Нижегородский государственный университет им. Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Образовательный комплекс

Введение в методы параллельного программирования

Раздел 10.

Параллельные методы сортировки

Гергель В.П., профессор, д.т.н.
Кафедра математического обеспечения ЭВМ

Содержание

Постановка задачи

Сортировка является одной из типовых проблем обработки данных и обычно понимается как задача размещения элементов неупорядоченного набора значений

$$S = \{a_1, a_2, ..., a_n\}$$

в порядке монотонного возрастания или убывания

$$S \sim S' = \{(a_1', a_2', ..., a_n') : a_1' \le a_2' \le ... \le a_n'\}$$

Принципы распараллеливания…

Базовая операция – "*сравнить и переставить*" (*compare-exchange*)

```
// базовая операция сортировки
if ( A[i] > A[j] ) {
  temp = A[i];
  A[i] = A[j];
  A[j] = temp;
}
```

- Последовательное применение данной операции позволяет упорядочить данные
- В способах выбора пар значений для сравнения проявляется различие алгоритмов сортировки

Принципы распараллеливания…

Параллельное обобщение базовой операции при ***p = n***

(каждый процессор содержит 1 элемент данных)

- выполнить взаимообмен имеющихся на процессорах P_i и P_j значений (с сохранением на этих процессорах исходных элементов),
- сравнить на каждом процессоре P_i и P_j получившиеся одинаковые пары значений (a_i, a_j) и по результатам сравнения разделить данные между процессорами – на одном процессоре (например, P_i) оставить меньший элемент, на другом процессоре (т.е. P_j) запомнить большее значение пары

$$a_i' = \min(a_i, a_j) \qquad a_j' = \max(a_i, a_j)$$

Принципы распараллеливания…

Результат выполнения **параллельного алгоритма:**

- имеющиеся на процессорах данные упорядочены,
- порядок распределения данных по процессорам соответствует линейному порядку нумерации (т.е. значение последнего элемента на процессоре P_i меньше или равно значения первого элемента на процессоре P_{i+1} , где $0 \le i < p-1$).

Принципы распараллеливания…

Параллельное обобщение базовой операции при ***p < n***

(каждый процессор содержит блок данных размера n / p)

- упорядочить блок на каждом процессоре в начале сортировки,
- выполнить взаимообмен блоков между процессорами P_i и P_{i+1} ,
- объединить блоки A_i и A_{i+1} на каждом процессоре в один отсортированный блок с помощью операции слияния,
- разделить полученный двойной блок на две равные части и оставить одну из этих частей (например, с меньшими значениями данных) на процессоре P_i, а другую часть (с большими значениями соответственно) – на процессоре P_{i+1}

$$[A_i \cup A_{i+1}]_{copm} = A_i^{'} \cup A_{i+1}^{'} : \forall a_i^{'} \in A_i^{'}, \forall a_j^{'} \in A_{i+1}^{'} \Rightarrow a_i^{'} \le a_j^{'}$$

Рассмотренная процедура обычно именуется в литературе как операция "*сравнить и разделить*" (*compare-split*).

Пузырьковая сортировка: *последовательный алгоритм*

```
// Последовательный алгоритм пузырьковой сортировки
BubbleSort(double A[], int n) {
  for (i=0; i<n-1; i++)
    for (j=0; j<n-i; j++)
      compare_exchange(A[j], A[j+1]);
}
```

- Трудоемкость вычислений имеет порядок $O(n^2)$,
- В прямом виде сложен для распараллеливания

Пузырьковая сортировка: *алгоритм чет-нечетной перестановки*

```
// Последовательный алгоритм чет-нечетной перестановки
OddEvenSort(double A[], int n) {
  for (i=1; i<n; i++) {
    if ( i%2==1) // нечетная итерация
      for (j=0; j<n/2-1; j++)
        compare_exchange(A[2j+1],A[2j+2]);
    if (i%2==0) // четная итерация
      for (j=1; j<n/2-1; j++)
        compare_exchange(A[2j],A[2j+1]);
  }
}
```

Разные правила для выполнения четных и нечетных итераций

⇩

Возможности распараллеливания

Пузырьковая сортировка: *параллельный алгоритм чет-нечетной перестановки*…

```
// Параллельный алгоритм чет-нечетной перестановки
ParallelOddEvenSort ( double A[], int n ) {
  int id = GetProcId();  // номер процесса
  int np = GetProcNum(); // количество процессов
  for ( int i=0; i<np; i++ ) {
    if ( i%2 == 1 ) { // нечетная итерация
      if ( id%2 == 1 ) // нечетный номер процесса
        compare_split_min(id+1); // сравнение-обмен справа
      else compare_split_max(id-1); // сравнение-обмен слева
    }
    if ( i%2 == 0 ) { // четная итерация
      if( id%2 == 0 )  // четный номер процесса
        compare_split_min(id+1); // сравнение-обмен справа
      else compare_split_max(id-1); // сравнение-обмен слева
    }
  }
}
```

