Многопоточное программирование

Параллельные методы сортировки

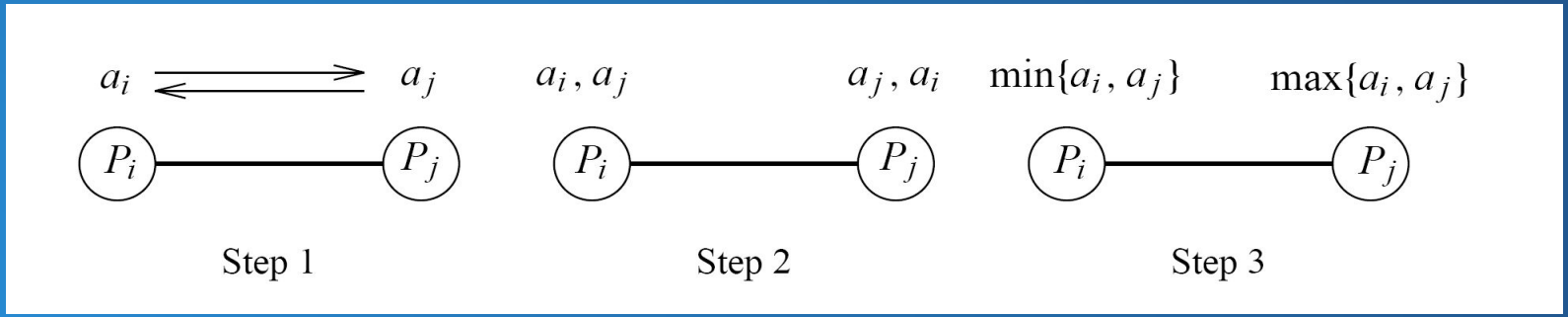
Сортировка

- $S = \{a_1, a_2, a_3 \ldots, a_n\}$
- $S \sim S' = \{a'_1, a'_2, a'_3 \ldots, a'_n\}: a'_i <= a'_j \ , i<j$
- $O(n^2)$, $O(n \log n)$
- swap(a_i, a_j)

```
temp = A[i];
A[i] = A[j];
A[j] = temp;
```

Compare-Exchange

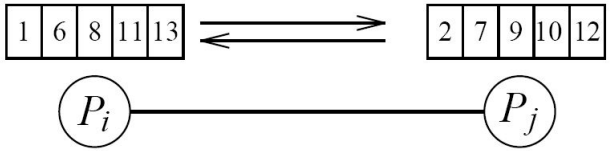
- p ~ n
- *compare-exchange*
- $t_s + t_w$.



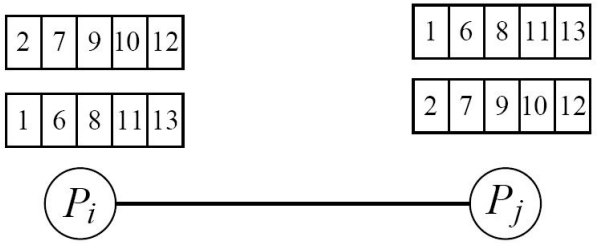
Compare-Split

- $p << n$
- $\{P_1, P_2, \ldots P_p\}$
- $\{A_1, A_2, \ldots A_p\}$
- Результат выполнения параллельного алгоритма ($A_i < A_j$)
- *compare-split* operation
- $t_s + t_w * n/p$.

