ПАРАЛЛЕЛЬНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМОВ ЧИСЛЕННОГО МОДЕЛИРОВАНИЯ

В.Э. Малышкин

Институт вычислительной математики
и математической геофизики СО РАН

Введение

Основная тема обсуждения – какими свойствами должны обладать алгоритмы и программы численного моделирования для их эффективного выполнения на мультикомпьютерах и какими способами нужные свойства могут быть обеспечены. Иными словами, обсуждаются вопросы параллельной реализации алгоритмов численного моделирования на мультикомпьютерах.

Проблема здесь в том, что нет единой теории параллельных вычислений, а раз нет теории, то нет и не может быть конкретных рецептов на все случаи жизни, как построить хорошо параллельно реализуемый алгоритм. Однако можно говорить о правильной точке зрения на алгоритм и на его параллельную реализацию.

1. Необходимые свойства параллельной программы

Начнем с обсуждения новых свойств параллельной программы (ПП). Эти свойства не требовались (или их отсутствие было не очень критичным) от последовательных программ, и в обеспечении этих свойств состоит одна из главных трудностей распараллеливания последовательных и создания параллельных программ. Любая хорошая ПП должна в большей или меньшей степени обладать рядом необходимых свойств общего характера [1].

- При исполнении параллельной программы на мультикомпьютере в каждый его процессорный элемент (ПЭ) загружается по крайней мере одна программа со своим набором входных данных для обработки и, таким образом, вся *параллельная программа представляется как множество взаимодействующих процессов*[*] .
- *Недетерминизм исполнения*. Последовательная программа тоже может быть представлена системой взаимодействующих процессов, но для нее это ненужная и даже вредная роскошь, так как ухудшается производительность программы. Для последовательного исполнения на множестве всех процессов

[*] *Процесс* – исполняемая программа + данные.

должен быть установлен линейный порядок, для каждого процесса точно указано, за кем он должен исполняться, а именно процессы должны выполняться в заданном линейном порядке. Поэтому последовательная программа, многократно исполняемая с одними и теми же данными, будет всегда вырабатывать один и тот же результат.

Для параллельного исполнения на множестве процессов устанавливается частичный порядок, и процессы исполняются, не нарушая этот порядок. При каждом конкретном исполнении ПП (с одними и теми же входными данными) ее процессы исполняются, вообще говоря, в разном порядке относительно друг друга, и в случае ошибки ПП может вырабатывать разные результаты при разных ее исполнениях. Поэтому локализация ошибок в ПП крайне затруднена. ПП может годами правильно работать, чтобы в самый «нужный» момент преподнести неприятный сюрприз.

Спешить избавляться от недетерминизма исполнения, однако, не следует, так как в ходе выполнения ПП многочисленные ресурсы мультикомпьютера используются разными процессами разных ПП или операционной системы, ресурсы освобождаются в разные непредсказуемые моменты времени. Если не допускать недетерминизма, то свободные ресурсы вычислителя не всегда могут быть немедленно использованы (например, процесс мог бы начать исполняться и загрузить свободный ресурс, но ему предписано ждать завершения некоторого другого процесса, в результате и ресурс простаивает и ПП медленнее исполняется) и, как следствие, производительность ПП может резко упасть. Некоторые другие важные свойства ПП также не могут быть обеспечены без недетерминизма исполнения.

• *Гибкая настройка на все доступные ресурсы мультикомпьютера*. Серьезная проблема – распределение ресурсов, включая время, мультикомпьютера так, чтобы обеспечить их равномерную загрузку (отображение прикладного алгоритма на ресурсы мультикомпьютера). При исполнении ПП не должно быть так, чтобы, например, все процессы, кроме одного, завершились, а этот один ещё долго бы работал. Работа ПП завершается, когда завершились все её процессы. В тех случаях, когда независимо исполняющиеся процессы требуют существенно разное количество ресурсов, хорошего распределения, как правило, нет. Да и количество необходимых ПП ресурсов чаще всего становится известным лишь в ходе исполнения программы, до начала вычислений этой информации обычно нет.

