Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра вычислительных технологий и моделирования

Бурачковский Андрей Игоревич

Алгоритмы работы с распределенными данными на расчетных сетках общего вида

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Научный руководитель:
д.ф-м.н., доцент
Ю.В.Василевский

Москва, 2016

Содержание работы

1. Введение

Для решения многих задач математической физики, необходимо построение динамических расчетных сеток, т.е. сеток меняющихся со временем. Например для задач с движущимися объектами или задач моделирования течения жидкости со свободной поверхностью необходимо сгущать сетку вокруг объекта и на поверхности жидкости, соответственно.

Для реализации процесса сгущения существует множество методов, но существует недостаток присущий каждому методу: время вычислений. Для распределенных сеток большой размерности время перераспределения сетки может достигать критичного значения, особенно для задач, в которых сетку необходимо перестраивать в каждый момент времени.

Для решения этой проблемы можно использовать распределенные сетки, т.е. сетки, ячейки которых хранятся в памяти различных процессоров. А именно: распределить сетку по вычислительным узлам (процессорам) и, используя некоторую технологию параллельного программирования, реализовать взаимодействие между узлами. Также важно понимать что при таком распределении необходимо учитывать сбалансированность вычислительной нагрузки на процессоры (все наши действия потеряют смысл если, например, на один из 10 процессоров будет приходиться 80% сетки).

В данной работе будет описана реализация распределенных динамических сеток на примере сгущения типа «octree» (восьмеричное дерево) [7], с использованием программной платформы для разработки численных моделей INMOST. Также используется технология балансировки ячеек по процессорам.

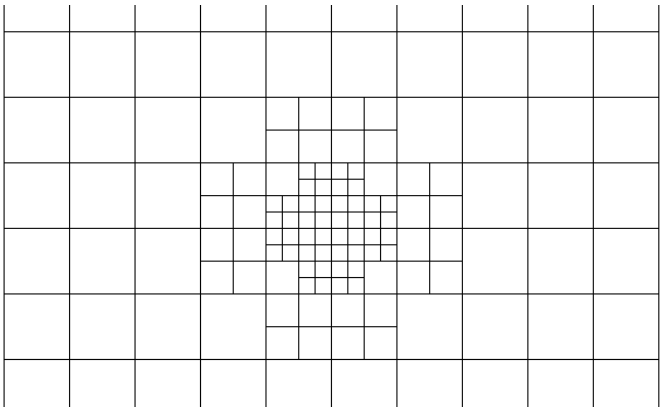

Рисунок 1: Пример сгущения сетки технологией octree

2. INMOST

В данной работе для реализации распределенных сеток используется программная платформа INMOST [1]. Достоинства данной платформы отлично подходят для выполнения поставленной задачи. Платформа INMOST использует библиотеку передачи данных MPI (Message passing Interface)[2],

причем основные межпроцессорные взаимодействия скрыты от пользователя средствами C++. Гибкость структуры сеточных данных позволяет использовать сетки типа octree, а технология OpenGL позволяет визуализировать сетки и увидеть распределение их по процессорам. Используя данную платформу можно пройти весь цикл решения численной задачи моделирования: построение расчетной сетки, построение дискретизации, формирование матриц и правой части, решения систем линейных алгебраических уравнений. В рамках данной работы мы остановимся только на первом пункте данной цепочки; на построении и перераспределении расчетных сеток.

2.1. Коротко об структуре INMOST

Рассмотрим основные механизмы и структуры, необходимые для построения расчетной сетки в платформе INMOST. Вообще говоря, сетки бывают разных пространственных размерностей (2D, 3D), далее мы будем рассматривать только трехмерные сетки. Перечислим базовые сеточные *элементы* (Elements):

- *Узел сетки (Node)* - точка в пространстве и сопутствующая ей информация;
- *Ребро (Edge)* - полностью определяется двумя узлами;
- *Грань (Face)* - в общем случае многоугольник, опирающийся на множество ребер;
- *Ячейка (Cell)* - в общем случае многогранник, опирающийся на множество граней.

Основным элементом в сетке является ячейка, собственно говоря, сетка и формируется как набор ячеек.

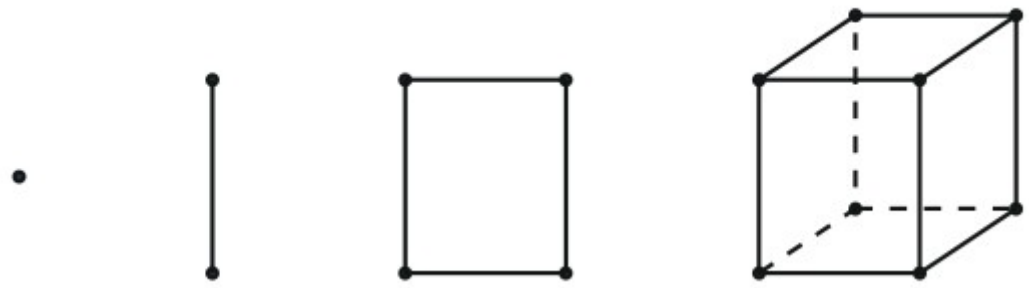

Рисунок 2: Основные элементы 3D сетки: узел, ребро, грань, ячейка.

Платформа INMOST сама реализует хранение этих элементов, предоставляя итераторы для доступа у ним. Кроме того каждый элемент может хранить в себе некоторые данные, например, координаты узла, поток через грань, центр ячейки и т.п. Причем мы сами можем выбирать что сохранять в элементах. Данная гибкость очень востребована для различных задач математической физики, такая возможность пригодится нам и для задачи построения динамических сеток, к которым мы вернемся позже.

Работая с распределенными сетками, может происходить по следующим схемам:

- Сетка может храниться в специальном файле с расширением *.pvtk. Средства INMOST предоставляют удобный функционал для импорта сетки из таких файлов. Файл *.vtk — это файл, в котором описана некая геометрическая структура (в нашем случае сетка). Файл *.pvtk — это файл, в котором указаны названия *.vtk файлов, соответствующих сетке определенного процессора.
- Мы можем создавать распределенную сетку с нуля, т.е. на каждом процессоре создавать свои ячейки. В данном подходе следует соблюдать согласованность (например чтобы одна ячейка не принадлежала двум процессорам) и конформность сеток.
- Мы можем создать всю сетку целиком на одном процессоре и вызвать специальный механизм, которой сам распределит сетку по процессорам. Например, INMOST позволяет использовать внешние пакеты для перераспределения данных (ParMETIS, Zoltan)[3] и [4].

Последними мы и будем пользоваться, т.к. для динамических сеток сбалансированное распределение является одной из первоочередных задач.

Визуализация распределенных сеток. Для визуализирования сеток можно использовать сторонние программы (например paraview[5]) или средства OpenGL. В нашем случае мы будем пользоваться вторым вариантом.

