МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий

Кафедра параллельных вычислений

Направление подготовки: 09.04.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Магистерская программа: высокопроизводительные вычислительные системы

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Разработка и реализация алгоритмов динамической балансировки нагрузки на мультикомпьютер для моделирования процессов типа «взрыв» методом частиц-в-ячейках

Сумбатянца Ильи Ильича

Тема диссертации утверждена распоряжением по НГУ №104 от «20» марта 2014г.

«К защите допущена»	**Научный руководитель**
Заведующий кафедрой,	зав. кафедрой, ИВМиМГ СО РАН
д.т.н., проф.	д.т.н., проф.
Малышкин В. Э./……..	Малышкин В. Э./……...
(фамилия , И., О.) / (подпись, МП)	(фамилия , И., О.) / (подпись, МП)
«……»……20…г.	«……»……20…г.

Дата защиты: «……»……20…г.

Новосибирск, 2015г.

Содержание

ВВЕДЕНИЕ

При параллельной реализации больших численных методов на системах с распределенной памятью (мультикомпьютерах) часто возникает проблема обеспечения равномерного распределения вычислительной нагрузки на узлы мультикомпьютера. Эта проблема может быть обусловлена низкой производительностью вычислений или невозможностью произвести вычисления из-за недостаточного количества ресурсов на узле мультикомпьютера. Для обеспечения равномерного распределения вычислительной нагрузки часто используют алгоритмы динамической балансировки вычислительной нагрузки.

Один из широко используемых характерных примеров больших численных методов – метод частиц-в-ячейках [7]. В этом методе есть пространство моделирования, в котором под действием поля (полей) движутся частицы. Поля дискретизируются с помощью разбиения пространства моделирования регулярной сеткой на ячейки. На практике при параллельной реализации такой задачи на мультикомпьютерах используют пространственную декомпозицию, которая приводит к разбиению пространства моделирования на фрагменты, содержащие части сеточных значений с значениями частиц, принадлежащих фрагменту; и последующее отображение фрагментов на узлы мультикомпьютера. Так как частицы перемещаются по сетке, то в определенные моменты может складываться ситуация, когда одни узлы мультикомпьютера будут содержать большее количество нагрузки, чем другие. Этот факт приводит к необходимости в динамической балансировке нагрузки для устранения дисбаланса нагрузки. Кроме этого могут возникать ситуации, когда фрагмент будет содержать количество частиц, превосходящее количество доступных для исполнения ресурсов на одном узле. Характерный пример такой ситуации – моделирование «взрыва». В определенный момент модельного времени большинство частиц будет содержаться в одном фрагменте или, что еще хуже, в одной ячейке, что приведет к отсутствию возможности продолжения исполнения даже после проведения балансировки нагрузки, потому что ни на одном узле не будет достаточного количества ресурсов для вычисления такого фрагмента. Для решения этой проблемы, имеет смысл реализовать механизм создания распределенной реализации фрагмента сетки: производится разбиение фрагмента и отображение получившихся частей на несколько узлов так, что появляется возможность независимо произвести частичную обработку виртуального фрагмента на нескольких узлах, после чего собрать результат вычислений на одном узле.

Актуальной темой является динамическая балансировка нагрузки для моделирования процессов типа «взрыв» на мультикомпьютерах, в процессе исполнения которых возникают случаи, требующие дополнительных средств для продолжения моделирования. Как было описано выше, к таким случаям относятся ситуации, в которых нет возможности продолжить исполнение из-за недостатка ресурсов на любом из узлов мультикомпьютера. Таким образом, в силу специфики проблемы нужны локальные и распределенные алгоритмы динамической балансировки нагрузки для обеспечения достаточного уровня масштабируемости.

0.1. Обзор

В настоящее время существует несколько основных классов алгоритмов динамической балансировки нагрузки, которые используются при реализации больших численных методов. В обзоре будут рассмотрены и исследованы основные идеи этих классов алгоритмов.

0.1.1. Предметно-ориентированные методы

Рассмотрим алгоритм балансировки вычислительной нагрузки для двумерной задачи моделирования плазмы [5]. Пространство моделирования фрагментируется послойно и равномерно распределяется по узлам мультикомпьютера. Основная идея алгоритма – это представление двумерного пространства моделирования в виде одномерного с последующим рефрагментированием в случаях, когда частицы распределены неравномерно по узлам мультикомпьютера.

Рассмотрим пример распределения частиц в определенный момент модельного времени. При заданном контуре плотности (рис. 1) распределение плотности по оси y имеет следующий вид (рис. 2).

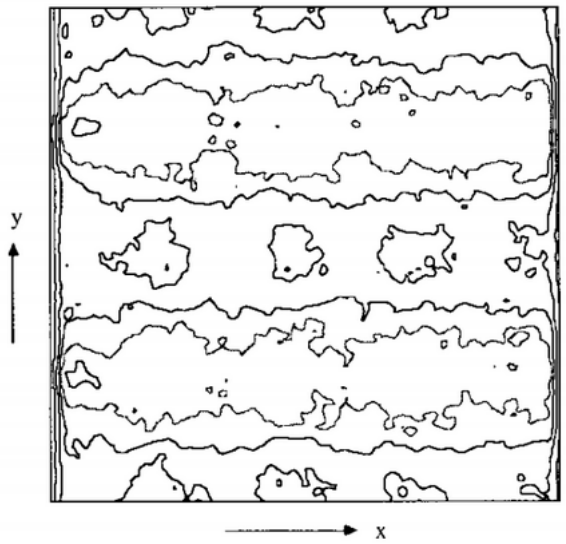

Рисунок 1: Контуры плотности

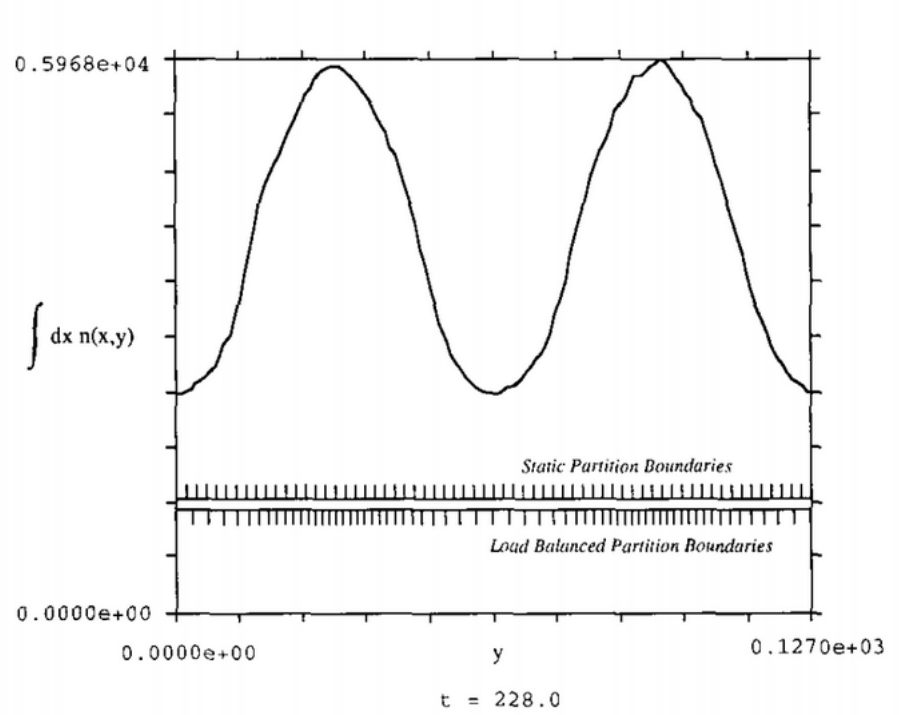

Рисунок 2: Распределение плотности

Как можно видеть на (рис. 2), при статической фрагментации в некоторый момент времени возникнет дисбаланс вычислительной нагрузки, динамическая фрагментация же позволяет избежать дисбаланса нагрузки.

Основная задача рефрагментирования – решить следующее уравнение:

$$p\frac{N_p}{N_{proc}} = \int_0^{y_l} dyn(y)$$

$$\begin{cases} y_l(0) = 0 \\ y_l(p) = y_r(p+1) \\ y_l(N_{proc}) = L_y \end{cases}$$

N_p – общее количество частиц, N_{proc} – количество вычислителей, p – логический номер вычислителя, y_l – левая граница фрагмента, $n(y)$ – функция вычисления плотности.

Алгоритм балансировки следующий:

а) Частичное значение плотности фрагмента вычисляется локально и распространяется по процессорам

б) Каждый процессор считает плотность фрагмента

в) Каждый процессор вычисляет границы и передает правую границу правому соседу

Данный алгоритм решает поставленную задачу на кольце процессоров, но не подходит по следующим критериям:

– На отличных от кольца топологиях алгоритм не будет эффективным из-за отсутствия учета соседства фрагментов

– Алгоритм балансировки ограничивает возможности декомпозиции задачи

Аналогичный алгоритм балансировки нагрузки был представлен в статье, связанной с моделированием поведения плазмы методом частиц-в-ячейках [10].