• *Динамическая балансировка загрузки*. В зависимости от входных данных процессы могут потреблять разное количество ресурсов (например, исполняться за разное время в зависимости от числа итераций внутреннего

цикла). В ходе исполнения ПП количество потребляемых процессами ПП ресурсов также может существенно изменяться (подробный пример приведен ниже). Во всех этих случаях нужна динамическая (в ходе исполнения) балансировка загрузки мультикомпьютера.

- *Переносимость в классе мультикомпьютеров*. Параллельная программа должна быть хорошо переносима в классе мультикомпьютеров, т.е. хорошо исполняться на всех мультикомпьютерах некоторого класса. Ведь даже на одном и том же мультикомпьютере число доступных ПЭ меняется от одного запуска ПП на счет до другого. А ПЭ больших мультикомпьютеров вообще разделяются между задачами, и при каждом старте ПП ей выделяется разное количество ПЭ. Конечно, ПП должна хорошо (если можно, оптимально) выполняться на всех тех ресурсах, что ей выделены при текущем исполнении.
- *Динамизм поведения*. При реализации численных моделей большого размера ПП должна уметь учитывать поведение модели и подстраиваться под него для достижения высокой производительности. Детальный пример рассматривается далее.

Все перечисленные свойства очень непросто реализуются, а динамические свойства исключительно сложны в отладке. Достичь перечисленных свойств при создании параллельной программы (распараллеливании последовательной) можно на пути создания мелко фрагментированных программ, чья фрагментированность сохраняется в ходе вычислений. Каждый фрагмент кода определяет отдельный процесс. При этом должны соблюдаться два сильных условия:

1) все процессы должны потреблять примерно одинаковое количество ресурсов;
2) на множестве всех процессов должен существовать такой частичный порядок, что все процессы взаимодействуют только со своими непосредственными соседями, т.е. процесс p_i может взаимодействовать с процессом p_j, если $p_i < p_j$, и не существует процесса p такого, что $p_i < p < p_j$.

Явные численные алгоритмы на прямоугольных сетках – хороший пример класса алгоритмов, допускающих такое представление и нередко хорошо реализуемых на мультикомпьютерах. Примерами фрагментов могут служить вычисления внутри одной ячейки сетки или вычисления внутри всех ячеек строки либо слоя ячеек. Исполнение системы параллельно выполняющихся и взаимодействующих процессов на ПЭ мультикомпьютера можно организовать так, что они будут вести себя в мультикомпьютере подобно жидкости в системе сообщающихся сосудов: если некоторый ПЭ окажется перегруженным, то лишняя работа (процессы) «переливается» в

соседние (связанные физическими линками) процессоры. Таким способом достигаются высокое качество динамического распределения ресурсов и гибкая настройка параллельной программы на все доступные ресурсы, а значит, и ее переносимость в классе мультикомпьютеров. Программа может вообще ничего не знать о структуре коммуникационной сети мультикомпьютера и будет очень устойчива (практически 100%-ная устойчивость) к разным сбоям оборудования.

2. Параллельная реализация PIC

2.1. Вычислительная схема реализации метода частиц-в-ячейках

При моделировании физического явления методом частиц-в-ячейках [2-5] исходное физическое пространство представляется в компьютере *пространством моделирования* (ПМ). В тех задачах, которые мы решали (моделирование разлета облака плазмы в замагниченном фоне, взаимодействие электромагнитного пучка с пластинкой), ПМ имеет форму прямоугольного параллелепипеда.

Плазма представляется множеством модельных частиц, характеризующихся координатами, скоростью и величиной заряда. Частицы распределены в пространстве моделирования в соответствии с некоторым законом, который изменяется во времени.

В отличие от прямого метода частица-частица, в котором рассматриваются все парные взаимодействия между частицами, в PIC методе частицы непосредственно друг с другом не взаимодействуют. Частицы влияют друг на друга, изменяя значения электромагнитных полей.

Для дискретизации значений электрического E и магнитного H полей все пространство моделирования разбивается прямоугольной сеткой на ячейки. В вершинах ячеек определяются значения электрического E и магнитного H полей. Чем меньше шаг сетки, то есть чем больше ячеек, тем точнее представление полей (рис.1).

