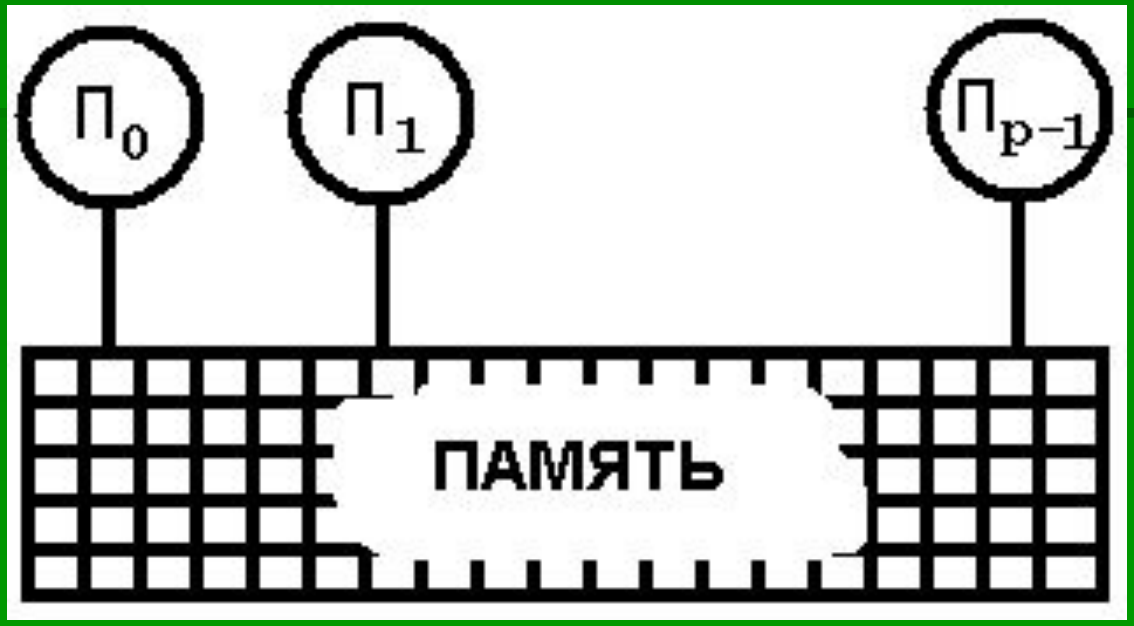
Построение и анализ параллельных алгоритмов

PRAM: модель параллельных вычислений с общей памятью

Модель PRAM

- Модель PRAM: Parallel Random-Access Memory
- Позволяет учитывать ограничения, связанные с одновременным доступом к памяти
- Является идеализированной моделью архитектуры SMP (Symmetric MultiProcessor, Shared Memory Processor)

Модель **PRAM**



Процессоры П0, П1, …, Пр–1 используют общую память, состоящую из множества ячеек.

Время доступа каждого процессора к каждой ячейке памяти одинаково и не зависит от количества процессоров.

Модель PRAM

Один шаг (такт) работы PRAM-машины синхронизирован по фазам:

1. Чтение данных из памяти.
2. Обработка данных.
3. Запись результата в память.

Режимы чтения и записи

- Режимы чтения данных из памяти:
  - Одновременное (Concurrent Read)
  - Исключающее (Exclusive Read)
- Режимы записи в память:
  - Одновременная (Concurrent Write)
  - Исключающая (Exclusive Write)

Варианты одновременной записи

- Одновременная запись одинакового значения
- Произвольная запись: сохраняется произвольное значение из множества записываемых
- Запись в зависимости от приоритетов процессоров
- Комбинация записываемых значений

Виды **PRAM** машин и алгоритмов

EREW (Exclusive Read, Exclusive Write): исключающее чтение и исключающая запись	**CREW (Concurrent Read, Exclusive Write): одновременное чтение и исключающая запись**
ERCW (Exclusive Read, Concurrent Write): исключающее чтение и одновременная запись	CRCW (Concurrent Read, Concurrent Write): одновременной чтение и одновременная запись

Пример CREW-алгоритма

ЗАДАЧА НАХОЖДЕНИЯ КОРНЕЙ ДВОИЧНОГО ЛЕСА

Пример CREW-алгоритма

Дано: Лес, состоящий из бинарных деревьев. Деревья заданы следующим образом: для каждой вершины имеется указатель на её родителя. Для корней деревьев этот указатель пуст.