0.1.2. Рекурсивная бисекция

Такие алгоритмы основаны на рекурсивном фрагментировании пространства моделирования для уменьшения коммуникационных затрат и используются как для статической балансировки нагрузки [2, 18], так и для динамической [14, 16, 19].

Основная идея алгоритмов рекурсивной бисекции заключается в том, что пространство задачи рекурсивно разбивается на равные по нагрузке части (рис. 3). Получившиеся части, равные по вычислительной нагрузке, параллельно исполняются на узлах мультикомпьютера. При возникновении дисбаланса нагрузки в процессе исполнения производится уточнение границ частей области, в которой возник дисбаланс, с последующим рефрагментированием [19].

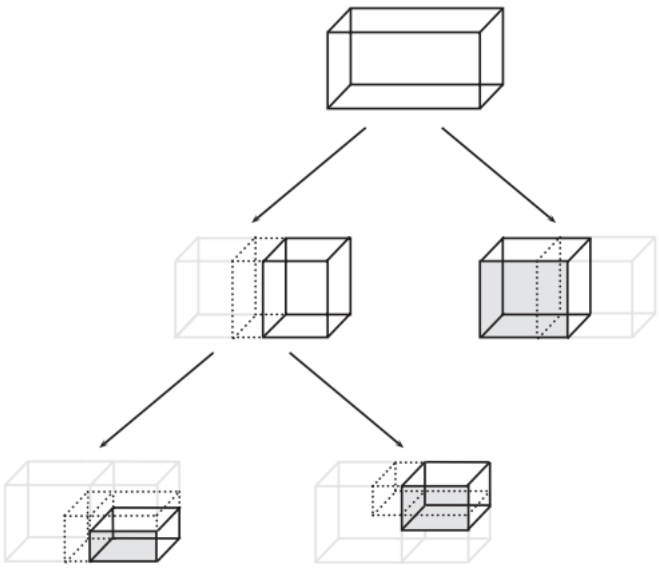

Рисунок 3: Пример результата работы ортогональной рекурсивной бисекции

Такие алгоритмы не подходят для задачи моделирования «взрыва» методом частиц-в-ячейках, так как при интенсивном изменении распределения нагрузки может потребоваться повторное использование рекурсивной бисекции для устранения дисбаланса. Эту операцию нельзя произвести распределенно, что означает большие затраты для проведения балансировки.

0.1.3. Диффузионные алгоритмы балансировки нагрузки

Основная идея диффузионных алгоритмов балансировки состоит в распределении нагрузки от более нагруженных узлов менее нагруженным в некоторой окрестности узла [3].

Существует множество методик расчета количества нагрузки, которую необходимо передать соседним узлам [4, 8].

Алгоритм балансировки в общем виде:

а) Каждый узел определяет количество избыточной нагрузки, в соответствии с нагрузкой узлов в окрестности

б) Производится передача нагрузки (фрагментов данных, частей пространства моделирования и т. д., в зависимости от типа задачи) в соответствии с методикой расчета количества нагрузки, необходимой для передачи

Диффузионные алгоритмы хорошо маштабируются и являются децентрализованными. Этот класс алгоритмов используется для задач численного моделирования [11, 12, 17].

Но с другой стороны, такие алгоритмы могут допускать возникновение глобального дисбаланса в рамках допустимого градиента нагрузки.

0.1.4. Выводы

В исследованных работах не было найдено решения проблемы. Многие алгоритмы балансировки используют для численного моделирования на мультикомпьютерах, но почти все из них узкоспециализированные, имеют ряд ограничений и показывают хорошие результаты только на определенных классах задач.

Лучше всего для основы решения проблемы подходит класс диффузионных алгоритмов в силу его универсальности и достаточного уровня масштабируемости.

0.2. Цели и задачи

Цель данной работы – разработка и реализация распределенных локальных алгоритмов динамической балансировки нагрузки для случая моделирования взрыва методом частиц-в-ячейках. Предполагается, что алгоритмы балансировки будут использоваться для задач большого размера, моделируемых на мультикомпьютерах. Таким образом, к алгоритмам предъявляются следующие требования:

а) Обеспечение равномерной нагрузки на узлы

б) Возможность производить балансировку в процессе исполнения

в) Принятие решения о перераспределении нагрузки только в рамках локальной окрестности узла

г) Децентрализованность

Эффективность таких алгоритмов (в смысле способности обеспечивать баланс нагрузки процессоров и в смысле накладных расходов на балансировку) сложно оценить теоретически. В том числе вследствие того, что эффективность может существенно зависеть от конфигурации вычислителя и входных данных прикладной программы. Поэтому важную роль играет тестирование эффективностии алгоритмов балансировки нагрузки на процессоры.

Такое тестирование должно охватывать широкий спектр ситуаций, чтобы получить представление о работе алгоритма балансировки в целом. В частности, должны варьироваться такие параметры, как классы прикладных задач, входные данные задачи, размер задачи, количество вычислительных узлов, фоновая нагрузка на сеть, параметры самого алгоритма балансировки нагрузки (если имеются), и т.п.

Для автоматизации проведения таких тестов целесообразно создание отладочного тестового стенда – программы, принимающей на вход реализацию некоторого алгоритма балансировки нагрузки и выполняющей серию тестов с различными параметрами. В процессе тестирования производится реальное (не имитационное) исполнение задачи на мультикомпьютере. Результаты тестов позволяют судить о том, каковы качественные и количественные характеристики эффективности исследуемого алгоритма балансировки нагрузки на процессоры.

Для достижения цели были выделены следующие задачи:

а) Разработка и реализация отладочного тестового стенда
б) Разработка и реализация алгоритмов балансировки нагрузки
в) Реализация задачи моделирования взрыва методом частиц-в-ячейках на отладочном тестовом стенде
г) Тестирование алгоритмов балансировки на стенде и анализ полученых результатов

0.3. Характеристика результатов

Разработан и реализован отладочный тестовый стенд, предназначенный для испытания различных характеристик алгоритмов балансировки нагрузки на процессоры. Реализован ряд синтетических тестов, проверяющих правильность работы тестового стенда. Разработаны и реализованы алгоритмы динамической балансировки нагрузки. Реализована задача моделирования взрыва методом частиц-в-ячейках. Приведены результаты тестирования алгоритмов балансировки на целевой задаче и их анализ.

Полученные результаты ориентированы на современные системы фрагментированного программирование (например, систему LuNA [9, 13]). При поддержке в таких системах интерфейсов, разаработанных в данной работе, и некоторых механихзмов, позволяющих реализовать фрагмент на нескольких узлах мультикомпьютера, можно переиспользовать балансировщики и ожидать таких же результатов балансировки, какие были получены при тестировании на отладочном тестовом стенде.

Научную новизну работы составляют:

– Разработка локальных распределенных алгоритмов балансировки нагрузки для случая моделирования «взрыва» методом частиц-в-ячейках
– Архитектура и анализ отладочного тестового стенда

Глава 1

Постановка задачи

1.1. Необходимые определения

1.1.1. Метод частиц-в-ячейках

Метод частиц-в-ячейках – это численный метод решения дифференциальных уравнений в частных производных, который используется для моделирования физических явлений [7]. Физическое пространство представляется пространством моделирования, которое дискретизируется разбиением сеткой на ячейки. С каждой ячейкой связаны частицы, которые находятся в ней в данный момент времени. Частицы под действием полей, заданных в ячейках, меняют свое местоположение в пространстве моделирования. Процесс моделирования – серия временных шагов, на которых вычисляются новые координаты частиц и значения полей в ячейках, после перемещения частиц. Причем шаг выбирается таким образом, что частицы не перемещаются дальше одной ячейки.

Моделирование поведения вещества под действием гравитационных сил

Рассмотрим целевую задачу, для которой необходима балансировка нагрузки – моделирование поведения вещества под действием гравитационных сил.

Физическое пространство представляется в виде разбитого регулярной сеткой на ячейки пространства моделирования, каждая ячейка может содержать частицы вещества в модельный момент времени. [1]

Ячейки определяются следующими параметрами:

- Плотность в центре ячейки
- Гравитационный потенциал в центре ячейки
- Гравитационные силы в центрах граней ячейки

Частицы определяются следующими парамеетрами:

- Масса
- Вектор скорости
- Координаты

Итерационный шаг метода частиц-в-ячейках при этом будет выглядеть следующим образом:

а) Сдвиг частиц под действием сил гравитации.

б) Вычисление распределения плотности

в) Вычисление распределения гравитационного потенциала

г) Вычисление гравитационных сил

Рассмотрим эти этапы более детально.