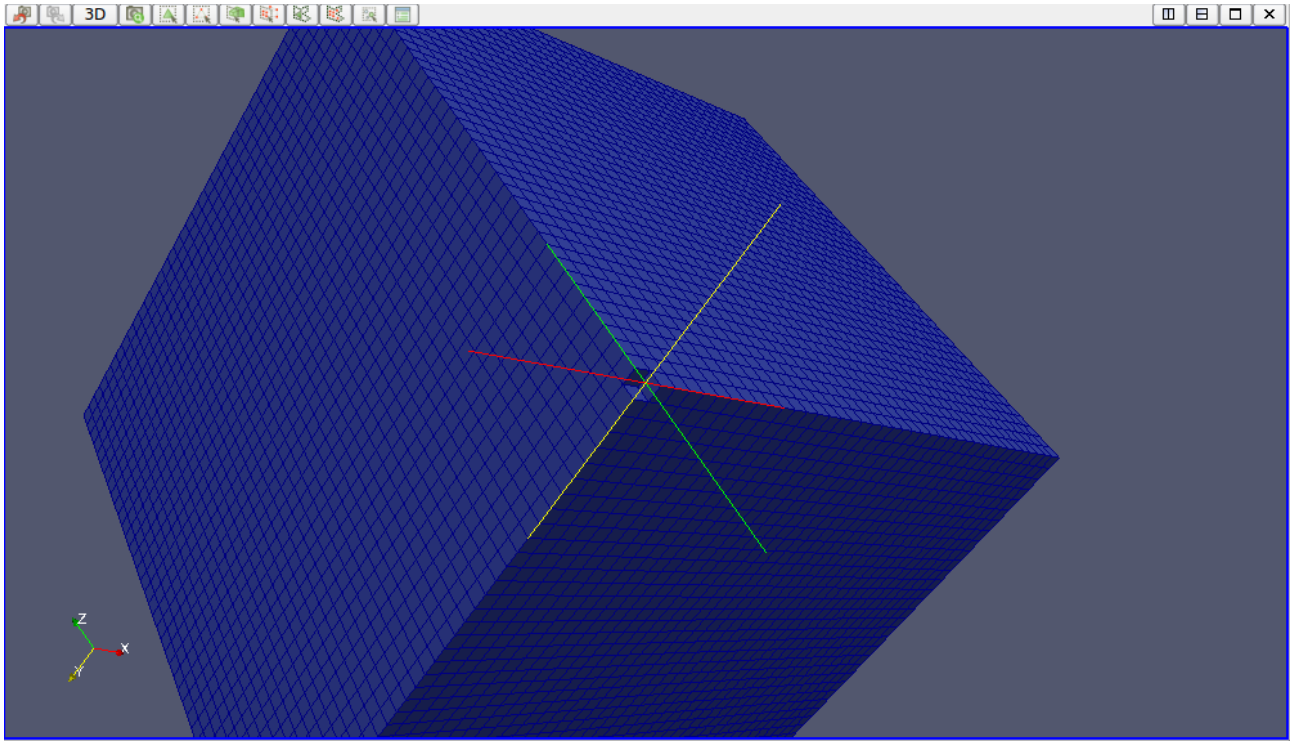

Рисунок 3: Пример визуализации сетки в программе paraview

3. Octree

3.1. Общие сведения

Восьмеричное дерево (октодерево, octree) — древовидная структура, в которой у каждого элемента может быть восемь потомков. Мы будем использовать данный тип наследования для *сгущения* и *разгрубления* сетки, рассматривая ячейку как узел дерева. Таким образом, каждая ячейка может быть поделена на 8 более мелких ячеек. Затем каждая ячейка-потомок может тоже быть разбита еще на 8 ячеек. Таким образом мы получаем *глубину сгущения*. Глубину сгущения можно регулировать в зависимости от требований задачи. Остановимся пока на глубине 2 (рис. 4).

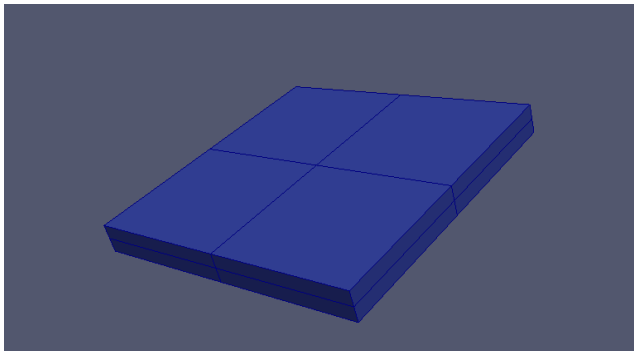

Рисунок 4: Ячейка с 8ю потомками

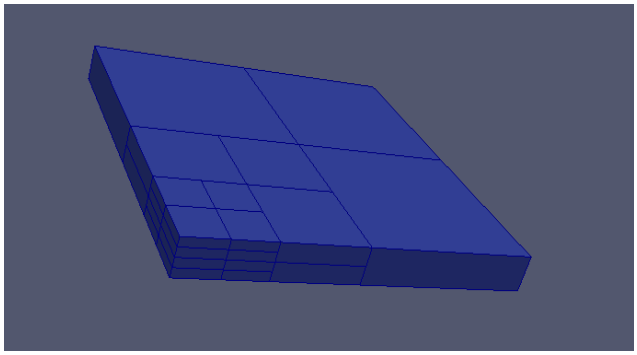

Рисунок 5: Сгущение типа Octree глубины 2

При построении сгущения такого вида важно учитывать очень важное правило: ячейка с уровнем сгущения *n* должна граничить только с ячейкой с уровнем сгущения *n-1* или *n+1*. Иными словами: если мы сгущаем ячейку до второго уровня глубины, то мы обязаны сгустить соседние ячейки либо до уровня 1, либо до 2, либо до 3 (назовем это правило «*»). Благодаря этому правилу мы получаем полезное свойство: *«Грани соседний ячеек соотносятся либо одна ко одной, либо одна к четырем.»*

3.2. Реализация

Первая задача реализации состоит в построении динамически сгущающейся сетки. Для тестирования и демонстрации сгущения мы будем использовать сетку размером 10x10x3, которая будет отслеживать движение курсора мыши по экрану. В данной задаче применяются 2 действия: *сгустить сетку в ячейке* и *разгрубить несколько ячеек в одну*. К счастью, решение данной задачи уже было реализовано, но в другой программной платформе MSPP [6] (предшественник INMOST). В MSPP имеется однопроцессорный пример, демонстрирующий сгущение сетки вокруг курсора мыши. Однако использование устаревшей библиотеки MSPP не позволило бы нам в полной мере реализовать удобный механизм распределения динамической сетки. Поэтому было решено перенести данный пример из MSPP в INMOST. В этом состоит первая часть данной работы.

3.3. Описание структуры и алгоритма примера

В приведенном примере MSPP для реализации сгущения и разгрубления были введены дополнительные структуры. Для реализации сгущения типа восмидерево в каждой ячейке необходимо хранить дополнительную информацию, например:

- массив номеров ячеек потомков;
- номер ячейки родителя;
- уровень глубины ячейки;
- является ли ячейка листом;
- и т.п.

