Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования

**«Уральский федеральный университет имени первого Президента России Б. Н. Ельцина»**

Физико-технологический институт
Кафедра теоретической физики и прикладной математики

# Пояснительная записка к курсовой работе по теме РЕАЛИЗАЦИЯ МНОГОПОТОКОВОГО АЛГОРИТМА ЧИСЛЕННОГО РЕШЕНИЯ КЛАССИЧЕСКОЙ СПИНОВОЙ МОДЕЛИ МЕТОДОМ ГЛОБАЛЬНОЙ ОПТИМИЗАЦИИ ПРИ ПОМОЩИ БИБЛИОТЕКИ MPI

Работа студента
группы ФтМ - 170403
Вяловой С.А.

Преподаватель:
д.ф.-м.н., профессор
Мазуренко В.В.

Консультант:
м.н.с.
Медведева Д. С.

Екатеринбург
2018

# Содержание

**Введение** **2**

**1. Теоретическая часть** **4**
1.1. MPI (Message Passing Interface) . . . . . 4
1.1.1. Терминология и обозначения . . . . . 4
1.1.2. Общие процедуры MPI . . . . . 5
1.1.3. Передача/прием сообщеий между отдельными процессами . . . 5
1.1.4. Передача /прием сообщеий с блокировкой . . . . . 6
1.1.5. Коллективные взаимодействия процессов . . . . . 7

**2. Практическая часть** **9**
2.1. Параметры системы . . . . . 9
2.2. Параметры запуска и блок-схема программы . . . . . 9
2.3. Разработка паралелльной версии программы. Технология MPI . . . 13

**Заключение** **20**

**Список использованных источников** **21**

# Введение

В настоящее время ряд задач, требующих для своего решения применения мощных вычислительных ресурсов, стремительно расширяется. Это связано с тем, что происходят фундаментальные изменения в самой организации научных исследований. Вследствие широкого внедрения вычислительной техники, значительно усилилось направление численного моделирования и численного эксперимента. Численное моделирование, заполняя промежуток между физическими экспериментами и аналитическими подходами, позволило изучать явления, которые являются либо слишком сложными для исследования аналитическими методами, либо слишком дорогостоящими или опасными для экспериментального изучения. При этом численный эксперимент позволил значительно удешевить процесс научного и технологического поиска. Стало возможным моделировать в реальном времени процессы интенсивных физико-химических и ядерных реакций, глобальные атмосферные процессы, процессы экономического и промышленного развития регионов и т.д. Очевидно, что решение таких масштабных задач требует значительных вычислительных ресурсов [2]. Это ставит перед нами задачу уменьшения времени, затрачиваемого на вычисления.

Вследствие вышесказанного, актуальной становится разработка программ, которые могут запускаться в параллельном режиме. Для этого могут быть использованы разные подходы, среди которых можно выделить интерфейс MPI (Message Passing Interface) – библиотеку функций, обеспечивающую взаимодействие параллельных процессов с помощью механизма передачи сообщений [3].

Ряд задач в теоретической физике требуют численного решения классических спиновых моделей. Проблема заключается заключается в поиске глобального минимума энергии таких систем. Поскольку зачастую есть необходимость решить такую задачу для большого числа однотипных спиновых систем, то, с точки зрения времени, затраченного на вычисления, разумно ее выполнять параллельно в несколько потоков.

Таким образом, целью данной работы является реализация многопотокового алгоритма численного решения классической спиновой модели методом глобальной оптимизации при помощи библиотеки функций MPI, а также оценка преимуществ проведения серии расчётов при помощи многопоточной программы относительно однопоточной программы. Для достижения поставленных целей были поставлены следующие задачи:

- Проанализировать имеющееся программное решение поставленной задачи;
- разработать параллельную версию программы, пригодную для запуска на различных типах машин;

- сравнить время выполнения данной задачи, выполняющейся в один поток и в несколько потоков.

Объектом данной курсовой работы является параллельное исполнение программы на CPU, предметом - классическая спиновая модель.

# Теоретическая часть

## 1.1. MPI (Message Passing Interface)

### 1.1.1. Терминология и обозначения

Интерфейс MPI представляет собой набор утилит и библиотечных функций для языков C/C++ и FORTRAN, позволяющих создавать и запускать приложения, работающие на параллельных вычислительных установках [5]. MPI-программа – это множество параллельных взаимодействующих процессов. Все процессы порождаются один раз, образуя параллельную часть программы. Каждый процесс работает в своем адресном пространстве, никаких общих переменных или данных в MPI нет. Основным способом взаимодействия между процессами является явная посылка сообщений.

Для локализации взаимодействия параллельных процессов программы можно создавать группы процессов, предоставляя им отдельную среду для общения - коммуникатор. Состав образуемых групп произволен. Группы могут полностью совпадать, входить одна в другую, не пересекаться или пересекаться частично. Процессы могут взаимодействовать только внутри некоторого коммуникатора. а сообщения, отправленные в разных коммуникаторах, не пересекаются и не мешают друг другу. Коммуникаторы имеют в языке Фортран тип **INTEGER**.

При старте программы всегда считается, что все порожденные процессы работают в рамках всеобъемлющего коммуникатора, имеющего предопределенное имя **MPI_COMM_WORLD**. Этот коммуникатор существует и всегда служит для взаимодействия всех запущенных процессов MPI-программы. Кроме него при старте программы имеется коммуникатор **MPI_COMM_SELF**, содержащий только текущий процесс, и коммуникатор **MPI_COMM_NULL**, не содержащий ни одного процесса. Все взаимодействия процессов протекают в рамках определённого коммуникатора, а сообщения, переданные в разных коммуникаторах, никак не мешают друг другу.