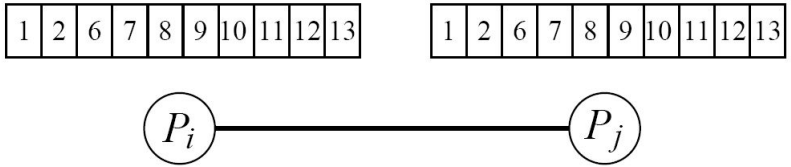
Compare-Split



Step 1



Step 2



Step 3

Step 4

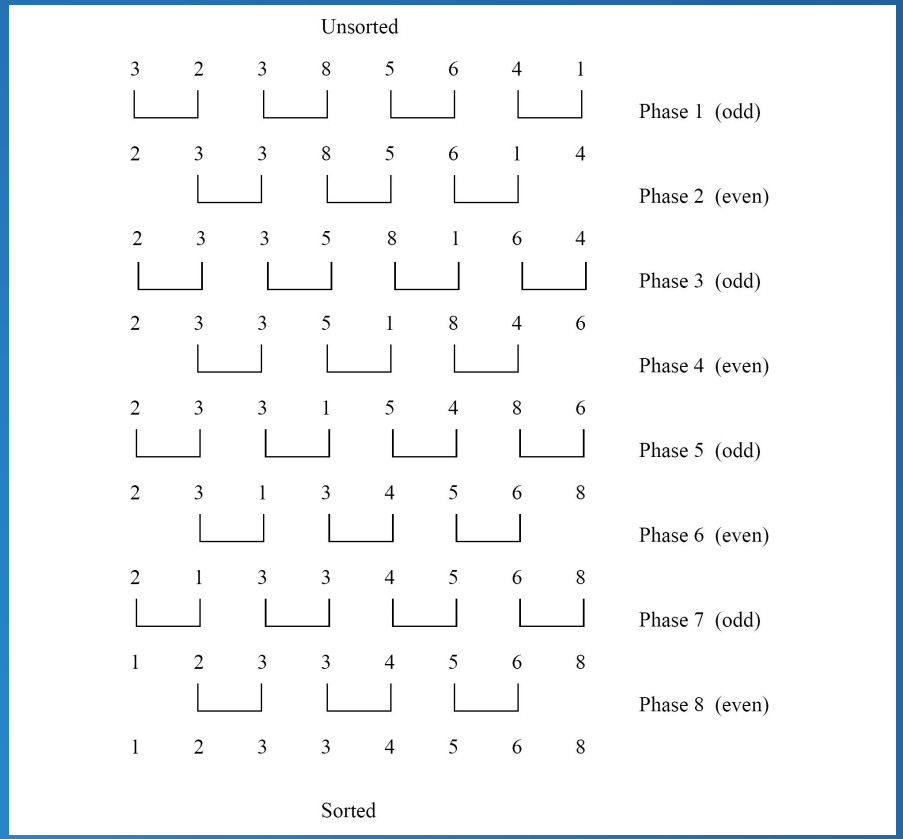
Bubble sort

```
procedure BUBBLE_SORT(n)
begin
    for i := n − 1 downto 1 do
        for j := 1 to i do
            compare-exchange(a_j, a_{j+1});
end BUBBLE_SORT
```

Odd-even transposition

```
procedure ODD-EVEN(n)
begin
    for i := 1 to n do
    begin
        if i is odd then
            for j := 0 to n/2 − 1 do
                compare-exchange(a_{2j+1}, a_{2j+2});
        if i is even then
            for j := 1 to n/2 − 1 do
                compare-exchange(a_{2j}, a_{2j+1});
    end for
end ODD-EVEN
```

Odd-even transposition



Parallel Odd-even transposition

```
procedure ODD-EVEN_PAR(n)
begin
    id := process's label
    for i := 1 to n do
    begin
        if i is odd then
            if id is odd then
                compare-exchange_min(id + 1);
            else
                compare-exchange_max(id − 1);
        if i is even then
            if id is even then
                compare-exchange_min(id + 1);
            else
                compare-exchange_max(id − 1);
    end for
end ODD-EVEN_PAR
```

Parallel Odd-even transposition

- Сравнение пар на одной итерации независимы
- Блоки по ***n/p***
- Первый шаг локальная сортировка
- Дальше ***p*** фаз на каждой из которых ***n/p*** операций сравнения и ***n/p*** коммуникативных операций

$$T_P = \overbrace{\Theta\left(\frac{n}{p}\log\frac{n}{p}\right)}^{\text{local sort}} + \overbrace{\Theta(n)}^{\text{comparisons}} + \overbrace{\Theta(n).}^{\text{communication}}$$

Parallel Odd-even transposition

$$S = \frac{\Theta(n \log n)}{\Theta((n/p)\log(n/p)) + \Theta(n)}$$

$$E = \frac{1}{1 - \Theta((\log p)/(\log n)) + \Theta(p/\log n)}$$

QuickSort

- Один из наиболее популярных алгоритмов
- В среднем O(nlog(n))
- Принцип разделяй и властвуй
- Эффективность зависит от выбора главного элемента
- В худшем случае $O(n^2)$

Quicksort

```
procedure QUICKSORT (A, q, r)
begin
    if q < r then
    begin
        x := A[q];
        s := q;
        for i := q + 1 to r do
            if A[i] ≤ x then
            begin
                s := s + 1;
                swap(A[s], A[i]);
            end if
        swap(A[q], A[s]);
        QUICKSORT (A, q, s);
        QUICKSORT (A, s + 1, r);
    end if
end QUICKSORT
```

Parallel quicksort

- Рекурсивная декомпозиция
  - Первый шаг на одном процессоре
  - Каждая подзадача запускается на новом процессора
- Неэффективно

Parallel quicksort

- $p = 2^N$
- Блоки по n/p
- Топология N-мерный гиперкуб
- Алгоритм:
  1. Выбрать ведущий элемент (средний на ведущем процессоре)
  2. Разделить на каждом процессоре на 2 множества
  3. Обменяться с соседним (разница в N бите)
  4. Для получившихся 2-х (N-1) - мерных гиперкубов повторить

Parallel quicksort

$$T_P = \overbrace{\Theta\left(\frac{n}{p}\log\frac{n}{p}\right)}^{\text{local sort}} + \overbrace{\Theta\left(\frac{n}{p}\log p\right) + \Theta(\log^2 p)}^{\text{array splits}}.$$

