7.3. Лабораторная работа «Игра "Жизнь" Дж. Конвея»

Работа предназначена для студентов 2 курса ФИТ НГУ
Разработал: к.т.н. Ю.Г. Медведев
medvedev@ssd.sscc.ru

Цель работы: практическое освоение методов реализации алгоритмов мелкозернистого параллелизма на крупноблочном параллельном вычислительном устройстве на примере реализации клеточного автомата «Игра "Жизнь" Дж. Конвея» с использованием неблокирующих коммуникаций библиотеки MPI.

Основные понятия и последовательная программная реализация

Топология клеточного массива представляет собой двумерную квадратную решетку с окрестностью Мура порядка 1, т.е. каждая клетка-квадрат имеет восемь соседних клеток (рис. 1). Клеточный автомат работает в синхронном режиме, т.е. состояние каждой клетки на каждой итерации обновляется одномоментно, по сигналу синхрогенератора. Состояние клетки булево, т.е. может принимать значение 1 (живая клетка) или 0 (мертвая клетка).

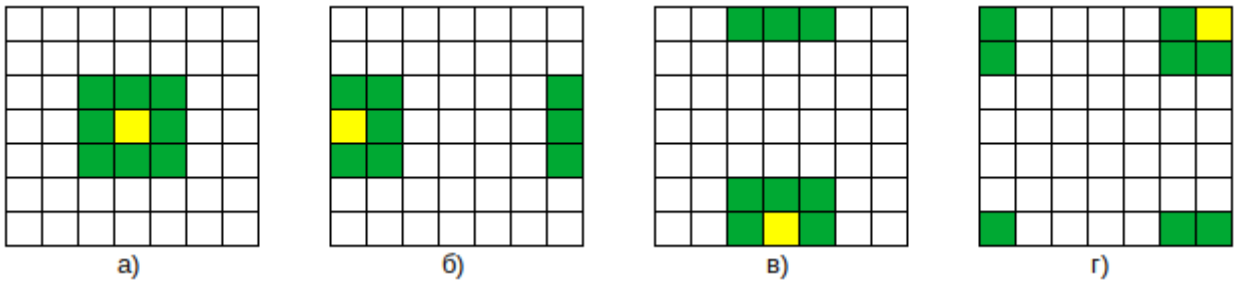

Рис. 1. Окрестность Мура для квадратного двумерного клеточного массива.
Желтым цветом обозначена некоторая клетка, зеленым – ее восемь соседей.
а) внутренние клетки, б) левая граница, в) нижняя граница, г) правый верхний угол.

Функция переходов клетки следующая. Живая клетка умирает (из состояния 1 переходит в состояние 0) от недостатка или от избытка социальных связей, если в окрестности ее соседства менее двух или более трех живых клеток соответственно (рис. 2б). Мертвая клетка оживает (из состояния 0 переходит в состояние 1), если ровно три клетки в ее окрестности живы. В остальных случаях состояние клетки не меняется. На рис. 2 приведен пример эволюции клеточного автомата с начальным состоянием в виде парусника (глайдера), который каждую четвертую итерацию приходит в исходную форму, смещенную на одну клетку по диагонали вправо-вниз.

В теории клеточный массив бесконечен. В практической реализации он имеет конечное количество строк и столбцов. Граничные условия могут быть двух типов. Первый – ограниченные границы – клетки первого и последнего столбца, как и клетки первой и последней строки всегда остаются мертвыми. Второй – периодические границы – левыми соседями клеток первого столбца являются

соответствующие клетки последнего столбца, а правыми соседями клеток последнего столбца – клетки первого столбца (рис. 1б). Так же определяются верхние и нижние соседи первой и последней строк (рис .1в). В угловых клетках оба эти правила сочетаются (рис. 1г). Такой способ замыкает границы клеточного массива в тор и не требует назначения особых функций переходов граничным клеткам, т.к. у каждой из них определено по 8 соседей, как и у всех остальных клеток.

