ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Московский институт электроники и математики им. А.Н. Тихонова

Хомутов Евгений Васильевич

АНАЛИЗ ВИРТУАЛЬНОГО ВРЕМЕНИ В ОПТИМИСТИЧЕСКОМ АЛГОРИТМЕ ПАРАЛЛЕЛЬНОГО МОДЕЛИРОВАНИЯ ДИСКРЕТНЫХ СОБЫТИЙ

Междисциплинарная курсовая работа
студента образовательной программы
«Прикладная математика»,
группы БПМ-151

Руководитель

_____
Е.В. Хомутов

_____
Л.Н. Щур,
д-р физ.-мат. наук, проф.

Москва 2018 г.

1. Аннотация

В работе проводится анализ модели эволюции профиля локальных времен оптимистического алгоритма синхронизации параллельного моделирования дискретных событий (далее ПДМС). Модель рассматривается на топологии *Small-World* (малый мир). Small World – это граф с малой концентрацией дальних связей. В результате выполнения работы была определена численная зависимость скорости роста профиля локальных виртуальных времен (далее ЛВВ) от параметра модели. Значение показателя степени указывает на принадлежность задачи к классу универсальности направленной перколяции. Показано, что показатель загруженности системы на топологии SW в модели эволюции профиля ЛВВ у оптимистического алгоритма выше, чем у консервативного.

2. Abstract

This paper includes synchronization aspects of an optimistic algorithm for parallel discrete event simulations (PDES). A model for the time evolution in optimistic PDES is presented. This model evaluates the local virtual time profile of the processing elements using the Small-World topology. I present results of the research: the numeric dependence of the velocity on the parameter. I introduce that the exponent is close to the critical exponent in the class of universality of directed percolation. It is shown that the system load on the SW topology in the model of local virtual time profile evolution in the optimistic algorithm is higher than that of the conservative one.

3. Оглавление

4. Введение

Компьютерное моделирование можно выполнять либо последовательно, либо параллельно [24]. Для реализации последовательного алгоритма достаточно использовать один вычислительный узел. Для реализации параллельного алгоритма необходим компьютер, состоящий из нескольких вычислительных узлов (вычислительный кластер).

Выполнение одной программы моделирования на большом количестве вычислительных узлов требует изменений в вычислительных алгоритмах. Один из методов, позволяющих выполнять одну программу моделирования одновременно на нескольких вычислительных узлах – метод параллельного моделирования дискретных событий (ПДМС) [1].

Состояние системы, моделируемой методом ПМДС, изменяется в дискретные моменты времени. Такое изменение называется дискретным событием. Например в системе Internet событиями являются прием/передача пакетов.

Главная идея ПМДС заключается в представлении моделируемой системы как набор подсистем, взаимодействующих между собой. Каждой подсистеме соответствует свой логический процесс (ЛП). ЛП – это подпрограмма, которая последовательно выполняет моделирование своей подсистемы. Каждый ЛП имеет входящую и исходящую очередь сообщений с временными метками, а также свое локальное виртуальное время (ЛВВ) [2]. ЛП взаимодействуют между собой посредством обмена сообщениями со штампами времени, содержащими время отправки сообщения и время наступления события. Совокупность всех ЛВВ образует профиль ЛВВ, который эволюционирует в процессе моделирования. Прогресс моделирования определятся глобальным виртуальным временем, равным значению минимального ЛВВ.

Во время моделирования ЛП последовательно выбирает сообщение из входящей очереди с наименьшей временной меткой, обрабатывает его, изменяет свое ЛВВ и при необходимости отправляет сообщения другим ЛП. Для корректной работы программы моделирования необходимо, чтобы все события обрабатывались строго в хронологическом порядке. Поскольку динамика системы является асинхронной, может возникнуть ситуация, когда ЛП получает сообщение с временной меткой

меньшей, чем его ЛВВ. В таком случае возникает нарушение причинности. Для соблюдения причинности вычислений используются алгоритмы синхронизации [3]. Их можно классифицировать на три группы: консервативные [4,5], оптимистические [6,7] и алгоритм Freeze and Shift (FaS) [8].

В консервативных алгоритмах синхронизации допускается исполнение только безопасных событий. Оптимистические алгоритмы допускают нарушение причинности, но имеют механизмы их исправления. При обнаружении нарушения причинно-следственной связи запускается механизм отката. В конечном случае причинность восстанавливается.

В работе изучается модель эволюции профиля ЛВВ в оптимистическом алгоритме синхронизации ПМДС на сети Small-World. Изучение профиля ЛВВ позволяет делать выводы о степени синхронизации логических процессов между собой и о скорости эволюции системы в зависимости от начальных параметров.

Постановка задачи.

Изучить свойства модели роста профиля ЛВВ в оптимистическом алгоритме синхронизации ПМДС.

Актуальность задачи.

Актуальность исследования обусловлена быстрым развитием вычислительной техники и необходимостью развития новых параллельных алгоритмов. ПМДС широко применяется в экономике, логистике, физике и других областях науки.

Новизна исследования состоит в изучении оптимистических алгоритмов на модели топологии сетей малого мира (Small-world). Топология малого мира представляет собой регулярную топологию с малым числом дальних связей между узлами графа [10]. Исследование модели роста профиля ЛВВ в консервативном алгоритме на этой топологии описано в [11]. Выбор такой топологии обусловлен тем, что многие реальные системы являются сетями малого мира, например, социальные сети [11, 23], сети цитирования [25, 26], система нейронов мозга [21], язык и коммуникации внутри группы людей [23].

Методы исследования.

Основной метод исследования – это численное исследование модели

поведения профиля локальных виртуальных времен.

Для проведения серии вычислительных экспериментов был использован язык программирования ***C++***, библиотека многопоточного программирования OpenMPI и библиотека псевдослучайных чисел RNGAVXLIB [12,13]. Все расчеты выполнялись на вычислительном кластере Научного Центра в Черноголовке Российской Академии Наук “Мантикор”. Для отрисовки графиков использовался графический пакет OriginLab2015.

Исследование проведено при поддержке гранта 14-21-00158 Российского Научного Фонда.

5. Теоретическая часть

5.1 Основные понятия ПДМС

В данном параграфе мы определим базовые концепции ПДМС – логический процесс, локальное и глобальное виртуальное время, профиль ЛВВ, сообщения, событие, причинность вычислений.

Под *логическим процессором* (ЛП) мы понимаем подпрограмму, исполняемую на отдельном процессорном элементе.

В каждый момент времени моделирования каждый ЛП имеет свое *локальное виртуальное время* [2]. ЛВВ – это переменная, принимающая неотрицательные значения. Набор всех ЛВВ составляет *профиль ЛВВ* (Рис. 1). ЛВВ изменяется в результате обработки события. В общем случае ЛВВ может как возрастать, так и убывать (например, при откате в оптимистическом алгоритме). Локальное виртуальное время необходимо для синхронизации ЛП.