Сдвиг частиц

На этом этапе частицы сдвигаются под действием гравитационных сил, заданных на сетке. Новые скорости и координаты вычисляются по формулам 1.1 1.2 1.3

$$F_x = (1-s'_x)[(1-s_y)((1-s_z)F_{x_{i',k,l}} + s_z F_{x_{i',k,l+1}}) + s_y((1-s_z)F_{x_{i',k+1,l}} + s_z F_{x_{i',k+1,l+1}})]+ \\ s'_x[(1-s_y)((1-s_z)F_{x_{i'+1,k,l}} + s_z F_{x_{i'+1,k,l+1}}) + s_y((1-s_z)F_{x_{i'+1,k+1,l}} + s_z F_{x_{i'+1,k+1,l+1}})] \tag{1.1}$$

$$F_y = (1-s_x)[(1-s'_y)((1-s_z)F_{y_{i,k',l}} + s_z F_{y_{i,k',l+1}}) + s'_y((1-s_z)F_{y_{i,k'+1,l}} + s_z F_{y_{i,k'+1,l+1}})]+ \\ s_x[(1-s'_y)((1-s_z)F_{y_{i+1,k',l}} + s_z F_{y_{i+1,k',l+1}}) + s'_y((1-s_z)F_{y_{i+1,k'+1,l}} + s_z F_{y_{i+1,k'+1,l+1}})] \tag{1.2}$$

$$F_z = (1-s_x)[(1-s_y)((1-s'_z)F_{z_{i,k,l'}} + s'_z F_{z_{i,k,l'+1}}) + s_y((1-s'_z)F_{z_{i,k+1,l'}} + s'_z F_{z_{i,k+1,l'+1}})]+ \\ s_x[(1-s_y)((1-s'_z)F_{z_{i+1,k,l'}} + s'_z F_{z_{i+1,k,l'+1}}) + s_y((1-s'_z)F_{z_{i+1,k+1,l'}} + s'_z F_{z_{i+1,k+1,l'+1}})] \tag{1.3}$$

Вычисление распределения плотности

На этом этапе вычисляется распределение плотности частиц. Значения плотности задаются в центрах ячеек. Вклад каждой частицы определяется ядром метода частиц. В данном случае использовано ядро PIC: оно определяет вклад частицы в ближайшие узлы сетки обратной линейной интерполяцией по каждой координате:

$$\tilde{R}(x,y,z) = \begin{cases} \dfrac{1}{h_x h_y h_z}(1-\dfrac{x}{h_x})(1-\dfrac{y}{h_y})(1-\dfrac{z}{h_z}) & \text{если } |x| \le h_x, |y| \le h_y, |z| \le h_z \\ 0 & \text{иначе} \end{cases} \tag{1.4}$$

Вклады одной частицы в ближайшие узлы в таком случае определяются следующими соотношениями:

$$
\begin{aligned}
\rho_{i,k,l} &= (1-s_x)(1-s_y)(1-s_z)m \\
\rho_{i,k,l+1} &= (1-s_x)(1-s_y)s_z m \\
\rho_{i,k+1,l} &= (1-s_x)s_y(1-s_z)m \\
\rho_{i,k+1,l+1} &= (1-s_x)s_y s_z m \\
\rho_{i+1,k,l} &= s_x(1-s_y)(1-s_z)m \\
\rho_{i+1,k,l+1} &= s_x(1-s_y)s_z m \\
\rho_{i+1,k+1,l} &= s_x s_y(1-s_z)m \\
\rho_{i+1,k+1,l+1} &= s_x s_y s_z m
\end{aligned}
\tag{1.5}
$$

Вычисление распределения гравитационного потенциала

На данном подшаге решается уравнение Пуассона – по распределению плотности вещества ρ на сетке вычисляется распределение гравитационного потенциала Φ. Трехмерное уравнение Пуассона аппроксимируется семиточечной разностной схемой второго порядка:

$$
\frac{\Phi_{i+1,k,l} - 2\Phi_{i,k,l} + \Phi_{i-1,k,l}}{h_x^2} + \frac{\Phi_{i,k+1,l} - 2\Phi_{i,k,l} + \Phi_{i,k-1,l}}{h_y^2} + \frac{\Phi_{i,k,l+1} - 2\Phi_{i,k,l} + \Phi_{i,k,l-1}}{h_z^2} = 4\pi\rho_{i,k,l}
\tag{1.6}
$$

Вычисление гравитационных сил

На этом подшаге из распределения гравитационного потенциала Φ в центрах ячеек вычисляются значения гравитационных сил $F = (F_x, F_y, F_z)$ в центрах граней сетки.

Сила, действующая на частицу единичной массы, задается соотношением $F = -\nabla\Phi$. И аппроксимация этого соотношения с помощью центральных разностей выглядит следующим образом:

$$
F_{x_{i+\frac{1}{2},k,l}} = -\frac{\Phi_{i+1,k,l} - \Phi_{i,k,l}}{h}
$$

$$
F_{y_{i,k+\frac{1}{2},l}} = -\frac{\Phi_{i,k+1,l} - \Phi_{i,k,l}}{h}
$$

$$
F_{z_{i,k,l+\frac{1}{2}}} = -\frac{\Phi_{i,k,l+1} - \Phi_{i,k,l}}{h}
$$

Параллельная реализация метода частиц-в-ячейках

Существует несколько способов декомпозиции задач, моделируемых методом частиц-в-ячейках:

а) Декомпозиция множества частиц

б) Декомпозиция пространства моделирования

Декомпозиция множества частиц подходит для реализации задач на системах с общей памятью из-за небольших затрат на чтение и запись в память между процессорами, но она плохо масштабируется, поэтому не подходит для задач больших размеров и не подходит для моделирования задач на системах с распределенной памятью. Декомпозиция пространства моделирования, в свою очередь, хорошо масштабируется и может быть использована для реализации больших задач, моделируемых на системах с распределенной памятью [5, 15].

1.2. Постановка задачи

1.2.1. Целевой класс задач

Моделирование процессов типа «взрыв» методом частиц-в-ячейках – это задача, параллельная реализация которой требует динамической балансировки нагрузки. Это требование обусловлено следующим:

- Параллельная реализация задач большого размера, моделируемых на мультикомпьютерах (системах с разделяемой памятью) методом частиц-в-ячейках, требует пространственной декомпозиции из-за необходимости в масштабировании.
- Процессы типа «взрыв» приводят к одному из крайних случаев неравномерного распределения нагрузки на узлы мультикомпьютера, который может возникать при моделировании методом частиц-в-ячейках: в определенный момент процесса моделирования все частицы будут расположены в определенной области сетки (вплоть до одной ячейки), после чего начнется интенсивное распространение частиц по всей сетке.
- В случае скопления частиц в одной ячейке может возникнуть ситуация, когда на вычислительном узле, на который отображена ячейка, не будет хватать ресурсов для исполнения и хранения ячейки.

Требования к целевому классу задач, для которого необходимо разработать и реализовать алгоритмы балансировки:

а) Моделирование процесса типа «взрыв» методом частиц-в-ячейках
б) Параллельная реализация – декомпозиция пространства моделирования

1.2.2. Балансировка нагрузки

Исходя из особенностей целевого класса задач, параллельная реализация такой задачи – это разбиение пространства моделирования на непересекающиеся смежные области S_i. Каждая область S_i содержит в себе значения части сетки и значения частиц, которые расположены в этой части сетки. Построение иного пространственного разбиения в процессе моделирования будем называть *динамической балансировкой вычислительной нагрузки, или балансировкой нагрузки*. Построение нового разбиения может зависеть от результатов построения предыдущих разбиений.

Вычислительная сложность итерационного шага метода частиц-в-ячейках в основном зависит от количества частиц, так как самые затратные подшаги — это сдвиг частиц и вычисление распределения плотности. Таким образом, для целевого класса задач *вычислительной нагрузкой* будем называть количество частиц, которое расположено в области разбиения пространства моделирования.

Так как предполагается исполнение задач большого размера на мультикомпьютерах, балансировка нагрузки должна производиться в рамках некоторой локальной окрестности узла, на котором была обнаружена ситуация неравномерного распределения вычислительной нагрузки. То есть, новое пространственное разбиение будет уточняться только в некоторой локальной области и результат может зависеть только от локальной части результатов построения предыдущих разбиений.