В примере, приведенном в MSPP, использована внешняя по отношению к MSPP структура, которая содержит в себе все эти данные, а также указатель на ячейку в формате MSPP. Таким образом получается, что в данном примере реальная ячейка участвует в 2 структурах. Алгоритм сгущения реализуется проверкой для каждой ячейки функции *cell_should_split*, которая принимает на вход ячейку и возвращает 1, если ячейку надо разбить на 8 потомков. Соответственно, функция *cell_should_split,* может быть какой угодно, в нашем случае, она возвращает значение 1, если расстояние от позиции курсора до центра ячейки во второй метрике не превосходит заранее заданного числа (радиуса). Ячейка, которая должна быть разбита, разбивается на глубину заранее заданного уровня (в нашем случае два уровня, т.е. происходит разбиение ячейки на 8 ячеек, а потом аналогичное разбиение делается для каждого из потомков). Далее, если ячейка должна быть разбита, то необходимо будет проверить, что глубина разбиения соседних ячеек отличается от глубины самой ячейки не более чем на 1 (правило *). Если правило (*) не выполняется, то необходимо провести разбиение и для соседних ячеек. Таким образом мы получаем разбиение, представленное не рисунке 9.

Аналогично сгущению происходит и обратная операция - разгрубление сетки. Используется функция cell_should_unite, и если она возвращает значение 1, необходимо удалить дочерние ячейки (если они есть) и проверить правило (*). Такая проверка делается после каждого изменения позиции курсора мыши, получая динамическое изменение сетки.

Так как программная платформа MSPP является предшественником INMOST, то том этапе изменения были незначительны. С программной точки зрения основные изменения касались лишь изменения имен функций. Пришлось несколько модифицировать структуру данных, а также вместо указателей на объекты использовать без существенных изменений ссылки на объекты. В остальном структура программы осталась такая же. В итоге был получен пример на INMOST для демонстрации динамически сгущающихся сеток. Далее представлены иллюстрации работы данного примера.

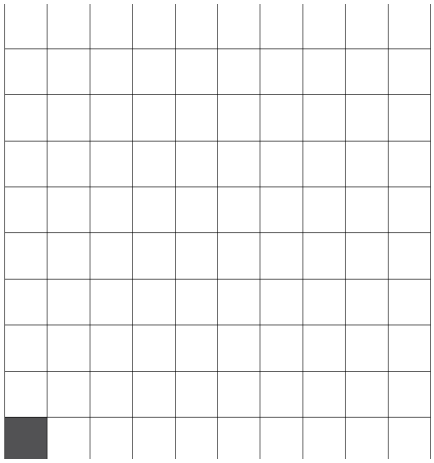

Рисунок 6: Вид сетки в примере Octree

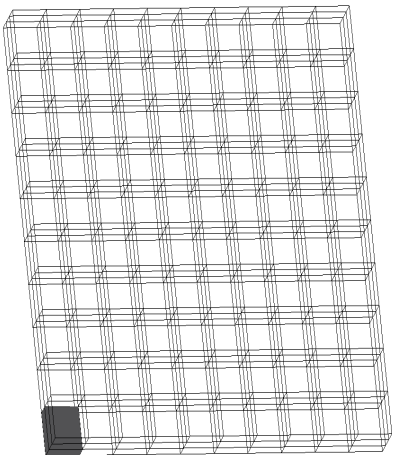

Рисунок 7: Поворот сетки в пространстве

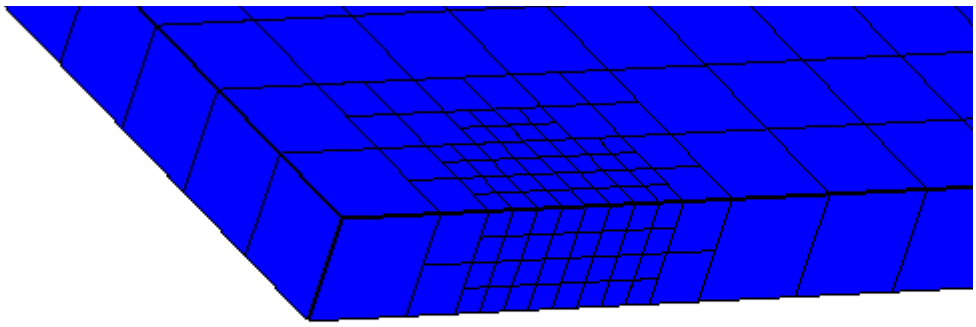

Рисунок 8: Сгущение производится по всем направлениям

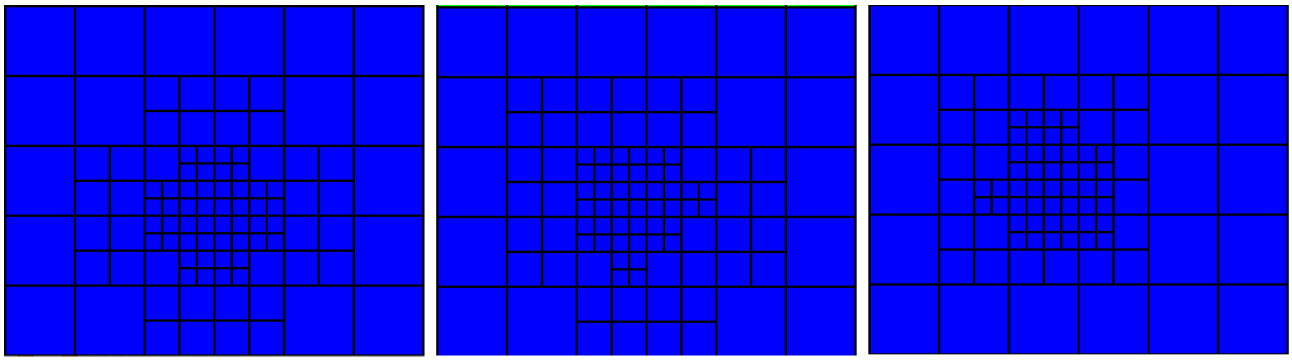

Рисунок 9: Сгущение сетки по движению курсора мыши

Данная версия примера работает только на одном процессоре. Поэтому следующим шагом в данной работе будет добавление возможности работать в параллельном режиме**.**

3.4. Переход к параллельной версии примера

Основная задача данной работы: создать механизм для построения сбалансированной распределенной динамической сетки. Для организации параллельной версии мы будем использовать технологию MPI. В данном подходе у каждого процессора будет свой набор ячеек. Например, в последовательной версии в начале программы вызывается функция *grid_init*, которая строит сетку указанного размера (в нашем случае 10x10x3). В параллельной версии каждый процессор вызывает эту функцию, но генерирует лишь свою часть сетки (рис. 10).

