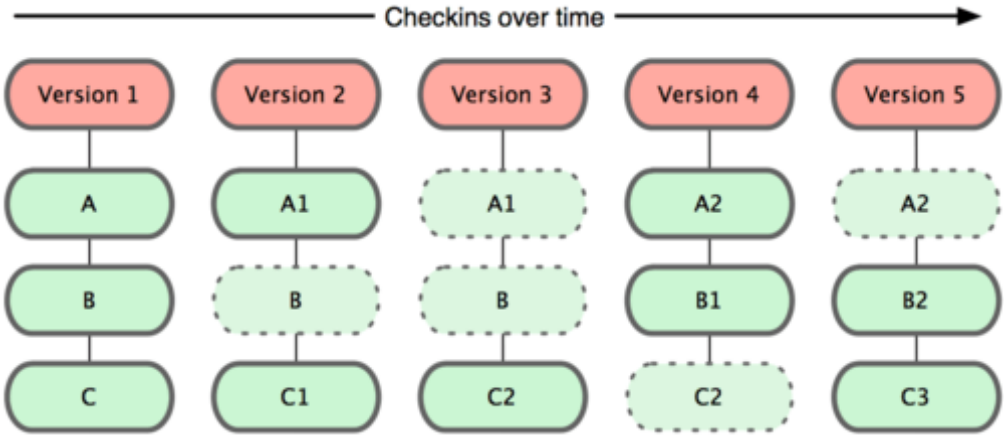


Рис. 2: Git хранит данные как слепки состояний проекта во времени.

За счет этого, для большинства операций в Git нужны только локальные ресурсы и файлы. Что в свою очередь определяет два основных преимущества Git перед остальными СКВ.

- Быстродействие. Поскольку вся история проекта хранится локально у вас на диске, большинство операций кажутся практически мгновенными(в отличии от централизованных системам, где практически на каждую операцию накладывается сетевая задержка).
- Возможность работать(делать коммиты) без доступа к сети или VPN.

3.2.2 Основные команды

В целом, следующая картинка (Рис. 3) наглядно демонстрирует основные команды Git, знание которых достаточно, чтобы начать им пользоваться.

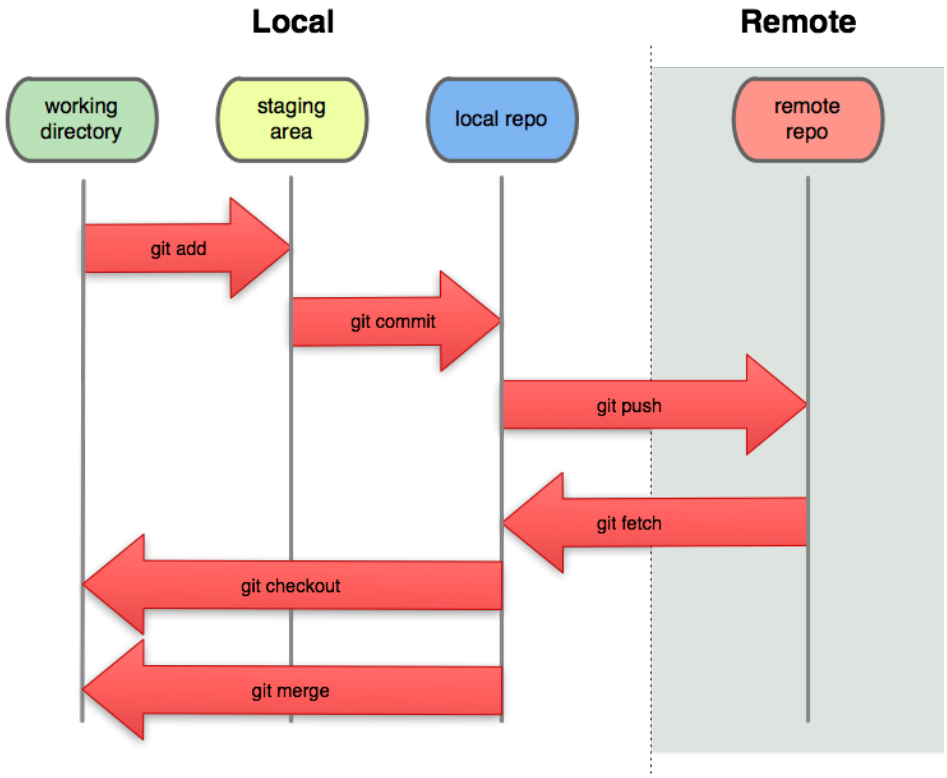


Рис. 3: Основные команды при работе с Git.

3.3 GitHub

GitHub — крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки. Основан на системе контроля версий Git и разработан на Ruby on Rails и Erlang компанией GitHub, Inc (ранее Logical Awesome).

В ходе прохождения практики, на сайте GitHub была создана учетная запись MorozovSD. В данной учетной записи был создан репозиторий Practice-2016 по которому можно легко отследить процесс прохождения практической работы.

4 Паралельные вычисления

Т.к. практика предполагает не доскональное изучение параллельного программирования, а лишь сравнение функций замера времени в программах, работающих на основе парралельных вычислений, то данная глава носит более ознакомительных характер, содержащий тот минимум знаний, необходимый для работы с этой области.