Глобальное виртуальное время (ГВВ) – это минимум из всех локальных виртуальных времен. Глобальное виртуальное время никогда не уменьшается и служит показателем общего прогресса моделирования системы.

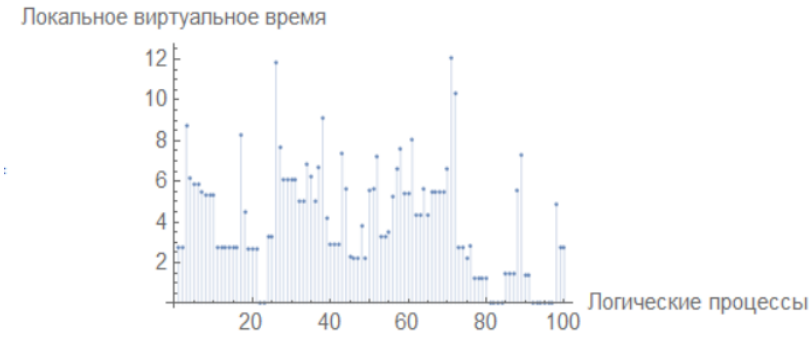

Рис.1 Мгновенный снимок профиля локальных виртуальных времен.

Сообщения, генерируемые ЛП, помимо информации о событии, содержат имя отправителя (номер ЛП, который отправляет сообщение), локальное виртуальное время отправителя, имя получателя и локальное виртуальное время доставки сообщения.

Событие – это последовательность вычислений, производимых процессом и включающая 0 или более следующих операций:

1. Процесс может обновлять свои переменные;
2. Процесс может читать свое виртуальное время;
3. Процесс может получать любое количество сообщений с указанным отправителем и виртуальным временем доставки и читать содержимое этих сообщений;
4. Процесс может отправлять любое количество сообщений, которые автоматически будут содержать метки отправителя - его номер и локальное виртуальное время отправления.

Система, основанная на концепции виртуального времени, должна подчиняться двум фундаментальным правилам [2]:

- **Правило 1:**

  Локальное виртуальное время каждого сообщения должно быть меньше локального виртуального времени его доставки.

- **Правило 2:**

  Локальное виртуальное время каждого события на любом из ЛП должно быть меньше локального виртуального времени следующего события.

Исходя из определенных выше правил, получаем определение причинности вычислений:

Если событие A является причинной события B, то они должны стоят в очереди так, что событие A закончилось до начала события B.

5.2 Классификация алгоритмов

5.2.1Консервативные алгоритмы

Первые разработанные алгоритмы синхронизации для ПМДС являлись консервативными [4,5]. Характерным признаком консервативных алгоритмов ПМДС является выполнение только *безопасных* процессов. Процесс считается безопасным, если заведомо известно, что его выполнение не приведет к нарушению причинно-следственной связи. Более строго это правило озвучил Reynolds в работе [15]:

Для любых ЛП A и B в консервативном алгоритме процесс A никогда не получит сообщение от процесса B с меньшим временем, чем локальное время процесса A.

Существует несколько подходов, гарантирующих выполнение этого правила. Первый заключается в том, что логический процесс проводит опрос всех ЛП с которыми он взаимодействует. В результате такого опроса формируется список событий этого ЛП. После этого ЛП выбирает событие с меньшим временем и выполняет его. Недостатком такого метода являются мертвые состояния – состояния, при которых несколько ЛП ждут сообщения друг от друга, находясь в заблокированном состоянии. Для решения этой проблемы Chandy и Misra [4,5] предложили алгоритм нахождения и исправления мертвых состояний.

Второй подход заключается в избегании мертвых состояний путем отправки пустых сообщений. Этот подход также имеет существенный недостаток – отправка пустых сообщений занимает процессорное время, что ведет к снижению скорости работы программы [1].

Примером консервативных алгоритмов синхронизации ПМДС может служить совместное наступление войск, когда различные боевые единицы должны сохранять

цепь наступления, двигаясь с определенной скоростью, для чего используются алгоритмы синхронизации, схожие с консервативными.

5.2.2Оптимистические алгоритмы

Оптимистический алгоритм предложил Jefferson, и этот алгоритм имеет название Time Warp [2,6,7].

В оптимистических алгоритмах допускается нарушение причинно-следственной связи. При этом имеются механизмы обнаружения и устранения ошибок. Все ЛП проводят свои расчеты независимо друг от друга в течение некоторого временного окна. В случае возникновения нарушения причинности действия ЛП могут быть отменены путем рассылки антисообщений. *Антисообщения* – это сообщения, которые отличаются от сообщений только знаком. Если в одной очереди встречаются сообщение и его антисообщение (два таких сообщения будут отличаться только знаками), они аннигилируют, по аналогии с частицей и античастицей в физике. В любой произвольный момент времени сумма всех сообщений и антисообщений должна быть равна нулю.

Особую роль при таком механизме устранения ошибок начинает играть глобальное виртуальное время. Глобальное виртуально время – это минимальное значение, до которого может откатиться локальное виртуальное время процесса. Все состояния ЛП до ГВВ могут быть удалены.

Примером системы с оптимистическим алгоритмом синхронизации может быть сборка автомобиля бригадой, каждый член которой делает свою определенную задачу. В течение некоторого времени каждый член бригады выполняет свое задание, после все обмениваются сообщениями и исправляют ошибки. Ошибка также может быть выявлена на этапе выполнения работы – к примеру попытка насадить колеса на неустановленную ось.

5.2.3Алгоритм FaS

В работе [8] было показано, что эволюция профиля ЛВВ в консервативном алгоритме ПМДС может быть описано уравнением Бюргерса. Эта связь позволила

классифицировать алгоритмы ПМДС в зависимости от граничных условий уравнения. Предложенный новый алгоритм *Freese and Shift* заключается в “заморозке” поверхностных ЛП в течение некоторого промежутка времени. Во время этой фазы их локальные времена не изменяются. С течением времени точка фиксации распространяется вглубь кластера ЛВВ, пока не будет остановлена половина всех ЛП. Тогда начинается процесс обмена сообщениями, который должен сохранять причинно-следственную связь.

Алгоритм синхронизации FaS был использован для решения уравнений в частных производных [19].

6. Практическая часть

6.1 Модель

Мы рассматриваем модель роста профиля ЛВВ в оптимистическом алгоритме ПМДС на топологии малого мира (Small World). Такую топологию описывает граф, вершинами которого являются ЛП, а ребрами – связи (зависимости) между ЛП (см. Рис.2). Small World – это вариант неполносвязной топологии, в котором большинство узлов не являются соседями, но достигнуть любого узла из любого другого можно за относительно маленькое число переходов. Графы такого типа относятся к сетям малого мира [10,11]. Такая топология часто встречается в реальных сетях – например топология сети малого мира хорошо описывает социальные сети.