Пузырьковая сортировка: *параллельный алгоритм чет-нечетной перестановки*…

- **Анализ эффективности**
  - Общая оценка показателей ускорения и эффективности

$$S_p = \frac{p \log_2 n}{\log_2 (n/p) + 2p}, \qquad E_p = \frac{\log_2 n}{\log_2 (n/p) + 2p}$$

Пузырьковая сортировка: *параллельный алгоритм чет-нечетной перестановки…*

- **Анализ эффективности** (уточненные оценки)

- Время выполнения параллельного алгоритма, связанное непосредственно с вычислениями, составляет

$$T_p(calc) = (n/p)\log_2(n/p) + 2n)\tau$$

- Длительность выполнения операции сбора данных при использовании модели Хокни определяется при помощи следующего выражения:

$$T_p(comm) = p \cdot (\alpha + w \cdot (n/p)/\beta)$$

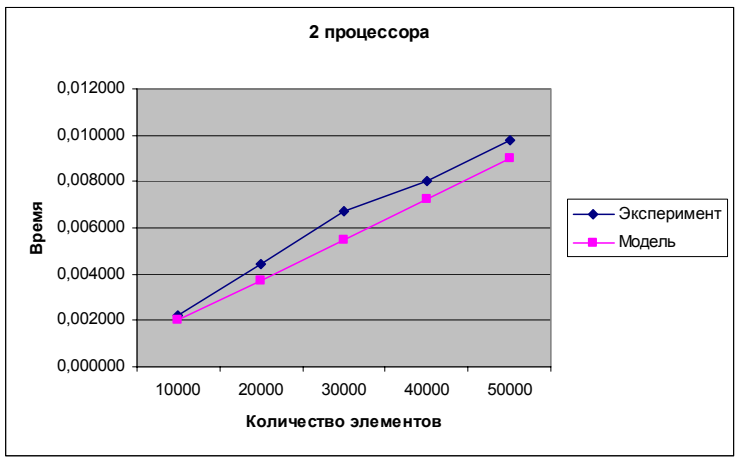
Общее время выполнения параллельного алгоритма составляет

$$T_p = ((n/p)\log_2(n/p) + 2n)\tau + p \cdot (\alpha + w \cdot (n/p)/\beta)$$

Пузырьковая сортировка: *параллельный алгоритм чет-нечетной перестановки*…

- **Результаты вычислительных экспериментов**
  - Сравнение теоретических оценок и экспериментальных данных

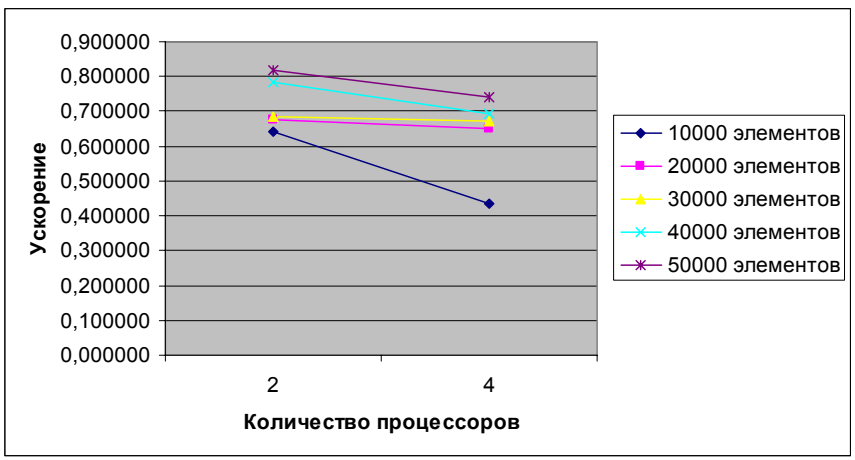
Размер данных	Параллельный алгоритм			
	2 процессора		4 процессора	
10 000	0,002003	0,002210	0,002057	0,003270
20 000	0,003709	0,004428	0,003366	0,004596
30 000	0,005455	0,006745	0,004694	0,006873
40 000	0,007227	0,008033	0,006035	0,009107
50 000	0,009018	0,009770	0,007386	0,010840