Каждый процесс MPI-программы имеет в каждой группе, в которую он входит, уникальный атрибут номер процесса, который является целым неотрицательным числом. С помощью этого атрибута происходит значительная часть взаимодействия процессов межлу собой. В одном и том же коммуникаторе все процессы имеют различные номера, но, поскольку процесс может одновременно входить в разные коммуникаторы, то его номер в одном коммуникаторе может отличаться от его номера в другом. Отсюда становятся понятными два основных атрибума процесса: коммуникатор и номер в коммуникаторе.

Посылка сообщений - основной способ общения процессов между собой. Со-

общение - это набор данных некоторого типа. Каждое сообщение имеет несколько атрибутов, в частности, номер процесса-отправителя, номер процесса-получателя, идентификатор сообщения и другие. Одним из важных атрибутов сообщения является его идентификатор или тэг. По идентификатору процесс, принимающий сообщение, может различать два сообщения, пришедшие к нему от одного и того же процесса. Сам идентификатор сообщения является целым неотрицательным числом, лежащим в диапазоне от 0 до **MPI_TAG_UP**, и гарантируется, что **MPI_TAG_UP** не меньше 32767. [1]

### 1.1.2. Общие процедуры MPI

Рассмотрим ряд процедур, необходимых в каждой содержательной параллельной программе.

- **MPI_INIT(IERR)**
  **INTEGER IERR**

Инициализация параллельной части программы. Все другие процедуры MPI могут быть вызваны только после вызова **MPI_INIT**. Инициализация параллельной части для каждого приложения должна выполняться только один раз.

- **MPI_FINALIZE(IERR)**
  **INTEGER IERR**

Завершение параллельной части приложения. Все последующие обращения к любым процедурам MPI, в том числе **MPI_INIT(IERR)**, запрещены. К моменты вызова **MPI_FINALIZE(IERR)** каждым процессов программы все действия, требующие его участия в обмене сообщениями, должны быть завершены

- **MPI_COMM_SIZE(COMM, SIZE, IERR)**
  **INTEGER COMM, SIZE, IERR**

В аргументе **SIZE** процедура возвращает число параллельных процессов в коммуникаторе **COMM**.

### 1.1.3. Передача/прием сообщеий между отдельными процессами

Практически все программы, написанные с использованием коммуникационной технологии MPI, должны содержать средства не только для порождения и завершения параллельных процессов, но и для взаимодействия запущенных

[1] Для описния использованных директив использован Антонов А.С. Учебное пособие "Параллельное программирование с использованием MPI".

процессов между собой. Такое взаимодействие и осуществляется в MPI посредством явной посылки сообщений.

Все процедуры передачи сообщений в MPI делятся на две группы. В одну группу входят процедура, которые предназначены для взаимодейсвия только двух процессов программы. Такие операции называются индивидуальными или операциями типа точка-точка. Процедуры другой группы предполагают, что в операцию должны быть вовлечены все процессоры некоторого коммуникатора. Такие операции называются коллективными.

Рассмотрим процедуры обмена сообщениями на примере операций типа точка-точка. В таких взаимодействиях участвуют два процесса, причём один процесс является отправителем сообщения, а другой - получателем. Процесс-отправитель должен вызвать одну из процедур передачи данных и явно указать номер в некотором коммуникаторе процесса-получателя, а процесс-получатель должен вызвать одну из процедур приема с указанием того же коммуникатора.

Все процедуры данной группы, в свою очередь, делятся на два класса: процедуры с блокировкой (с синхронизацией) и процедуры без блокировки (асинхронные). Процедуры обмена с блокировкой приостанавливают работу процесса до выполнения некоторого условия, а возврат из асинхронных процедур происходит немедленно после инициализации соответствующей коммуникационной операции.

### 1.1.4. Передача /прием сообщеий с блокировкой

- **MPI_SEND(BUF, COUNT, DATATYPE, DEST, MSGTAG, COMM, IERR)**

  **<type> BUF(*)**

  **INTEGER COUNT, DATATYPE, DEST, MSGTAG, COMM, IERR**

Блокирующая посылка массива **BUF** с идентификатором **MSGTAG**, состоящего из **COUNT** элементов типа **DATATYPE**, процессу с номером **DEST** в коммуникаторе **COMM**. Все элементы посылаемого сообщения должны быть расположены подряд в буфере **BUF**. Операция начинается независимо от того, быа ли инициализирована соответствующая процедура приема. Значение **COUNT** может быть нулем. Процессу разрешается передавать сообщение самому себе, однако это может привести к возникновению тупиковой ситуации.

Блокировка гарантирует корректность повторного использоваия всех параметров после возврата из процедуры. Это означает, что после возврата из **MPI_SEND** можно использоват любые присутствующие в вызове данной процедуре переменные без опасения испортить передаваемое сообщение.

- **MPI_RECV(BUF, COUNT, DATATYPE, DEST, MSGTAG, COMM, STATUS, IERR)**

**<type> BUF(*)**

**INTEGER COUNT, DATATYPE, DEST, MSGTAG, COMM, IERR, STATUS(MPI_STATUS_SIZE)**

Блокирующий прием в буфер **BUF** не более **COUNT** элементов сообщения типа **DATATYPE** с идентификатором **MSGTAG** от процесса с номером **SOURCE** в коммуникаторе **COMM** с заполнением массива атрибутов приходящего сообщения **STATUS**. Если число принятых элементов меньше значения **COUNT**, то гарантируется что в буфере **BUF** изменятся только элементы, соответствующие элементов принятого сообщения. Если количество элементов в принимаемом сообщении больше значения **COUNT**, то возникает ошибка переполнения.