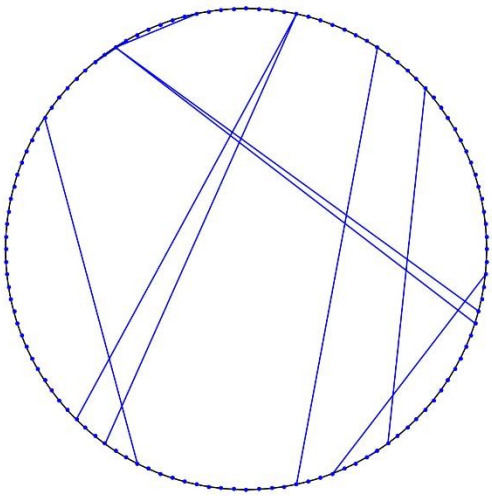

Рис 2. Вариант топологии малого мира

Первым этапом исследования был сбор информации о поведении ЛВВ – минимальное ЛВВ, среднее ЛВВ, максимальное ЛВВ и ширину профиля ЛВВ. Для этого была написана программа, моделирующая поведение профиля ЛВВ (см. приложение 1), которая может быть представлена псевдокодом (см. Рис 3):

Устанавливаем параметры N, M, p, b
Создаем сеть малого мира с pN случайных дальних связей.
for M раз **do**
  for N раз **do** $\tau_i(t) += \eta_i$
  K = РаспределениеПуассона(b)
  for $i = 1..(K \cdot N)$ **do**
    Выбираем случайный PE_j
    Выбираем случайного соседа у PE_j PE_r
    Если $\tau_j(t) > \tau_r(t)$ то $\tau_j(t) = \tau_r(t)$
  Высчитываем интересующие нас значения:
  $\tau_{min} = min(\tau(t))$
  $\tau_{avg} = \sum \tau_i(t)/N$
  $w^2(t) = \sum(\tau_i(t) - \tau_{avg})^2/N$
  N_{active} = количество ЛП, у которых ЛВВ меньше или равно времени его соседей
  $utilization(t) = N_{active}/N$

Рис 3. Псевдокод моделирования поведения профиля ЛВВ,

где τ – ЛВВ, w^2 – ширина профиля ЛВВ

При моделировании профиля ЛВВ мы использовали библиотеку псевдослучайных чисел RNGAVXLIB [12,13].

Далее, 1000 значений для каждого набора параметров, полученные в результате работы программы 1, были усреднены и вычислены их квадратичные отклонения от среднего по формуле несмещенного квадратичного отклонения. (см. Приложение 2). Программа 3 (см. Приложение 3) рассчитывает из полученных раннее данных новые величины, например шаг эволюции профиля ЛВВ, и разбивает данные на отдельные файлы для их дальнейшего анализа. При написании программы 3 была использована стандартная библиотека алгоритмов C++, что позволило сократить код в несколько раз и повысить его читаемость.

6.2 Исследование модели

Два основных параметра системы:

q – отношение числа шагов вперед на сумму шагов вперед и количества откатов назад;

p - процент дальних связей в топологии;

Для изучения системы определим главные показатели ПМДС: скорость эволюции и ширину локального профиля (все ЛП) виртуальных времен.

Средняя скорость эволюции показывает среднюю величину прироста локального времени за один шаг.

Ширина профиля показывает среднестатистическое отклонение локальных временен каждого ЛП от среднего значения.

Другие обозначения:

$\tau_i(t)$ - локальное время i-того процесса на шаге времени t;

τ(t) - глобальное виртуальное время системы на шаге t;

$w^2(t)$ - ширина профиля локальных виртуальных времен;

$u(t)$ - скорость эволюции профиля локальных виртуальных времен;

где t – дискретный шаг моделирования (wall-clock time).

Каждый ЛП i характеризуется своим локальным виртуальным временем (ЛВВ) τ_i(t) в момент времени t. Обозначим среднюю скорость роста профиля ЛВВ на шаге t как:

$$< u_i(t) > = < \tau_i(t+1) > - < \tau_i(t) >$$

Усредняя сначала по всем ЛП, а затем по времени, получаем значение $<u>$ - среднее значение роста профиля ЛВВ для данного набора параметров.

Под шириной профиля ЛВВ $\sqrt{<w^2>}$ будем понимать квадрат среднее квадратичное отклонение ЛВВ от среднего значения:

$$< w^2 >= \frac{1}{N}\sum(\tau_i(t) - \bar{\tau}(t))^2 \ ,$$

где $\bar{\tau}(t) = \frac{1}{N}\sum \tau_i(t)$

Профиль ЛВВ растет одинаково при различных параметрах p (см. Рис. 4). Для параметра $p = 0$ получены результаты, идентичные результатам в работе [9].

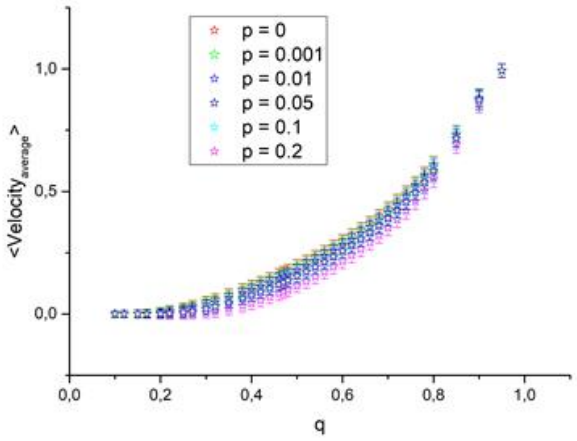

Рис. 4 Зависимость средней скорости локальных времен от параметра q.

До значения $q = 0.8$ графики могут быть аппроксимированы степенной функцией:

$$u = u_0(q - q_c)^{\nu},$$

где параметры зависят от значения p. Полученные значения u_0, q_c и ν для систем с разными параметрами p приведены в таблице 1.l.

p	u_0	q_c	ν
0	1.090(4)	0.143(2)	1.66(1)
0.001	1.166(6)	0.166(2)	1.70(2)
0.01	1.165(8)	0.166(2)	1.71(1)
0.05	1.27(2)	0.202(2)	1.74(2)
0.1	1.38(1)	0.224(2)	1.79(2)
0.2	1.53(2)	0.250(2)	1.86(2)

Таблица 1. Таблица зависимости параметров функции роста средних скоростей от процента случайных связей в системе.

Показатель степени ν при q = 0 указывает на принадлежность к классу направленной перколяции [20]. ($\nu_{\text{направленной перколяции}} = 1.733847(6)$)

Анализ ширины профиля ЛП указывает на то, что профиль ЛП выходит на константу с флуктуациями (см. Рис 5).