Пузырьковая сортировка: *параллельный алгоритм чет-нечетной перестановки*…

- **Результаты вычислительных экспериментов**
  - Ускорение вычислений

Количество элементов	Последовательный алгоритм	Параллельный алгоритм			
		2 процессора		4 процессора	
		Время	Ускорение	Время	Ускорение
10 000	0,001422	0,002210	0,643439	0,003270	0,434862
20 000	0,002991	0,004428	0,675474	0,004596	0,650783
30 000	0,004612	0,006745	0,683766	0,006873	0,671032
40 000	0,006297	0,008033	0,783891	0,009107	0,691446
50 000	0,008014	0,009770	0,820266	0,010840	0,739299



Пузырьковая сортировка: *параллельный алгоритм чет-нечетной перестановки*

- Параллельный вариант алгоритма *работает медленнее* исходного последовательного метода пузырьковой сортировки:
  - – объем передаваемых данных между процессорами является достаточно большим и сопоставим с количеством выполняемых вычислительных операций,
  - – этот дисбаланс объема вычислений и сложности операций передачи данных увеличивается с ростом числа процессоров

Сортировка Шелла: *последовательный алгоритм…*

- Общая идея *сортировки Шелла* состоит в сравнении на начальных стадиях сортировки пар значений, располагаемых достаточно далеко друг от друга в упорядочиваемом наборе данных (сортировка таких пар обычно требует большого количества перестановок, если используется сравнение только соседних элементов):
  - На первом шаге алгоритма происходит упорядочивание элементов *n/2* пар (a_i, $a_{n/2+i}$) для $1 \le i \le n/2$,
  - На втором шаге упорядочиваются элементы в *n/4* группах из четырех элементов (a_i, $a_{n/4+1}$, $a_{n/2+1}$, $a_{3n/4+1}$) для $1 \le i \le n/4$ и т.д.,
  - На последнем шаге упорядочиваются элементы сразу во всем массиве (a_1, a_2,…, a_n).
- Общее количество итераций алгоритма Шелла является равным $log_2 n$

Сортировка Шелла: *последовательный алгоритм*

```
// Последовательный алгоритм сортировки Шелла
ShellSort ( double A[], int n ){
  int incr = n/2;
  while( incr > 0 ) {
    for ( int i=incr+1; i<n; i++  ) {
      j = i-incr;
      while ( j > 0 )
        if ( A[j] > A[j+incr] ){
          swap(A[j], A[j+incr]);
          j = j - incr;
        }
        else j = 0;
    }
    incr = incr/2;
  }
}
```

Трудоемкость вычислений имеет порядок $O(n\log_2 n)$.

