Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

Дисциплина «Высокопроизводительные вычисления»

**Лабораторная работа № 1**

**Исследование средств измерения затрат времени исполнения кода**

Выполнил:

Студент группы ИВТАПбд-41

Галацков И.А.

Проверил:

Негода В. Н.

Ульяновск

2024

**Цель работы**

Цель работы заключается в исследовании и оценке различных методов измерения затрат времени исполнения кода в контексте высокопроизводительных вычислений. В данной работе следующие цели:

* изучить общие механизмы замеров затрат времени;
* определить разрешающую способность средств измерений;
* исследовать повторяемость результатов измерения;
* исследовать ошибки оценки затрат времени по эмпирической формуле с линейной зависимостью от размера рабочей нагрузки.

Основной задачей является анализ эффективности и точности различных механизмов измерения времени, а также их применимости к конкретным задачам вычислительного характера.

Функция по варианту: *ai \* th*(*xi*) - *bi* \* *sch*(*xi*).

Задача 1. Оценка влияния режима сборки debug на время исполнения кода

В данной задаче будет проведён сравнительный анализ режимов debug и release. Замеры затрат времени будет проводится тремя способами: clock, QPC, TSC.

Для подсчета времени следует выполнить следующие действия – разделить число тактов на частоту используемого таймера:

* Для TSC – частота процессора
* Для QPC – частота, определяемая функцией QueryPerformanceFrequency()
* Для clock – константа CLOCKS\_PER\_SEC

Замеры затрат времени будут сведены в Таблицу 1.1. Функция, которая будет замерена – функция Фибоначчи (Листинг 1.1).

Таблица 1.1. Замеры времени для функции Фибоначчи

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Время | | Debug | | Release | | T\_Debug / T\_Release | |
| n=10 | n=40 | n=10 | n=40 | n=10 | n=40 |
| clock | t\_code (ns) | 4530  COUNT = 100000 | 8149000000  COUNT = 1 | 470  COUNT = 1000000 | 841500000  COUNT = 10 | 9.638 | 9.684 |
| t\_end – t\_start (такты) | 453 | 8149 | 470 | 8415 | - | - |
| QPC | t\_code (ns) | 4510  COUNT =10 | 7920410000  COUNT = 1 | 486  COUNT = 100 | 849868000  COUNT = 1 | 9.280 | 9.320 |
| t\_end – t\_start (такты) | 451 | 79204099 | 486 | 8498676 | - | - |
| TSC: t\_code (ns) | | 4879.81 | 7798660000 | 468.76 | 841154000 | 10.410 | 9.271 |

Листинг 1.1. Функция Фибоначчи

|  |
| --- |
| unsigned \_\_int64 fibRecursive(unsigned \_\_int64 n) {  if (n < 2) return n;  return fibRecursive(n - 1) + fibRecursive(n - 2);  } |

У debug и release примерно одинаковое количество тактов, за которой функция выполняется 1 раз, но release примерно в 10 раз быстрее по времени выполняет такое же количество проходов одной функции.

Из приведенной таблицы видно, что время подсчета чисел Фибоначчи при запуске программы в Release меньше в 9-11 раза для любого из счетчиков. Данная закономерность объясняется следующими факторами:

* Оптимизации кода: В режиме release компилятор обычно применяет более агрессивные оптимизации кода, чтобы сделать исполняемый файл более эффективным по скорости выполнения. В режиме debug такие оптимизации обычно отключены, чтобы облегчить отладку и соответствие исходному коду.
* Дополнительная отладочная информация: В режиме debug в исполняемый файл включается дополнительная отладочная информация, такая как символьные таблицы, информация о переменных, отслеживание стека и т.д. Это позволяет использовать отладчик для анализа программы на уровне исходного кода.

В режиме Release уровень оптимизации обычно выше, чем в режиме Debug. Это означает, что компилятор старается создать исполняемый файл, который работает быстрее за счет применения более сложных оптимизаций.

**Задача 2. Исследование разрешающей способности**

Разрешающая способность средств измерений – это наименьшее изменение измеряемой величины, различимое с помощью данного средства измерения, в данной работе: clock, QPC, TSC.