Исходя из всего вышеизложенного алгоритмы динамической балансировки вычислительной нагрузки для целевого класса задач должны обладать следующими свойствами:

- Алгоритм балансировки должен иметь возможность производить балансировку нагрузки по данным предыдущего пространственного разбиения
- Алгоритм балансировки должен опираться на распределение нагрузки в рамках локальной окрестности (1-2-окрестности)
- Алгоритм балансировки должен быть децентрализованным

И алгоритмы балансировки могут иметь следующие вход и выход:

- Вход:
  а) Информация о локальной части разбиения пространства моделирования
  б) Информация о нагрузке всего пространства моделирования
  в) Параметры алгоритма балансировки
- Выход:
  а) Решение по построению нового разбиения

1.2.3. Отладочный тестовый стенд

Отладочный тестовый стенд предназначен для тестирования алгоритмов балансировки на целевом классе задач. Итак, опираясь на свойства целевого класса задач и алгоритмов балансировки можно сформулировать требования, которым должен удовлетворять стенд:

- Поддержка распределенных реализаций итерационных сеточных численных методов (далее просто задач)
- Поддержка распределенных локальных динамических алгоритмов балансировки нагрузки
- Возможность независимо заменять реализации алгоритма балансировки и задачи

И вход/выход стенда:

- Вход: реализации задачи и алгоритма балансировки

– Выход: данные об исполнении задачи, представляющие интерес для анализа и оценки качества алгоритмов балансировки

Глава 2

Теоретические результаты

2.1. Отладочный тестовый стенд

Отладочный тестовый стенд (ОТС) – это программа, принимающая на вход реализацию некоторого алгоритма балансировки нагрузки и выполняющая серию тестов с различными параметрами. В процессе тестирования производится реальное исполнение задачи на мультикомпьютере. Результаты тестов позволяют судить о том, каковы качественные и количественные характеристики эффективности исследуемого алгоритма балансировки нагрузки. Соответственно, возникает потребность в унификации интерфейса модуля балансировки нагрузки (балансировщика). Кроме того, поведение прикладных программ с точки зрения баланса нагрузки на процессоры может быть очень многообразным. Как следствие, целесообразно не вкладывать все возможные ситуации в ОТС, а предоставить возможность закладывать в него различные прикладные программы. Для этого в настоящей работе была предложена модель вычислений, в которой может быть представлен прикладной алгоритм из заданного класса.

Рассмотрим компоненты ОТС, а именно методику оценки алгоритмов, модели исполнения прикладной задачи, балансировщика.

2.1.1. Оценка алгоритмов балансировки нагрузки

Для оценки и анализа работы балансировщика ОТС собирает данные о процессе исполнения численного метода. Такие данные должны быть достаточно полными и полезными для анализа. Так как стенд нацелен на практическое использование для определенного класса задач, то был исследован ряд статей, в которых были представлены алгоритмы статической и динамической балансировки нагрузки для характерного примера из целевого класса задач: итерационных сеточных методов и моделирования физических процессов методом частиц-в-ячейках.

Так было определено, что для анализа и тестирования алгоритмов балансировки нагрузки авторы использовали следующие данные:

а) Общее время исполнения и время моделирования [5, 6, 11]
б) Развертка максимума и минимума количества частиц на всех вычислительных узлах по времени моделирования [5]
в) Максимальное количество частиц отображенных на один вычислительный узел [11]
г) Развертка распределения вычислительной нагрузки по времени моделирования [19]
д) Время исполнения итерации на каждом узле [19]

Помимо этого существенными представляются такие показатели как:

а) Общее количество балансировок
б) Развертка нагрузки на коммуникационную сеть по времени

2.1.2. Структура ОТС

Модель балансировщика

Балансировщик – это реализация локального распределенного алгоритма балансировки. Балансировщик должен устранять дисбаланс нагрузки и не допускать ситуаций, когда исполнение не может быть продолжено из-за отсутствия свободных ресурсов на узле. Балансировщик может опираться на информацию о нагрузке в локальной окресности и на входные параметры, изменяя которые, можно менять его поведение.

Для ОТС балансировщик — это модуль, который содержит реализацию алгоритма балансировки и некоторый предикат, который позволяет определить, необходимо ли производить балансировку. В определенные моменты времени ОТС проверяет необходимость в балансировке, и после положительного результата инициирует процесс балансировки в некоторой окрестности текущего вычислительного узла.

2.1.3. Модель вычислений прикладной программы

Прикладной алгоритм представляется в виде графа $< F, N, op, R >$, где F – множество вершин графа (далее – фрагментов), N – множество ребер на множестве F (отношение соседства), op – оператор, который применяется к каждому элементу F, R – оператор редукции, который применяется ко всем элементам F. Процесс вычислений происходит в дискретном времени $T = \{t_1 \dots t_n\}$, и в каждый момент времени t_i элемент $f \in F$ имеет состояние (значение) f_{t_i}. Каждое следующее состояние фрагмента $f_{t_{i+1}}$ зависит от его текущего состояния f_{t_i}, от текущих состояний соседних фрагментов $\{fn_{t_i} : (fn \in F) \wedge ((fn, f) \in N)\}$ и от редукционных данных времени t_i. Тогда в общем виде процесс вычислений выглядит следующим образом (рис. 2.1):

$$\forall f \in F(f_{t_i} = op(f_{t_{i-1}}, \{fn_{t_{i-1}} : (fn \in F) \wedge ((fn, f) \in N)\}, R(t_{i-1}))$$

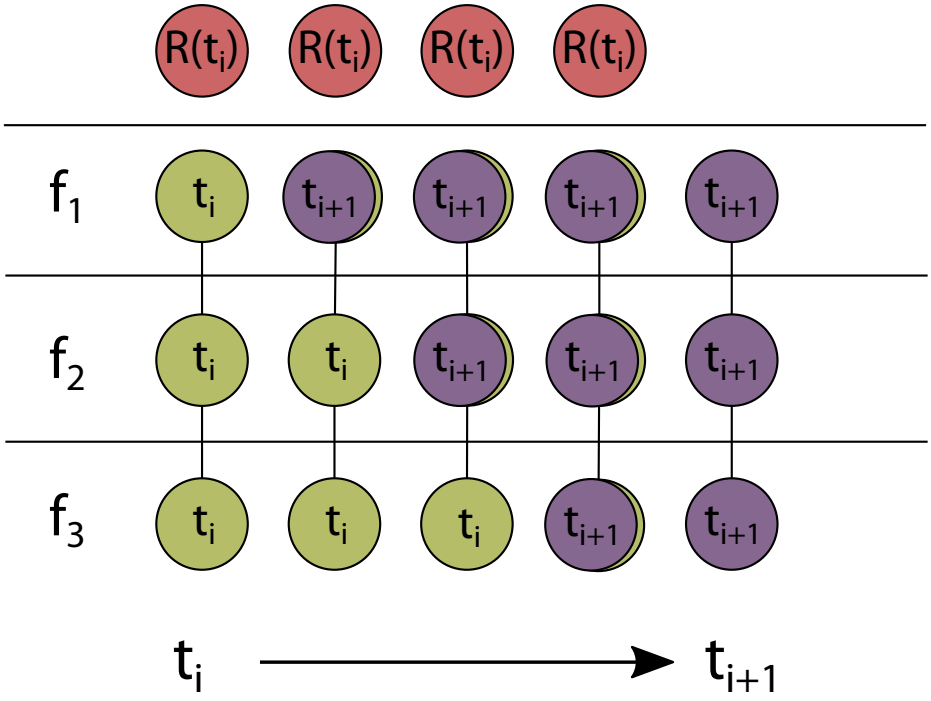

Рисунок 2.1: Графическое представление процесса исполнения

В такую модель укладываются итерационные численные методы, включая метод частиц-в-ячейках и многие клеточные автоматы. В случае метода частиц-в-ячейках фрагментом будет некоторая определенная часть сетки с ее значениями и значениями частиц, принадлежащих этой части сетки. Оператор скрывает в себе вычисление новых координат частиц и новых сеточных значений, а редукционный оператор применяется для определения таких величин, как суммарная энергия системы.

Реализация вычислений на мультикомпьютере

Тестовый стенд реализует описанный прикладной вычислительный процесс на мультикомпьютере. Каждый узел мультикомпьютера может обмениваться сообщениями с ограниченным множеством других узлов. Конкретная топология соединений задается стендом. Далее будем называть такое множество *локальной окрестностью узла*, а узлы из этого множества – *соседними узлами*.

В ходе исполнения ОТС может предоставить фрагменту данные соседних фрагментов, передать данные от одного фрагмента другому и произвести редукцию данных между всеми фрагментами.

Начальное отображение фрагментов на узлы мультикомпьютера можно производить несколькими способами:

а) Статически определять место создания фрагментов до начала исполнения
б) Создавать фрагменты на нескольких выделенных узлах после начала исполнения, для построения начального отображения с помощью балансировщика

По требованию балансировщика система может перемещать фрагменты на соседние узлы мультикомпьютера в рамках локальной окрестности и создавать распределенную реализацию фрагмента, если на узле нет достаточного количества ресурсов для его хранения или обработки.

2.1.4. Анализ

Виртуальные фрагменты

Опираясь на свойства целевого класса задач, а именно вероятность скопления частиц в небольшой области сетки, можно сделать вывод, что подобная ситуация может возникнуть в рамках модели вычислений задачи. При возникновении такой ситуации на задачах достаточно большого размера дальнейшее исполнение не сможет быть продолжено из-за недостаточного количества ресурсов на одном узле, так как в исходной модели фрагмент является атомарной сущностью, которая отображается на один вычислитель. Таким образом, необходим некоторый механизм, который бы позволял отображать один фрагмент на несколько вычислителей для «искуственного» разделения вычислительной нагрузки на узел. В дальнейшем будем называть фрагмент, отображенный на несколько вычислительных узлов, *виртуальным фрагментом*.