Требуется: для каждой вершины найти корень дерева, которому она принадлежит

Пример CREW-алгоритма

Представление входных данных:

- вершины пронумерованы,
- ребра деревьев заданы с помощью массива *parent*: элемент *parent*[*i*] представляет номер вершины, являющейся родителем для вершины с номером *i*.

Пример CREW-алгоритма

Результат работы алгоритма — массив *root*. В ячейке *root*[*i*] хранится вершины, являющейся корнем дерева, в которое входит вершина *i*.

Массивы *parent* и *root* хранятся в общей памяти.

CREW-алгоритм

1. Для каждого процессора P_i выполнить
2. Если *parent*[*i*] = NIL, то *root*[*i*] := *i*;
3. Пока существует узел *i*, для которого *parent*[*i*] ≠ NIL, выполнять:
4. Для каждого процессора *i* выполнить
5. Если *parent*[*i*] ≠ NIL, то
6. {
7. *root*[*i*] := *root*[*parent*[*i*]];
8. *parent*[*i*] := *parent*[*parent*[*i*]];
9. }

Анализ CREW-алгоритма

Временная сложность алгоритма:

$$O(\log 2\ d),$$

где d — наибольшая глубина дерева в заданном лесе.

Можно показать, что ни один EREW-алгоритм не может решить эту задачу за время, меньшее $O(\log 2\ n)$, где n — количество вершин в лесе

Пример CRCW-алгоритма

НАХОЖДЕНИЕ МАКСИМАЛЬНОГО ЭЛЕМЕНТА В МАССИВЕ

Пример CRCW-алгоритма

Дано: Массив *n* элементов

Требуется: Найти максимальный элемент

Пример CRCW-алгоритма

Способ решения

- Количество процессоров: n^2.
- Каждый процессор нумеруется парой индексов.
- Процессор с номером (*i*,*j*) сравнивает A[*i*] и A[*j*].
- Используется вспомогательный булевский массив *m*[*i*]. После выполнения сравнений *m*[*i*]=*true* ⇔ A[*i*] — наибольший элемент массива.
- Результат помещается в переменную *max*.

CRCW-алгоритм

1. Для всех *i* от 0 до *n*–1 выполнить:

   m[*i*] := *true*;
2. Для всех *i* от 0 до *n*–1 и для всех *j* от 0 до *n*–1 выполнить:
3. Если A[*i*] < A[*j*], то *m*[*i*] := *false*;
4. Для всех *i* от 0 до *n*–1 выполнить:
5. Если *m*[*i*] = *true*, то *max* := A[*i*];
6. Вернуть *max*.

Анализ CRCW-алгоритма

- Без использования параллельного чтения невозможно решить эту же задачу быстрее, чем за время O(*log n*).
- Представленный CRCW-алгоритм работает за время O(1) и требует n^2 процессоров. Наилучший последовательный алгоритм работает за время O(*n*). Поэтому эффективность составляет 1/*n*, т.е. алгоритм не является эффективным по затратам.

Рекомендуемые источники

- Адигеев М.Г. Анализ сложности параллельных алгоритмов. Метод. указания. — Ростов-на-Дону: Изд-во Ростовского гос. ун-та, 2007 г. — 37 с.
- Кормен Т.Х., Лейзерсон Ч.И., Ривест Р.Л. Алгоритмы: построение и анализ. — М.: Бином, 2004. — 960 с.
- Кузюрин Н.Н. Эффективные алгоритмы и сложность вычислений.

  http http:// http://discopal http://discopal. http://discopal.ispras http://discopal.ispras. http://discopal.ispras.ru http://discopal.ispras.ru/ http://discopal.ispras.ru/ru http://discopal.ispras.ru/ru. http://discopal.ispras.ru/ru.book http://discopal.ispras.ru/ru.book- http://discopal.ispras.ru/ru.book-advanced http://discopal.ispras.ru/ru.book-advanced- http://discopal.ispras.ru/ru.book-advanced-algorithms http://discopal.ispras.ru/ru.book-advanced-algorithms. http://discopal.ispras.ru/ru.book-advanced-algorithms.htm)
- Bertsekas D.P., Tsitsiklis J.N. Parallel and Distributed