Сортировка Шелла: *параллельный алгоритм…*

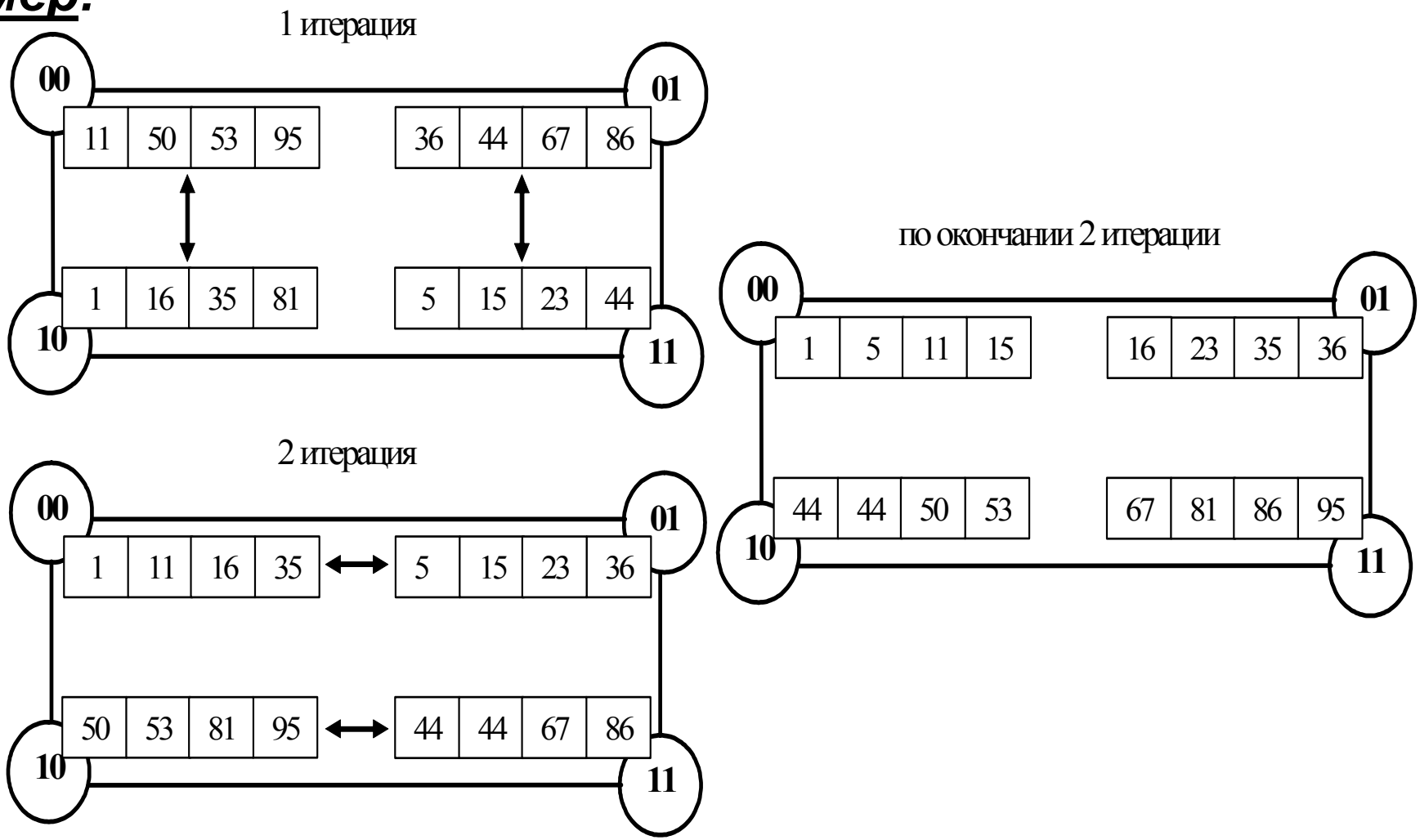
Пусть топология коммуникационной сети имеет вид ***N***-мерного гиперкуба (т.е. количество процессоров равно $p=2^N$).

Действия алгоритма состоят в следующем:

- *Первый этап (**N** итераций)*: выполнение операции "сравнить и разделить" для каждой пары процессоров в гиперкубе. Формирование пар процессоров происходит по правилу – на каждой итерации i, $0 \le i < N$, парными становятся процессоры, у которых различие в битовых представлении их номеров имеется только в позиции $N-i$,
- *Второй этап*: реализация обычных итераций параллельного алгоритма чет-нечетной перестановки. Итерации выполняются до прекращения фактического изменения сортируемого набора. Общее их количество L может быть различным - от 2 до p.

Сортировка Шелла: *параллельный алгоритм…*

Пример:



Сортировка Шелла: *параллельный алгоритм…*

- **Анализ эффективности**
  - Общая оценка показателей ускорения и эффективности

$$S_p = \frac{p \log_2 n}{\log_2 (n/p) + 2p}, \qquad E_p = \frac{\log_2 n}{\log_2 (n/p) + 2p}$$

Сортировка Шелла: *параллельный алгоритм…*

❑ Анализ эффективности (уточненные оценки)

- Время выполнения параллельного алгоритма, связанное непосредственно с вычислениями, составляет

$$T_p(calc) = (n/p)\log_2(n/p)\tau$$

- Длительность выполнения операции сбора данных при использовании модели Хокни определяется при помощи следующего выражения:

$$T_p(comm) = (\log_2 p + L)[(2n/p) + (\alpha + w\cdot(n/p)/\beta)]$$

Общее время выполнения параллельного алгоритма составляет

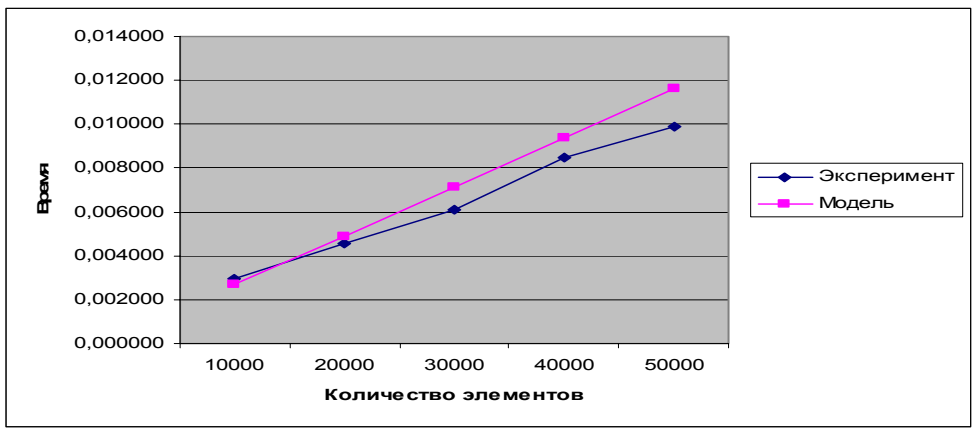
$$T_p = (n/p)\log_2(n/p)\tau + (\log_2 p + L)[(2n/p) + (\alpha + w\cdot(n/p)/\beta)]$$

Сортировка Шелла: *параллельный алгоритм…*

❑ Результаты вычислительных экспериментов

– Сравнение теоретических оценок и экспериментальных данных T_p^*

Размер данных	Параллельный алгоритм			
	2 процессора		4 процессора	
10 000	0,002684	0,002959	0,002938	0,007509
20 000	0,004872	0,004557	0,004729	0,009826
30 000	0,007100	0,006118	0,006538	0,012431
40 000	0,009353	0,008461	0,008361	0,017009
50 000	0,011625	0,009920	0,010193	0,019419

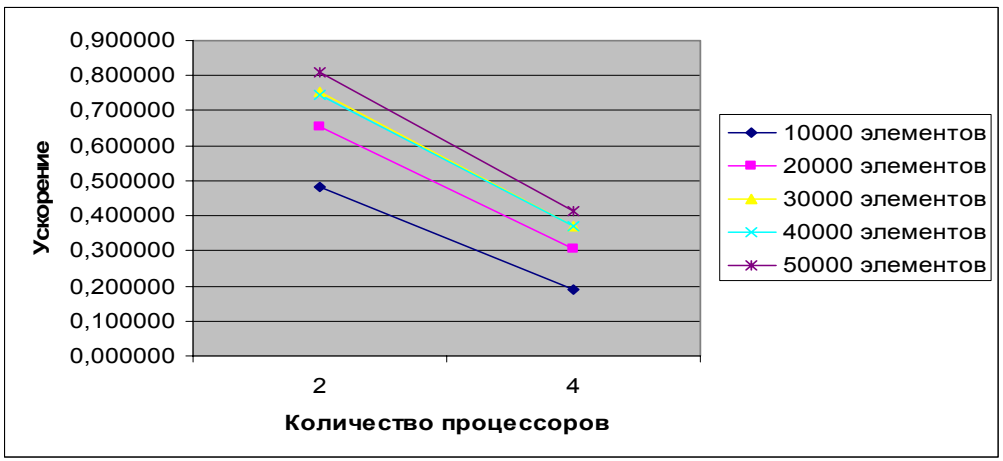


Сортировка Шелла: *параллельный алгоритм*

❑ Результаты вычислительных экспериментов

– Ускорение вычислений

Количество элементов	Последовательный алгоритм	Параллельный алгоритм			
		2 процессора		4 процессора	
		Time	Speedup	Time	Speedup
10 000	0,001422	0,002959	0,480568	0,007509	0,189373
20 000	0,002991	0,004557	0,656353	0,009826	0,304396
30 000	0,004612	0,006118	0,753841	0,012431	0,371008
40 000	0,006297	0,008461	0,744238	0,017009	0,370216
50 000	0,008014	0,009920	0,807863	0,019419	0,412689



Быстрая сортировка: *последовательный алгоритм…*

- *Алгоритм быстрой сортировки*, предложенной Хоаром (*Hoare C.A.R.*), основывается на последовательном разделении сортируемого набора данных на блоки меньшего размера таким образом, что между значениями разных блоков обеспечивается отношение упорядоченности (для любой пары блоков все значения одного из этих блоков не превышают значений другого блока):
  - – На первой итерации метода осуществляется деление исходного набора данных на первые две части – для организации такого деления выбирается некоторый *ведущий элемент* и все значения набора, меньшие ведущего элемента, переносятся в первый формируемый блок, все остальные значения образуют второй блок набора,
  - – На второй итерации сортировки описанные правила применяются рекурсивно для обоих сформированных блоков и т.д.
- При оптимальном выборе ведущих элементов после выполнения $log_2 n$ итераций исходный массив данных оказывается упорядоченным.