2.1.5. Характеристика

Такой отладочный тестовый стенд соответствует требованиям, описанным в постановке задачи:

- В модель иполнения задачи укладывается класс итерационных сеточных численных методов
- В модель балансировщика укладывается класс распределенных локальных динамических алгоритмов балансировки нагрузки
- Модели балансировщика и исполнения задачи не зависят друг от друга, а значит есть возможность независимо заменять их реализации без внесения изменений.
- Вход тестового стенда — реализации задачи и алгоритма балансировки
- На выходе тестовый стенд предоставляет данные об исполнении задачи, соответствующие данным, которые использовали авторы статей в процессе анализа алгоритмов балансировки нагрузки

2.2. Алгоритмы балансировки

Исходя из обзора и постановки задачи, для решения проблемы балансировки нагрузки для моделирования процессов типа «взрыв» методом частиц-в-ячейках подходит класс диф-

фузионных алгоритмов балансировки нагрузки, так как этот класс обладает следующими свойствами:

- Диффузионные алгоритмы децентрализованные
- Принятие решения о перераспределении нагрузки основывается на информации в локальной окрестности
- Диффузионные алгоритмы подходят для балансировки в динамике

В общем виде алгоритмы диффузионной балансировки выглядят следующим образом:

а) Если возник дисбаланс нагрузки в окрестности, начать балансировку:
   1) Определить количество избыточной нагрузки на узле
   2) Распределить изыбыточную нагрузку в локальной окрестности в соответствии со *стратегией распределения нагрузки*

В этом описании для простоты опущена логика обмена информацией о распределении нагрузки в окрестности.

Таким образом, для разработки диффузионных алгоритмов балансировки нагрузки необходимо:

- Разработать логику определения возникновения дисбаланса нагрузки в окрестности
- Разработать логику определения количества избыточной нагрузки
- Разработать стратегию распределения нагрузки в локальной окрестности

Кроме этого, исходя из необходимости в механизмах отображения фрагмента на несколько узлов, алгоритмы балансировки должны содержать логику инициации такого отображения.

2.2.1. Порог дисбаланса

Возникновение дисбаланса в локальной окрестности – это ситуация, в которой один из узлов окрестности содержит количество вычислительной нагрузки, превосходящее некоторый *порог дисбаланса*.

Порог дисбаланса можно определить несколькими способами:

а) Допустимая абсолютная величина разницы количества нагрузки между каждой парой узлов

б) Допустимое отклонение от теоретической средней нагрузки, приходящейся на один узел

Первое определение порога дисбаланса – универсальное, потому как может использоваться для задач с изменяющимся количеством общей вычислительной нагрузки в процессе исполнения. Второе определение более узкоспециализированное, потому как требует знания об изменении общего количества нагрузки. Для целевой задачи оба этих определения подходят, потому как общее количество вычислительной нагрузки константно на всем времени исполнения – частицы не вылетают за пределы пространства моделирования.

Для алгоритмов балансировки в данной работе в качестве порога дисбаланса была выбрана допустимая абсолютная величина разницы нагрузки между каждой парой узлов, потому как это типичный случай для диффузионных алгоритмов: [17]

Алгоритмы разрабатывались на основе анализа существующих алгоритмов и в значительной степения являются их адаптацией под конкретную ситуацию.

2.2.2. Стратегии распределения нагрузки

Логика определения количества избыточной нагрузки зависит от стратегии распределения нагрузки в локальной окрестности. Поэтому два этих пункта разработки будут рассмотрены в одном разделе.

Итак, было разработано две стратегии распределения нагрузки в локальной окрестности:

а) Стратегия достижения баланса нагрузки в локальной окрестности

б) Стратегия избыточного распределения нагрузки в локальной окрестности

Первая стратегия заключается в том, что при обнаружении дисбаланса в окрестности, перегруженный узел старается распределить избыток нагрузки равномерно для достижения баланса нагрузки в его окрестности.

Вторая стратегия заключается в том, что перегруженный узел стремится избавиться от большего количества нагрузки, чем надо для достижения баланса в локальной окрестности.

Смысл первой стратегии достаточно очевиден с точки зрения поведения узла при наличии избыточного количества нагрузки. Но он может быть недостаточно эффективным, в случаях, когда на нескольких узлах при моделировании некоторого процесса скапливается нагрузка. Такая стратегия приведет к большому числу передач нагрузки с узла со скоплением нагрузки к соседним узлам. С другой стороны, такую ситуацию можно предугадывать и использовать стратегию избыточного распределения нагрузки для освобождения места под нагрузку, которая скапливается в процессе моделирования. Характерный пример такой ситуации – моделирование процессов типа «коллапс» методом частиц-в-ячейках.

2.2.3. Алгоритм балансировки №1

Этот алгоритм балансировки основан на стратегии достижения баланса нагрузки.

Перед описанием введем несколько обозначений:

- Пронумеруем узлы окрестности, причем узел на котором исполняется (далее просто узел) алгоритм балансировки будет 0, а узлы его окрестности $N = 1...k$
- Пусть n_i – количество нагрузки на i узле
- Пусть m_i – количество нагрузки, которое 0 узел планирует передать узлу i
- Пусть порог дисбаланса – ε

В процессе исполнения может возникнуть несколько типов ситуаций:

а) В окрестности есть узлы, которые имеют меньшую нагрузку, чем узел:

$$\exists i \in N(|n_i - n_0| > \varepsilon \wedge n_i < n_0)$$

б) Нагрузка узлов в окрестности меньше нагрузки на узле:

$$\forall i \in N(|n_i - n_0| > \varepsilon \wedge n_i < n_0)$$

в) В окрестности есть узлы, которые имеют большую нагрузку, чем узел:

$$\exists i \in N(|n_i - n_0| > \varepsilon \wedge n_i \geq n_0)$$

г) Нагрузка узлов в окрестности больше нагрузки на узле:

$$\forall i \in N(|n_i - n_0| > \varepsilon \wedge n_i > n_0)$$

д) Отсутствие дисбаланса нагрузки:

$$\forall i \in N(|n_i - n_0| < \varepsilon)$$

Очевидно, в случае 5 балансировка нагрузки не требуется. В случаях 4, 3 и 1 балансировка нагрузки не требуется, так как теоретически после проведения балансировки нагрузки на соседних узлах при дальнейшем исполнении балансировка нагрузки не потребуется. В противном случае после проведения балансировки на соседних узлах, узел придет к ситуации 2, в которой, в свою очередь, балансировка требуется.

Рассмотрим подробнее ситуацию 2. Для определения количества нагрузки m_i, которую надо распределить в локальной окрестности необходимо решить систему линейных уравнений:

$$\begin{cases} n_0 - \sum_k (m_i) = n_1 + m_1 \\ n_0 - \sum_k (m_i) = n_2 + m_2 \\ \vdots \\ n_0 - \sum_k (m_i) = n_k + m_k \end{cases}$$

Если $\exists i(m_i < 0)$, это значит, что нет возможности достигнуть баланса нагрузки в данной окрестности. Для решения этой проблемы необходимо решить эту же систему уравнений, исключив соседей, для которых на предыдущем этапе решение по миграции количества нагрузки было отрицательным. Такое решение не приведет к балансу, но зато создаст ситуацию возникновения дисбаланса на соседних узлах, которые были проигнорированы при распределении нагрузки. После нескольких балансировок такой дисбаланс должен быть устранен.

2.2.4. Алгоритм балансировки №2

Этот алгоритм балансировки основан на стратегии избыточного распределения нагрузки в локальной окрестности. Обозначения и ситуации возникновения дисбаланса остаются прежними.

Очевидно, в случае 5 балансировка нагрузки не требуется. В случаях 4, 3 и 1 балансировка нагрузки не требуется, так как теоретически после проведения балансировки нагрузки на соседних узлах балансировка нагрузки не потребуется. В противном случае после проведения балансировки на соседних узлах, узел придет к ситуации 2, в которой, в свою очередь, балансировка требуется.

Рассмотрим подробнее ситуацию 2. Для определения количества нагрузки m_i, которую надо распрелить в локальной окрестности необходимо решить систему линейных уравнений:

$$\begin{cases} n_0 - \sum_k (m_i) = n_1 + m_1 \\ n_0 - \sum_k (m_i) = n_2 + m_2 \\ \vdots \\ n_0 - \sum_k (m_i) = n_k + m_k \end{cases}$$

Как и в случае с первым алгоритмом балансировки, существование $\exists i (m_i < 0)$ значит, что нет возможности достигнуть баланса нагрузки в данной окрестности. Для решения этой проблемы необходимо решить эту же систему уравнений, исключив соседей $M = m_i | m_i > 0$, для которых на предыдущем этапе решение по миграции количества нагрузки было отрицательным. Таким образом после решения уравнения алгоритм балансировки решит передать $rm_0 = \sum_{m \in M} m$ нагрузки. Для достижения избыточности распределения необходимо передать дополнительно $(n_0 - rm_0)k$, где k – это параметр алгоритма. Таким образом после окончания алгоритма балансировки каждый узел, который был учтен при решении уравнения получит следующиее количество нагрузки:

$$m_i + \frac{(n_0 - rm_0)k}{|M|}$$

Такое решение приведет к избыточной передаче нагрузки на узле, на котором возник дисбаланс.