Получить значение разрешающей способности можно, последовательно увеличивая объем обработки данных между двумя засечками времени до устойчивого получения значения, отличного от 0. Достаточно получения двух подряд значений, отличных от 0.

Для TSC разрешающая способность может быть получена без изменения рабочей нагрузки просто двумя вызовами \_\_rdtsc(), т.е. при пустом блоке CODE. И это будет не длительность такта, а задержка самой функции \_\_rdtsc().

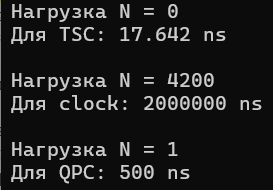
Для того чтобы найти разрешающую способность у clock и QPC, необходимо пошагово увеличивать рабочую нагрузку. После того как результат будет отличаться от 0, и если после следующего шага результат так же будет отличен от 0, то это будет нашей разрешающей способности (Рис. 2.1).

Рис. 2.1 – Исследование разрешающей способности

**Задача 3. Исследование повторяемости результатов измерения**

В данной задаче необходимо проверить повторяемость результатов в одних и тех же условиях.

Используются вектора A, B, X. После генерации входных параметров, находится результирующий вектор Y (Листинг 2.1).

Листинг 2.1. Функция нахождения результирующего вектора

|  |
| --- |
| double func(double x, double a, double b) {  return a / log(1-x) + b \* log(1+x);  } |

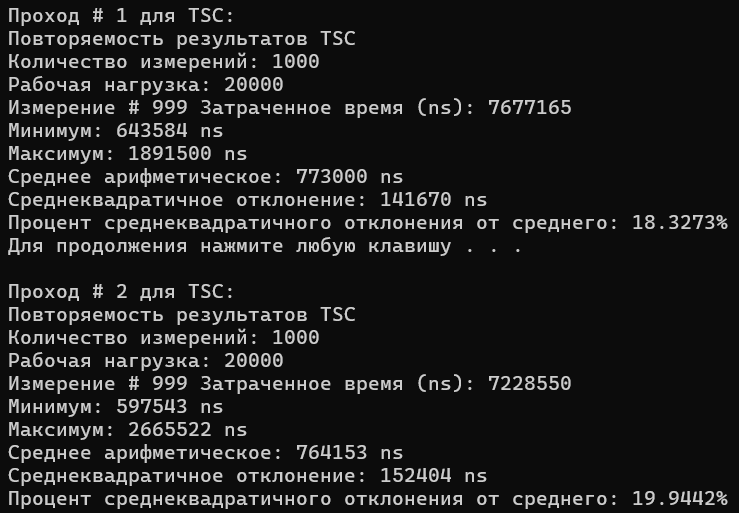
Для QPC и clock c помощью функций get\_size\_QPC() и get\_size\_clock() находится необходимая нагрузка, чтобы количество тактов было примерно одинаковая (Листинг 2.2). Для TSC же задаётся рабочая нагрузка.

Листинг 2.2. Нахождение рабочей нагрузки

|  |
| --- |
| #define STEP\_CLOCK 10000  #define STEP\_QPC 100  int get\_size\_clock() {  int count = 50000;  clock\_t t\_start\_clock, t\_end\_clock;  t\_start\_clock = clock();  t\_end\_clock = clock();  while (t\_end\_clock - t\_start\_clock < MIN\_COUNT\_TAKTS) {  x\_src.resize(count);  a\_src.resize(count);  b\_src.resize(count);  vector <double> y(count);  generate\_input\_vector(x\_src, count);  generate\_input\_vector(a\_src, count);  generate\_input\_vector(b\_src, count);  t\_start\_clock = clock();  for (int i = 0; i < y.size(); i++) {  y[i] = func(x\_src[i], a\_src[i], b\_src[i]);  }  t\_end\_clock = clock();  count += STEP\_CLOCK;  }  return count ;  }  int get\_size\_QPC() {  int count = 0;  LARGE\_INTEGER t\_start\_QPС, t\_finish\_QPС, freqQPС;  QueryPerformanceFrequency(&freqQPС);  QueryPerformanceCounter(&t\_start\_QPС);  QueryPerformanceCounter(&t\_finish\_QPС);  while (t\_finish\_QPС.QuadPart - t\_start\_QPС.QuadPart < MIN\_COUNT\_TAKTS\_QPC) {  x\_src.resize(count);  a\_src.resize(count);  b\_src.resize(count);  vector <double> y(count);  generate\_input\_vector(x\_src, count);  generate\_input\_vector(a\_src, count);  generate\_input\_vector(b\_src, count);  QueryPerformanceFrequency(&freqQPС);  QueryPerformanceCounter(&t\_start\_QPС);  for (int i = 0; i < x\_src.size(); ++i) {  y[i] = func(x\_src[i], a\_src[i], b\_src[i]);  }  QueryPerformanceCounter(&t\_finish\_QPС);  count += STEP\_QPC;  }  return count;  } |