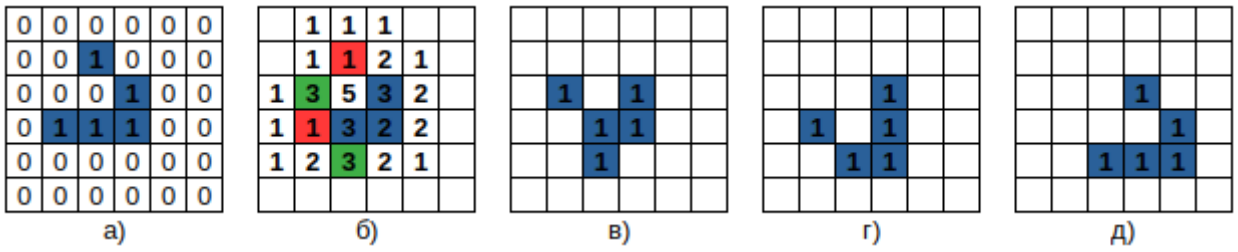

Рис. 2. Функция переходов клеточного автомата. Синим цветом обозначены живые клетки, белым – мертвые, зеленым – рождающиеся, красным – умирающие.

а) исходное состояние, 1 – живые клетки, 0 – мертвые, б) количество живых соседей в исходном состоянии, в) результат первой итерации, г) результат двух итераций, д) результат четырех итераций.

Начальная конфигурация задается произвольно путем назначения состояния 0 или 1 каждой клетке массива. Т.к. на последовательном или крупноблочном параллельном вычислителе невозможно вычислить состояния всех клеток одномоментно, используют второй, вспомогательный экземпляр клеточного массива для последовательной записи получившихся значений (рис. 3а), чтобы старое значение состояния клетки не затиралось и могло быть использовано для вычисления состояния соседних клеток до конца текущей итерации (рис. 3б). По завершении каждой итерации получившиеся состояния клеток вспомогательного массива будут содержать новые значения. На следующей итерации массивы меняются ролями. В этом случае игра завершается после выполнения наперед заданного количества итераций.

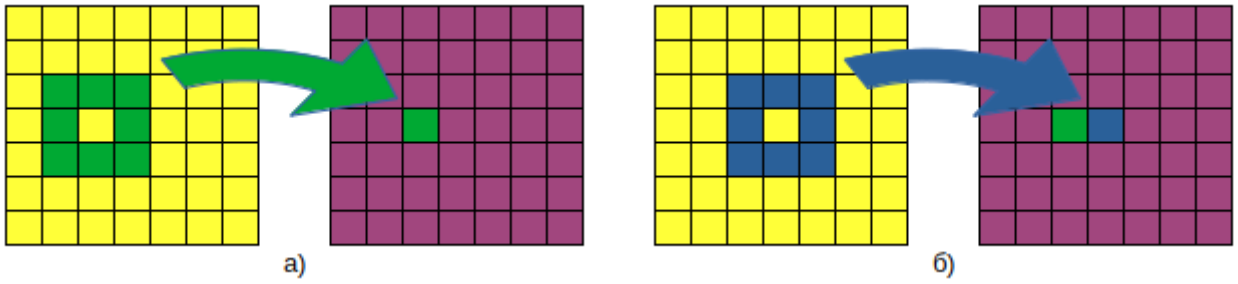

Рис. 3. Реализация синхронного автомата с помощью вспомогательного массива. Желтым цветом обозначен основной массив, фиолетовым – вспомогательный. а) из состояний соседей клетки в основном массиве вычисляется и записывается во вспомогательный массив новое состояние этой клетки, б) то же повторяется для соседней клетки с использованием старого состояния предыдущей клетки.