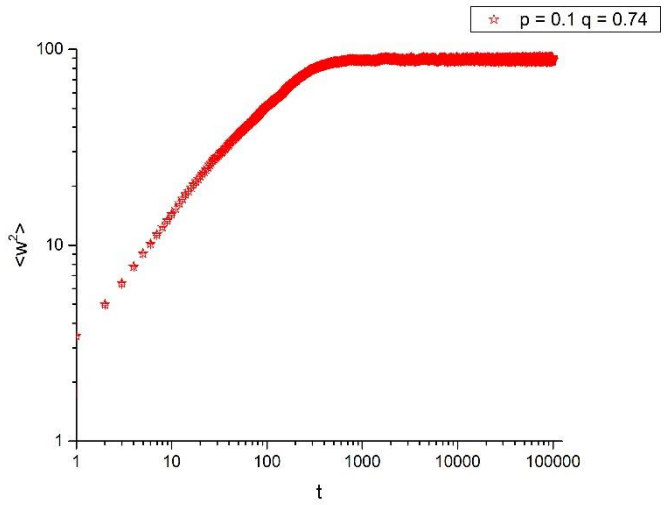

Рис 5. Профиль ЛП при параметрах системы
p = 0.1 и q = 0.74 в log-log координатах

В консервативном алгоритме средняя загруженность системы равна примерно 0.25 [16]. Результаты для оптимистического алгоритма отражены в Рис. 6 и Рис. 7.

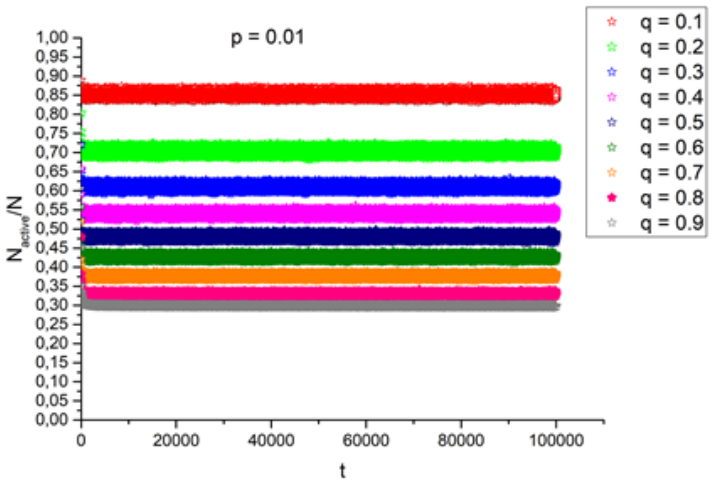

Рис. 5 Нагруженность системы при
p = 0.01 проценте дальних связей

Важно заметить, что высокая степень загруженности системы не означает что ЛВВ движутся вперед; высокая степень загруженности может быть связана с большим числом откатов системы.

В работе [16] проводится анализ консервативных алгоритмов, и устанавливается их связь с ростом поверхности. Важным результатом этой статьи является факт, что средняя нагруженность системы – отношение числа активных ЛП к общему числу ЛП – равно 0.25.

Загруженность системы может быть аппроксимирована функцией вида (см. Рис. 6):

$$utilization = U_0 + q_0 * a^q$$

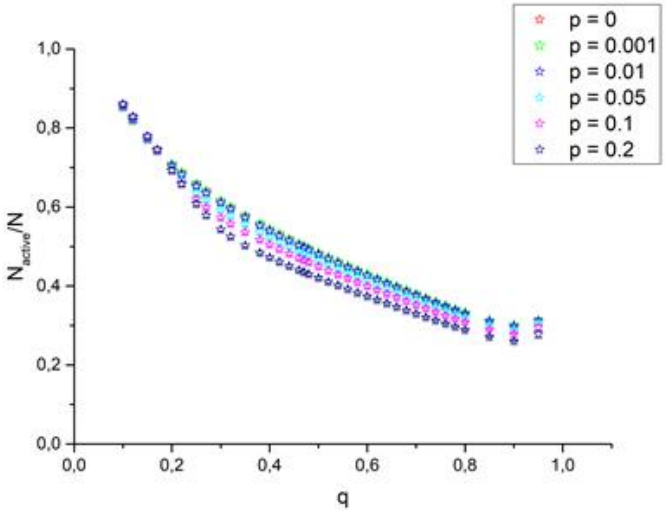

Рис. 6 Зависимость загруженности системы от параметров p и q

Параметры для функции определяются из таблицы 2.

p	U_0	q_0	a
0	0.039(8)	0.0866(5)	0.263(7)
0.001	-0.05(1)	0.940(9)	0.321(8)
0.01	0.046(9)	0.0871(6)	0.246(8)
0.05	0.07(3)	0.80(1)	0.24(3)
0.1	0.046(9)	0.871(6)	0.22(1)
0.2	0.179(7)	0.82(2)	0.082(8)

Таблица 2. Таблица зависимости параметров загруженности системы от процента случайных связей.

7. Заключение

В работе изучена эволюция профиля ЛВВ в оптимистическом алгоритме ПМДС. Модель дает возможность прогнозировать скорость роста профиля ЛВВ в зависимости от исходных параметров моделирования. Результаты работы могут применяться при моделировании различных дискретных моделей в таких областях, как физика, экономика, военное дело и другие.

Основные результаты, полученные в ходе исследования:

- Получены численные зависимости средней скорости локальных времен от параметра q.
- Численно получено критическое значение параметра q_c. При $q < q_c$ профиль не растет, процесс моделирования останавливается. При $q > q_c$ профиль растет с постоянной скоростью. Чем выше q, тем больше средняя скорость роста профиля ЛВВ.
- Загруженность системы на сети SW в модели эволюции профиля ЛВВ в оптимистическом алгоритме выше, чем консервативном.

Выражаю благодарность Л. Зигануровой за проведенные консультации!

Исследование проведено при поддержке гранта 14-21-00158 Российского Научного Фонда.

8. Список используемой литературы

1) R. M. Fujimoto, Parallel Discrete Event Simulation. — Commun. ACM., 1990.

2) D. Jefferson, Virtual time // ACM Transactions on Programming Languages and Systems (TOPLAS). — 1985. — Vol. 7, no. 3.

3) S. Jafer, Synchronization methods in parallel and distributed discreteevent simulation / S. Jafer, Q. Liu, G. Wainer // Simulation Modelling Practice and Theory. — 2013. — Vol. 30. — P. 54–73.

4) K. Chandy, Asynchronous distributed simulation via a sequence of parallel computations / K.M. Chandy, J. Misra // CACM.— 1981.— Vol. 24, no. 4. —P. 198–205.

5) K. Chandy, Distributed simulation: A case study in design and verification of distributed programs / K.M. Chandy, J. Misra // Software Engineering, IEEE Transactions. — 1989. — Vol. SE-5, no. 5.