Определение рабочей нагрузки. Перед началом измерений определяется размер рабочей нагрузки, который увеличивается в ходе цикла для создания различных уровней нагрузки.

Происходят 2 прохода замеров для каждого способа измерения. После завершения измерений проводится статистический анализ: минимума, максимума, среднего и стандартного отклонения времени выполнения для каждого метода измерения.

В результате работы программы показаны на рисунках 3.1 – 3.3.

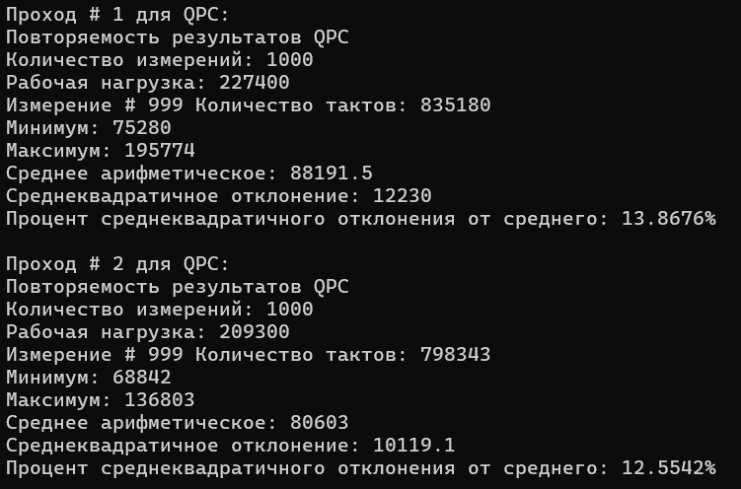
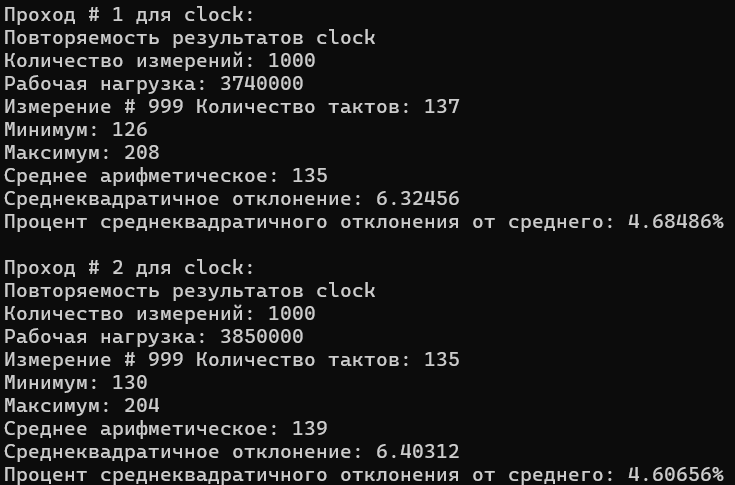
Рис. 3.1 – Исследование повторяемости TSC

Рис. 3.2 – Исследование повторяемости QPC

Рис. 3.3 – Исследование повторяемости clock

**Задача 4. Исследование ошибок оценки затрат времени по эмпирической формуле с линейной зависимостью от размера рабочей нагрузки**

Линейная зависимость времени от объема рабочей нагрузки выражается формулой T = k \* w, где w – объем рабочей нагрузки, k – коэффициент пропорциональности.