Технически более сложным критерием завершения игры является повтор состояния клеточного массива, т.е. совпадение состояний всех его клеток со своим состоянием на некоторой предыдущей итерации. В этом случае необходимо запоминать эволюцию клеточного массива, т. е. состояние клеточного массива на всех прошедших итерациях, поэтому вспомогательный массив на каждой последующей итерации становится основным, а новый вспомогательный массив располагается в новой области памяти. После каждой итерации проводится проверка на совпадение состояния массива с состоянием на каждой предыдущей итерации. Этот процесс требует много вычислительных ресурсов, поэтому целесообразно его распараллелить.

Схема параллельной программной реализации

Для реализации клеточного автомата на многомашинном вычислителе (кластере) используется метод декомпозиции клеточного массива. Этот метод также будет эффективен и на архитектурах с общей памятью. Каждому вычислительному ядру назначается свой участок массива, разрезанного, например, на прямоугольные элементы вдоль строк (рис. 4а). Для хранения соседей верхней и нижней строки, приходящихся на участок массива, принадлежащего соседнему ядру, используется пара дополнительных строк – по одной копии сверху и снизу этого участка (рис. 4б). На каждой итерации дополнительные строки обновляются, они заполняются актуальными значениями состояний клеток, полученными от соседних ядер путем приема-передачи сообщений средствами библиотеки MPI. При замкнутых в тор границах первое и последнее ядро также будут соседними.

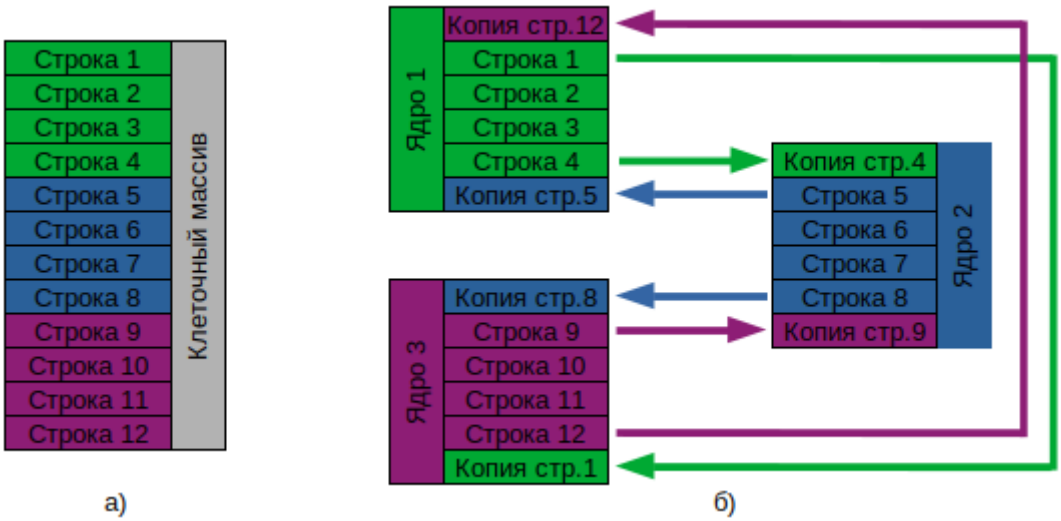

Рис. 4. Распределение клеточного массива по процессорным ядрам.
Цветами обозначены процессорные ядра
а) распределение строк в исходном массиве, б) закрепленные за ядрами участки массива, дополнительные строки и схема пересылки границ

Программа должна корректно работать, когда заданное количество строк массива не делится нацело на число используемых процессорных ядер. В этом случае количество строк массива, приходящихся на каждое ядро, не должно отличаться друг от друга более чем на одну строку. Граничные строки вычисляются, например,

делением общего количества строк на количество ядер и округлением к большему целому (рис. 5). Количество ядер, допустимое к использованию на клеточном массиве заданного размера, не должно превышать половину от количества его строк, т.к. на каждое ядро должно приходиться не менее двух строк массива.