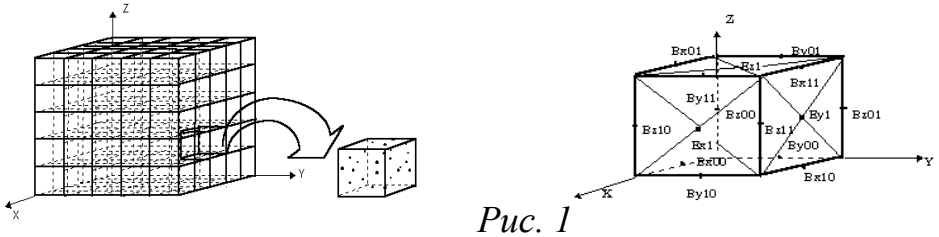

Рис. 1

С каждой ячейкой связаны частицы, находящиеся в этой ячейке в данный момент времени. Электромагнитные поля, заданные в вершинах

ячейки, действуют на частицы внутри нее, изменяя их скорости и координаты.

В свою очередь, “перелетевшие” в новую точку ПМ частицы влияют на значения электрического и магнитного полей в вершинах ячейки. Заряды частиц внутри ячейки интерполируются в вершины ячейки, таким образом вычисляются значения средней плотности тока и заряда в узлах сетки. Эти значения используются для пересчета значений электромагнитных полей. Назовем значения, заданные в узлах сетки, *сеточными значениями*.

Процесс моделирования состоит из серии временных шагов. На каждом временном шаге в каждой ячейке:

1) на основе значений электромагнитных полей вычисляется сила, действующая на частицу;
2) вычисляются новые скорости и координаты частиц;
3) из новых координат и скоростей частиц вычисляются значения средней плотности тока и заряда в узлах сетки;
4) из полученных значений средней плотности тока и заряда в узлах сетки пересчитываются значения электромагнитных полей.

Количество шагов моделирования определяется условиями физического эксперимента. Так как координаты частиц изменяются в процессе моделирования, частицы “перелетают” из ячейки в ячейку. Для устойчивости решения временной шаг выбирается столь малым, чтобы в течение одного шага частица улетала не далее соседней ячейки.

Заметим, что в первых трех стадиях вычисления ведутся в цикле по всем частицам. Время, затрачиваемое на выполнение этого цикла, составляет в зависимости от задачи от 70 до 90% всего счетного времени.

2.2. Особенности параллельной реализации метода частиц

Обычно задачи, решаемые с использованием метода частиц, характеризуются большим объемом данных и вычислений. На каждом временном шаге для вычисления новых скоростей и координат одной частицы требуется порядка ста арифметических операций. До того момента, когда можно наблюдать интересные физические результаты, требуется провести более 1000 временных шагов.

В алгоритме реализации PIC заложен внутренний параллелизм. Так как частицы между собой непосредственно не взаимодействуют, действия, связанные с перевычислением координат и скоростей частиц, можно проводить независимо друг от друга.

При решении задач на мультикомпьютерах возникает вопрос: каким образом распределить данные между ПЭ. Так как в каждый момент времени частица находится в некоторой ячейке, для уменьшения коммуникационных

издержек в процессе моделирования желательно так распределять данные, чтобы сеточные значения, заданные в вершинах ячейки, хранились в том же ПЭ, что и атрибуты частицы.

Чтобы удовлетворить этим условиям, можно

- либо дублировать массивы всех сеточных значений (во всех узлах сетки) в памяти каждого ПЭ. В этом случае частицы равномерно распределяются между ПЭ, вне зависимости от их расположения в ПМ. В этом случае, однако, на каждом временном шаге требуется обмениваться массивами сеточных значений между всеми ПЭ;
- либо применить пространственную декомпозицию – разрезать ПМ на подобласти по числу ПЭ. В этом случае массивы сеточных значений также разрезаются на части (с дублированием граничных строк и столбцов). Тогда в каждый ПЭ помещаются части массивов сеточных значений и те частицы, которые находятся в соответствующей подобласти ПМ. Пространство моделирования разрезается на подобласти таким образом, чтобы в каждом ПЭ находилось примерно одинаковое количество частиц. Таким образом, в начальный момент времени достигается балансировка загрузки ПЭ.