### 1.1.5. Коллективные взаимодействия процессов

В операциях коллективного взаимодействия процессов участвуют все процессы коммуникатора. Соответствующая процедура может быть вызвана каждым процессом со своим набором параметров. Возврат из процедуры коллективного взаимодействия может произойти в тот момент, когда участие процесса в данной операции уже закончено. Как и для блокирующих процедур, возврат означает то, что разрешен свободный доступ к буферу приема или посылки.

В коллективных операциях не используются идентификаторы сообщений (теги). Таким образом, коллективные операции строго упорядочены согласно их появлению в тексте программы.

- **MPI_BARRIER(COMM, IERR)**

  **INTEGER COMM, IERR**

Процедура используется для барьерной синхронизации процессов. Работа процессов блокируется до тех пор, пока все оставшиеся процессы коммуникатора **COMM** не выполнят эту процедуру. Только после того, как последний процесс коммуникатора выполнит данную процедуру, все процессы будут разблокированы и продолжат выполнение дальше. Данная процедура является коллективной. Все процессы должны вызвать **MPI_BARRIER**, хотя реально исполненные вызовы различными процессами коммуникатора могут быть расположены в разных местах программы.

- **MPI_BCAST(BUF, COUNT, DATATYPE, ROOT, COMM, IERR)**

  **INTEGER COUNT, DATATYPE, ROOT, COMM, IERR**

Рассылка **COUNT** элементов данных типа **DATATYPE** из массива **BUF** от процесса **ROOT** всем процессам данного коммуникатора **COMM**, включая сам рассылающий процесс. При возврате из процедуры содержимое буфера **BUF**

процесса **ROOT** будет скопировано в локальный буфер каждого процесса коммуникатора **COMM**. Значения параметров **COUNT**, **DATATYPE**, **ROOT** и **COMM** должны быть одинаковыми у всех процессов.

# Практическая часть

## 2.1. Параметры системы

Сборка и запуск программы велись в коммандной оболочке Bash при помощи утилит mpifort и mpirun из пакета mpich. [1] В качестве платформ для тестирования были использованы машины со следующими параметрами:

- операционная система Gentoo GNU/Linux. Процессор Intel(R) Core(TM) i3-3110M 2.40GHz, ОЗУ 6,00 Гб.
- операционная система Debian GNU/Linux. Процессор Intel(R) Core(TM) i7-3770 3.40GHz, ОЗУ 8,00 Гб.

## 2.2. Параметры запуска и блок-схема программы

Рассматриваемый программный код решает задачу поиска глобального минимума энергии как функции многих переменных для заданного количества однотипных спиновых систем. Начальные параметры спиновых систем задаются при помощи трёх конфигурационных файлов - Basic, Exchange и Frozen. Начальные направления спинов задаются хаотически для каждой спиновой системе в программе.

Входные файлы содержат значения следующих входных параметров:

- $N\_Spins$ - число спинов в системе;
- $H\_mag$ - величина магнитного поля;
- $N\_Steps$ - число конфигураций, для которых будет посчитан глобальный минимум энергии;
- $num$ - номер спина и соответствующий ему модуль обменного коэффициента S_Value;
- $N\_Couples$ - число взаимодействующих спиновых пар;
- $Exchange$ - коэффициент обменного взаимодействия для каждой пары;
- $N\_Frozen$ - число замороженных спинов.

На выходе программы для каждого уникального состояния генерируются файлы, содержащие следующую информацию:

- Энергия полученного состояния;

[1] MPICH is a high performance and widely portable implementation of the Message Passing Interface (MPI) standard.

- номера спинов;
- продольный и поперечные углы, соответсвующие каждому спину;
- магнитный момент каждого спина;
- суммарный магнитный момент системы.

Блок-схема программы представлена на рисунке 2.1. Поскольку рассматриваемая программа производит расчёт $N\_Steps$ конфигураций системы, то среди полученных конфигураций встречаются такие, минимальные энергии которых одинаковы. Записывать в выходные файлы повторяющиеся конфигурации не имеет смысла, поэтому в программе реализован анализ состояний при помощи соответствующей подпрограммы $State\_Analysis$. Блок-схема данной подпрограммы представлена на рисунке 2.2.