4.1 Немножко теории

Мультипрограммирование — параллельное выполнение нескольких программ. Мультипрограммирование позволяет уменьшить общее время их выполнения.

Под параллельными вычислениями понимается параллельное выполнение одной и той же программы. Параллельные вычисления позволяют уменьшить время выполнения одной программы. Чаще всего, хороший последовательный алгоритм не является таковым для параллельного выполнения.(а параллельные алгоритмы могут не являться эффективными при работе с одним процессором), поэтому одна из задач паралельных вычислений это разработка эффективных алгоритмов, полностью использующие количество преподставленных процессоров.

Т.к. тема практики — измерение времени работы пареллельно работающей программы, то необходимо получить оценку времени выполения программы одним процессором T_1 для идеализированного случая, когда число процессоров не ограничивается — T_∞. А так же оценить верхнюю и нижнюю границы времени выполнения конечным число процессоров T_p[4].

Для введенных характеристик очевидно следующее соотношение:

$$T_\infty \leq T_p \leq T_1 \tag{1}$$

Для T_p справедлива следующая оценка снизу:

[4]В отчете будут представленны только конечные формулы, без доказательств

$$T_p \geq \frac{T_1}{p} \tag{2}$$

Для T_p справедлива следующая оценка сверху:

$$T_p \leq \frac{T_1}{p} + T_\infty \tag{3}$$

Исходя из вышеперечисленных неравенств, неравенство(1) может быть заменено более точным:

$$\frac{T_1}{p} \leq T_p \leq \frac{T_1}{p} + T_\infty \tag{4}$$

На графике это выглядит следующим образом (Рис. 4):

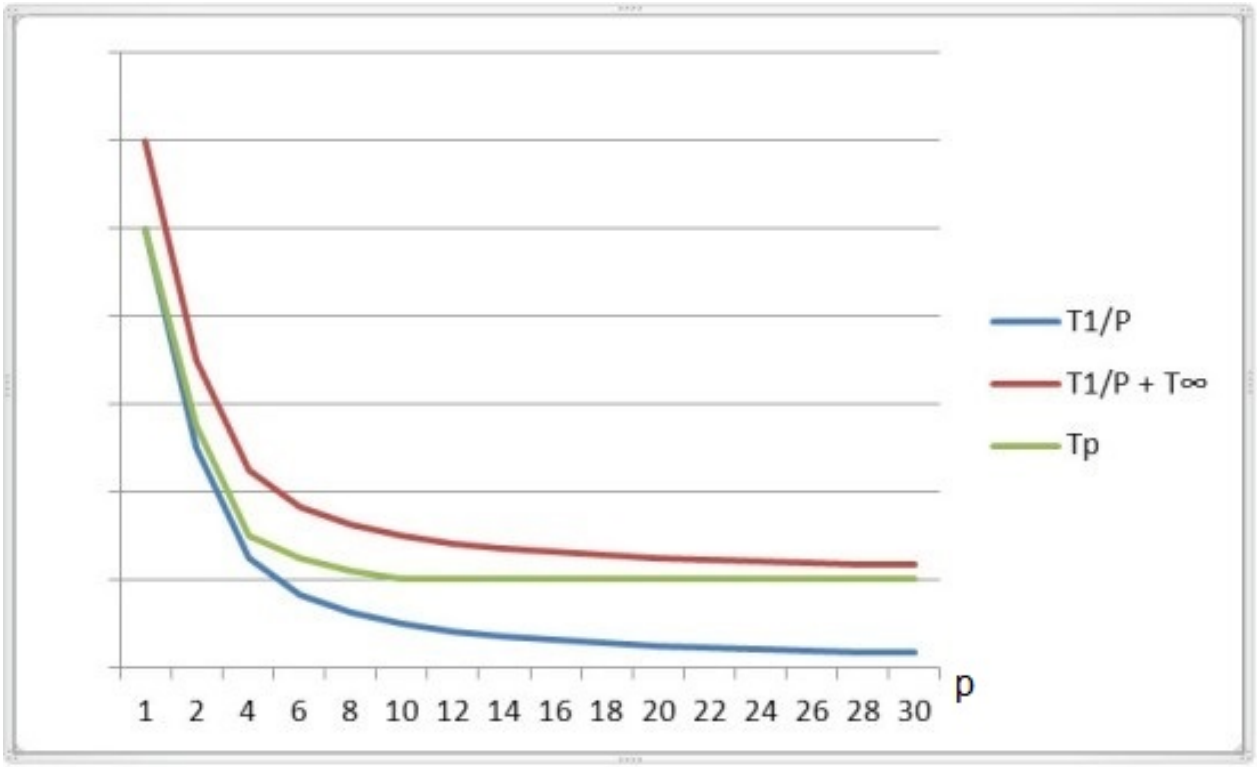


Рис. 4: Графическая интерпритация поведения функции T_p

Очевидно, что при $p = 1$ графики функций T_p и $\frac{T_1}{p}$ совпадают. Так же происходит совпадение графиков T_p и $\frac{T_1}{p} + T_\infty$ при $p \to \infty$.