Для оценки погрешностей аналитического процесса относительно реального необходимо организовать серию измерений при разных значениях рабочей нагрузки с получением ряда значений (T(0), T(Δw), T(2\*Δw), T(3\*Δw),…), где Δw – шаг изменения рабочей нагрузки. Затем нужно оценить значение коэффициента пропорциональности либо путем линейной аппроксимации на основе полученного ряда значений. Для каждого счётчика производится 1015 измерений и отбрасываются определённое количество самых минимальных и максимальных значений.

На Листинг 4.1 кода представляет функцию calculateCoefficient(), которая вычисляет коэффициент пропорциональности по методу наименьших квадратов. Этот метод используется для оценки линейной зависимости между переменными.

Листинг 4.1. Нахождение коэффициента пропорциональности

|  |
| --- |
| double calculateCoefficient(vector<unsigned \_\_int64>& line\_stat, int step) {  unsigned \_\_int64 sum\_xy = 0;  unsigned \_\_int64 sum\_x = 0;  unsigned \_\_int64 sum\_y = 0;  unsigned \_\_int64 sum\_x2 = 0;  // Подсчет сумм  for (int i = 0; i < line\_stat.size(); ++i) {  unsigned \_\_int64 x = (i+1) \* step; // Δw \* i  unsigned \_\_int64 y = line\_stat[i];  sum\_xy += x \* y;  sum\_x += x;  sum\_y += y;  sum\_x2 += x \* x;  }  // Вычисление коэффициента пропорциональности по методу наименьших квадратов  double coefficient = (double)(line\_stat.size() \* sum\_xy - sum\_x \* sum\_y) / (line\_stat.size() \* sum\_x2 - sum\_x \* sum\_x);  coefficient = abs(coefficient);  return coefficient;  } |

В данной задаче модель предполагается линейной, где зависимость между переменными x и y выражается уравнением прямой: y = mx+c, где m - наклон прямой (коэффициент пропорциональности), а c - свободный член (константа).

Функция calculateCoefficient вычисляет коэффициент пропорциональности 𝑚m с помощью формулы:

Далее находится вектор абсолютного отклонения от линейной функции коэффицента. Такая же статистика как в задаче 3 получается для вектора абсолютных отклонений (Листинг 4.2).

|  |
| --- |
| vector<\_\_int64> deviations\_QPC;  for (int i = 0; i < line\_stat\_TSC.size(); ++i) {  \_\_int64 deviation = abs(line\_stat\_TSC[i] - calculateCoefficient(line\_stat\_TSC,  STEP\_WORKLOAD\_FOR\_TSC) \* i \* STEP\_WORKLOAD\_FOR\_TSC);  deviations\_QPC.push\_back(deviation);  } |

В результате работы программы получается статистика линейной зависимости для разных методов измерения затрат времени (Рис. 4.1).