6) D. Jefferson, B. Beckman, F. Weiland et al. - Distributed simulation and the time warp operating system // ACM SIGOPS Operating Systems Review. — 1987. — Vol. 21, no. 5. — P. 77–93.

7) D. Jefferson, Fast Concurrent Simulation Using the Time Warp Mechanism / D. Jefferson, H. Sowizral. — Santa Monica, Calif. : Rand Corp., 1982.

8) Л.Н. Щур, М.А. Новотный - Эволюция горизонта времен при параллельном моделировании дискретных событий // Труды Семинара по вычислительным технологиям в естественных науках. Вып. 1. Вычислительная физика / Под ред. Р. Р. Назирова. М.: Изд-во КДУ, 2009, с. 6-13

9) L. Ziganurova, M. A. Novotny, L. N. Shchur - Model for the evolution of the time profile in optimistic parallel discrete event simulations // International Conference on Computer Simulation in Physics and Beyond 2015

10)H. Gulcu, G. Korniss, M. Novotny et al. - Synchronization landscapes in small-world-connected computer networks // Physical Review Letters. — 2000. — Vol. 84, no. 1351.

11) Watts, D. J., Strogatz S.H. Collective dynamics of «small-world» networks // Nature. — 1998. — Vol. 393.

12) M.S. Guskova, L. Yu Barash and L. N. Shchur. "RNGAVXLIB: Program library for random number generation, AVX realization." Computer Physics Communications 200 (2016): 402405.

13) М.С. Гуськова. Л.Ю. Бараш, Л.Н. Щур “Библиотека случайных чисел RNGAVXLIB ” http://cpc.cs.qub.ac.uk/summaries/AEIT_v3_0.html Дата обращения 10.10.2017

14) G. Korniss, Z. Toroczkai, M. A. Novotny, P. A. Rikvold - Virtual time horizon control via communication network design /// Physical Review Letters. — 2006. — Vol. 73, no. 066115.

15) P. F. Reynolds, A spectrum of options for parallel simulation / P. F. Reynolds // In Processing of the 1988 Winter Simulatio Conference. — 1988. — P. 325–332.

16) G. Korniss, Z. Toroczkai, M. A. Novotny, P. A. Rikvold - From massively parallel algorithms and fluctuating time horizons to_nonequilibrium surface growth // Physical Review Letters. — 2000. — Vol. 84, no. 1351.

17) W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, eds - PARALLEL DISCRETE EVENT SIMULATION: THE MAKING OF A FIELD - Proceedings of the 2017 Winter Simulation Conference

18) R. M. Fujimoto, (1993) Feature Article—Parallel Discrete Event Simulation: Will the Field Survive. ORSA Journal on Computing 5(3):213-230

19) M.A. Novotny, L.N. Shchur Virtual Times for Parallel Computations of Partial Differential Equations // The Sixth Mississippi State — UAB Conf. on Differential Equations & Computational Simulations. Starkville, Mississippi State, USA. 2005.

20) G. Ódor - Universality classes in nonequilibrium lattice systems - Reviews of modern physics, 2004 – APS

21) D. S. Bassett , E. Bullmore - Small-World Brain Networks - *Neuroscientist 2006; 12; 512*

22) C. P. Herrero - Ising model in small-world networks - Phys. Rev. E 65, 066110 (2002).

23) R. F. i Cancho, R. V. Solé - The small world of human language – The Royal Society, 2261 – 2265

24) В.В. Воеводин, Вл. В. Воеводин – Параллельные вычисления - БХВ-Петербург, 2002. — 608 с. ISBN 5-94157-160-7.

25) M. E. J. Newman - Scientific collaboration networks. I. Network construction and fundamental results - Phys. Rev. E 64, 016131 – Published 28 June 2001

26) M. E. J. Newman - Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality - Phys. Rev. E 64, 016132 – Published 28 June 2001; Erratum Phys. Rev. E 73, 039906 (2006)