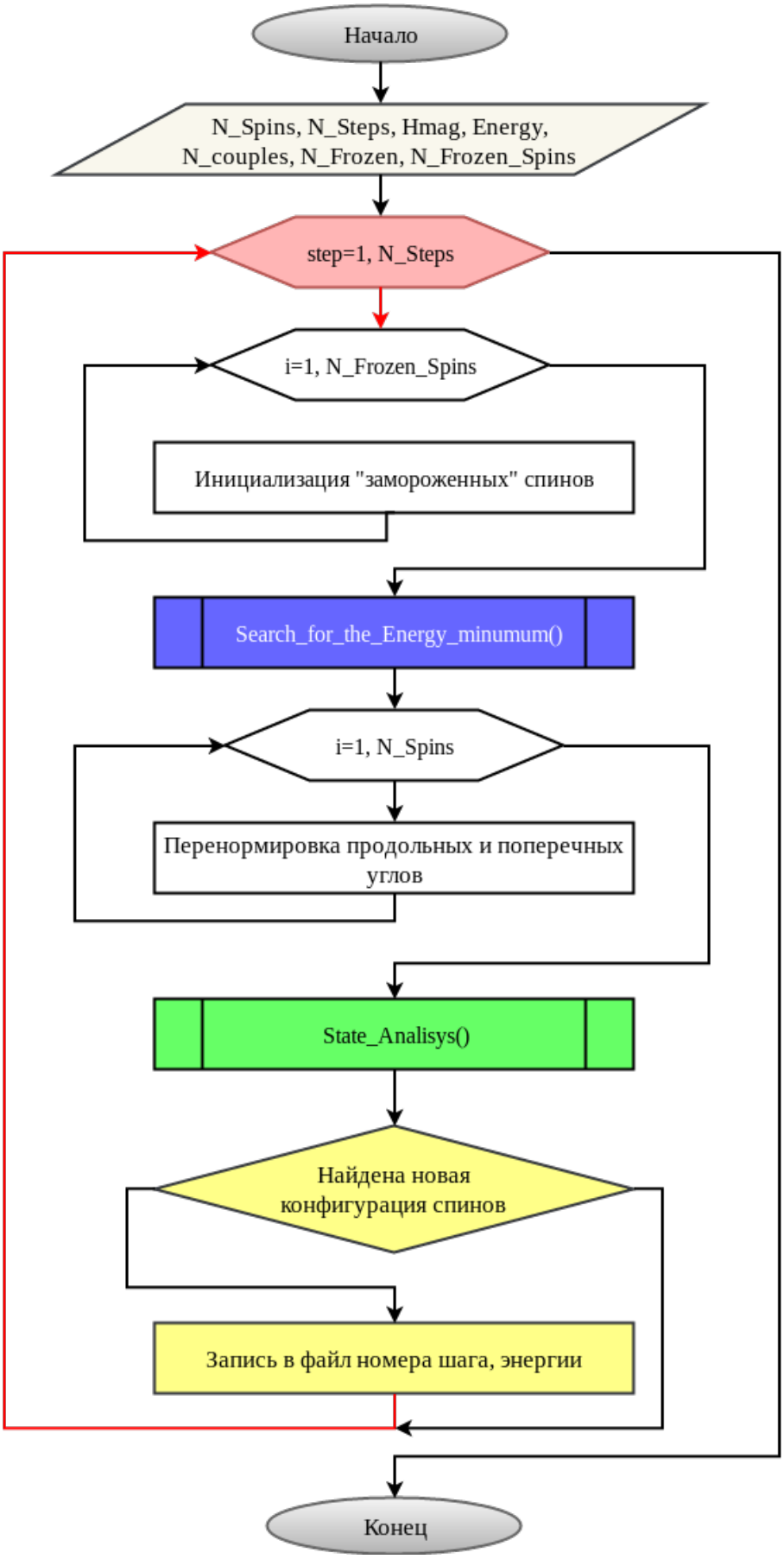


**Рис. 2.1.** Блок-схема программы.

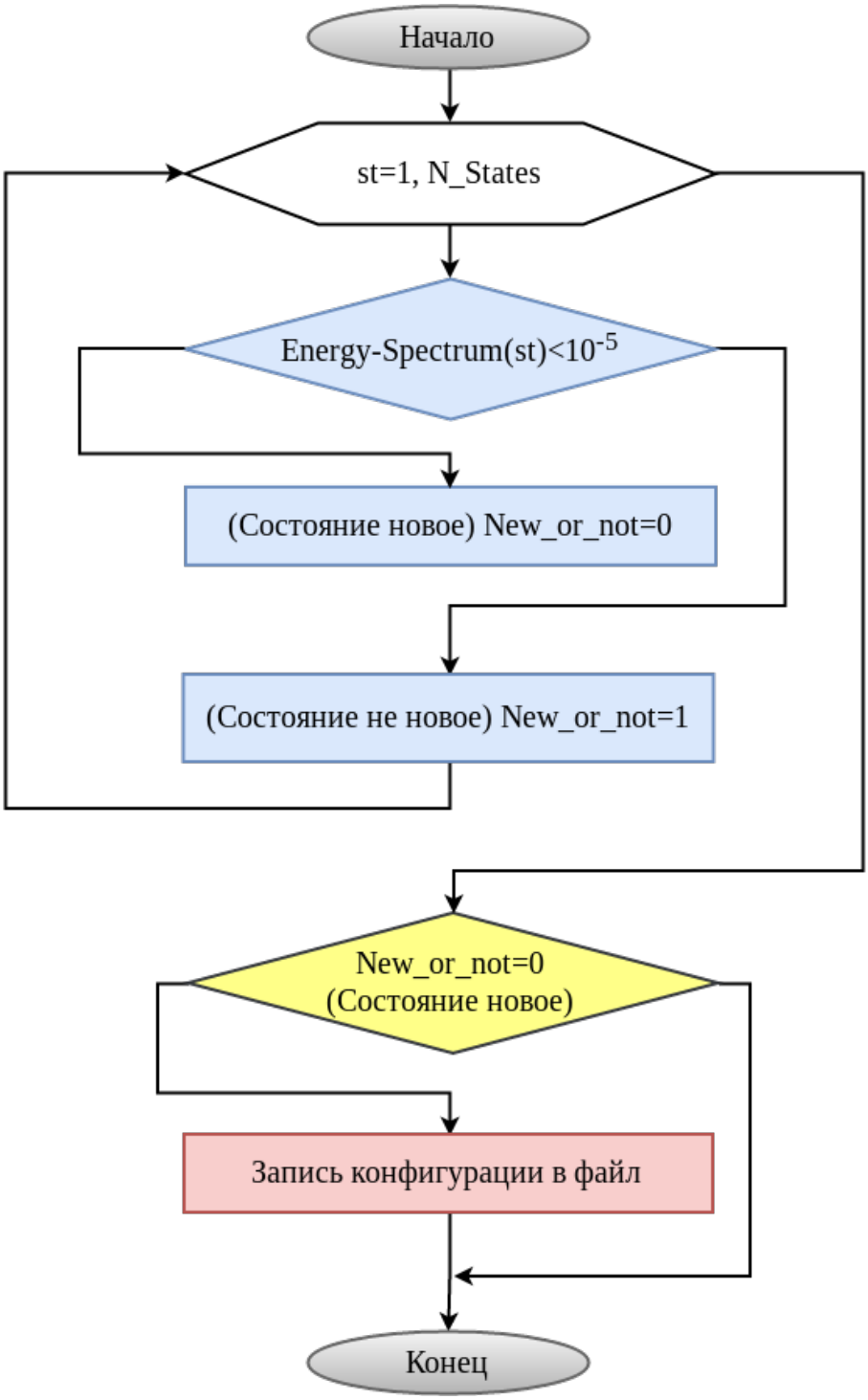


**Рис. 2.2.** Блок-схема подпрограммы State_Analysis.

## 2.3. Разработка паралелльной версии программы. Технология MPI

Расчёт параметров основного состояния спиновой модели(энергии, продольного и поперечного углов спинов), реализованный в рассматриваемой программе, происходит в цикле, и параметры, необходимые для расчёта основного состояния, не пересекаются. Поэтому разумно блоки программ, реализующих расчёт данных величин, выполнять на разных процессорах.

MPI позволяет решить задачу поиска глобального минимума энергии как функции многих переменных распределить на $N$ потоков, выполняющихся параллельно. Для этого основной цикл программы, в которой $N\_Steps$ раз вычисляется глобальный минимум энергии системы, начальное состояние которой задаётся произвольно, можно разбить на $N$ частей так, чтобы в каждом потоке считались $\frac{M}{N}$ конфигураций, соответствующих глобальному минимуму энергии системы.

Для этого необходимо разбить параметр $N\_Steps$ на $N$ частей. Тогда конечной точкой $step\_end$ для каждого потока с идентификатором $myid$ будет $step\_end = (myid + 1) \cdot h$, где $h = \frac{N\_Steps}{N}$, а стартовой точкой $step\_begin$ для каждого потока будет число $step\_begin = step\_end - h + 1$

Блок-схема параллельной версии программы представлена на рисунке 2.3.

Поскольку не все $N\_Steps$ конфигураций, соответствующих минимуму энергии системы, являются оригинальными, и многие полученные наборы состояний спинов повторяются, то в программе происходит отбор состояний, энергии которых отличаются, и лишь оригинальные состояния записываются в спектр. Этот блок программы также можно выполнять параллельно. Для этого назначим поток, идентификатор которого равен 0, главным потоком ($master$). После части кода, в которой