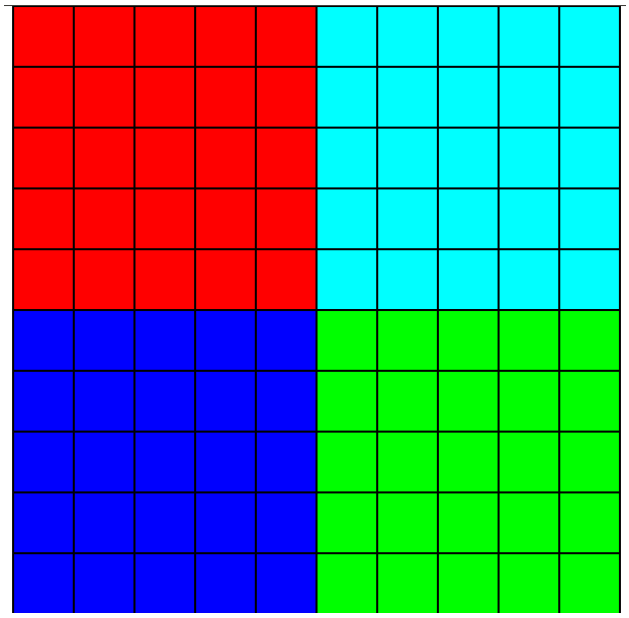

Рисунок 10: Пример распределения сетки (10x10x3) по 4ым процессорам

В первый момент каждый процессор ничего не знает о ячейках других процессоров. Далее при изменении курсора мыши происходит процесс сгущения (проверка cell_should_split, cell_should_unite), у каждого процессора свой. Каждый процесс, независимо от других, производит сгущение своей сетки. При таком подходе важно помнить о правиле (*), которое может не выполняться на граничных участках. В первых версиях программы так и происходило, при сгущении сетки на границе двух процессоров получалось так, что на одном процессоре ячейка имела глубину 2, а на втором соседняя ячейка имела глубину 0 (рис. 11). Такая проблема возникала из-за того, что для процессора 1 ячейка номер 1 являлась граничной, т.к. процессор 1 не знает о ячейке номер 2 (рис. 13). Т.е. если бы обе эти ячейки были бы на одном

процессоре, то сначала сгустилась бы ячейка 2, а потом для выполнения правила (*) сгустилась бы ячейка 1, но у процессора 2 нет ячейки 1.

Для решения этой проблемы была усовершенствована функция механизма поиска ячеек, которые должны быть разбиты. Теперь помимо простой проверки cell_should_split, выполняется еще одна проверка cell_should_split но с меньшей глубиной и большим радиусом. Таким образом ячейка 1 у процессора 1 попадет под действие cell_should_split и будет разбита (рис. 12).

После этого мы получаем динамическую распределенную сетку, которая сгущается при движении курсора мыши.