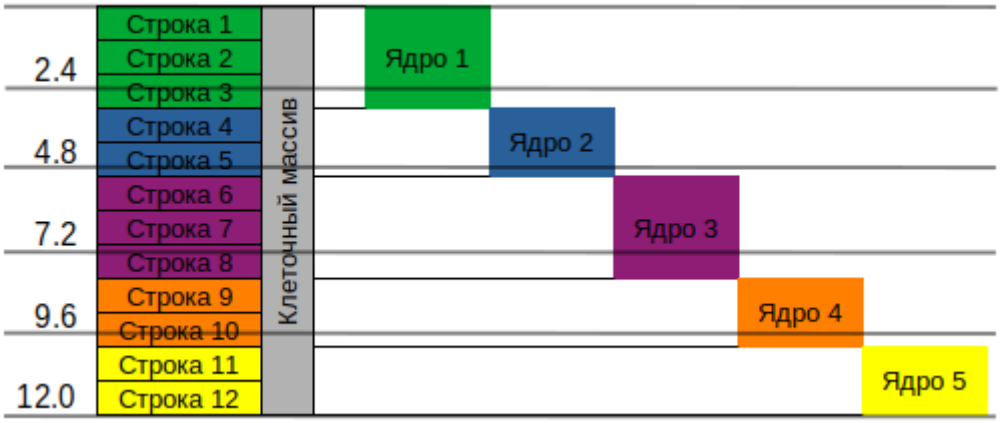

Рис. 5. Округление границ участков массива, приходящихся на процессорные ядра: 12 строк делятся на 5 ядер, на одно ядро в среднем приходится по 2.4 строки, округление границы производится к большему целому.

Завершение программы происходит, если текущее состояние всех клеток массива совпадает с их состоянием на какой-либо из предыдущих итераций. В параллельной версии это условие должно выполняться на каждом процессорном ядре для одной и той же предыдущей итерации. Для того, чтобы это проверить, после каждой итерации каждое ядро должно сообщить всем остальным, на каких итерациях это условие выполняется для его участка массива. Для этого каждое ядро вычисляет булев вектор флагов останова длиной на единицу меньше номера текущей итерации. Компонент этого вектора равен 1, если текущее состояние участка массива этого ядра совпадает с состоянием на соответствующей этому компоненту предыдущей итерации, и равен нулю – в противном случае. После того, как все ядра произведут обмен этими векторами флагов, каждое ядро сможет однозначно определить, нужно ли завершать программу. Останов происходит, когда один и тот же компонент вектора флагов равен 1 во всех присланных векторах.

Для того, чтобы сократить время работы программы, можно одновременно с пересылкой между соседними процессорными ядрами состояний граничных строк их участков массива вычислять на каждом ядре состояния внутренних (не граничных) строк, не требующих пересылаемой с соседних ядер информации. В таком случае следует использовать неблокирующие функции MPI_Isend и MPI_Irecv. Состояние флага останова следует передавать с помощью функции MPI_Ialltoall. Ожидание приема неблокирующих сообщений осуществляется функцией MPI_Wait.

```
int MPI_Isend(const void *buf, int count, MPI_Datatype datatype, int dest,
        int tag, MPI_Comm comm, MPI_Request *request)
Параметры:
buf       - адрес буфера, из которого передавать,
```

```
count     - количество передаваемых элементов,
datatype  - тип передаваемых элементов,
dest      - кому передавать,
tag       - идентификатор сообщения,
comm      - идентификатор группы,
request   - адрес идентификатора передачи для MPI_Wait.

int MPI_Irecv(void *buf, int count, MPI_Datatype datatype, int source, int tag,
        MPI_Comm comm, MPI_Request *request)
Параметры:
buf       - адрес буфера, куда принимать,
count     - количество принимаемых элементов,
datatype  - тип принимаемых элементов,
source    - от кого принимать,
tag       - идентификатор сообщения,
comm      - идентификатор группы,
request   - адрес идентификатора приема для MPI_Wait.

int MPI_Ialltoall(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
        void *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm,
        MPI_Request *request)
Параметры:
sendbuf   - адрес буфера, из которого передавать,
sendcount - количество передаваемых элементов,
sendtype  - тип передаваемых элементов,
recvbuf   - адрес буфера, куда принимать,
recvcount - количество принимаемых от каждого процесса элементов,
recvtype  - тип принимаемых элементов,
comm      - идентификатор группы,
request   - адрес идентификатора приема для MPI_Wait.

int MPI_Wait(MPI_Request *request, MPI_Status *status)
Параметры:
request   - адрес идентификатора приема/передачи,
status    - адрес структуры параметров сообщения.
```

Последовательность действий на каждой итерации следующая.