4.2 Характеристики параллельных вычислений

4.2.1 Ускорение

Ускорение $S_p(n)$[5] определяют как отношение:

$$S_p(n) = \frac{T_1(n)}{T_p(n)} \tag{5}$$

Интерпритировать данную формулу следует как отношение время наилучшего алгоритма, для которого достаточно одного процессора, и время наилучшего параллельного алгоритма, который может использовать p имеющихся процессоров.

4.2.2 Эффективность

Эффективность $E_p(n)$ определяют как отношение:

$$E_p(n) = \frac{S_p(n)}{p} \tag{6}$$

При оптимальном ускорении[6] эффективность равна 1. Если же эффективность существенно ниже 1, то часто число процессоров целесообразно уменьшить, используя их более эффективно.

4.2.3 Упущенная эффективность

Мера неиспользованных возможностей — упущенной выгоды — $U(n)$, определяют следующим образом:

$$U(n) = \frac{T_p(n)}{T_{p_{opt}}} - 1 \tag{7}$$

Оптимальное время, которое можно достичь, используя p процессоров, дается нижней оценкой для T_p, поэтому получаем:

[5]Все вводимые характеристики рассматриваются как функции параметра n, характеризующего сложность решаемой задачи. Обычно n понимается как объем входных данных.

[6]Оптимальное ускорение достигается когда $T_p = \frac{T_1}{p}$

$$U(n) = p\frac{T_p(n)}{T_1} - 1 \tag{8}$$

Если для компьютера с p ядрами время решения задачи оптимально и сокращается в $\sim p$ раз в сравнении с решением задачи на одноядерном компьютере, то наши потери равны нулю, возможности компьютера полностью используются. Если же задача решается за время T_1 — столь же долго, как на одноядерном компьютере, то потери пропорциональны числу неиспользованных ядер.

4.3 Основные проблемы паралельного программирования

4.3.1 Синхронизация

Синхронизация нужна для того, чтобы согласовать обмен информацией между модулями (между параллельно выполняемыми множествами операций). Синхронизация может привести к простою процессора, т.к. после достижения точки синхронизации он должен ждать, пока другие задания достигнут точки синхронизации. Задержка с подачей в процессор необходимых данных ведет к простою процессора и снижению эффективности параллельной обработки

4.3.2 Гонка данных

Проблема «гонки данных» возникает для мультипроцессорных компьютеров с общей памятью. В одни и те же моменты времени процессоры могут получать доступ к одним и тем же данным,хранимым в общей памяти, как для чтения, так и для записи.

Если с чтением данных проблем не возникает,то одновременная запись двух разных значений в одну и ту же ячейку памяти не возможна. Запись всегда идет последовательно, следовательно в памяти останется храниться значение, пришедшее последним (причем не известно какое значение каким

прийдет). Конкурирование процессоров за запись в одну и ту же ячейку памяти и есть «гонка данных».

Один из способов справиться с этой проблемой — это закрытие доступа к ресурсу первым пришедшим процессором. Остальные процессоры прерывают выполнение и становятся в очередь за обладание ресурсом. Обладатель ресурса спокойно выполняет свою работу, а по ее окончании открывает ресурс, с которым теперь начинает работать тот, кто первым стоит в очереди.

4.3.3 Взаимная блокировка (Deadlock)

Блокировка — хороший механизм решения проблемы «гонки данных». Однако блокировка может прервать выполнение всей программы, когда наступает ситуация, называемая взаимной блокировкой, клинчем, смертельным объятием или deadlock'ом.

В качестве примера рассмотрим простейшую ситуацию, приводящую к возникновению клинча. Пусть есть два конкурента A и B, претендующие на два ресурса x и y. Пусть гонку за ресурс x выиграл A и соответственно закрыл этот ресурс для B. Гонку за ресурс y выиграл B и закрыл ресурс для A. Но A, чтобы закончить свою работу нужен ресурс y, поэтому он стал в очередь, ожидая освобождения ресурса. Симметрично, B находится в очереди, ожидая освобождения ресурса x. Возникает ситуация вечного ожидания (которое может разрешить только внешнее воздействие), когда ни A, ни B не могут продолжить свою работу.(Рис. 5):

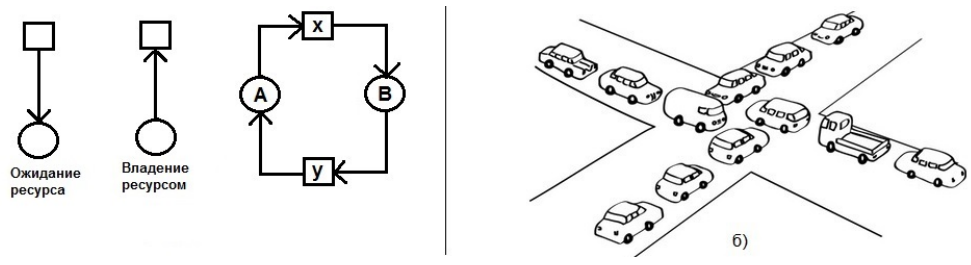


Рис. 5: Взаимная блокировка (клинч): а) Общий вид. б) На примере перекрестка.

5 Функции замера времени

Эта глава посвещена функциям замера времени языка Си.