Быстрая сортировка: *последовательный алгоритм…*

```
// Последовательный алгоритм быстрой сортировки
QuickSort(double A[], int i1, int i2) {
  if ( i1 < i2 ){
    double pivot = A[i1];
    int is = i1;
    for ( int i = i1+1; i<i2; i++ )
      if ( A[i] ≤ pivot ) {
        is = is + 1;
        swap(A[is], A[i]);
      }
    swap(A[i1], A[is]);
    QuickSort (A, i1, is);
    QuickSort (A, is+1, i2);
  }
}
```

Среднее количество операций $1.4n\log_2 n$

Быстрая сортировка: *параллельный алгоритм…*

Пусть топология коммуникационной сети имеет вид ***N***-мерного гиперкуба (т.е. количество процессоров равно $p=2^N$). **Действия алгоритма** состоят в следующем:

- ❑ выбрать каким-либо образом ведущий элемент и разослать его по всем процессорам системы (например, в качестве ведущего элемента можно взять среднее арифметическое элементов, расположенных на ведущем процессоре),
- ❑ разделить на каждом процессоре имеющийся блок данных на две части с использованием полученного ведущего элемента,
- ❑ образовать пары процессоров, для которых битовое представление номеров отличается только в позиции ***N***, и осуществить взаимообмен данными между этими процессорами; в результате таких пересылок данных на процессорах, для которых в битовом представлении номера бит позиции ***N*** равен 0, должны оказаться части блоков со значениями, меньшими ведущего элемента; процессоры с номерами, в которых бит ***N*** равен 1, должны собрать, соответственно, все значения данных, превышающие значения ведущего элемента,
- ❑ перейти к подгиперкубу меньшей размерности и повторить описанную выше процедуру

Быстрая сортировка: *параллельный алгоритм…*

- **Анализ эффективности**
  - Общая оценка показателей ускорения и эффективности

$$S_p = \frac{p \log_2 n}{\log_2 (n/p) + 2p}, \qquad E_p = \frac{\log_2 n}{\log_2 (n/p) + 2p}$$

Быстрая сортировка: *параллельный алгоритм…*

❑ Анализ эффективности (уточненные оценки)

- Время выполнения параллельного алгоритма, связанное непосредственно с вычислениями, составляет

$$T_p(calc) = [(n/p)\log_2 p + (n/p)\log_2(n/p)]\tau$$

- Длительность выполнения операции сбора данных при использовании модели Хокни определяется при помощи следующего выражения:

$$T_p(comm) = (\log_2 p)^2(\alpha + w/\beta) + \log_2 p\ (\alpha + w(n/2p)/\beta)$$

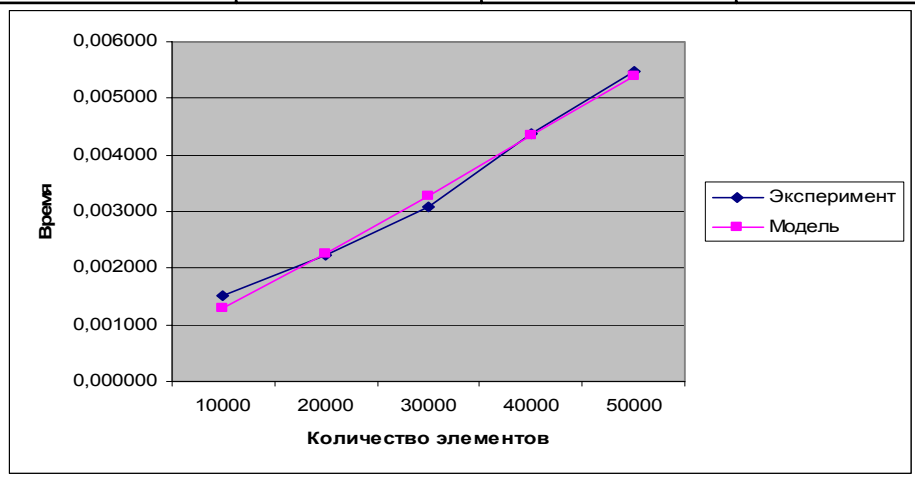
Общее время выполнения параллельного алгоритма составляет

$$T_p = [(n/p)\log_2 p + (n/p)\log_2(n/p)]\tau + (\log_2 p)^2(\alpha + w/\beta) + \log_2 p\ (\alpha + w(n/2p)/\beta)$$