Однако в процессе моделирования частицы изменяют свое местоположение в ПМ, перелетая из ячейки в ячейку. Для того чтобы частицы по-прежнему находились в том же ПЭ, что и соответствующие ячейки, иногда приходится передавать атрибуты частицы из одного ПЭ в другой. Таким образом, в процессе моделирования может накопиться большой дисбаланс загрузки ПЭ – в некоторые ПЭ прилетит много больше частиц, чем в другие. Поэтому время от времени необходимо производить динамическую перебалансировку загрузки ПЭ.

Второй подход используется в наших параллельных реализациях метода частиц. Реализация основана на применении сборочного подхода к конструированию параллельной программы [6, 7], при котором определяется минимальный неделимый фрагмент вычислений, а вся программа собирается из этих атомарных фрагментов. Фрагментарная структура сборочной программы сохраняется и в исполняемом коде, что позволяет при необходимости передавать фрагменты на исполнение из одного ПЭ в другой. В нашем случае таким фрагментом является ячейка (вместе с частицами и сеточными значениями, необходимыми для вычисления силы, которая действует на частицу, а также со всеми вычислениями над этими данными). Поэтому частицы и сеточные переменные, заданные в вершинах ячейки, всегда оказываются в одном ПЭ.

3. Параллельная реализация PIC на линейке ПЭ

При реализации PIC на линейке ПЭ в качестве минимального фрагмента выбирается слой ячеек с тем, чтобы сохранить регулярность структур – хранить сеточные значения по-прежнему в массивах и производить вычисления в цикле над массивами. В начальный момент времени слои распределяются между ПЭ таким образом, чтобы в каждом ПЭ было примерно одинаковое количество частиц. Смежные слои назначаются в один или соседние ПЭ. Смежные слои, размещенные в одном ПЭ, образуют блок (рис.2).

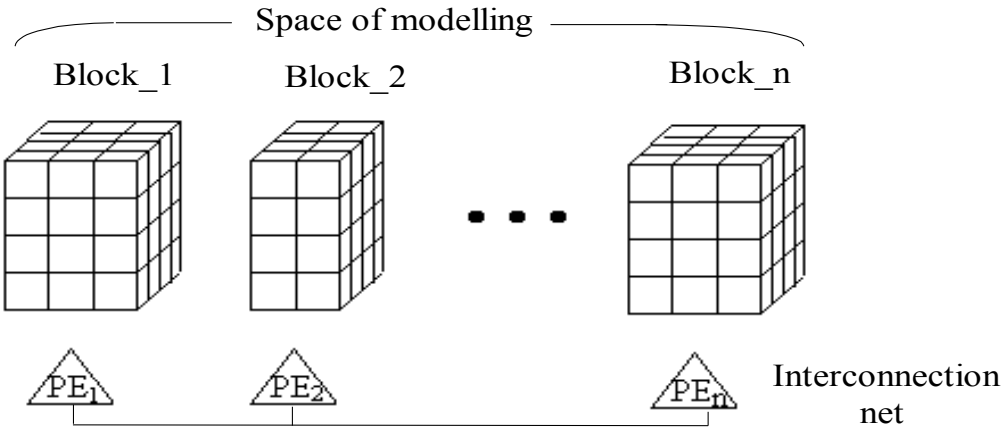

Рис. 2

Такая форма представления PIC называется *линеаризованной*.

4. Параллельная реализация PIC на решетке ПЭ

Линеаризованный PIC хорошо подходит для исполнения на 2-D решетке ПЭ. Отображение линеаризованного PIC на 2-D решетку ПЭ мультикомпьютера выполняется следующим способом.

Пусть дана двумерная решетка $l \times m$ процессорных элементов. Тогда **Block_i** назначается на обработку на i-ю строку ПЭ, $1 \le i \le l$. **Block_i** формируется таким образом, чтобы обеспечить равную нагрузку каждой строки ПЭ мультикомпьютера. Понятно, что это линеаризованное представление PIC. Затем **Block_i** делится на m подблоков **Block_ij**, j=1,...,m, которые назначаются на обработку на m ПЭ строки ПЭ. Каждый **Block_ij** формируется таким образом, чтобы обеспечить равную загрузку всех ПЭ строки (рис.3).