2.2.5. Характеристика

Представленные алгоритмы балансировки соответствуют предъявленным требованиям:

– Алгоритмы балансировки могут производить балансировку нагрузки по данным предыдущего пространственного разбиения

– Алгоритм балансировки опираются на распределение нагрузки в рамках локальной окрестности (1-2-окрестности)
– Алгоритм балансировки децентрализованные

И алгоритмы балансировки имеют следующие вход и выход:

– Вход:
    а) Информация о нагрузке в локальной части разбиения пространства моделирования
    б) Параметры алгоритма балансировки (алгоритм №2)
– Выход:
    а) Решение по построению нового разбиения

Глава 3

Практические результаты

3.1. Тестовый стенд

3.1.1. Реализация ОТС

Предложенный ОТС был реализован в виде программного прототипа. Рассмотрим его. ОТС содержит интерфейс для подключения балансировщика, интерфейс для подключения модуля, реализующего прикладной алгоритм и систему, обеспечивающую исполнение прикладного алгоритма (рис. 3.1). Таким образом, на вход ОТС приниимает реализацию задачи и реализацию балансировщика.

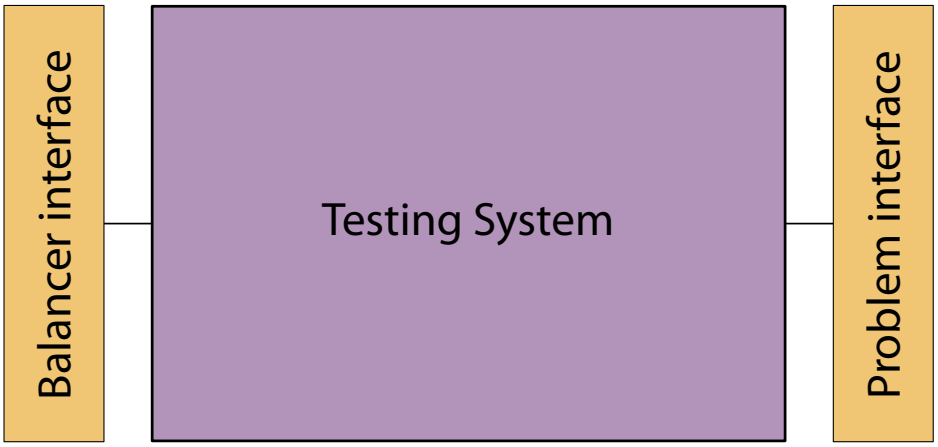

Рисунок 3.1: Архитектура тестового стенда

Реализация задачи

В соответствии с моделью вычислений, прикладная задача – это множество фрагментов, отношение соседства на множестве фрагментов и операторы шага итерации и редукции. ОТС предоставляет интерфейсы для описания множества фрагментов, которые инкапсулируют в себе эти операторы и отношение соседства. ОТС и интерфейсы были реализованы на языке C++. Таким образом, со стороны пользователя, прикладная задача – это реализация

соответствующего интерфейса ОТС на языке C++. Таких реализаций для описания задачи может быть несколько, с разными поведением и функциями.

Пользователь ОТС может определять любые данные в реализации множества фрагментов, которые будут определять состояние фрагмента. Интерфейсы содержат управляемый пользователем внутренний счетчик, который определяет модельный момент времени (шаг модельного времени или номер итерации). Для упрощения описания процесса исполнения помимо счетчика модельного времени был введен счетчик прогресса исполенния на данном шаге итерации (подшаг итерации). Информация об отношении соседства на множестве фрагментов хранится распределенно: каждый объект из множества фрагментов содержит информацию о его локальной окрестности (локальная окрестность задается пользователем в момент создания экземпляра фрагмента). Реализации операторов шага итерации и редукции – это реализации методов интерфейсов фреймворка. Ниже представлена существенная часть исходного кода интерфейса для описания множества фрагментов:

```
class Fragment {
public:
  // General
  Fragment(ID id);
  virtual ~Fragment();
  ID id();

  // Neighbours and their location
  bool isNeighbour(const ID& id);
  void updateNeighbour(ID id, ts::NodeID node);
  void addNeighbour(ID id, ts::NodeID node);

  // Fragment steps defined by user
  virtual ReduceData* reduce() = 0;
  virtual ReduceData* reduce(ReduceData* data) = 0;
  virtual void reduceStep(ReduceData* data) = 0;
  virtual void runStep(std::vector<Fragment*> neighbours) = 0;

  // Flag setters
  void setEnd();
  void setUpdate();
  void setReduce();
  void setNeighbours(uint64_t iteration, uint64_t progress);

  // State setters
```

```
  void nextIteration();

  // State getters
  uint64_t iteration();
  uint64_t progress();

  virtual Fragment* split() = 0;
  virtual void merge(Fragment*) = 0;
};
```

Реализация задачи в терминах стенда – это множество объектов (фрагментов) из реализованного множества фрагментов, которые передаются ОТС в качестве входа. Начальное распределение фрагментов по узлам мультикомпьютера может быть задано статически: в процессе порождения фрагментов на мультикомпьютера, либо для этой цели может быть использован балансировщик: фрагменты порождаются на нескольких выделенных узлах, и балансировщик во время порождения начинает распределять фрагменты по соседним узлам мультикомпьютера.

ОТС скрывает в себе такие операции как: применение оператора к фрагменту, поиск соседних фрагментов для применения оператора, редукция данных, миграция фрагмента.

Исполнение задачи

ОТС содержит множество фрагментов, итерационные шаги которых необходимо исполнять. Исполнение производится по подшагам. Управление ходом исполнения производится с помощью установки флагов: при необходимости в редукции данных; необходимость в соседних фрагментах для применения оператора; необходимость в обновлении состояния фрагмента для глобального использования; для указания того, что на следующем подшаге будет следующая итерация. ОТС передает фрагменты на исполнение в соответствии с выставленными флагами. Если фрагменту для данного подшага не нужны соседние фрагменты, то он сразу передается на исполнение. Если же есть необходимость в соседних фрагментах, то сначала производится поиск соответствующих фрагментов на данном узле, если не все фрагменты найдены, то данный фрагмент пропускается, пока на узел не придут все соседние фрагменты. Во многих задачах нет необходимости во всем фрагменте (например, в методе частиц-в-ячейках необходимы теневые грани ячеек), поэтому для оптимизации была введена такая абстракция как *частичная реализация фрагмента*, которая определяется пользователем. И именно частичная реализация фрагмента передается на узел с фрагментом, которому для применения оператора необходимы соседние фрагменты.

Балансировка

Балансировщик реализуется как внешняя динамически подключаемая к ОТС библиотека. Соответственно, балансировщик может быть реализован на любом языке программирования, который позволяет собрать динамическую библиотеку. Реализация алгоритма балансировки работает в терминах нагрузки: на вход принимает количество вычислительной нагрузки на локальный узел и на соседние узлы, а на выходе предоставляет информацию о том, как нагрузка должна быть распределена. После этого ОТС сама решает, какие именно фрагменты необходимо передать. Величина нагрузки определяется ОТС в единицах, зависящих от задачи (и определяемых пользователем при описании прикладной задачи).

Система тестирования периодически обращается к балансировщику для определения необходимости инициации балансировки на узле.

Миграция фрагмента с узла i на узел j производится следующим образом: оповещается узел j о начале миграции; оповещаются все узлы, на которых есть соседние фрагменты, о том, что фрагмент будет перемещен на узел j; на узел j передается фрагмент и все частичные реализации фрагментов, которые были переданы ранее на узел i.

3.1.2. Ограничения

- Тестирование можно проводить только на топологии «кольцо»
- Логика выбора фрагментов для миграции может быть реализована только как часть стенда. В текущей реализации используется стратегия выбора «первый подходящий».

3.2. Реализация прикладной задачи

Задача моделирования взрыва методом частиц-в-ячейках была реализована на тестовом стенде. Параллельная реализация была сделана путем декомпозиции пространства моделирования. Фрагмент был реализован следующим образом:

- Данные:
  - Часть сеточных значений в соответствии с пространственной декомпозицией задачи. На рисунке[1] (рис. 3.2) части сетки, принадлежащие разным фрагментам, выделены разными цветами.
  - Границы сеточных значений соседних фрагментов
  - Значения частиц, которые расположены в данном фрагменте сетки
- Итерационный шаг метода частиц-в-ячейках был реализован с учетом необходимости создания виртуальных фрагментов.

В качестве входных данных реализация задачи принимает файл с описанием местоположения частиц в пространстве моделирования.