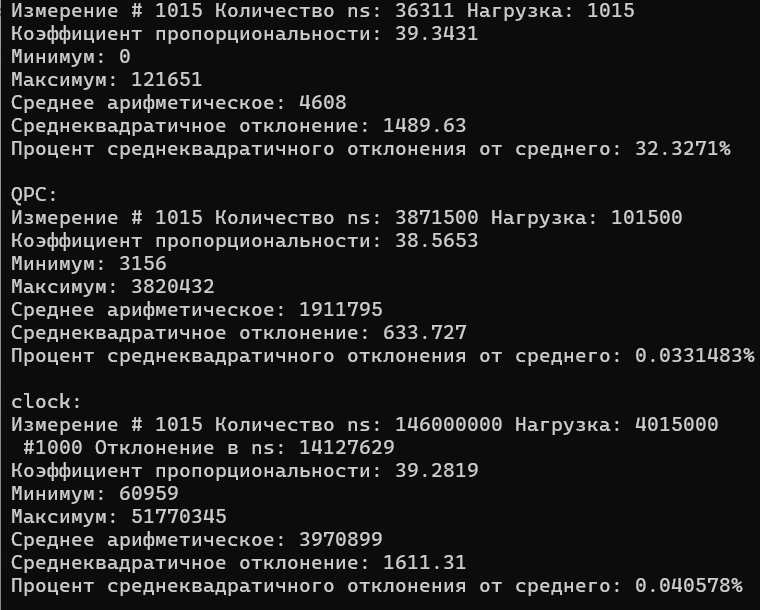


Рис. 4.1 – Исследование линейной зависимости

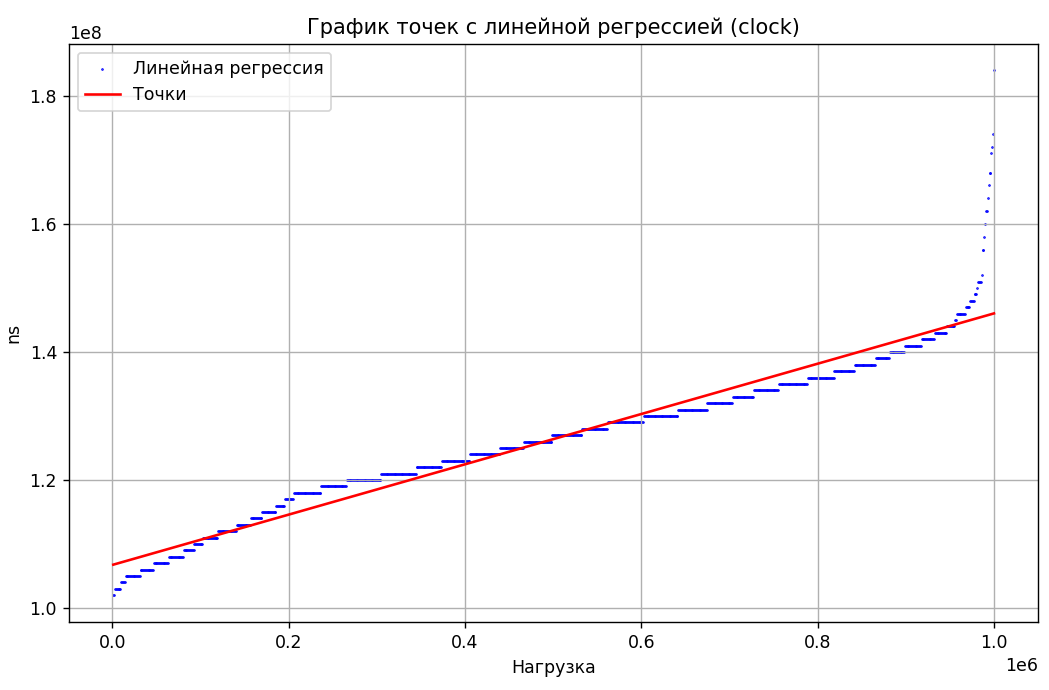


Рис. 4.2 – График линейной зависимости clock

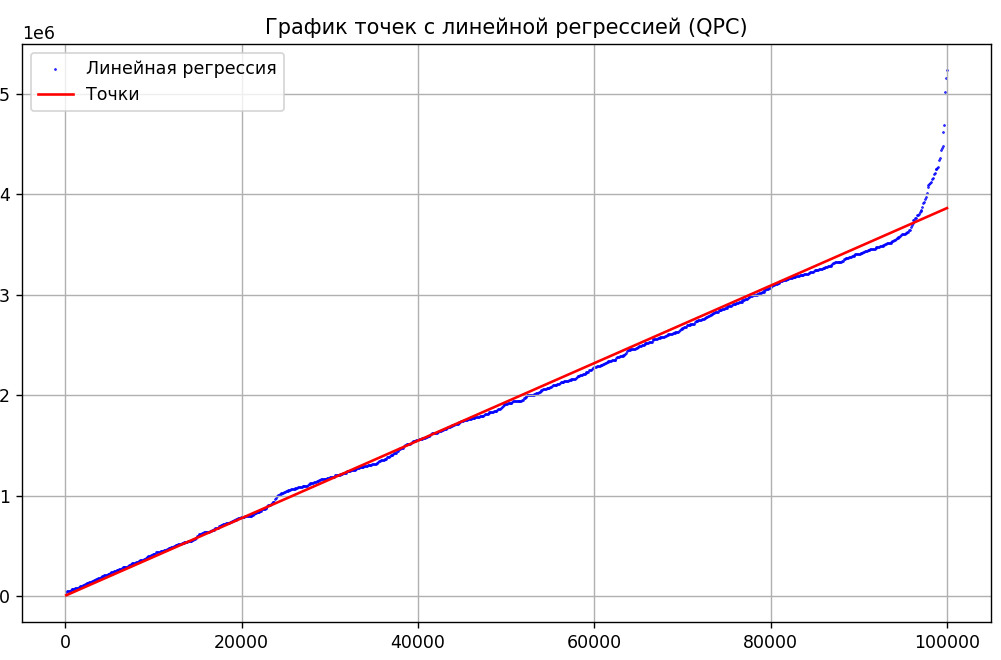


Рис. 4.3 – График линейной зависимости QPC

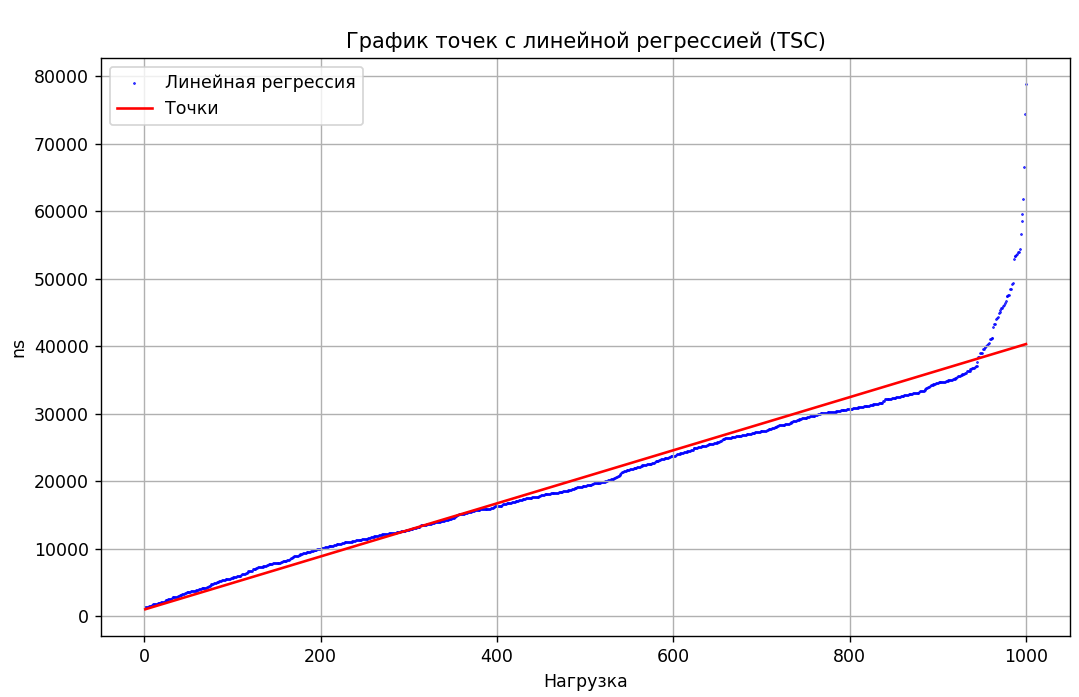


Рис. 4.4 – График линейной зависимости TSC

**Вывод**

В ходе выполнения работы были достигнуты следующие результаты:

1. Оценка влияния режима сборки на время исполнения кода:

* проведен сравнительный анализ режимов Debug и Release;
* показано, что режим Release работает значительно быстрее благодаря более агрессивным оптимизациям и отсутствию дополнительной отладочной информации.

1. Исследование разрешающей способности:
   * определена разрешающая способность средств измерений (clock, QPC, TSC) методом последовательного увеличения объема обработки данных до устойчивого получения значений, отличных от нуля.
2. Исследование повторяемости результатов измерения:
   * Проведена проверка повторяемости результатов в одинаковых условиях с использованием статистического анализа для каждого метода измерения.
3. Исследование ошибок оценки затрат времени:
   * реализован метод наименьших квадратов для оценки коэффициента пропорциональности между временем исполнения кода и размером рабочей нагрузки;
   * проведен анализ абсолютных отклонений от линейной зависимости для различных методов измерения затрат времени.

В результате работы были получены данные, позволяющие сделать вывод о эффективности и точности различных методов измерения времени исполнения кода в вычислительных задачах. Полученные результаты могут быть использованы для оптимизации процессов высокопроизводительных вычислений и выбора наиболее подходящего метода замера времени в конкретных сценариях использования.