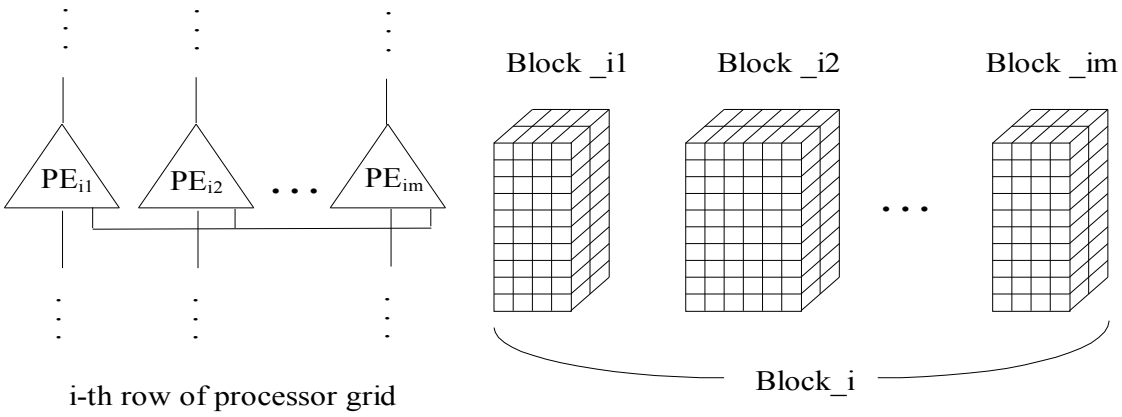

Рис. 3

Понятно, что в качестве минимального неделимого фрагмента вычислений здесь используется колонка ячеек пространства моделирования.

5. Балансировка загрузки ПЭ

Разработан эвристический алгоритм *начальной балансировки загрузки*. В каждом ПЭ строятся два массива:

- *A* содержит данные о количестве частиц в каждом слое ПМ (*A*[i] – количество частиц в *i*-м слое);
- в *S* заносятся данные о распределении слоев между ПЭ (*S*[*i*]) – номер первого слоя блока, размещенного в ПЭ$_i$).

Таким образом, каждый ПЭ "знает", какие слои назначаются в определенный ПЭ, это собственно *карта распределения нагрузки* между ПЭ.

Алгоритм имеет сложность *O(km)*, где *km* – количество слоев ПМ. Обозначим общее количество частиц как *NP*, число ПЭ – как *NPE*, среднее количество частиц в каждом ПЭ *ANP = NP/NPE*. Нулевой слой назначается в ПЭ$_0$ (нумерация ПЭ и слоев начинается с нуля). Далее в цикле по элементам массива *A*, слой *j*+1 назначается в ПЭ$_i$, если при этом количество частиц в первых *i*+1 ПЭ менее отличается от *(i+1)×ANP*, чем без частиц этого слоя. Этот эвристический алгоритм на всех проведенных тестах давал хорошие результаты.

Динамическая балансировка загрузки. Рассмотрим, как работает механизм перебалансировки загрузки. Каким-либо образом определяется порог разрешенного дисбаланса *BP* – максимальное количество частиц свыше среднего количества частиц, которое может обрабатываться одним ПЭ. Например, его можно вначале задать константой. После каждого шага по времени каждый ПЭ подсчитывает количество частиц в нем. Если это количество отличается от среднего количества частиц *ANP* больше, чем на заданный порог, ПЭ сигнализирует о необходимости проведения пере-

балансировки загрузки. Выделенный ПЭ собирает сведения о количестве частиц в каждом слое, записывает данные в уже описанный выше массив *A* и рассылает этот массив всем ПЭ. Далее, алгоритмом начальной балансировки загрузки каждый ПЭ строит массив *S*, в который заносятся данные о новом распределении слоев между ПЭ. После чего производится обмен слоями между соседними ПЭ – сначала (если необходимо) одновременно обмениваются слоями каждый $ПЭ_{2*i}$ и $ПЭ_{2*i+1}$, затем каждый $ПЭ_{2*i+1}$ и $ПЭ_{2*i+2}$.

Определение порога разрешенного дисбаланса BP. Были проведены эксперименты для *BP,* равного 0% (когда расписание перебалансировки строилось на каждом шаге), и 5% от общего числа частиц. При *BP*=5% были проведены 2 серии экспериментов. В первой серии новая карта распределения нагрузки строилась каждый раз, когда уровень дисбаланса превышал заданный порог. Во второй серии экспериментов новая карта строилась только тогда, когда увеличивалось максимальное количество частиц по сравнению с предыдущим шагом.