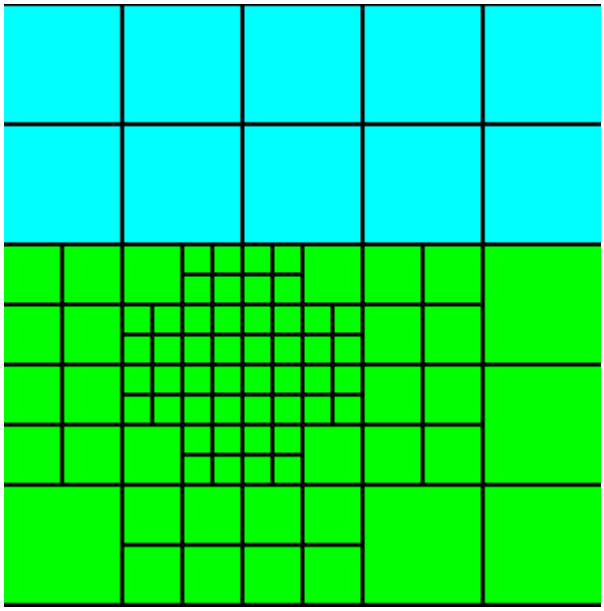

Рисунок 11: Нарушение правила (*). Ячейка глубины 2 граничит с ячейкой глубины 0

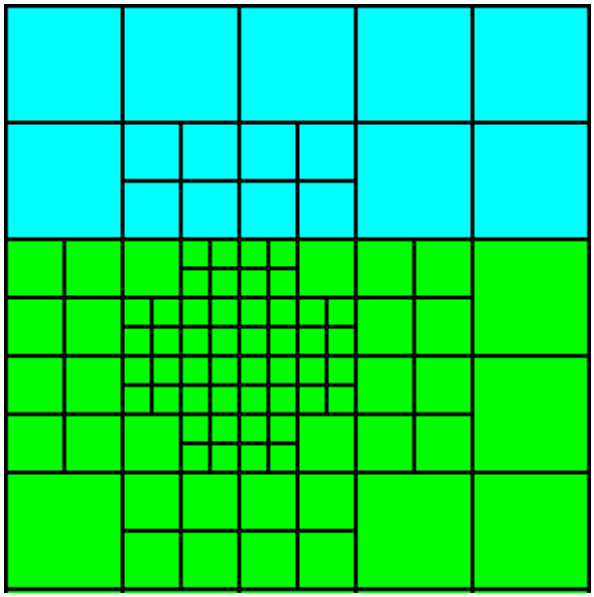

Рисунок 12: Правило (*) выполняется

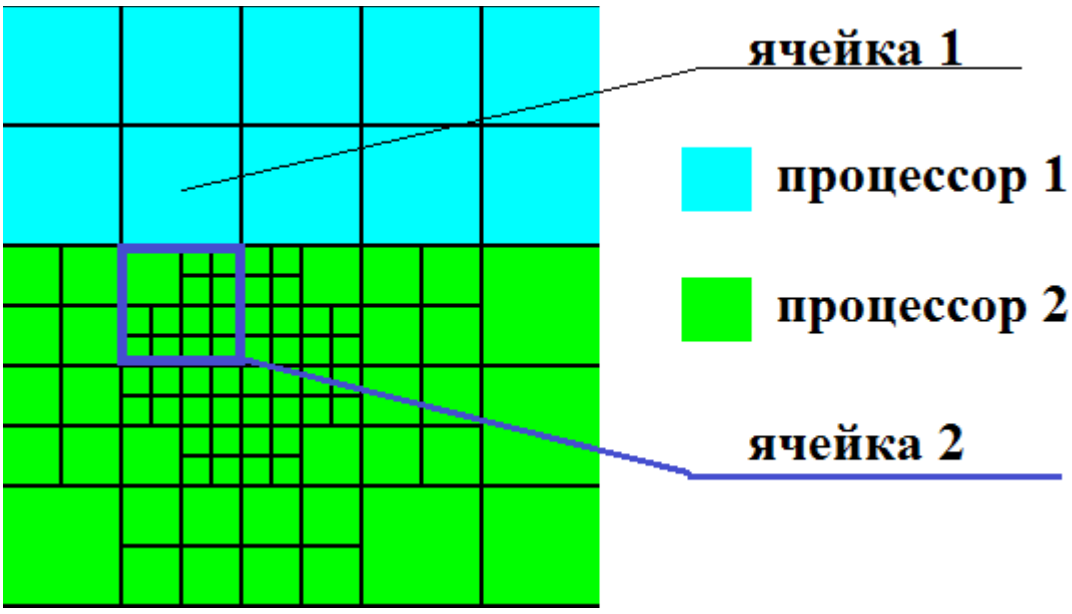

Рисунок 13: Проблема сгущения на границе процессоров

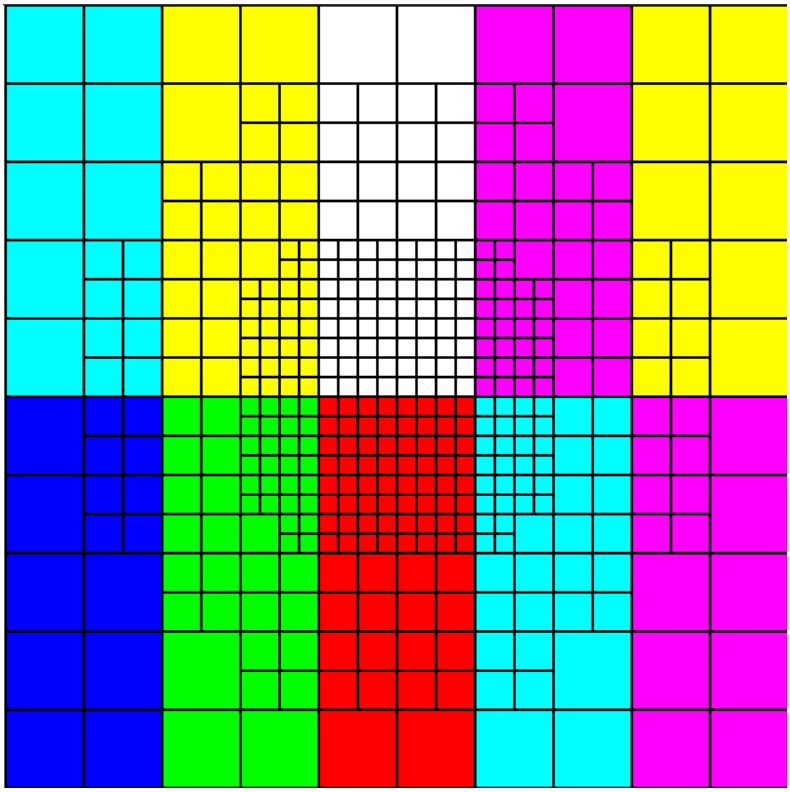

Рисунок 14: Разбиение с большим радиусом на 10 процессорах

3.5. Примеры распределения сеток

Иллюстрации, которые представлены ранее, являются результатом работы тестового примера, который мы разрабатываем. В примере из MSPP уже было реализовано рисование сеток средствами OpenGL, однако это выполнялось только на одном процессоре. Для демонстрации распределенной сетки необходимо нарисовать всю сетку на одном из процессов. Однако, т.к. процессоры имеют данные только о своих ячейках, то здесь необходимо использовать средства пересылки данных MPI. Для решения этой задачи было решено использовать схему master-slaves, т.е. один процессор рассматривается как главный (головной, master), а остальные как вспомогательные (slaves). Соответственно, головной процессор и будет все рисовать (только у него будет вызвана функция *glutMainLoop).* А остальные процессоры встают в ожидание сообщения от головного процессора. При каждом изменении структуры сетки, головной процессор отправляет команду вспомогательным, чтобы те прислали ему координаты своих ячеек. Получив координаты ячеек с других процессоров, он рисует всю сетку различными цветами, соответствующими номеру процессора обладателя ячейки (рис. 14).

4. Балансировка сеток

На данном этапе мы имеем распределенную динамическую сетку, а также соответствующий функционал для ее демонстрации. Однако это распределение является статичным, т.е. каждый процессор имеет одно и то же количество корневых ячеек. В идеальном случае желательно чтобы каждый процессор имел одинаковое количество ячеек, однако учитывая, что при сгущении одна ячейка делится на 8, то процессор, на который приходится большая часть сгущающихся ячеек сильно обгонит остальных по количеству ячеек. Приведем пример, наглядно демонстрирующий это: рассмотрим сетку 10x10x3 на 4х процессорах. На начальный момент на каждый процессор приходится 5*5*3 = 75 ячеек. Сделаем сгущение только на втором процессоре (зеленый). На остальных процессорах как было 75 ячеек так и осталось. А на втором, учитывая, что выполнялось разбиение 2 уровня, получаем 12*4*3 + 6*4*3 + 16*3 = 636. Разница в 8 раз, и это учитывая, что сетка очень маленькая (в реальных задачах используются сетки гораздо большего размера). Очевидно, что балансировка здесь крайне необходима, иначе все действия по распределению сеток потеряют смысл.

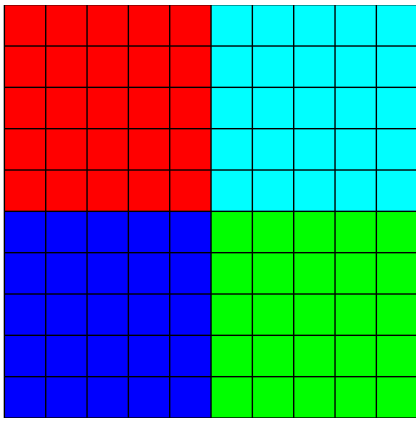

Рисунок 15: Разбиение сетки 10x10x3 по 4 процессорам

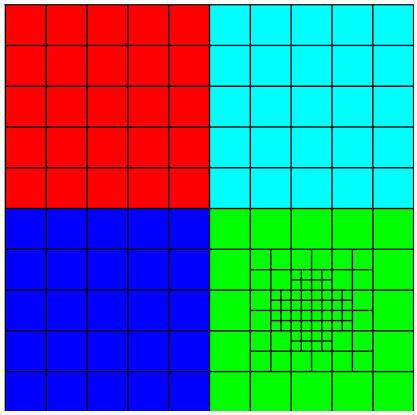

Рисунок 16: Сгущение сетки только на одном процессоре

4.1. Перераспределение ячеек.

Для балансировки сетки необходимо после каждой операции сгущения проводить *перераспределение ячеек*. В данной задаче нам помогут пакеты перераспределения данных (Partitions), о которых мы упоминали в пункте 2.1. Программный пакет INMOST предоставляет удобный функционал для подключения внешних пакетов, в нашем случае мы будем использовать пакет ParMETIS [3]. Вообще, процесс перераспределения ячеек в INMOST происходит следующим образом: сначала мы с помощью некоторого механизма «раскрашиваем» ячейки в «цвет» процессора, т.е. ставим каждой ячейке номер ее процессора, а потом вызываем специальную функцию Redistribute библиотеки INMOST, которая по заданным номерам производит обмен ячейками между процессорами. После данной операции мы получаем новые ячейки у каждого процессора.

Пакет ParMETIS используется на этапе раскрашивания ячеек. При распределении ячеек, можно выбрать один из трех режимов, раскраски ячеек:

- Partition;
- Repartition;
- Refine;

Partition — производит полное распределение сетки с нуля.
Repartition — производит перераспределение, учитывая предыдущее распределение.
Refine — совсем чуть-чуть перераспределяет ячейки.

Для наших целей мы будем пользоваться всеми тремя вариантами распределения сеток. Учитывая, что сгущение сетки следует за курсором, т.е. оно непрерывно, то после такого перемещения логично применять балансировку *Refine* или *Repartition*. При таком подходе долго удержать баланс ячеек не получится, поэтому после нескольких итераций, можно применить балансировку *Partition*, которая выровняет количество ячеек на процессорах, но потребует более длительного времени работы.

Однако, при такой балансировке, ParMETIS не учитывает специфику задачи; основная его цель — распределить ячейки по процессорам, по возможности сохранив их связность в рамках одного процессора. При таком подходе мы сталкиваемся с некоторыми проблемами, специфичными для сгущения типа octree.