9. Приложение

9.1 Основная программа

```
/*********************************************************

The simple model of time evolution in optimistic PDES (parallel
discrete event simulation) algorithm on small-world topology.

Date: 15.09.2017

Authors:         Ziganurova Liliia, Khomutov Evgeniy

Purpose:        Parallel Discrete Event Simulation Research

Compile:        mpic++ -o optimisticSW.out SWv3+mpi+mt19937.cpp -std=c++11 -I../include -L../lib -lrngavx (g++ -o
test ./OSW.cpp -std=c++11 -I./include -L./lib -lrngavx)


Usage:          ./optimisticSW.out -N -M -r -q -p

Reference:      I use RNGAVXLIB library to generate random numbers
(http://cpc.cs.qub.ac.uk/summaries/AEIT_v3_0.html)

File formats:   The ouptut files are:  txt
*********************************************************/

#include <mpi.h>
#include <string>
#include <vector>
#include <algorithm>
#include <iterator>
#include "stdio.h"
#include "stdlib.h"
#include "time.h"
#include <fstream>
#include "mt19937.h"
#include <sys/stat.h>
#include <sys/types.h>

using namespace std;


//Function CreateOneDimSmallWorldTopolgy returns a graph of N nodes
//with pN random links added on the top of regular 1D topology(ring)

vector< vector<int> > CreateOneDimSmallWorldTopolgy(int N, float p, mt19937_state &state);

//Functiont PrintGraph prints graph : )
void PrintGraph(vector< vector<int> > graph);
/* Function ForwardPropogation updates array of LVT. We assume that all
LPs may update their times.
*/
void ForwardPropogation(double array_of_LVT[], size_t number_of_LP, double r, mt19937_state &state);

//Function Rollback choses a random node index and then compares its time with time of its random neighbour.
//If the time of the neigbour is lower, the time of node index becomes equal to the time of the neigbour.
//This procedure is made K*N times.
void  Rollback(double array_of_LVT[], vector< vector<int> > graph, size_t number_of_LP, double b, mt19937_state
&state);
/*============================================================

Function CreateOneDimSmallWorldTopolgy returns a graph of N nodes
with pN random links added on the top of regular 1D topology (ring)

Functiont PrintGraph prints graph :)

*/

vector< vector<int> > CreateOneDimSmallWorldTopolgy(int N, float p, mt19937_state &state) {

        vector< vector<int> > graph;
        int right_neighbour_ind, left_neighbour_ind;
```

```
        int from_node, to_node;

        //First Create a regular topology
        for (int index = 0; index < N; index++) {

                //Define boundaries
                if (index == 0) {
                        right_neighbour_ind = 1;
                        left_neighbour_ind = N - 1;
                }
                else if (index == N - 1) {
                        left_neighbour_ind = index - 1;
                        right_neighbour_ind = 0;
                }
                else {
                        left_neighbour_ind = index - 1;
                        right_neighbour_ind = index + 1;
                }

                // Add closest neighbours
                graph.push_back(vector <int>());
                graph[index].push_back(right_neighbour_ind);
                graph[index].push_back(left_neighbour_ind);
        }


        // Add random links
        int k = 0;
        int counter = 0;
        while (k < N*p)
        {
                from_node = mt19937_generate_(&state) % N;
                to_node = mt19937_generate_(&state) % N;

                //Проверка, чтобы не было кратных дуг и петель
                if ((from_node != to_node) &&
                        (find(graph[from_node].begin(), graph[from_node].end(), to_node) ==
graph[from_node].end())) {
                        graph[from_node].push_back(to_node);
                        graph[to_node].push_back(from_node); //так как граф не ориентированный
                        k += 1;
                }
        }
        return(graph);

} //GraphCreate



void PrintGraph(vector< vector<int> > graph) {
        int N = graph.size();

        for (int i = 0; i< N; i++) {
                cout << "Элемент " << i << " соседствует с ";
                for (int k = 0; k < graph[i].size(); k++)
                {
                        cout << graph[i][k] << " ";
                }
                cout << "\n";
        }
}

/*=============================================================*/

/* Function ForwardPropogation updates array of LVT. We assume that all
LPs may update their times.
*/

void ForwardPropogation(double array_of_LVT[], size_t number_of_LP, double r, mt19937_state &state) {

        double sluch, J;

        for (int i = 0; i<number_of_LP; i++) {
                // Генерация случайного числа от 0 до 1 (равномерное распределение)
                // Вычисление числа J (пуассоновское распределение с параметром r)

                sluch = mt19937_generate_uniform_float_(&state);
```

```
                if (sluch == 0) {
                        do { sluch = mt19937_generate_uniform_float_(&state); }
                        while (sluch == 0);
                        J = -r * log(sluch);
                }
                else {
                        J = -r * log(sluch);
                }

                // Прибавление к случайному ЛП числа J
                int index = mt19937_generate_(&state) % number_of_LP;

                //int index = rand() % number_of_LP;
                array_of_LVT[index] = array_of_LVT[index] + J;
        } // Завершение цикла по N_PE


}



/* Function Rollback choses a random node index and then compares its time with time of its random neighbour.
If the time of the neigbour is lower, the time of node index becomes equal to the time of the neigbour.
This procedure is made K*N times.
*/
void  Rollback(double array_of_LVT[], vector< vector<int> > graph, size_t number_of_LP, double b, mt19937_state
&state) {

        //Генерация числа K
        double sluch = mt19937_generate_uniform_float_(&state);
        int K;
        int nNeighbours, indNeighbour;
        int index;

        if (sluch == 0) {
                do { sluch = mt19937_generate_uniform_float_(&state); }
                while (sluch == 0);
                K = round(-b * log(sluch));
        }
        else { K = round(-b * log(sluch)); }

        // K*N раз
        for (int i = 0; i < K * number_of_LP; i++) {

                //Выбор случайного ЛП
                index = mt19937_generate_(&state) % number_of_LP;

                nNeighbours = graph[index].size(); // определили количество соседей

                indNeighbour = mt19937_generate_(&state) % nNeighbours;//выбрали случайного

                if (array_of_LVT[index] > array_of_LVT[graph[index][indNeighbour]])
                {
                        array_of_LVT[index] = array_of_LVT[graph[index][indNeighbour]];
                }


        } // Завершение цикла по K*N

}
/*=============================================================*/

/* Here is a block of simple function of calculating average, minimum,
width and local minima density of a given array */


struct Observables {
        double minimum;
        double max;
        double average;
        double width;
        double utilization;
public:
        //Observables():minimum(0), max(0), average(0), width(0), utilization(0) {};
        friend ostream& operator<<(ostream& os, const Observables& _this);
};
```

```
ostream& operator<<(ostream& os, const Observables& this_observables)
{
	os << this_observables.minimum << " " << this_observables.max << " " << this_observables.average << " " << 
this_observables.width << " " << this_observables.utilization << endl;
	return os;
}

Observables CalucateObservables(double array_of_LVT[], size_t number_of_LP, vector< vector<int> > graph) {

	double min = array_of_LVT[0];
	double max = array_of_LVT[0]; //!
	double sum = 0.0;
	double sum_square = 0.0;
	int number_of_active_LP = 0;
	int nNeighbours;
	double avg;
	bool Flag = 0;

	for (int i = 0; i < number_of_LP; i++) {


		sum += array_of_LVT[i];

		if (array_of_LVT[i] < min) min = array_of_LVT[i];
		if (array_of_LVT[i] > max) max = array_of_LVT[i];

		nNeighbours = graph[i].size();
		for (int q = 0; q < nNeighbours; q++) {
			if (array_of_LVT[i] > array_of_LVT[graph[i][q]]) {
				Flag = 0;
				break;
			}
			else Flag = 1;

		}
		if (Flag == 1) number_of_active_LP += 1;

	}
	avg = double(sum) / number_of_LP;

	for (int i = 0; i < number_of_LP; i++) {
		sum_square += pow((array_of_LVT[i] - avg), 2);
	}

	return{ min,max, avg , sum_square / number_of_LP, double(number_of_active_LP) / number_of_LP };
}

string give_path(const size_t number_of_LP, const size_t number_of_timesteps, const  float r, const float q, const 
float p) {
	return "N" + to_string(number_of_LP) + "M" + to_string(number_of_timesteps) + "r" + to_string(r) + "q" + 
to_string(q) + "p" + to_string(p);
}

int main(int argc, char* argv[]) {

	std::ios_base::sync_with_stdio(false);
	mkdir("result", 0777);

	int Ranks;
	int myRank;
	MPI_Init(&argc, &argv);
	MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
	MPI_Comm_size(MPI_COMM_WORLD, &Ranks);

	// Initializing  Generator
	mt19937_state state; // Initializating state.
	int number_of_LP = atoi(argv[1]); // Number of logical processes
	int number_of_timesteps = atoi(argv[2]); // Number of time steps
	float r = atof(argv[3]); //Percentage of random connections in range [0:1]
	float q = atof(argv[4]);
	float b = 1/q-1;
	float p = atof(argv[5]);
	mt19937_init_sequence_(&state, (unsigned long long)(time(NULL) + myRank));

	for (size_t j = 100 * (myRank); j < 100 * (myRank + 1); j++)
	{
```

```
                Observables my_observables;
                double *tau;
                vector< vector<int> >  my_topology;
                my_topology = CreateOneDimSmallWorldTopolgy(number_of_LP, p, state);
                tau = (double*)malloc(number_of_LP*sizeof(double)); //contains local virual times

        // Initialize array of local virual times
                for (int i = 0; i < number_of_LP; i++)  tau[i] = 0.0;

                //Open file for writing
                string path = "result/";
                path += give_path(number_of_LP, number_of_timesteps, r, q, p);
                mkdir(path.c_str(), 0777);
                path = path + "/test" + to_string(j) + ".txt";
                ofstream out(path, ios_base::trunc);
                out.precision(8);

                // Begin to model the evolution of LVT profile
                for (int t = 0; t < number_of_timesteps; t++)
                {
                        ForwardPropogation(tau, number_of_LP, r, state);
                        Rollback(tau, my_topology, number_of_LP, b, state);
                        my_observables = CalucateObservables(tau, number_of_LP, my_topology);
                        out << t << " " << my_observables;
                }
                out.close();
        }

        MPI_Barrier(MPI_COMM_WORLD);
        MPI_Finalize();

        return 0;
}
```