Bucketsort

- n элементов равномерно распределены в отрезке [a; b]
- Алгоритм:
  - Создаем k блоков
  - Каждый элемент помещаем в свой блок
  - Локальной сортируем каждый блок
  - Сливаем блоки\
- Θ(nlog(n/m)) -> Θ(n)

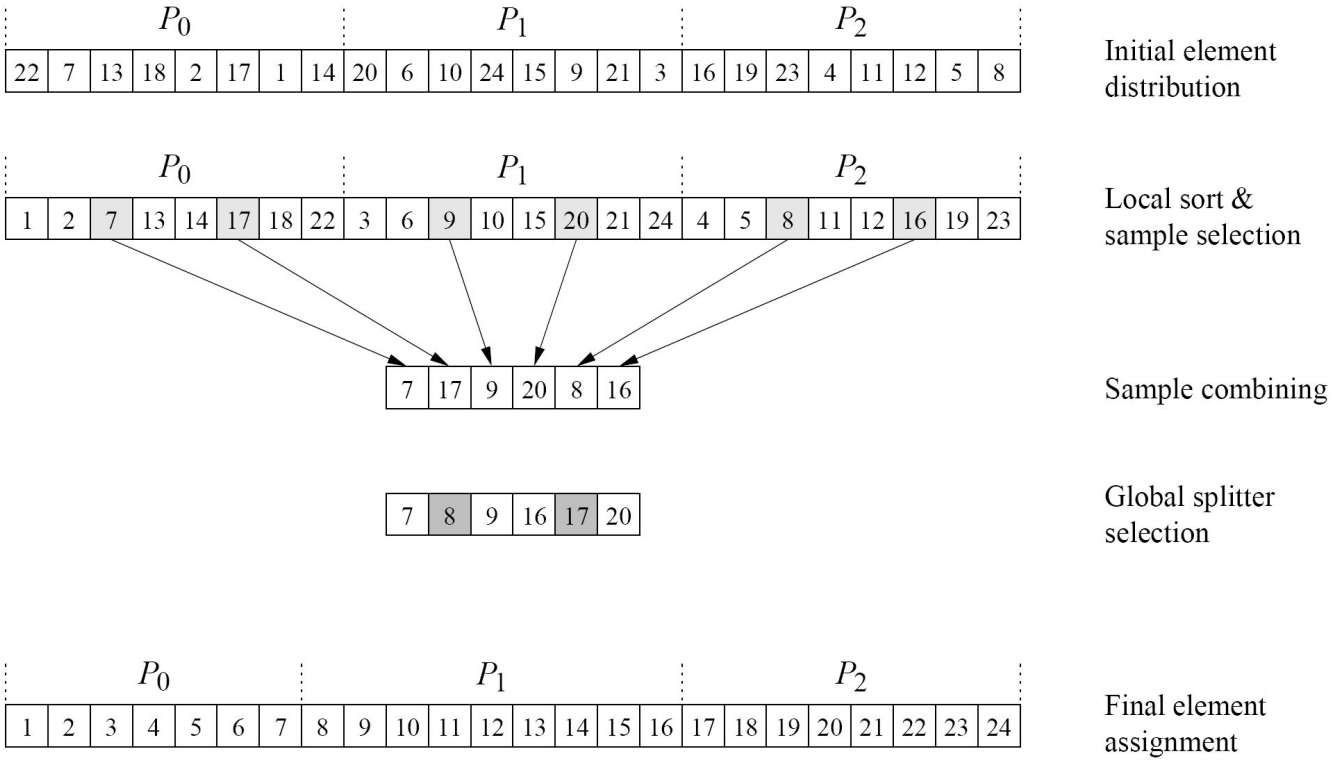
Parallel bucketsort

- Выбираем количество блоков равное количеству процессов
- В рамках каждого процесса распределяем элементы по p блокам
- Обмениваемся нужными блоками с другим процесса
- Сортируем данные в рамках каждого процесса

Bucket and Sample sort

- Основная проблема - разделить элементы равномерно по блокам.
- Для этого выбираемся spliter
  - Сортируем все блоки размера n/p
  - Выбираем sample - p-1 элемент из каждого блока(всего p* (p-1))
  - Сортируем sample и выбираем p-1 равномерно распределенный элемент.
  - Полученный результат - критерий для разделения по блокам
- Далее Bucket-sort

Bucket and Sample sort



Parallel Bucket and Sample sort

1. Локально отсортировать эдементы блока
2. Broadcast p выбранных элементов
3. Выделение Splitter-a
4. Разделение по блокам
5. Объеденить значения

$$T_P = \overbrace{\Theta\left(\frac{n}{p}\log\frac{n}{p}\right)}^{\text{local sort}} + \overbrace{\Theta\left(p^2\log p\right)}^{\text{sort sample}} + \overbrace{\Theta\left(p\log\frac{n}{p}\right)}^{\text{block partition}} + \overbrace{\Theta(n/p).}^{\text{communication}}$$