4.2. Проблема согласованности структур данных

Как отмечалось ранее, в нашем примере для представления ячейки используется 2 структуры (рис. 17):

- Одна хранит информацию о структуре воьсмидерева (ссылки на дочерние ячейки, ссылки на родительскую ячейку, глубину ячейки и т.п.) и находится вне INMOST.
- Другая хранит ячейку в формате INMOST, содержит геометрическую информацию о ячейке и информацию о связях.

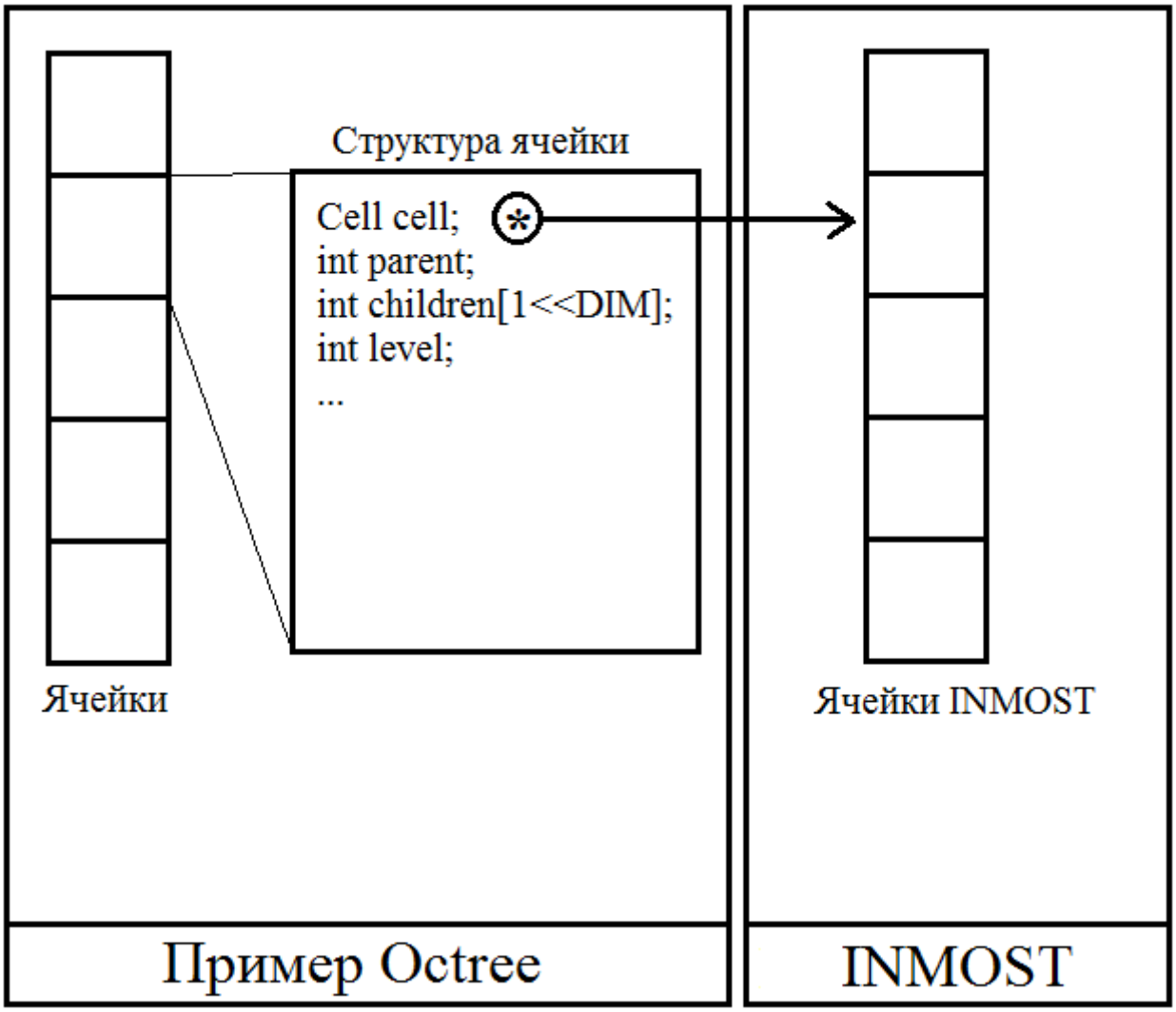

Рисунок 17: Двойная структура ячеек

Данный подход не является гибким и универсальным и для балансировки ячеек в нашей задачи не подходит. Проблема заключается в том, что при применении балансировки, ParMETIS использует только данные о ячейках, которые сохранены в структуре INMOST, поэтому после распределения, меняется структура сетки располагающаяся в INMOST. Структура, которую мы используем вне INMOST остается прежней. Происходит несоответствие ячеек (рис. 18-19). Для решения данной проблемы можно применить один из следующих подходов:

- После каждой балансировки делать синхронизацию двух структур.
- Отказаться от структуры вне INMOST и сохранить всю вспомогательную информацию в ячейках внутри INMOST.

В первом варианте к имеющимся действиям добавится еще одно трудоемкое действие, что скажется на скорости работы. Во втором варианте, вся информация о ячейках будет перераспределена вместе с ячейками INMOST. Второй вариант более универсальный, т.к. подходит не только для данной задачи, но и для других видов сгущения. Поэтому было решено остановиться на втором варианте.