9.2 Код программы для усреднения результатов по 1000 моделей.

```
#include <fstream>
#include <string>
#include <cmath>
#include "stdio.h"
#include "stdlib.h"
#include <sys/stat.h>
#include <sys/types.h>

using namespace std;

// Структура, как для среднего значения, так и для среднего квадратичного
// Используем массив таких структур из number_of_timesteps элементов
struct value {
        double minimum = 0;
        double max = 0;
        double average = 0;
        double width = 0;
        double utilization = 0;
};

// Считаем среднее
void average(value &sr);
// Берем корень из суммы квадратов отклонений на number_of_timesteps;
void root(value &error);
// Функция, которая добавляет значения из файлов:
void add(ifstream &in, value &sr, value &error);

void count(value &sr, value &error);
// Указываем путь к файлу из которого будем читать
string give_path(const int i);
// Выписываем в файл
void write(ofstream &out, size_t j, value Val, value Val2);

// Количество проведенных тестов
const size_t number_of_simulation = 1000;
// Количество шагов в одной симуляции
const size_t number_of_timesteps = 100000;



int main(int argc, char* argv[])
{
        std::ios_base::sync_with_stdio(false);
        double r = 1.0;
        double p = atof(argv[1]);
        size_t number_of_LP = 1024;
        mkdir(("Result" + to_string(p)).c_str(), 0777);
        ofstream OUT("OUT" +to_string(p)+".txt", ios_base::trunc);
        for (double q = 0; q < 1.1; q += 0.1) {

                string name = "N";
                name += to_string(number_of_LP);
                name += "M";
                name += to_string(number_of_timesteps);
                name += "r";
                name += to_string(r);
                name += "q";
                name += to_string(q);
                name += "p";
                name += to_string(p);
                name +="/";
OUT << name << std::endl;
                // Массивы средних значений, для каждого шага
                value sr[number_of_timesteps];
                value error[number_of_timesteps];
                // Добавляем все значения на каждом шаге
                for (int i = 0; i < number_of_simulation; i++) {
                        ifstream in(name+give_path(i), ios_base::in);
                        for (size_t j = 0; j < number_of_timesteps; j++)
                        {
                                add(in, sr[j], error[j]);
```

```
                        }
                        in.close();
                }
                // Высчитываем среднее значение на каждом шаге и записываем в файл
                for (size_t j = 0; j < number_of_timesteps; j++)
                {
                        // Делим на количество симуляций
                        average(sr[j]);
                        average(error[j]);
                        // Вычитаем из суммы квадратов деленных на K квадрат суммы деленной на K
                        count(sr[j], error[j]);
                }
                string ut = "Result" + to_string(p)+"/result";
                ut += to_string(q);
                ut += ".txt";
                ofstream out(ut, ios_base::trunc);
                // Считаем корень из суммы квадратов отклонений на number_of_timesteps;
                // И сразу пишем в файл
                for (size_t j = 0; j < number_of_timesteps; j++)
                {
                        // Берем корень из разности суммы квадратов деленной на K и квадраты суммы деленной на K
деленную на K - 1
                        root(error[j]);
                        write(out, j, sr[j], error[j]);
                }
                out.close();
        }
        return 0;
}
string give_path(const int i) {
        string path = "test";
        path += to_string(i);
        path += ".txt";
        return path;
}
void average(value & sr)
{
        sr.minimum = sr.minimum / number_of_simulation;
        sr.max = sr.max / number_of_simulation;
        sr.average = sr.average / number_of_simulation;
        sr.width = sr.width / number_of_simulation;
        sr.utilization = sr.utilization / number_of_simulation;
}
void root(value & error)
{
        error.minimum = sqrt(error.minimum / (number_of_simulation- 1));
        error.max = sqrt(error.max / (number_of_simulation - 1));
        error.average = sqrt(error.average / (number_of_simulation - 1));
        error.width = sqrt(error.width / (number_of_simulation - 1));
        error.utilization = sqrt(error.utilization / (number_of_simulation - 1));

}
void add(ifstream & in, value & sr, value & error)
{
        size_t k;// Переменная, считывающая первую столбец из матрицы, то есть номер шага
        double _minimum = 0;
        double _max = 0;
        double _average = 0;
        double _width = 0;
        double _utilization = 0;
        in >> k >> _minimum >> _max >> _average >> _width >> _utilization;

        sr.minimum += _minimum;
        sr.max += _max;
        sr.average += _average;
        sr.width += _width;
        sr.utilization += _utilization;

        error.minimum += (_minimum)*(_minimum);
        error.max += (_max)*(_max);
        error.average += (_average)* (_average);
        error.width += (_width)*(_width);
        error.utilization += (_utilization)*(_utilization);
}
void count(value & sr, value & error){

        error.minimum -= sr.minimum*sr.minimum;
```

```
        error.max -= sr.max*sr.max;
        error.average -= sr.average*sr.average;
        error.width -= sr.width*sr.width;
        error.utilization -= sr.utilization*sr.utilization;
}
void write(ofstream &out, size_t j, value Val, value Val2) {
        // Выводим первые 8 значащих знака после запятой
        out.precision(8);
        out << j << " " << Val.minimum << " " << Val2.minimum << " "
                << Val.max << " " << Val2.max << " "
                << Val.average << " " << Val2.average << " "
                << Val.width << " " << Val2.width << " "
                << Val.utilization << " " << Val2.utilization << endl;;
}
```

