3.3.6 **Декомпозиция, последовательная обработка интерфейсных граней**.

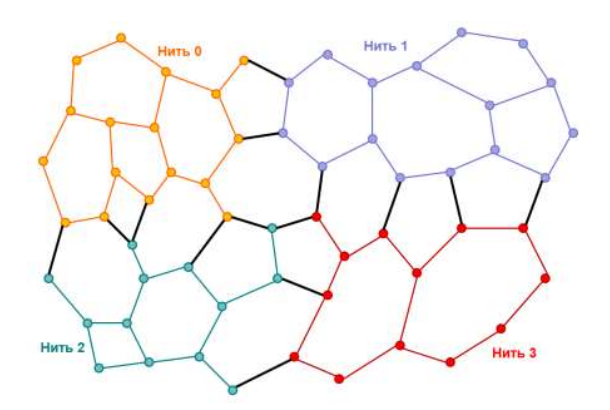
В разделе 3.3.6 мы рассмотрем процесс декомпозиции и последовательной обработки интерфейсных граней. Для начала, мы распределяем задачи между нитями статически, разделяя вершины графа между параллельными потоками, и присваиваем каждому потоку цвет для удобства отображения. Для этого нам нужно назначить цвета вершинам.

Этот процесс декомпозиции реализуется внешним алгоритмом или библиотекой, о которых подробно рассказано в разделе 4.3.1. Этот алгоритм распределит вершины графа таким образом, чтобы:

Разбиение было сбалансированным, с приблизительно равным числом вершин одного цвета.

Количество рёбер между вершинами разных цветов (так называемых интерфейсных рёбер) было минимальным.

Именно интерфейсные рёбра могут вызывать состояния гонки. Для наглядности, после раскраски вершин мы окрашиваем рёбра: рёбра, у которых обе вершины одного цвета, окрашиваются в этот цвет, а интерфейсные рёбра получают отдельный цвет (на изображении они обозначены черным цветом), который не соответствует ни одной из вершин. Пример показан на картинке.



Move to next slide.

A computer screen shot of text

Description automatically generated

Как и в предыдущем случае, мы переупорядочим все рёбра по цветам. В итоге у нас получается NC различных цветов рёбер. Значение NC равно NT+1, где NT - количество нитей (T - потоков). Для более удобного управления этими цветами мы создаем массив цветов C размером NC+1. В этом массиве C[k] содержит позицию начала k-го цвета в упорядоченном массиве всех рёбер E. Рёбра k-го цвета будут находиться в интервале от ie = C[k] до C[k+1]-1. Следовательно, в последнюю позицию C[NC] мы записываем общее количество рёбер NE.

У каждой нити есть свой цвет, что обеспечивает параллельное выполнение задач. Однако возникает вопрос о том, как обрабатывать интерфейсные рёбра. Для обработки интерфейсных рёбер мы выделяем одну нить, которая будет работать с ними в последовательном режиме.

Explain the code if possible.

Минусы: последовательная обработка интерфейса, деградация с ростом числа нитей. Если нитей мало, а ребер много, вес интерфейса будет небольшим, и мы получим приемлемое ускорение. Но когда нитей много, последовательная обработка интерфейса всё портит. Попробуем это исправить

Move to next slide.

A diagram of a molecule

Description automatically generated with medium confidence

3.3.7 **Декомпозиция, дублирование вычислений по интерфейсу**.  
Как уже было предложено ранее, мы рассматривали вариант с дублированием вычислений по всей сетке. Однако почему бы нам не рассмотреть возможность применить дублирование только для интерфейса? Это позволит нам увеличить скорость обработки данных по сравнению с последовательной обработкой, особенно при количестве потоков больше двух. В данном варианте потоки будут перебирать интерфейс, пропускать рёбра, если обе инцидентных вершины не принадлежат этому потоку, и выполнять вычисления, если хотя бы одна из вершин принадлежит этому потоку. Таким образом, каждое интерфейсное ребро будет обработано дважды.

Move to next slide.

A computer screen with text and numbers

Description automatically generated

Изменим только интерфейсный блок в коде как-то так:

Теперь как это будет в коде. Пусть цвета вершин хранятся в массиве P (Partition).