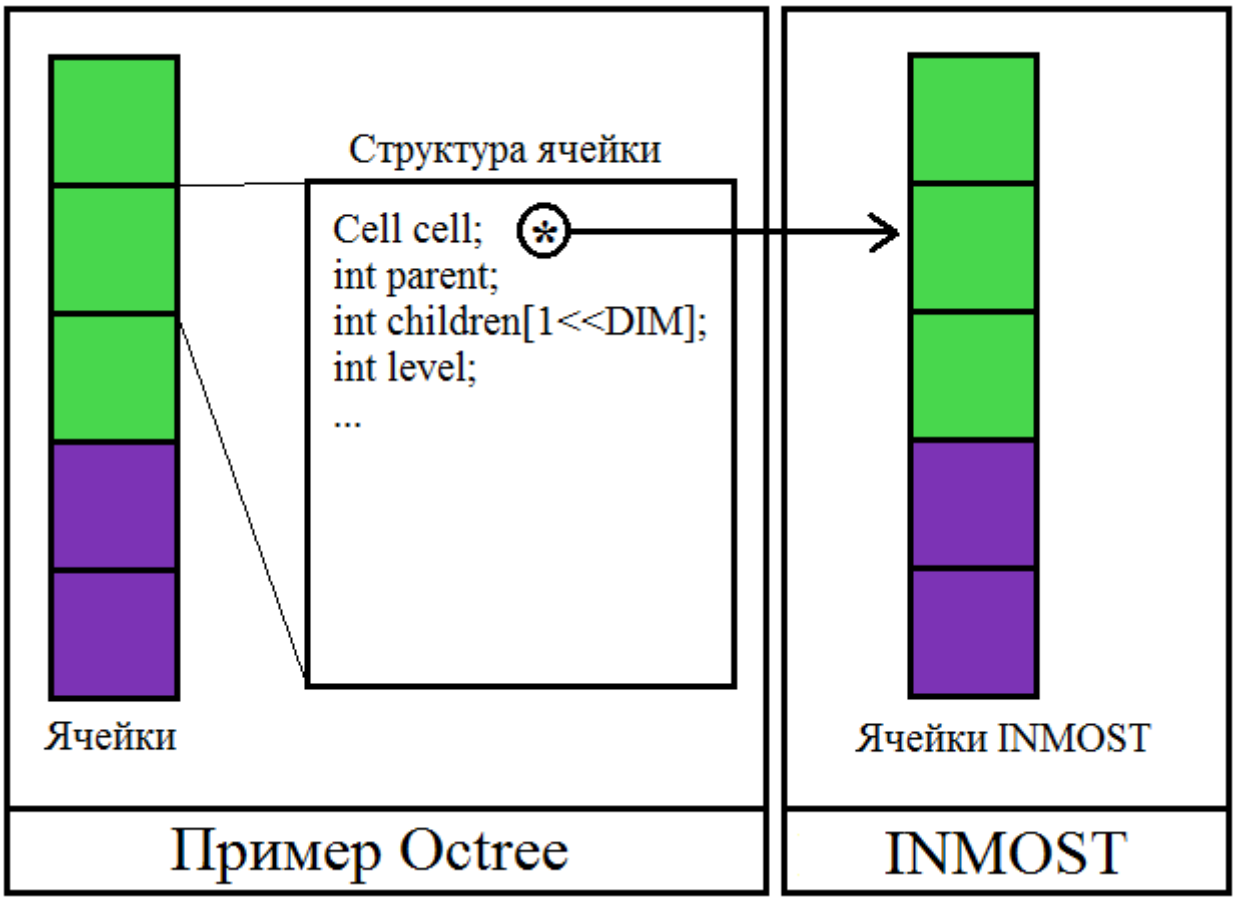

Рисунок 18: Структура распределенной сетки

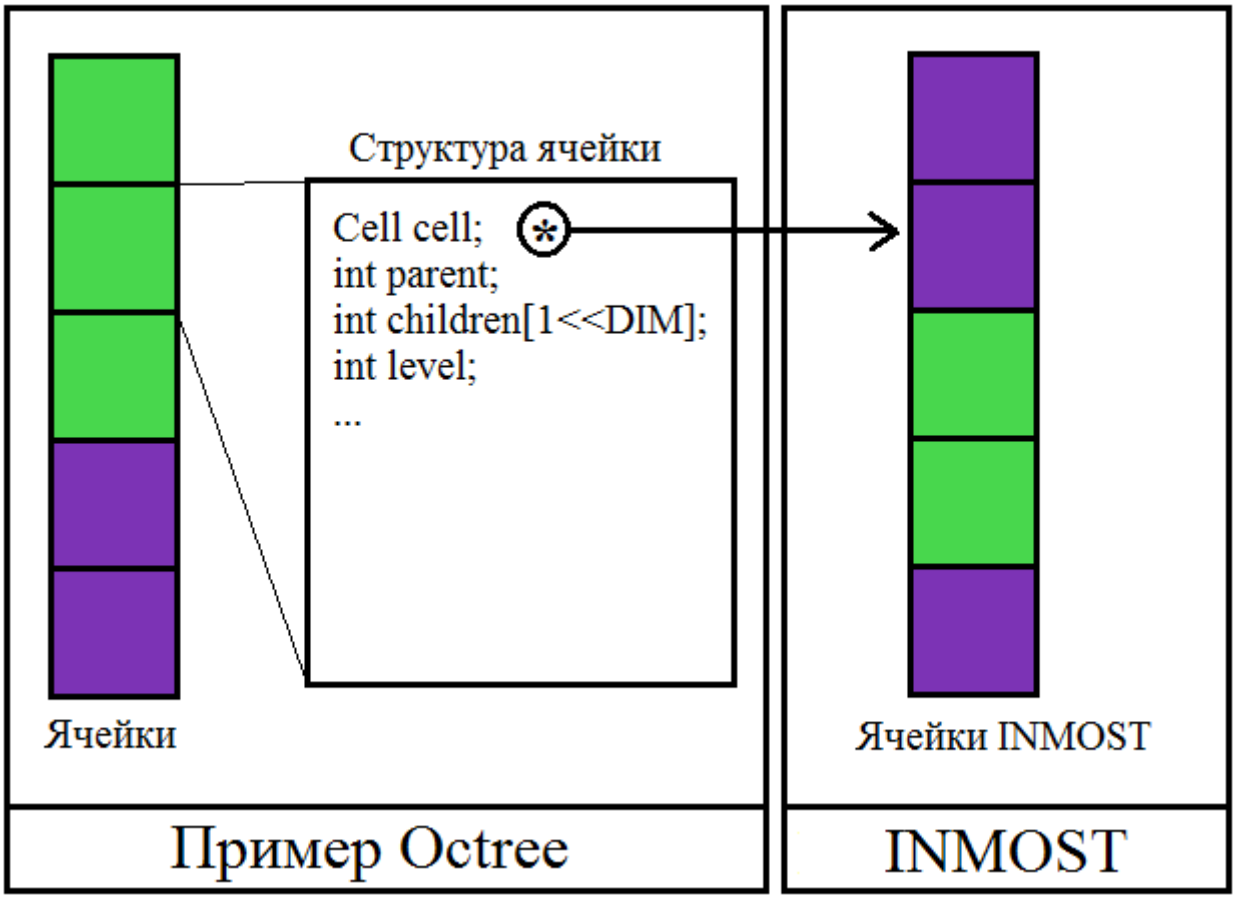

Рисунок 19: Структура распределенной сетки после балансировки

Перед нами стоит задача: сохранить всю информацию о ячейке внутри INMOST. Программная платформа позволяет связывать некоторые данные с ячейкой. Управления этими данными осуществляется с помощью механизма тегов. Тег (ярлык, атрибут) в INMOST — средство для доступа к данным в элементе. Тег имеет уникальное имя, содержит информацию о том где и как искать данные в элементе каждого типа (ячейке, грани, ребре, узле, множестве, сетке), содержит информацию о типе данных. Также в теге хранится информация о том, является ли тег «плотным», т.е. создан в каждом элементе данного типа, или «разреженным» т.е. принадлежит только некоторым элементам данного типа.

Таким образом мы можем создать теги: *parent*, *children*, *level* и остальные, и хранить их в ячейке INMOST. С программной точки зрения изменений пришлось делать много, т.к. полностью поменялась структура программы. Однако реализация такого подхода позволит нам не заботиться о синхронизации ячеек после балансировки.

4.3. Проблема разделения потомков

Как отмечалось ранее, балансировщику ParMETIS ничего не известно про структуру ячеек, которые он распределяет; его конечная цель — сбалансировать нагрузку на процессоры. В связи с этим в нашем примере может возникнуть проблемы с тем, что потомки одной ячейки попадут на разные процессоры (рис. 20). С точки зрения использованного алгоритма, проблема будет заключаться в том, что операция сгущения и разгрубления опирается на данные о родителе сгущаемых и разгрубляемых ячеек. При распределении дочерней и родительской ячеек на разные процессоры, информации о друг друге у них не будет и алгоритм не сможет правильно произвести сгущение и разгрубление.