Быстрая сортировка: *параллельный алгоритм…*

- **Результаты вычислительных экспериментов**
  - Сравнение теоретических оценок и экспериментальных данных

Размер данных	Параллельный алгоритм			
	2 процессора		4 процессора	
10 000	0,001280	0,001521	0,001735	0,003434
20 000	0,002265	0,002234	0,002321	0,004094
30 000	0,003289	0,003080	0,002928	0,005088
40 000	0,004338	0,004363	0,003547	0,005906
50 000	0,005407	0,005486	0,004175	0,006635

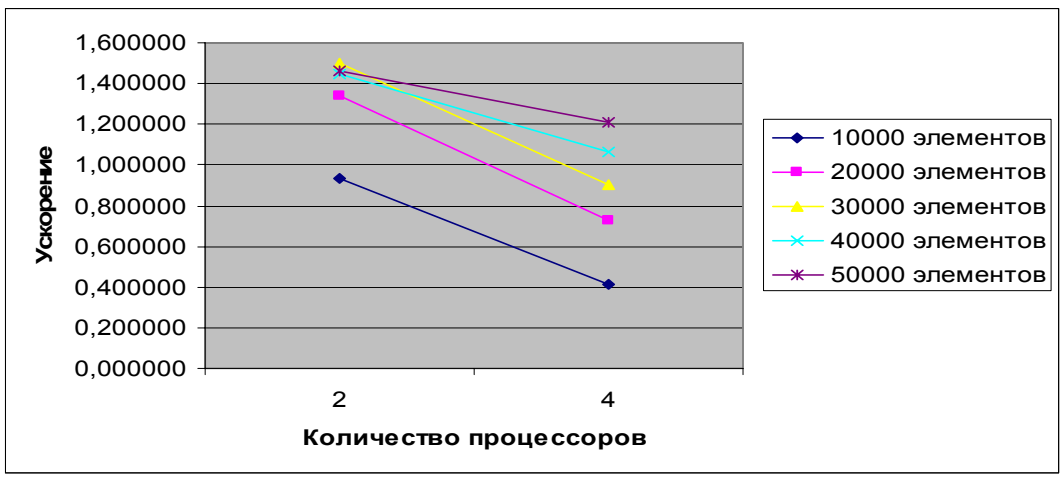


Быстрая сортировка: *параллельный алгоритм*

❑ Результаты вычислительных экспериментов

– Ускорение вычислений

Количество элементов	Последовательный алгоритм	Параллельный алгоритм			
		2 процессора		4 процессора	
		Время	Ускорение	Время	Ускорение
10 000	0,001422	0,001521	0,934911	0,003434	0,414094
20 000	0,002991	0,002234	1,338854	0,004094	0,730581
30 000	0,004612	0,003080	1,497403	0,005088	0,906447
40 000	0,006297	0,004363	1,443273	0,005906	1,066204
50 000	0,008014	0,005486	1,460809	0,006635	1,207837



Обобщенная быстрая сортировка: *параллельный алгоритм…*

- Основное отличие от предыдущего алгоритма – конкретный способ выбора ведущего элемента
- Сортировка блоков выполняется в самом начале выполнения вычислений. В качестве ведущего выбирается средний элемент какого-либо блока (например, на первом процессоре вычислительной системы). Выбираемый подобным образом ведущий элемент в отдельных случаях может оказаться более близок к настоящему среднему значению всего сортируемого набора, чем какое-либо другое произвольно выбранное значение
- Для поддержки упорядоченности в ходе вычислений процессоры выполняют операцию слияния частей блоков, получаемых после разделения

Обобщенная быстрая сортировка: *параллельный алгоритм…*

- **Анализ эффективности**
  - Общая оценка показателей ускорения и эффективности

$$S_p = \frac{p \log_2 n}{\log_2 (n/p) + 2p}, \qquad E_p = \frac{\log_2 n}{\log_2 (n/p) + 2p}$$

Обобщенная быстрая сортировка: *параллельный алгоритм…*

❑ Анализ эффективности (уточненные оценки)

- Время выполнения параллельного алгоритма, связанное непосредственно с вычислениями, составляет

$$T_p(calc) = [(n/p)\log_2(n/p) + (\log_2(n/p) + (n/p))\log_2 p]\tau$$

- Длительность выполнения операции сбора данных при использовании модели Хокни определяется при помощи следующего выражения:

$$T_p(comm) = (\log_2 p)^2(\alpha + w/\beta) + \log_2 p\,(\alpha + w(n/2p)/\beta)$$

Общее время выполнения параллельного алгоритма составляет

$$T_p = [(n/p)\log_2(n/p) + (\log_2(n/p) + (n/p))\log_2 p]\tau + (\log_2 p)^2(\alpha + w/\beta) + \log_2 p\,(\alpha + w(n/2p)/\beta)$$

Обобщенная быстрая сортировка: *параллельный алгоритм…*

- **Программная реализация**
  - Первый этап: Инициализация и распределение данных между процессорами:
    - получение размера сортируемого массива,
    - определение исходных данных для сортируемого массива,
    - пересылка исходных данных между процессорами вынесено в отдельную функцию *DataDistribution*.