5.1 Категории функций

Функции работы со временем можно отнести к трем категориям:

- Функции календарного времени
- Функции для измерения прошедшего времени CPU
- Функции для установки будильников и таймеров [7]

5.1.1 Календарное время

Осуществляет для слежения за датами и временем согласно Грегорианскому календарю. Существуют несколько способов представления информации даты и времени. Т.к. нас интересует разрешающая способность (точность замеров времени) рассмотрим эти способы представления в зависимости от их разрешающей способности.

- Тип данных time_t — компактное представление, обычно дает число секунд, истекающих начиная с некоторого основного времени. Разрешающая способность одна секунда
- Тип данных struct timeval — представление времени с большей точностью. Разрешающая способность до микросекунд [8]
- Тип данных timespec — представление времени с более большей точностью. Разрешающая способность до наносекунд

[7] Данная категория функций не используется в практической работе, и рассматриваться не будет

[8] На некоторых платформах возможно отслеживание времени только в пределах разрешающей способности системного таймера, который в общем случае устанавливается на значение 100 Гц

5.1.2 Время процессора

Процессорное время отлично от фактических часов, тем что оно не включает временя выполнения другого процесса и все потраченное время на ожидание ввода-вывода. Процессорное время представляется типом данных clock_t, и дано как ряд импульсов времени относительно произвольного базового времени [9] , отмечающего начало одиночного вызова программы.

Важно! В зависимости от архитектуры компьютера и операционной системы способ слежения за процессорным временем может быть разным. Общее для внутренних часов процессора то, что разрешающая способность где-то между тысячной и милионной долей секунды.

5.2 Функции

В данной главе будут описаны функциии замера времени в операционных системах Windows и Linux.

5.2.1 Кроссплатформенные функции

clock

$$clock\_t\ clock\ (void)$$

Функция возвращает прошедшее процессорное время. Базовое время произвольно, но не изменяется внутри одиночного процесса. Если процессорное время не доступно или не может представляться, clock возвращает значение (clock_t) (-1). Т.к. процессорное время считается по разному в разных ОС, результат работы clock меняется в зависимости от используемой ОС.

[9] Для перевода количества импульсов в секунды, количество импульсов неоходимо делить на CLOCKS_PER_SEC (число импульсов времени clock в секунду)

time

$$time\_t\ time(time\_t\ *\ timeptr)$$

Функция возвращает текущее календарное значение времени в секундах. Если аргумент не является нулевым указателем, ей передается значение времени типа time_t.

omp_get_wtime

$$double\ omp\_get\_wtime(void)$$

функция возвращает значение с плавающей запятой двойной точности, эквивалентное истеченному реальному времени в секундах с момента, прошедшего от некоторого "времени в прошлом которое гарантированно не затрагивает программа. Время измеряется для каждого потока , никакой гарантии не может быть сделано , что два различных потока измеряют то же самое время и время каждого потока не обязательно быть глобально одинаковым во всех потоках, учавствующих в приложении.

gmtime

$$struct\ tm\ *\ gmtime(const\ time\_t\ *\ timeptr)$$

Функция преобразует системное время в секундах в дату по Гринвичу. Результат помещается в структуру типа tm и функция возвращает указатель на эту структуру.

localtime

$$struct\ tm\ *\ localtime(const\ time\_t\ *\ timeptr)$$

Функция преобразовывает текущее значение времени, передаваемое как аргумент, через указатель timeptr на time_t в структуру tm (местное время). Так же функция возвращает указатель на эту структуру.

asctime

$$char * asctime (const\ struct\ tm * m_time)$$

Функция преобразует локальное (местное) время представленное в виде структуры типа struct tm, на которую указывает аргумент m_time в текстовую строку. Результат преобразования возвращается функцией в виде указатель на строку содержащую дату и время. Возвращаемая строка имеет следующий формат:

«ННН МММ ДД ЧЧ: ММ: СС ГГГГ \ n \ 0», где

ННН — это день недели,
МММ — месяц,
ДД — день,
ЧЧ: ММ: СС — время,
ГГГГ — год.

5.2.2 Windows

В ОС Windows многие старые функции CRT имеют безопасные версии (в названии присутствует приставка "_s"). Если безопасная функция существует, то старая менее безопасная версия помечена как нерекомендуемая.

Отличие безопасных функций в том, что они перехватывают ошибки при их возникновении. Они выполняют дополнительные проверки на выполнение условий возникновения ошибки, а в случае ошибки вызывают обработчик ошибок. В нашем случае речь идет о функциях localtime, ctime и acstime (т.е. их безопасных версиях localtime_s, ctime_s, acstime_s). Отличие безопасных аналогов в том, что в параметрах так же указывается ссылка на буффер (ctime_s, acstime_s) или на структуру tm (localtime_s), благодаря чему, эти функции могут быть использованы например в параллельном программировании (каждая функция может использовать свой буффер, что исключает различные ошибки, которые могут возникнуть при работе с одним буффером).

GetTickCount

$$DWORDWINAPIGetTickCount(void);$$

$$ULONGLONGWINAPIGetTickCount64(void);$$

Функции извлекает число миллисекунд, которые истекли с тех пор как система была запущена.