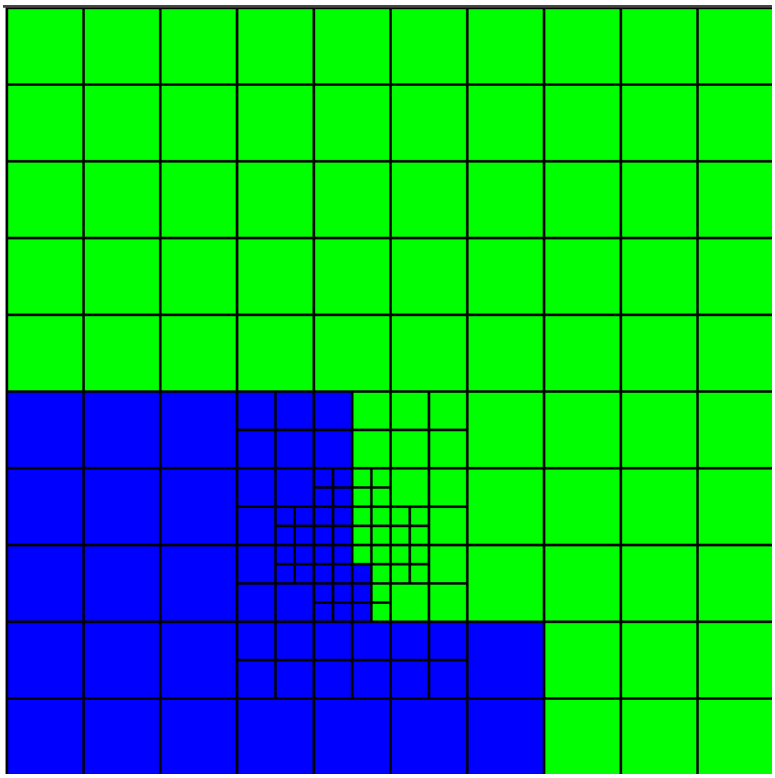

Рисунок 20: Применение балансировки ParMETIS::Partition

Для решения данной проблемы можно использовать следующие подходы:

- Изменить алгоритм сгущения сетки, чтобы он не зависел от информации о дочерних и родительских ячеек.
- Дополнить ячейки дополнительной информацией (связями), чтобы балансировщик не распределял дочерние ячейки по разным процессорам.

Мы будем использовать оба этих подхода. Во первых можно реорганизовать алгоритм, чтобы после перераспределения ячеек, процессоры сообщали друг другу недостающие дочерние ячейки (рис. 21).

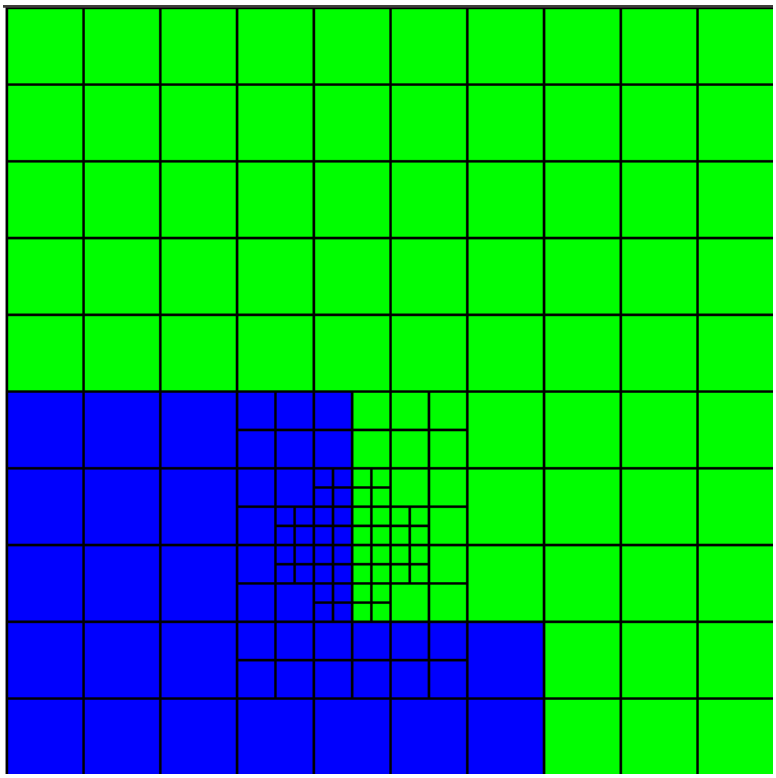

Рисунок 21: Корректировка дочерних ячеек

Во вторых можно задать связи между ячейками таким образом, чтобы при балансировке ParMETIS старался не разделять ячейки с сильными связями. Т.е. дочерние ячейки связать между собой сильной связью, а корневые ячейки связать слабой связью.

Таким образом будет решена проблема разделения потомков и алгоритм будет работать корректно.

5. Заключение

Решалась задача о построении распределенных динамических сеток, построенных на основе технологии octree.

В данной работе было сделано:

- Был рассмотрен и изучен последовательный пример динамических сгущающихся сеток реализованный с помощью платформы MSPP (предшественник платформы INMOST).
- Последовательный пример Octree был перенесен на платформу INMOST.
- Было реализовано статическое разбиение структуры данных по процессорам для параллельной реализации примера Octree.
- Было реализован механизм динамического сгущения распределенных сеток.
- Был разработан и реализован механизм отрисовки параллельной структуры сетки для примера Octree.
- Подключение балансировщика сеток ParMETIS к примеру Octree.
- Была разработана и реализована универсальная структура данных для реализации технологии octree на распределенных сетках. Теперь все структуры данных целиком хранятся в INMOST, что позволяет организовать обмен данных и перераспределение ячеек.
- Был разработан и реализован новый алгоритм сгущения сетки типа Octree, с использованием новой структуры данных.
- Организация согласованного перераспределения сетки с учетом связей parents/children.

Дальнейшее развитие работы:

- Совершенствование механизма перераспределение сеток Refine, в том числе дополнение его собственными алгоритмами.
- Включение этого примера Octree в платформу INMOST.

Список используемой литературы:

[1] INMOST — программная платформа и графическая среда для разработки параллельных численных моделей на сетках общего вида. Ю.В. Василевский, И.Н. Коньшин, Г.В. Копытов, К.Н. Терехов. Издательство МГУ, 2013, 144 с. http://www.inmost.org (2016) A toolkit for distributed mathematical modeling.

[2] MPI (Message passing interface). https://www.mpi-forum.org (2016).

[3] ParMETIS - Parallel Graph Partitioning and Fill-reducing Matrix Ordering. glaros.dtc.umn.edu/gkhome/metis/parmetis/overview (2016).

[4] Zoltan: Parallel Partitioning, Load Balancing and Data-Management Services. http://www.cs.sandia.gov/zoltan (2016).

[5] Paraview. http://www.paraview.org (2016).

[6] MSPP (Mesh and solver parallel platform).

[7] К.М. Терехов. Применение адаптивных сеток типа восьмеричное дерево для решения задач фильтрации и гидродинамики. Диссертация к.ф.-м.н. ИВМ РАН, Москва, 2013.