1. Инициализировать отправку первой строки предыдущему ядру, используя MPI_Isend. Запомнить параметр request для шага 8.
2. Инициализировать отправку последней строки последующему ядру, используя MPI_Isend. Запомнить параметр request для шага 11.
3. Инициировать получение от предыдущего ядра его последней строки, используя MPI_Irecv. Запомнить параметр request для шага 9.
4. Инициировать получение от последующего ядра его первой строки, используя MPI_Irecv. Запомнить параметр request для шага 12.
5. Вычислить вектор флагов останова.
6. Инициализировать обмен векторами флагов останова со всеми ядрами, используя MPI_Ialltoall. Запомнить параметр request для шага 14.
7. Вычислить состояния клеток в строках, кроме первой и последней.
8. Дождаться освобождения буфера отправки первой строки предыдущему ядру, используя MPI_Wait и сохраненный на шаге 1 параметр request.

9. Дождаться получения от предыдущего ядра его последней строки, используя MPI_Wait и сохраненный на шаге 3 параметр request.
10. Вычислить состояния клеток в первой строке.
11. Дождаться освобождения буфера отправки последней строки последующему ядру, используя MPI_Wait и сохраненный на шаге 2 параметр request.
12. Дождаться получения от последующего ядра его первой строки, используя MPI_Wait и сохраненный на шаге 4 параметр request.
13. Вычислить состояния клеток в последней строке.
14. Дождаться завершения обмена векторами флагов останова со всеми ядрами, используя MPI_Wait и сохраненный на шаге 6 параметр request.
15. Сравнить вектора флагов останова, полученные от всех ядер. Если для какой-то итерации все флаги равны 1, завершить выполнение программы.

Исходные данные задачи

Клеточный массив размером X × Y клеток (не менее 100 × 100) заполнен нулями. Единицами инициализированы пять клеток (1,2), (2,3), (3,1), (3,2) и (3,3), согласно рис. 6. Такая конфигурация с периодом в 4 итерации воспроизводит саму себя со смещением на одну клетку по диагонали вправо-вниз и называется парусником (глайдером).

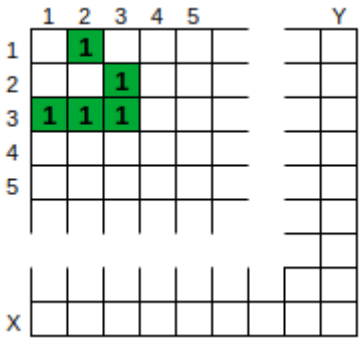

Рис. 6. Исходное заполнение клеточного массива

Задание к лабораторной работе

1. Написать параллельную программу на языке C/C++ с использованием MPI, реализующую клеточный автомат игры "Жизнь" с завершением программы по повтору состояния клеточного массива в случае одномерной декомпозиции массива по строкам и с циклическими границами массива. Проверить корректность исполнения алгоритма на различном числе процессорных ядер и различных размерах клеточного массива, сравнив с результатами, полученными для исходных данных вручную.

2. Измерить время работы программы при использовании различного числа процессорных ядер: 1, 2, 4, 8, 16, … . Размеры клеточного массива X и Y подобрать таким образом, чтобы решение задачи на одном ядре занимало не менее 30 секунд. Построить графики зависимости времени работы, ускорения и эффективности распараллеливания от числа используемых ядер.

3. Произвести профилирование программы и выполнить ее оптимизацию. Попытаться достичь 50-процентной эффективности параллельной реализации на 16 ядрах для выбранных X и Y.