Если разница между двумя вызовами в функции GetTickCount составляет более чем 49.7 дней. Эта проблема решена у функции GetTickCount64, переполнение которых крайне маловероятно.

5.2.3 Linux

Функции ctime(), asctime(), помещает результат строку в статический буфер , который повторно используется каждый раз, когда вы называете ctime() или asctime(). Вызов gmtime() или localtime() может также изменить дату в этом статическом буфере. В параллельных вычислениях это может привести к некорректным результатам, поэтому в Linux при работе с данными функциями следует использовать аналоги данных функций с суффиксом _r.

localtime_r

$$struct\ tm\ *\ localime\_r\ (const\ time\_t\ *\ s\_time,\ struct\ tm\ *\ m\_time);$$

Функция преобразовывает текущее значение времени, передаваемое как аргумент, через указатель timeptr на time_t в структуру tm (местное время), на которую указывает аргумент m_time. Так же функция возвращает указатель на эту структуру.

asctime_r

$$char\ *\ asctime\_r\ (const\ struct\ tm\ *\ m\_time,\ char\ *\ buf)$$

Функция преобразует локальное (местное) время представленное в виде структуры типа struct tm, на которую указывает аргумент m_time в текстовую строку длинною 26 символов. Результат преобразования помещается в строку, на которую указывает аргумент buf. Возвращаемая строка имеет формат аналогичный фортату строки возвращаемый функцией asctime.

ctime_r

$$char\ *\ ctime\_r(const\ time_t\ *\ clock,\ char\ *\ buf)$$

Функция эквивалентна последовательному выполнению функций localtime_r() и asctime_r().

gettimeofday

$$int\ gettimeofday(struct\ timeval\ *tv,\ struct\ timezone\ *tz)$$

Функция возвращает системное время в виде структуры timeval.

clock_gettime

$$int\ clock\_gettime(clockid\_t\ clk\_id,\ struct\ timespect\ *tp)$$

Функция обеспечивает доступ к нескольким видам системных таймеров и имеет наносекундное разрешение. clk_id — задает вид таймера, например:

- CLOCK_THREAD_CPUTIME_ID — Таймер процессора работающий с каждым потоком с высокой разрешающей способностью
- CLOCK_PROCESS_CPUTIME_ID — Таймер процессора работающий с каждым процессом с высокой разрешающей способностью
- CLOCK_REALTIME — Таймер реального времени в масштабе всей системы

5.3 Сравнение функций

Для сравнения функций были выбранны следующие критерии:

- Кроссплатформенность. То есть на каких ОС работает данная функция.
- Реентабильность и Thread-safery. Потоко-безопасная функция может вызываться одновременно из нескольких потоков, даже когда вызовы используют общие данные, потому что все ссылки на общие данные упорядочиваются. Реентерабельная функция также может вызываться одновременно из нескольких потоков, но только тогда, когда каждый вызов использует свои собственные данные. Т.е. каждая реентабильная функция является thread-safery функцией, но не каждая thread-safery функция является реентабильной.
- Возвращаемое значение. Данный критерий выбран только для наглядности, указывает какое значение (int, double, time_t и т.д.) возращает функция.
- Точность. Какова точность (или разрешающая способность) функции.

Функции	Критерии оценивания			
	Платформа	Реент. и thread-safery	Возв. значение	Точность
clock	Кросс.	+/+	clock_t	$10^{-3} - 10^{-9}$ сек
time	Кросс.	+/+	time_t	1 сек
gettimeofday	Linux	+/+	timeval	10^{-6} сек
clock_gettime	Linux	+/+	int	10^{-9} сек
omp_get_wtime	Кросс.	+/+	double	10^{-6} сек
GetTickCount	Windows	+/+	DWORDW	10^{-6} сек
gmtime	Кросс.	-/-	struct tm	–
localtime	Кросс.	-/-	struct tm	–
localtime_r	Linux	+/+	struct tm	–
localtime_s	Windows	+/+	struct tm	–
asctime	Кросс.	-/-	char	–
asctime_r	Linux	+/+	char	–
asctime_s	Windows	+/+	char	–
ctime	Кросс.	-/-	char	–
ctime_r	Linux	+/+	char	–
ctime_s	Windows	+/+	char	–

6 Практическая часть

6.1 Описание эксперементальной программы

Эксперименты проводились с программой код которой представлен ниже:

Листинг 1: Тестовая программа

```
#include <omp.h>
#include <stdio.h>

double run_parallel_experiment(int amount_of_threads,
    unsigned int experiment_size) {
  unsigned int i;
  double sum_of_squares = 0;
  omp_set_num_threads(amount_of_threads);
  #pragma omp parallel for reduction(+:sum_of_squares)
   for (i = 0; i < experiment_size; ++i) {
      sum_of_squares += i*i;
  }
  return sum_of_squares;
}

int main() {
    double result;
    double t1;
    double t2;
  //"Start timer"
    result = run_parallel_experiment(2, 100000000);
  //"Stop timer"
    printf("result=%e, time_in_milliseconds=%f\n", result,
       1000*(t2 - t1));

    return 0;
}
```

С ходе экспериментов в строки №19 и 21 подставлялись различные функции замера времени, что в свою очередь сказывалось на выводе программы[10].