[1] Для простоты иллюстрации, изображена двумерная проекция пространства моделирования

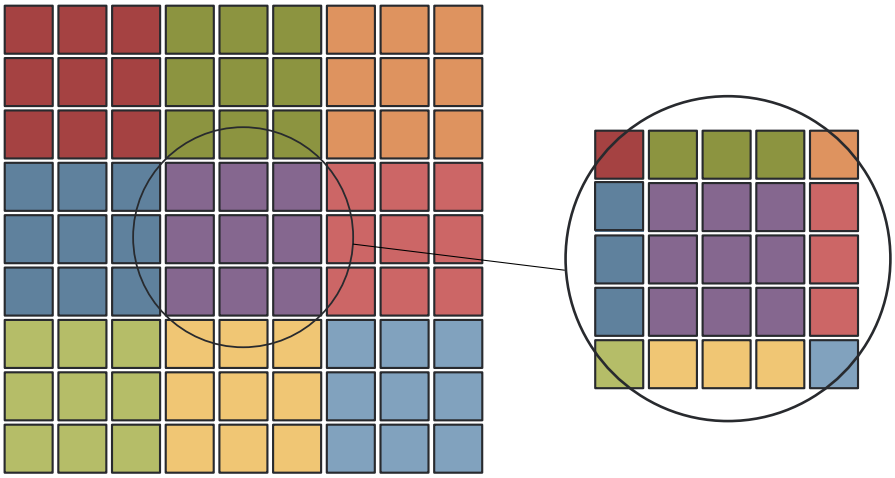

Рисунок 3.2: Декомпозиция пространства моделирования

3.3. Реализация алгоритмов балансировки

В соответствии с внешним интерфейсом ОТС для подключения балансировщиков алгоритмы балансировки были реализованы как внешние функции, которые соответствуют модельному описанию.

На вход балансировщики принимают информацию о текущем распределении нагрузки в локальной окрестности узла, на котором происходит исполнение – два аргумента: количество нагрузки на узле, массив с количеством нагрузки на соседних узлах.

На выходе балансировщики возвращают массив с количеством нагрузки, которую надо передать соседним узлам.

Так как ОТС поддерживает только топологию «кольцо», в реализации балансировщиков были использованы частные случаи алгоритмов для двух узлов в 1-окрестности.

3.4. Результаты тестирования

Представленные алгоритмы балансировки были протестированы на задаче моделирования «взрыва», реализованной на отладочном тестовом стенде. Пространство моделирования задачи было разбито сеткой 32x32x32, пространство моделирование содержало 10000 частиц. Исполнение производилось на 4 узлах, каждый из которых содержал по 8 фрагментов. Такие параметры являются близкими к используемым при реальном моделировании [12].

Для определения изменения распределения нагрузки на стенде был произведен запуск задачи без балансировки[2].

Как можно видеть на изображении (рис. 3.3), в начале исполнения узел №3 содержит около 95% всех частиц. Таким образом, можно сделать вывод, что взрыв оказался локализован

[2]Стоит отметить, что при исполнении «настоящей» задачи, значительно большего размера и с большим количеством частиц, график распределения нагрузки не удалось бы получить из-за перегрузки одного из узлов

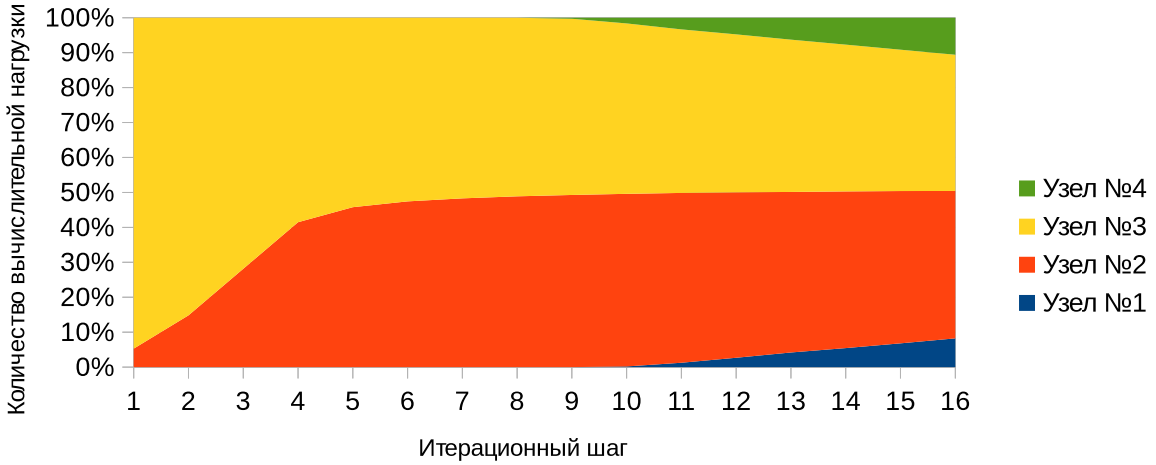

Рисунок 3.3: Распределение нагрузки по узлам в модельном времени (без балансировки)

на этом узле. Постепенно в процессе исполнения, распределение нагрузки выравнивается в силу того, что частицы начинают равномерно разлетаться по сетке.

Схематично модельный взрыв изображен на следующем рисунке (рис. 3.4)

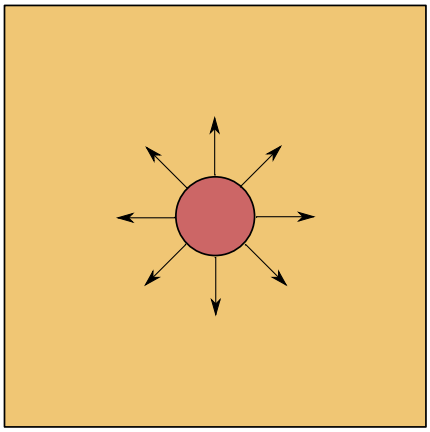

Рисунок 3.4: Схематичное изображение начала разлета частиц

3.4.1. Реализация алгоритма №1

Алгоритм балансировки, основанный на стратегии достижения баланса на локальной окрестности, не показал никаких результатов при балансировке без использования механизмов создания виртуальных фрагментов, из-за отсутствия возможности передать фрагменты, которые требуют виртуализации (рис. 3.5). На практике же это означает, что исполнение не может быть произведено из-за наличия фрагмента, размер которого превышает доступные ресурсы любого вычислителя.

Использование механизмов создания виртуального фрагмента при балансировке сделало возможным разбить фрагмент, содержащий 95% нагрузки на две половины и отобразить его на 2 узла (рис. 3.6).

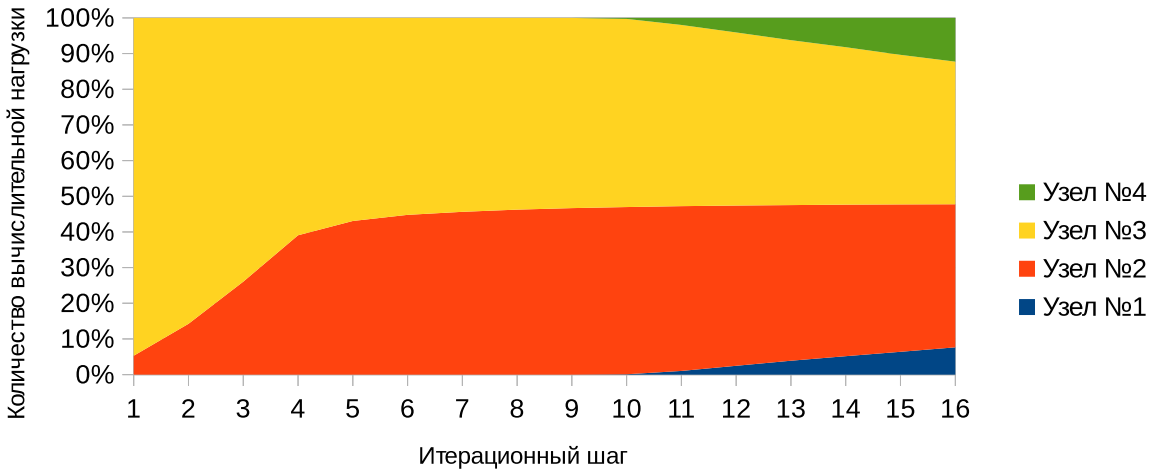

Рисунок 3.5: Распределение нагрузки по узлам в модельном времени (с балансировкой без виртуальных фрагментов)

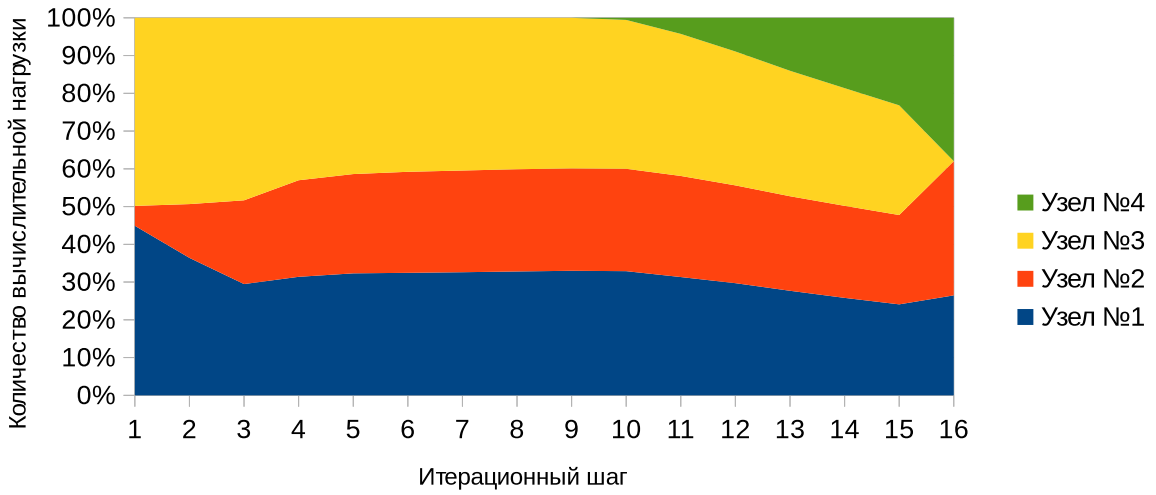

Рисунок 3.6: Распределение нагрузки по узлам в модельном времени (с балансировкой и виртуальными фрагментами)

Таким образом, можно сделать вывод, что такой алгоритм балансировки делает возможным исполнение задач моделирования процессов типа «взрыв» при наличии механизмов, позволяющих отображать фрагмент на несколько узлов мультикомпьютера.