В двух других сериях экспериментов *BP* вычислялся динамически из времени, затраченного на выполнение перебалансировки, и времени, затрачиваемого на обработку одной частицы. Таким образом, можно было подсчитать, обработке какого количества частиц эквивалентно выполнение одной перебалансировки. Это значение не является точным, так как время, затрачиваемое на проведение одной перебалансировки, может значительно отличаться от времени, затрачиваемого на проведение другой перебалансировки, в зависимости от того, сколько слоев передается. Результаты тестирования показали, что алгоритм с динамически вычисляемым *BP* имеет лучшие характеристики. Это и понятно, так как этот алгоритм лучше “чувствует” дисбаланс загрузки.

В процессе моделирования возможна ситуация, когда большинство частиц собирается в одном слое, или в одном столбце ячеек, или даже в одной ячейке. Для организации вычислений в таком случае используется понятие *виртуального слоя*. Слой ПМ (вычисления и сеточные значения, заданные в вершинах ячеек слоя) представляется несколькими копиями (виртуальными слоями), а множество частиц исходного слоя распределяется между этими *виртуальными* слоями и, следовательно, между ПЭ, обеспечивая тем самым их равную нагрузку. Основной недостаток такого решения заключается в том, что появляется потребность в “длинных” взаимодействиях, так как исходно смежные слои могут теперь располагаться не в соседних ПЭ.

Благодаря введению понятия виртуального слоя в начальный момент времени достигается сбалансированная загрузка. Граничные слои дублируются в соседних ПЭ, частицы из этих слоев распределяются между ПЭ таким образом, чтобы в каждом ПЭ находилось равное количество частиц. То

есть алгоритм начальной балансировки загрузки видоизменен следующим образом. Слой 0, как и ранее, назначается в $ПЭ_0$. В цикле по элементам массива *A* слой *j* назначается не только в $ПЭ_i$, но и в $ПЭ_{i+1}$, если при добавлении частиц слоя *j* в $ПЭ_i$ количество частиц в первых *i* ПЭ становится большим чем $i \times ANP$. При накоплении дисбаланса в каждом ПЭ по массиву *A* строится новый массив *S*. Если новый массив *S* отличается от предыдущего, ПЭ обмениваются слоями. После чего распределяются частицы из граничных дублируемых слоев.

Централизованный алгоритм балансировки при реализации PIC на решетке ПЭ. Если при реализации PIC на мультикомпьютерах с большим количеством ПЭ использовать только линейные связи между ПЭ, то во время выполнения централизованных алгоритмов динамической балансировки загрузки требуются длинные коммуникации. При использовании топологии связи типа “решетка” радиус системы значительно уменьшается. Другим преимуществом реализации PIC на решетке ПЭ является тот факт, что уменьшается размер минимального неделимого фрагмента вычислений.

Динамическая перебалансировка загрузки происходит в два этапа. Сначала перераспределяются слои между строками ПЭ решетки. Используется описанный выше алгоритм динамической балансировки для линейки ПЭ. Затем новые блоки распределяются между ПЭ строк (рис.4).

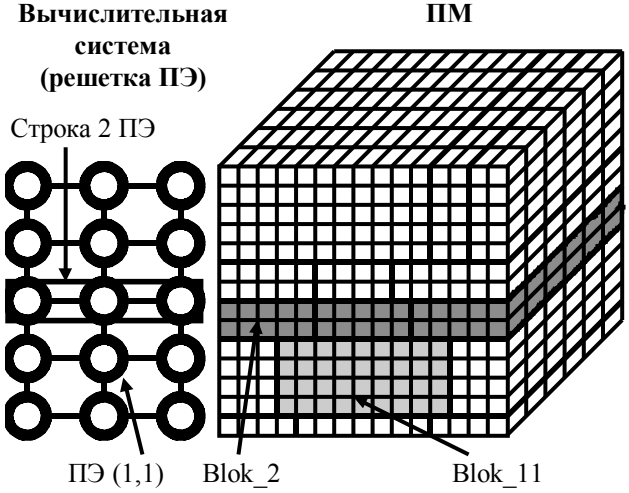

Рис. 4

Основной диффузионный алгоритм. В диффузионных алгоритмах решение о направлении передачи данных из более загруженных ПЭ в менее загруженные принимается локально. Цель – сбалансировать загрузку нескольких ПЭ в небольшой окрестности – локальной области. Все ПЭ мультикомпьютера покрываются такими локальными областями.