Для получения более точных и достоверных результатов были предприняты следующие шаги:

- Было отключено большинство работающих программ, процессов.
- Запуск программы осуществлялся несколько раз, с последующим определением среднего значения и доверительного интервала

Работы с Linux была осуществленна через программу Oracle VM VirtualBox, запущенной со следующими характеристиками (Рис. 6):

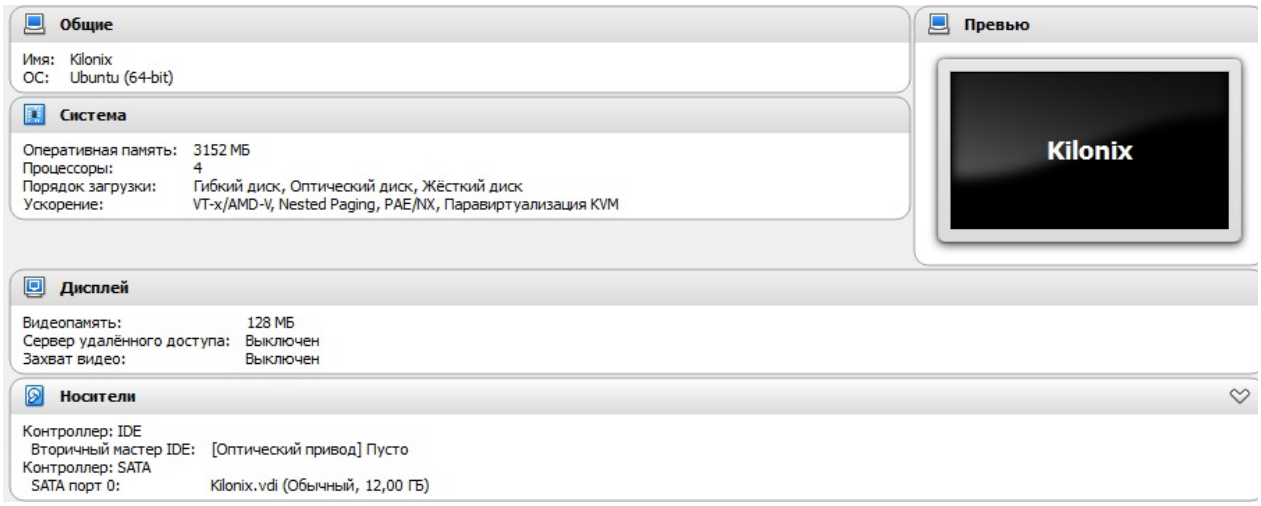
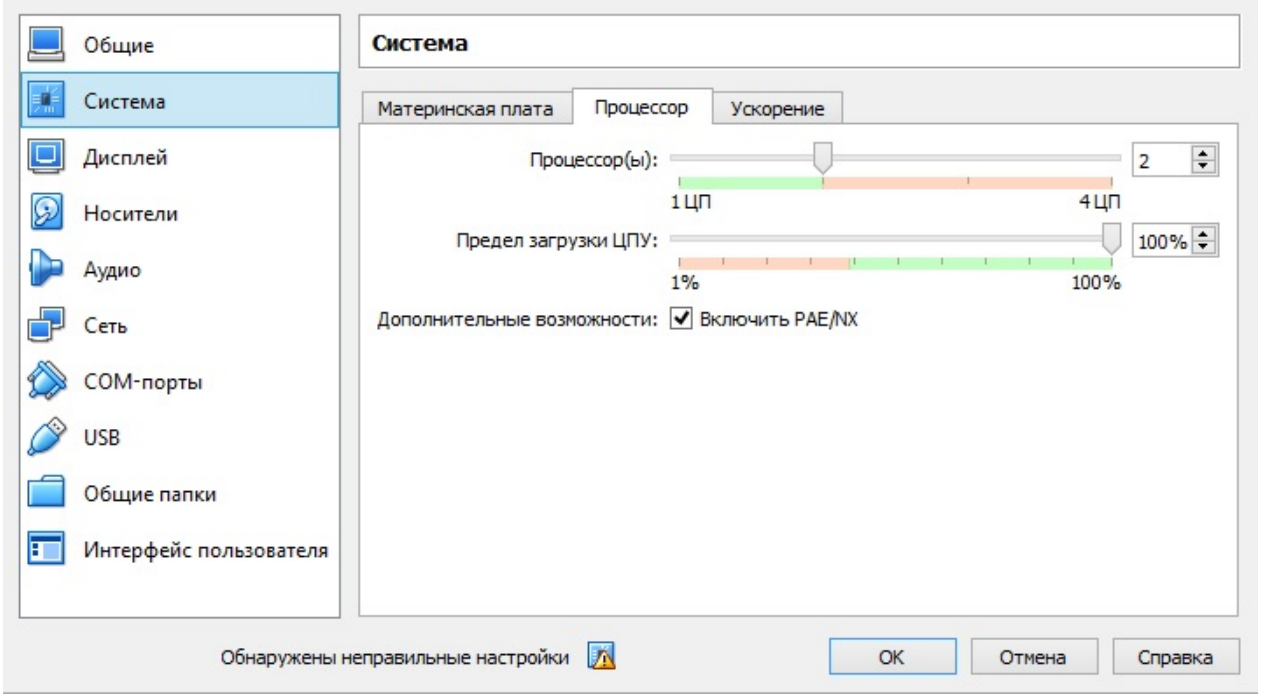


Рис. 6: Характеристики виртуальной машины.