выполняется поиск глобального минимума конфигурации, запускается процедура **MPI_BARRIER**. Ее задача - блокировка выполнения дальнейших задач до момента вычисления минимума энергии на каждом из потоков. В ходе отбора и записи этих состояний с остальных потоков на master значение минимума энергии будет последовательно передаваться процедурой **MPI_SEND**. Главный поток, в свою очередь, процедурой **MPI_RECV** будет принимать в переменную Energy_current значение полученной с других потоков энергии, иначе в эту переменную записывается значение энергии, полученное на потоке с номером $master$.

Чтобы потокам, идентификатор которых не соответствует потоку $master$, принять решение о записи в спектр нового состояния, им необходимо инициировать получение значения переменной $New\_or\_not$. Для этого каждый ненулевой поток после отправления на $master$ значения энергии инициируют получение значения переменной $New\_or\_not$ от потока $master$ процедурой **MPI_RECV**. Поток с идентификатором master при этом должен отправить значение переменной $New\_or\_not$ процедурой **MPI_SEND**.

Таким образом происходит обмен сообщениями между потоками, которые на момент вызова **MPI_BARRIER** хранят конфигурацию спинов, соответствующую минимуму энергии, а также само значение минимума энергии системы. Выполнение данной программы в несколько потоков уменьшает время выполнения программы за счёт того, что вычисление энергий и конфигураций происходит в $N$ потоков, а решение о записи конфигураций и энергий принимается только на одном потоке *master* за счёт отправления ему лишь значения энергии с других потоков. Запись файлы спектра состояний и конфигураций, в свою очередь, снова происходит в $N$ потоков. Блок-схема параллельной версии подпрограммы *State_Analysis* представлена на рисунках 2.4 и 2.5. Программный код параллельной версии программы доступен по ссылке TEX User Group.