3.4.2. Реализация алгоритма №2

Алгоритм балансировки, основанный на стратегии избыточного распределения нагрузки на локальной окрестности, так же не показал никаких результатов при балансировке без использования механизмов создания виртуальных фрагментов, из-за отсутствия возможности передать фрагменты, которые требуют виртуализации.

Использование механизмов создания виртуального фрагмента при балансировке сделало возможным разбить фрагмент, содержащий 95% нагрузки на две половины и отобразить его

на 2 узла, как и в случае с первой реализацией алгоритма балансировки. Но результат оказался таким же, потому как в силу специфики «взрыва» в начале разлета частиц существует только 2 типа фрагментов: с большим количеством нагрузки и практически без нагрузки. Таким образом при любом выборе параметра k, не найдется такое число соответствующих фрагментов, которые бы удовлетворяли решению по избыточному перемещению частиц:

$$m_i + \frac{(n_0 - rm_0)k}{|M|}$$

Таким образом, можно сделать вывод, что такой алгоритм балансировки делает возможным исполнение задач моделирования процессов типа «взрыв» при наличии механизмов, позволяющих отображать фрагмент на несколько узлов мультикомпьютера, но не демонстрирует стратегию, заложенную в основу алгоритма.

Заключение

Цели и задачи были достигнуты в полной мере, а именно: были разработаны отладочный тестовый стенд и локальные распределенные алгоритмы балансировки; тестовый стенд был реализован и проверен на ряде синтетических тестов; были реализованы разработанные алгоритмы балансировки и задача моделирования «взрыва» методом частиц-в-ячейках на тестовом стенде; алгоритмы балансировки были протестированы на целевой задаче с помощью стенда и был произведен анализ полученных результатов.

На защиту представлены следующие результаты:

а) Разработан и реалирован отладочный тестовый стенд
б) Разработанны и реализованы алгоритмы балансировки нагрузки
в) Реализована задача моделирования «взрыва» методом частиц-в-ячейках на тестовом стенде
г) Результаты тестирования алгоритмов балансировки на целевой задаче

Результаты работы были опубликованы и представлены:

а) 52-я Международная научная студенческая конференция МНСК-2014.
   Доклад и тезисы: «Система тестирования алгоритмов динамической балансировки нагрузки на узлы мультикомпьютера для итерационных задач»
б) Конференция молодых ученых ИВМиМГ СО РАН 2014.
   Доклад: «Стенд для отладки и тестирования качества работы локальных системных распределенных алгоритмов динамической балансировки нагрузки»
в) 53-я Международная научная студенческая конференция МНСК-2015.
   Доклад и тезисы: «Автоматизация тестирования распределеных алгоритмов динамической балансировки вычислительной нагрузки»
г) Конференция молодых ученых ИВМиМГ СО РАН 2015.
   Доклад: «Автоматизация тестирования качества работы системных распределенных алгоритмов балансировки вычислительной нагрузки»
д) Международная научная конференция Параллельные вычислительные технологии 2015
   Доклад и статья: «Стенд для отладки и тестирования качества работы локальных системных распределенных алгоритмов динамической балансировки нагрузки»
е) Вестник Южно-Уральского государственного университета. Серия «Вычислительная математика и информатика».

Статья (в печати): «Стенд для отладки и тестирования качества работы локальных системных распределенных алгоритмов динамической балансировки нагрузки»

В дальнейшем планируется развивать работу по следующим направлениям:

– Разработка новых алгоритмов балансировки
– Расширение класса задач, который поддерживает отладочный тестовый стенд

Список литературы

1. Киреев С. Е. Параллельная реалиация метода частиц в ячейках для моделирования задач гравитационной космодинамики // Автометрия. — 2006. — Т. 42, № 3. — С. 32–39.
2. Barnard, Stephen T. Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems / Barnard, Stephen T., Simon, Horst D. // Concurrency: Practice and Experience. — 1994. — Vol. 6, no. 2. — Pp. 101–117.
3. Cybenko George. Dynamic load balancing for distributed memory multiprocessors // Journal of parallel and distributed computing. — 1989. — Vol. 7, no. 2. — Pp. 279–301.
4. Elsässer Robert. Diffusion schemes for load balancing on heterogeneous networks / Elsässer Robert, Monien Burkhard, Preis Robert // Theory of Computing Systems. — 2002. — Vol. 35, no. 3. — Pp. 305–320.
5. Ferraro Robert D. Dynamic load balancing for a 2D concurrent plasma PIC code / Ferraro Robert D, Liewer Paulett C, Decyk Viktor K // Journal of computational physics. — 1993. — Vol. 109, no. 2. — Pp. 329–341.
6. Hiroshi Nakashima. OhHelp: a scalable domain-decomposing dynamic load balancing for particle-in-cell simulations / Hiroshi Nakashima, Yohei Miyake, Hideyuki Usui, Yoshiharu Omura // Proceedings of the 23rd international conference on Supercomputing / ACM. — 2009. — Pp. 90–99.
7. Hockney, Roger W. Computer simulation using particles. / Hockney, Roger W., Eastwood, James W. / CRC Press, 1988.
8. Hui Chi-Chung. Theoretical analysis of the heterogeneous dynamic load-balancing problem using a hydrodynamic approach / Hui Chi-Chung, Chanson Samuel T // Journal of Parallel and Distributed Computing. — 1997. — Vol. 43, no. 2. — Pp. 139–146.
9. Kireev Sergey. The LuNA library of parallel numerical fragmented subroutines / Kireev Sergey, Malyshkin Victor, Fujita Hamido // Parallel Computing Technologies. — Springer, 2011. — Pp. 290–301.
10. Kohring, Gregory Allen. Dynamic load balancing for parallelized particle simulations on MIMD computers // Parallel Computing. — 1995. — Vol. 21, no. 4. — Pp. 683–693.

11. M.A. Kraeva. Assembly technology for parallel realization of numerical models on MIMD-multicomputers / M.A. Kraeva, V.E. Malyshkin // Future Generation Computer Systems. — 2001. — 4. — Vol. 17. — Pp. 755–765.

12. M.A. Kraeva. Implementation of PIC method on MIMD multicomputers with assembly technology / M.A. Kraeva, V.E. Malyshkin // High-Performance Computing and Networking. — 1997. — 1. — Pp. 541–549.

13. Malyshkin Victor E. LuNA fragmented programming system, main functions and peculiarities of run-time subsystem / Malyshkin Victor E, Perepelkin Vladislav A // Parallel Computing Technologies. — Springer, 2011. — Pp. 53–61.

14. Van Driessche Rafael. An improved spectral bisection algorithm and its application to dynamic load balancing / Van Driessche Rafael, Roose Dirk // Parallel Computing. — 1995. — Vol. 21, no. 1. — Pp. 29–48.

15. Walker, David W. Characterizing the parallel performance of a large-scale, particle-in-cell plasma simulation code // Concurrency: Practice and experience. — 1990. — Vol. 2, no. 4. — Pp. 257–288.

16. Walshaw Chris. Dynamic load-balancing for PDE solvers on adaptive unstructured meshes / Walshaw Chris, Berzins Martin // Concurrency: Practice and Experience. — 1995. — Vol. 7, no. 1. — Pp. 17–28.

17. Watts Jerrell. A practical approach to dynamic load balancing / Watts Jerrell, Taylor Stephen // Parallel and Distributed Systems, IEEE Transactions on. — 1998. — Vol. 9, no. 3. — Pp. 235–248.

18. Williams, Roy D. Performance of dynamic load balancing algorithms for unstructured mesh calculations // Concurrency: Practice and experience. — 1991. — Vol. 3, no. 5. — Pp. 457–481.

19. Wolfheimer Felix. A parallel 3D particle-in-cell code with dynamic load balancing / Wolfheimer Felix, Gjonaj Erion, Weiland Thomas // Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment. — 2006. — Vol. 558, no. 1. — Pp. 202–204.