[10]Вывод программы есть ни что иное как время выполнение программы, по крайней мере мы хотим, чтобы это было временем выполнения.

Так же в качестве дополнительного задания, которое заключается в изучении изменения эффективности работы параллельных программ на виртуальной машине в зависимости от количеста отведенных им процессоров, изменялось количество процессоров предоставляемых виртуальной машине:



6.2 Результаты работы программы

Таблица 1: Windows

Функции	Кол-во ядер			
	1	2	3	4
clock	32.641±0.032	16.445±0.024	11.226±0.0289	8.652±0.043
time	32±0.5	16±0.5	11±0.5	8±0.5
omp_get_wtime	32.649±0.039	16.455±0.023	11.227±0.055	8.6175±0.029
GetTickCount	32.743±0.026	16.609±0.047	11.242±0.502	8.656±0.012

Таблица 2: Linux (VirtualBox Ubuntu, 4 процессора)

Функции	Кол-во ядер			
	1	2	3	4
clock	12.35767 ±0.103098	13.690974 ±0.852372	15.970829 ±0.60532	17.1831 ±0.37756
time	12±0.5	7±0.5	6±0.5	5±0.5
omp_get_wtime	12.441215 ±0.216493	7.256746 ±0.370947	6.124574 ±0.27953	5.771348 ±0.1799238
gettimeofday	12.580472 ±0.382535	7.075927 ±0.2374386	5.939196 ±0.291476	5.709618 ±0.275306
clock_gettime	12.459468792 ±0.262119073	7.35941419 ±0.537213588	6.078799968 ±0.251583524	5.674500551 ±0.116543728

Для наглядности, полученные данные можно представить в виде графиков (Рис. 7):

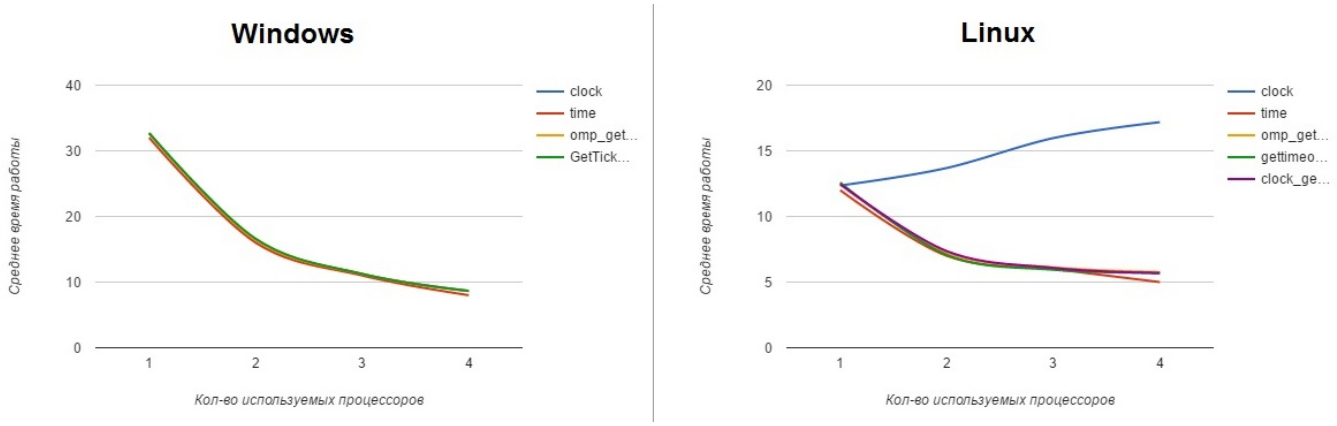


Рис. 7: График скорости выполнения программы.

6.3 Дополнительное задание

Как было сказанно ранее в дополнительном задание будет измерятся зависимость времени выполнения программы на виртуальной машине в зависимости от количества процессоров, предоставленных ВМ и программе. В качестве функции замера времени была выбрана функция omp_get_wtime.

Кол-во процессоров предоставленные ВМ	Кол-во процессоров предоставленные программе			
	1	2	3	4
4	12.656385 ±0.403936	7.035231 ±0.273854	6.180582 ±0.563768	5.601754 ±0.176031
3	12.743709 ±0.547703	7.285559 ±0.24012	6.110711 ±0.36705	-
2	12.908422 ±0.490109	7.50501 ±0.196944	-	-
1	13.971584 ±0.090051	-	-	-

На графике это выгллядит следующим образом (Рис. 8):