В основных диффузионных алгоритмах при выборе локальных областей должно выполняться условие, что нельзя разделить систему на части, не разделив при этом хотя бы одну локальную область (бесшовное покрытие). Балансировка загрузки в области происходит лишь на основе информации о текущей загрузке ПЭ области. Диффузионные алгоритмы отличаются размерами (и формами) локальных областей.

Пусть размер области ограничивается 2 ПЭ. Поведение системы во время принятия решения о направлении передачи данных синхронизировано. Принятие решения производится в несколько шагов. На четном шаге каждая пара ($ПЭ_{2i}$, $ПЭ_{2i+1}$) ($0 \le i < N/2$, где N – количество ПЭ) обменивается информацией о количестве частиц и принимает решение о передаче части частиц из более загруженного ПЭ в менее загруженный с целью сравнять загрузку. После чего каждый ПЭ имеет новую информацию о количестве частиц в нем (рис.5).

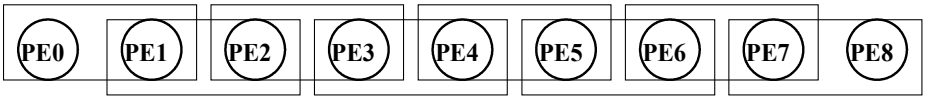

Рис. 5

На нечетном шаге каждая пара ($ПЭ_{2i+1}$, $ПЭ_{2i+2}$) ($0 \le i < (N-1)/2$, где N – количество ПЭ) обменивается информацией о количестве частиц и принимает решение о передаче части частиц из более загруженного ПЭ в менее загруженный с целью сравнять загрузку.

Такая диффузия может продолжаться в течение нескольких шагов, пока, наконец, данные не будут распределены равномерно. А в рассмотренном выше централизованном алгоритме перебалансировка производится за два шага.

Тестирование алгоритма показало, что хотя при увеличении количества шагов качество балансировки улучшается, общее время работы программы изменяется незначительно.

Заключение

В заключение можно сформулировать два рецепта:

- Строить мелко фрагментированные алгоритмы.
- Обеспечивать локальность взаимодействий между фрагментами, не допускать длинных транзитивных зависимостей, как в итеративных процессах.

Такое представление обеспечит хорошую параллельную реализацию алгоритмов. Конечно, совсем не обязательно всегда реализовать динамическую балансировку.

Список литературы

1. *Малышкин В.Э.* Параллельное программирование мультикомпьютеров: Учебное пособие. – Ярославль: Изд-во Ярослав. ун.-та, 1999. – 134 с.
2. *Березин Ю.А., Вшивков В.А.* Метод частиц в динамике разреженной плазмы. – Новосибирск: Наука, 1980. – 94 с.
3. *Вшивков В.А., Краева М.А., Малышкин В.Э.* Параллельная реализация метода частиц //Программирование. – 1997. – № 2. – С. 39-51
4. *Kraeva M.A., Malyshkin V.E.* Implementation of PIC Method on MIMD Multicomputers with Assembly Technology //The International Conference on High Performance Computing and Networking – HPCN Europe. – 1997. LNCS. – Vol.1225. – P. 541-549.
5. *Kraeva M.A., Malyshkin V.E.* Assembly Technology for Parallel Realization of Numerical Models on MIMD-Multicomputers //International Journal on Future Generation Computer Systems. – 2001. – Vol.17. – P. 755-765.
6. *Malyshkin V.E.* Functionality in ASSY System and Language of Functional Programming //Proceedings of the First International Symposium on Parallel Algorithms/Architectures Synthesis. 1995, Aizu, Japan. IEEE Comp. Soc. PressLos Alamitos, California. – 1995. – P. 92-97.
7. *Вальковский В.А., Малышкин В.Э.* Синтез параллельных программ и систем на вычислительных моделях. – Новосибирск: Наука, 1988. – 126 с.