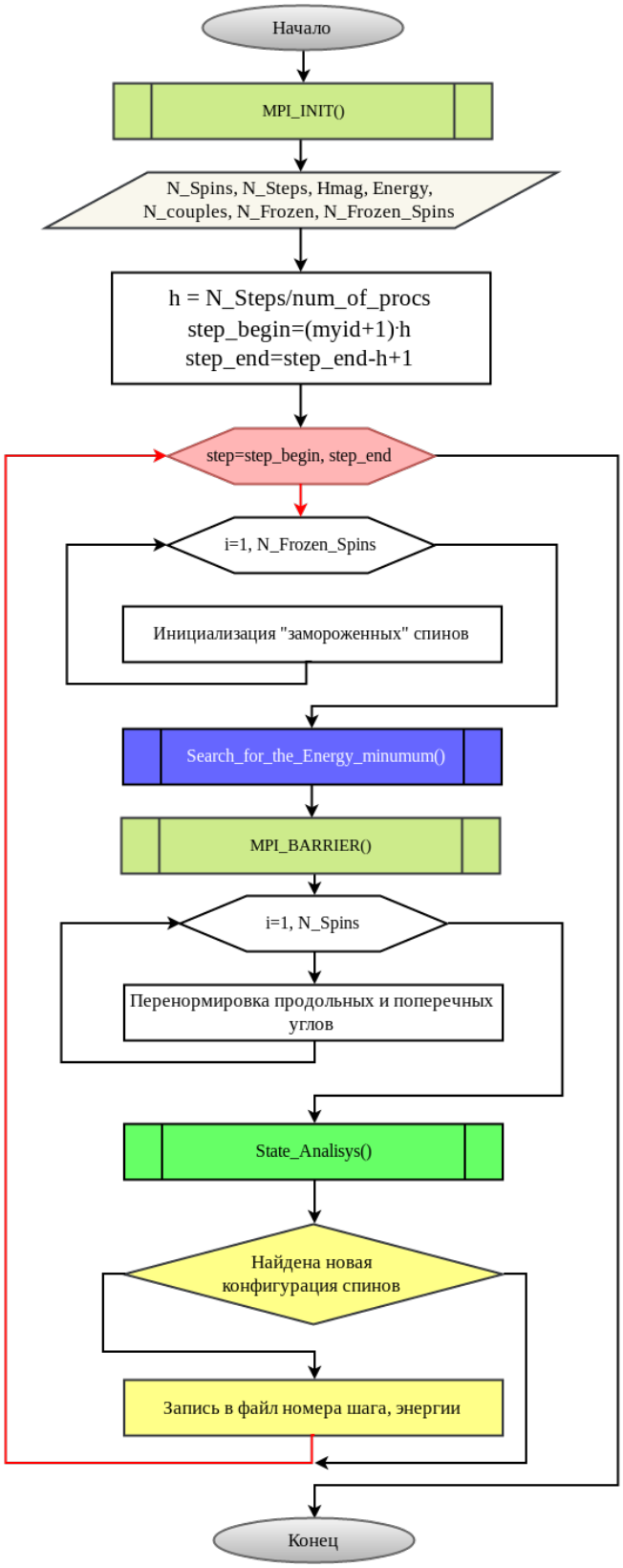


**Рис. 2.3.** Блок-схема параллельной версии программы .

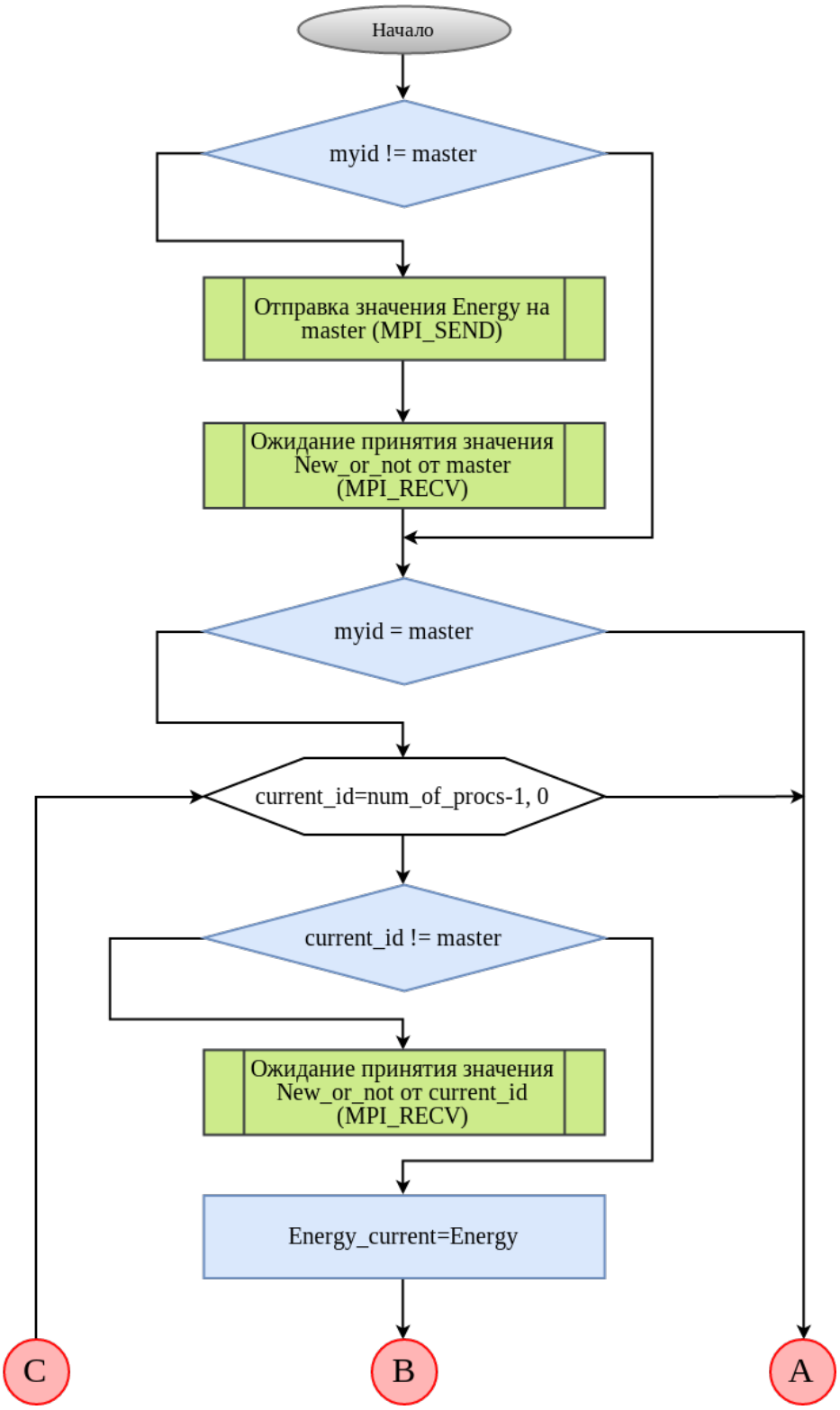


**Рис. 2.4.** Блок-схема параллельной версии подпрограммы State_Analysis (начало).

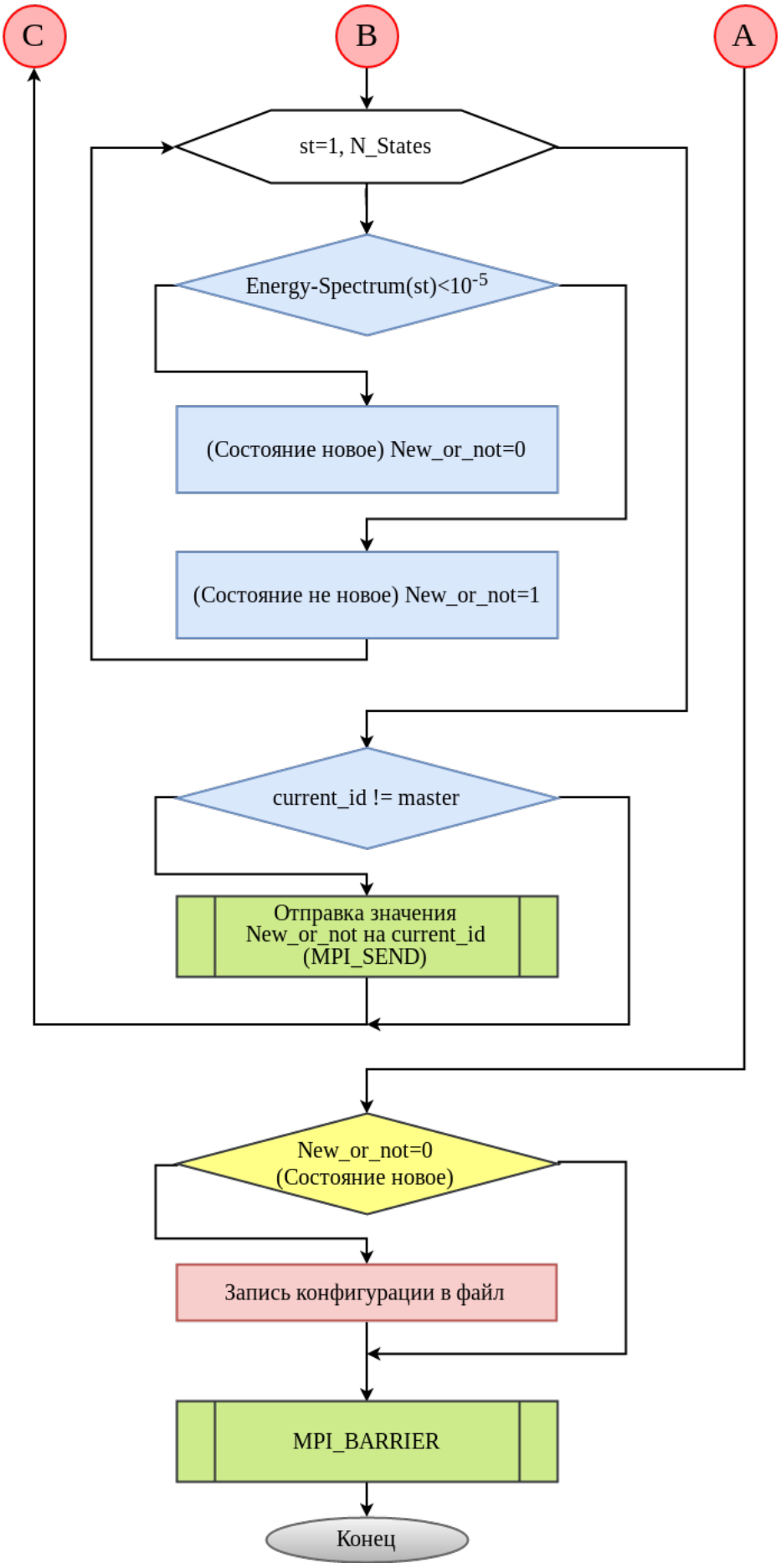


**Рис. 2.5.** Блок-схема параллельной версии подпрограммы State_Analysis (конец).

# Результаты

Разработанная параллельная версия программы запускалась при параметрах, представленных в таблице 3.1, на двух машинах с одинаковым программным обеспечением, но на различных аппаратных платформах.

Для начала сравним среднее время выполнения программы в зависимости от числа конфигураций $N\_Steps$, обсчитываемых программой, для Intel(R) Core(TM) i3-3110M 2.40GHz.

**Таблица 3.1.** Входные данные, используемые для сравнения времени выполнения программы.

| Параметр | Значение |
|---|---|
| $N\_Spins$ | 20 |
| $N\_Couples$ | 27 |
| $H\_mag$ | 0.1 |
| $N\_Frozen$ | 1 |

График зависимости среднего времени выполнения программы для различных значений числа обсчитываемых конфигураций представлен на рисунке 3.1. Усреднение проводилось по пяти измерениям. Линия синего цвета соответствует среднему времени выполнения программы в четыре потока, линия красного цвета - среднему времени выполнения программы в два потока, и линия зеленого цвета соответствует времени выполнения программы в один поток.