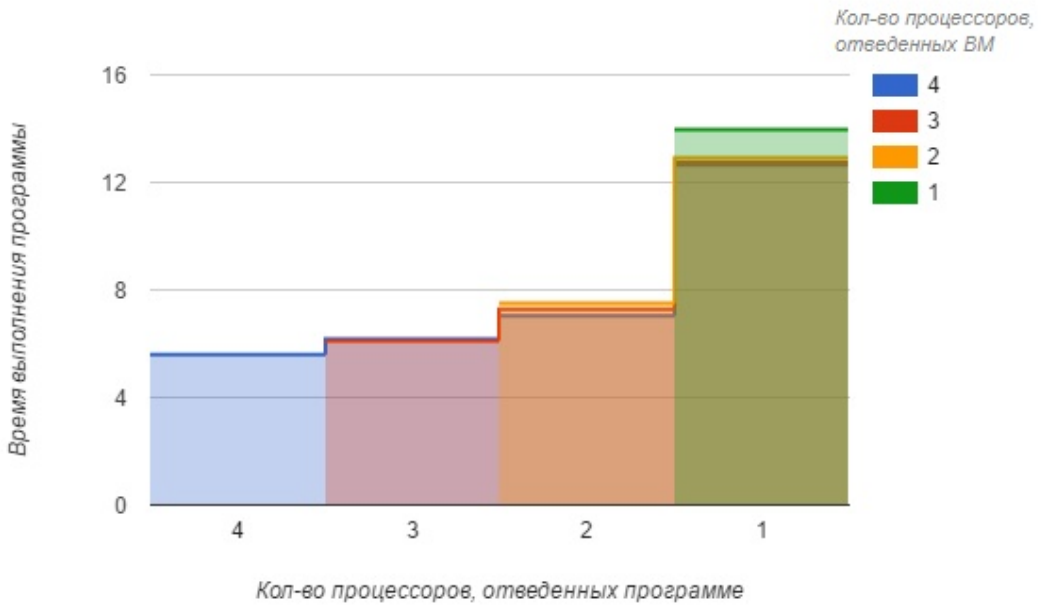


Рис. 8: График скорости выполнения программы.

7 Выводы

7.1 Вывод по основной части практического задания

На основе полученных результатов можно сделать следующие выводы:

- Lля работы с параллельными вычислениями не стоит использовать функцию time, всвязи с её малой точностью.
- В ОС Windows для работы с параллельными вычислениями следует использовать безопасные версии функций(если есть, суффикс $\_s$, вместо их устаревших версий.
- В Linux для работы с параллельными вычислениями не стоит использовать функцию clock (в качестве фунции рамера времени выполнения программы), т.к. в данной ОС она подсчитывает процессорное вре-

мя, которое в случае с параллельными вычеслениями не является реальным временем выполнения программы (в ОС Windows данная так же считает процессорное время, но результаты работы этой функции, как получилось в эксперименте, соответствуют реальному времени выполнения программы. Вероятно, это обусловленно разными способами расчета процессорного премени). Так же следует использовать вместо функций ctime(), asctime() localtime(), их аналоги localtime_r, ctime_r, acstime_r.

- Функция clock_gettime обладает наибольшей точностью, а благодаря тому, что первый параметром передается ID таймера, данная функция имеет очень большую область применения. Единственный её минус, который мне удалось найти, в том, что она не является кроссплатформенной (работает только в Linux).

- Из всех расмотренных функций, функция omp_get_wtime является наиболее оптимальной т.к. является кросплатформенной, реентабильной и обладает хорошей точностью.

Стоит отметить тот факт, что программа, время выполнения который рамерялось, была довольно простой. Возможно, в программах с более сложной структурой, некоторые из функций начали давать неккоректные результаты. Но сложные программы в практической работе не рассчатривались ввиду их сложности.

Так же можно заметить, что время выполнения программы в Linux в 2.5 раза быстрее времени выполнения той же программы в Windows. Причины разницы выполнения описаны не будут, т.к. это не является частью данной практической работы, и требуют отдельного рассмотрения.

7.2 Вывод по дополнительной части практического задания

На поддержания виртуальной машины тратится определенное количество ресурсов компьютера. VirtualBox советует отдавать виртуальной машине более двух процессоров(как показано на Рис. 6.1). Эксперементально было проверено, что при увеличении количества процессоров у ВМ, время выполения программы немного уменьшается, это обусловлено распределением нагрузки между процессорами на поддержание витруальной машины. Однако при предоставлении отдельной программе более двух процессоров (при предоставлении двух, время выполнения программы сокращается примерно в 2 раза), итоговое время выполнения программы изменяется не линейно, а на небольшую величину, это связано с тем, что процессоры и без того загруженны (поддерживают виртуальную машину), и не в состоянии предоставить все свои возможности данной программе.

7.3 Вывод по производственной практике

В ходе прохождения зимней практики была изучена система компьютерной верстки LaTeX, система контроля версий Git (а вместе с ней и GitHub). Были получены общие представления о параллельных вычислениях, а так же было рассмотренно множество функций замеров времени с поледующим их сравнением.

Знания, полученные в ходе прохождения, считаю полезными, т.к.:

- LaTeX является хорошим инструментом для написания диплома на 4 курсе, а так же для написания научных работ, диссертаций.
- Знание Git (или другой системы контроля версий) крайне полезно для любого программиста. Т.к. упрощает написание и работу с программами.
- Знание функций замера времени, а так же их точность и т.д. позволит лучшем образом их использовать.
- Знание (или знакомство) с параллельными вычислениями пригодится в дальнейшем (магистратура/работа), т.к. данная дисциплина крайне востребованна в наше время.