9.3 Код программы для сбора статистических данных.

```
#include "stdafx.h"
#include "fstream"
#include "numeric"
#include <algorithm>
#include <cmath>
#include <functional>
#include <vector>
#include "thread"
#include "string"
#include <direct.h> // <-- mkdir(("Result width p = " + P).c_str());


template <class T>
double Mean(const std::vector<T> data) {
	return std::accumulate(data.begin(), data.end(), 0.0f) / data.size();
}

template <class T>
double StandartDevesition(std::vector<T> data, double mean)
{
	std::transform(data.begin(), data.end(), data.begin(), std::bind2nd(std::minus<float>(), mean)); // <---
Вычитаем из каждого элемента среднее
	float deviation = std::inner_product(data.begin(), data.end(), data.begin(), 0.0f); // <-- Представляем
числитель нашей формулы как скалярное произведение
	return std::sqrt(deviation / (data.size() - 1)); // <-- Берем корень
}

struct Step {
	size_t StepNumber = 0;
	double Minimum = 0;
	double ErrorMinimum = 0;
	double Max = 0;
	double ErrorMax = 0;
	double Average = 0;
	double ErrorAverage = 0;
	double Width = 0;
	double ErrorWidth = 0;
	double Utilization = 0;
	double ErrorUtilization = 0;
public:
	friend std::istream& operator >> (std::istream& os, Step& oqject);
};

std::istream & operator >> (std::istream & os, Step & step)
{
	os >> step.StepNumber >> step.Minimum >> step.ErrorMinimum >>
		step.Max >> step.ErrorMax >> step.Average >> step.ErrorAverage >>
		step.Width >> step.ErrorWidth >> step.Utilization >> step.ErrorUtilization;
	return os;
}


const std::string p[] = {"0","0.001","0.01","0.05","0.1","0.2"}; // <-- Каие значения p у нас использовались
const size_t StepsNumber = 100000;

int Function(std::string p)
{
	_mkdir(("Min(t)/p = " + p).c_str());
	_mkdir(("Max(t)/p = " + p).c_str());
	_mkdir(("Average(t)/p = " + p).c_str());
	_mkdir(("Width(t)/p = " + p).c_str());
	_mkdir(("Utilization(t)/p = " + p).c_str());

	std::ofstream info("info/" + p + ".txt");

	std::ofstream VelocityMinOut("Min(q)/p =" + p + " Velocity(q) MIN.txt");
	std::ofstream VelocityMaxOut("Max(q)/p =" + p + " Velocity(q) MAX.txt");
	std::ofstream VelocityAverageOut("Average(q)/p =" + p + " Velocity(q) AVERAGE.txt");
	std::ofstream WidthOut("Width(q)/p =" + p + " Width(q).txt");
	std::ofstream UtilizationOut("Utilization(q)/p =" + p + "Utilization.txt");

	for (double q = 0.1; q < 1.0; q += 0.01) {
		std::ifstream from("p = " + p + "/result" + std::to_string(q) + ".txt");
```

```
		info << ("result" + std::to_string(q) + ".txt") << std::endl;
		if (from.is_open()) {
			// Инициализация необходимых массивов
			std::vector <double> Minimum;
			std::vector <double> Max;
			std::vector <double> Average;
			std::vector <double> Width;
			std::vector <double> Utilization;
			//
			// Чтение
			{
				Step step;
				// Тут инициализовать куда записать зависимости от t
				std::ofstream toMinimum("Min(t)/p = " + p + "/q = " + std::to_string(q) + ".txt");
				std::ofstream toMax("Max(t)/p = " + p + "/q = " + std::to_string(q) + ".txt");
				std::ofstream toAverage("Average(t)/p = " + p + "/q = " + std::to_string(q) +
".txt");
				std::ofstream ToWidth("Width(t)/p = " + p + "/q = " + std::to_string(q) + ".txt");
				std::ofstream ToUtilization("Utilization(t)/p = " + p + "/q = " + std::to_string(q)
+ ".txt");
				//
				for (size_t j = 0; j < StepsNumber; j++) {
					from >> step;
					// Тут выписываем зависимотси от t
					toMinimum << step.StepNumber << " " << step.Minimum << " " << step.Error-
Minimum << std::endl;
					toMax << step.StepNumber << " " << step.Max << " " << step.ErrorMax <<
std::endl;
					toAverage << step.StepNumber << " " << step.Average << " " << step.Er-
rorAverage << std::endl;
					ToWidth << step.StepNumber << " " << step.Width << " " << step.ErrorWidth
<< std::endl;
					ToUtilization << step.StepNumber << " " << step.Utilization << " " <<
step.ErrorUtilization << std::endl;

					Minimum.push_back(step.Minimum);
					Max.push_back(step.Max);
					Average.push_back(step.Average);
					Width.push_back(step.Width);
					Utilization.push_back(step.Utilization);
				}

				toMinimum.close();
				toMax.close();
				toAverage.close();
				ToWidth.close();
				ToUtilization.close();
			}
			from.close();
			//
			// Обрабатываем массивы с тау, чтобы получить скорость
			std::adjacent_difference(Minimum.begin(), Minimum.end(), Minimum.begin()); // Получаем раз-
ность между элементами t + 1 и t
			std::adjacent_difference(Max.begin(), Max.end(), Max.begin());
			std::adjacent_difference(Average.begin(), Average.end(), Average.begin());
			//
			// Надо убрать первые 500 значений
			Minimum.erase(Minimum.begin(), Minimum.begin() + 500);
			Max.erase(Max.begin(), Max.begin() + 500);
			Average.erase(Average.begin(), Average.begin() + 500);
			// Поменять стандартный вывод на свой
			{
				double MeanMinimum = Mean(Minimum);
				VelocityMinOut << q << " " << MeanMinimum << " " << StandartDevesition(Minimum,
MeanMinimum) << std::endl;

				double MeanMax = Mean(Max);
				VelocityMaxOut << q << " " << MeanMax << " " << StandartDevesition(Max, MeanMax) <<
std::endl;

				double MeanAverage = Mean(Average);
				VelocityAverageOut << q << " " << MeanAverage << " " << StandartDevesition(Average,
MeanAverage) << std::endl;

				double MeanWidth = Mean(Width);
				WidthOut << q << " " << MeanWidth << " " << StandartDevesition(Width, MeanWidth) <<
std::endl;
```

```
                                double MeanUtilization = Mean(Utilization);
                                UtilizationOut << q << " " << MeanUtilization << " " << StandartDevesition(Utiliza-
tion, MeanUtilization) << std::endl;

                        }
                }
        }
        info.close();
        VelocityMinOut.close();
        VelocityMaxOut.close();
        VelocityAverageOut.close();
        WidthOut.close();
        UtilizationOut.close();
        return 0;
}

int main() {
        std::ios_base::sync_with_stdio(false); // <-- Для ускорения работы программы

        _mkdir("info");

        _mkdir("Min(t)");
        _mkdir("Max(t)");
        _mkdir("Average(t)");
        _mkdir("Width(t)");
        _mkdir("Utilization(t)");
        _mkdir("Min(q)");
        _mkdir("Max(q)");
        _mkdir("Average(q)");
        _mkdir("Width(q)");
        _mkdir("Utilization(q)");

        std::thread P0(Function, p[0]);
        std::thread P1(Function, p[1]);
        std::thread P2(Function, p[2]);
        std::thread P3(Function, p[3]);
        std::thread P4(Function, p[4]);
        std::thread P5(Function, p[5]);

        P0.join();
        P1.join();
        P2.join();
        P3.join();
        P4.join();
        P5.join();
        }
```