A computer screen with text and numbers

Description automatically generated

Стало лучше, интерфейс теперь делается параллельно, но...

Минусы: в два раза больше вычислений по интерфейсу, деградация (в меньшей степени) с ростом числа нитей.

Move to next slide.

**3.3.8 Многоуровневая декомпозиция**

Мы уже показали, что можем эффективно выполнять операции в несколько тактов через барьеры, и этот подход применим и здесь.

Move to next slide.

A diagram of a molecule

Description automatically generated with medium confidence

После того как мы покрасили и переупорядочили рёбра, остаётся набор интерфейсных рёбер в конце массива. Но в чём их недостаток по сравнению с исходным графом? Фактически, ни в чём! Мы можем взять этот набор интерфейсных рёбер, убрать их из графа,

Move to next slide

A diagram of a molecule

Description automatically generated with medium confidence

взять вершины, инцидентные этим рёбрам, и создать новый граф.

Move to next slide

Move to next slide

A diagram of a molecule

Description automatically generated with medium confidence

Затем мы можем применить к этому новому графу ту же самую процедуру декомпозиции, повторяя то же, что делали с исходным графом. Мы продолжаем эту процедуру до тех пор, пока интерфейс не сократится до нескольких рёбер, которые можно обработать одним потоком. В результате получим несколько уровней декомпозиции. Таким образом, в параллельной реализации добавляется внешний цикл, который управляет уровнями. У каждого уровня будет свой набор рёбер, упорядоченных по цветам, что соответствует нитям.

Move to next slide

A computer screen with text

Description automatically generated

Для уровней сделаем массив указателей на их массивы диапазонов цветов, назовем его

L (Levels), а число получившихся уровней – NL. Получим следующий код.A computer screen with text

Description automatically generated

Минусы: немножко сложненько получается с этой многоуровневой декомпозицией. А так одни плюсы. С числом нитей не деградирует, последовательных частей не имеет, данные компактны в памяти, фиксированное распределение по нитям – легко делать NUMA placement.

Move to next slide.

A screenshot of a computer

Description automatically generated

Теперь давайте кратко рассмотрим те пункты, которые мы обсудили сегодня.

1. Полное дублирование вычислений:

Для устранения зависимостей по данным иногда приходится полностью дублировать вычисления. Это означает, что одни и те же вычисления выполняются несколько раз для разных частей задачи. Это может быть неэффективным, но иногда необходимо для обеспечения параллельности.

2. Атомики:

Атомарные операции - это специальные операции, которые гарантируют, что только один поток выполнения может выполнять их в данный момент. Они используются для предотвращения конфликтов при обновлении общих данных в параллельных вычислениях.

3. Разделение на две операции:

Иногда зависимости по данным можно разделить на две операции: чтение и запись. Это позволяет выполнять чтение данных параллельно, а затем собирать результаты и записывать их последовательно.

4. Раскраска графа по ребрам:

Раскраска графа по ребрам - это метод определения зависимостей по данным в графе вычислений. Различные цвета могут обозначать разные уровни зависимостей, и это помогает определить, какие вычисления можно выполнять параллельно.

5. Декомпозиция, последовательная обработка интерфейсных граней:

Декомпозиция - это разделение большой задачи на более мелкие подзадачи. Последовательная обработка интерфейсных граней означает, что грани между подзадачами обрабатываются последовательно, чтобы избежать конфликтов при доступе к данным.

6. Декомпозиция, дублирование вычислений по интерфейсу:

При декомпозиции задачи на подзадачи иногда приходится дублировать некоторые вычисления на интерфейсе между подзадачами. Это сделано для избежания конфликтов при обмене данными между подзадачами.

7. Многоуровневая декомпозиция:

Многоуровневая декомпозиция - это метод разделения задачи на более мелкие задачи, которые могут быть декомпозированы на ещё более мелкие подзадачи. Этот подход может увеличить уровень параллелизма в вычислениях.

Все эти методы и подходы используются для устранения зависимостей по данным и обеспечения эффективной параллельной обработки в сеточных методах параллельных вычислений.