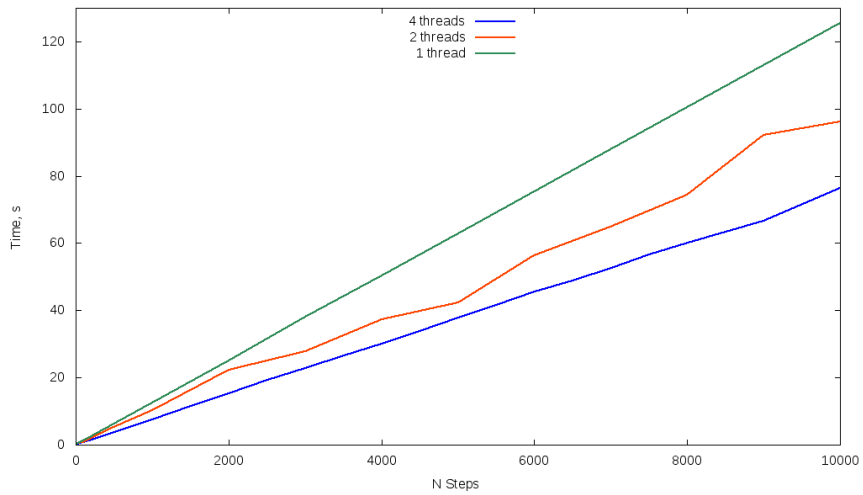


**Рис. 3.1.** График зависимости среднего времени выполнения программы для различных значений $N\_Steps$ на платформе Intel(R) Core(TM) i3-3110M 2.40GHz. Синим цветом обозначено время выполнения программы на 4-х потоках; красным - на 2-х потоках; зеленым - на 1-м потоке.

Полученный график зависимости 3.1 демонстрирует, что время выполнения параллельной версии программы в 4 потока в 1,64 раза меньше, нежели время выполнение однопоточной версии программы; программа, выполняемая в 2 потока, показала выигрыш по времени в 1,31 раз относительно однопоточной версии программы, что, однозначно, доказывает разумность выполнение такого типа задач в несколько потоков.

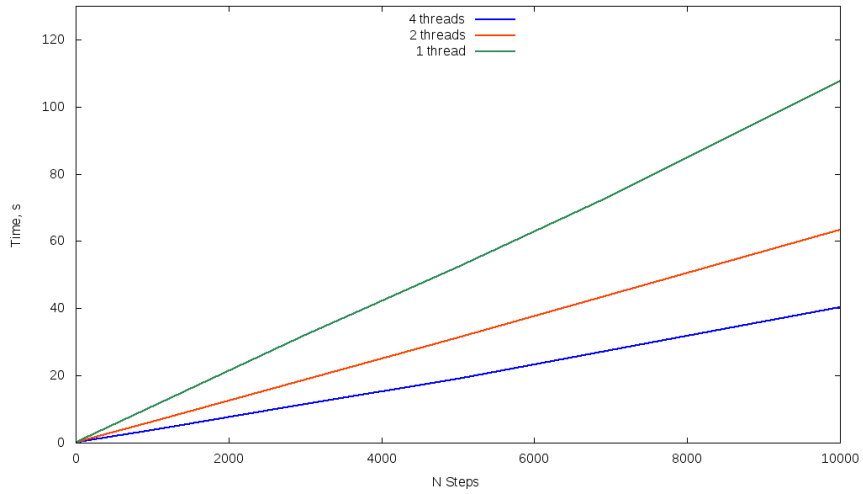


**Рис. 3.2.** График зависимости среднего времени выполнения программы для различных значений $N\_Steps$ на платформе Intel(R) Core(TM) i7-3770 3.40GHz. Синим цветом обозначено время выполнения программы на 4-х потоках; красным - на 2-х потоках; зеленым - на 1-м потоке.

Рассмотрим теперь время выполнения программы для второй машины с процессором Intel(R) Core(TM) i7-3770 3.40GHz. График зависимости времени выполнения программы от значения $N\_Steps$ представлен на рисунке 3.2. Как и на графике 3.1, время выполнения программы в большее, чем один, число потоков приводит к большему выигрышу по времени относительно однопоточной программы. В данном случае время сократилось в 2,67 раз. Как и ожидалось, вычисления, производимые на более мощном процессоре, привели к уменьшению времени, затрачиваемого на выполнение программы в 4 потока. Здесь относительно менее мощного процессора выигрыш по времени составил 30%.

Таким образом, можно сделать однозначный вывод, что выполнение задач, подобных рассматриваемой, где имеет место большое количество независимых однотипных вычислений, разумно в несколько потоков. В данном случае имеет место прирост производительности, в среднем, в 2 раза. Также, для ускорения вычислений, для выполнения программ подобного рода разумным выбором будет более мощная аппаратная платформа.

# Заключение

В ходе данной работы была разработана параллельная версия программы, которая осуществляет численное решение классической спиновой модели методом глобальной оптимизации при помощи интерфейса MPI. Были оценены преимущества во времени выполнения для данной программы, которая тестировалась запусками в два и четыре потока, относительно однопоточной версии программы; также было проведено сравнение полученных результатов на двух различных машинах с одинаковой программной платформой, аппаратные платформы которых различаются по мощности.

В ходе тестирования был получен прирост производительности по времени многопоточной версии программы относительно однопоточной в 1,6 раз для более слабой аппаратной платформы и в 2,67 раз для более мощной аппаратной платформы.

Исходя из результатов, был сделан вывод, что математические расчёты физических систем разумно выполнять в несколько потоков. Для разработки многопоточных программ MPI подтвердил свою эффективность. Более того, для большего прироста производительности эффективно использовать, одновременно с многопоточностью, более мощную аппаратную платформу.

# Список использованных источников

[1] Вонсовский С. В., Магнетизм. Магнитные свойства диа-, пара-, ферро-, антиферро-, и ферримагнетиков. - М.: Изд-во "Наука" , 1971. - 1032 с.

[2] Гэри М., Джонсон Д., Вычислительные машины и труднорешаемые задачи. - М.: Мир, 1982. - 416 с.

[3] Антонов А. С., Параллельное программирование с использованием технологии MPI: Учебное пособие - М.: Изд-во МГУ, 2004. - 71 с

[4] Корнеев В. Д., Параллельное программирование в MPI: Учебное пособие - М.: Изд-во ИВМиМГ, 2002. - 215 с.

[5] Богачёв К. Ю., Основы параллельного программирования: Учебное пособие - М.: Изд-во БИНОМ, 2015. - 345 с.