Программа

Обобщенная быстрая сортировка: *параллельный алгоритм…*

- **Программная реализация**
  - Второй этап: выполнение итераций параллельной обобщенной быстрой сортировки реализовано в функции ***ParallelHyperQuickSort***

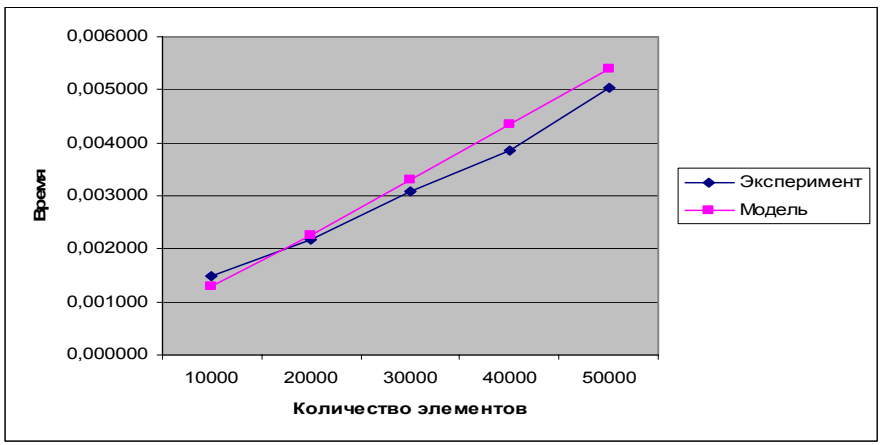
Программа

Обобщенная быстрая сортировка: *параллельный алгоритм…*

❑ Результаты вычислительных экспериментов

– Сравнение теоретических оценок и экспериментальных данных T_4^*

Размер данных	Параллельный алгоритм			
	2 процессора		4 процессора	
10 000	0,001281	0,001485	0,001735	0,002898
20 000	0,002265	0,002180	0,002322	0,003770
30 000	0,003289	0,003077	0,002928	0,004451
40 000	0,004338	0,003859	0,003547	0,004721
50 000	0,005407	0,005041	0,004176	0,005242

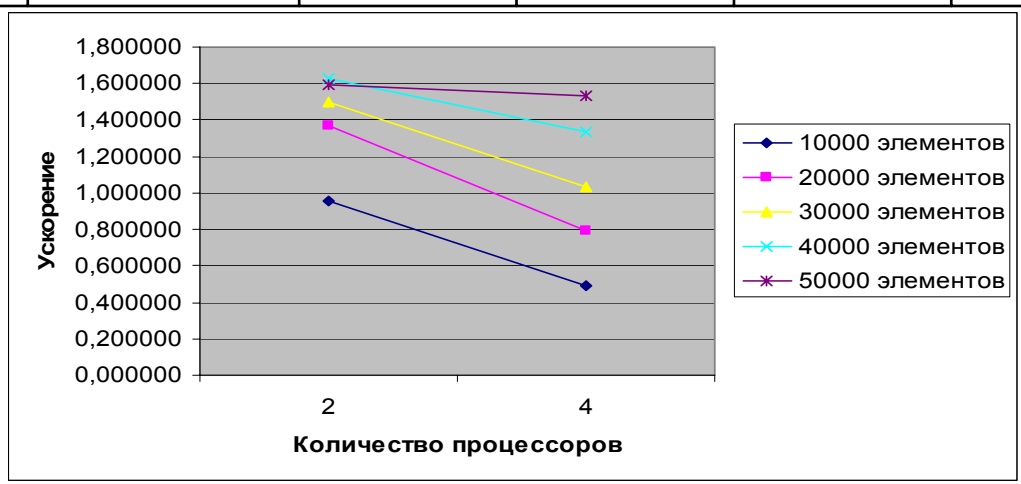


Обобщенная быстрая сортировка: *параллельный алгоритм*

❑ Результаты вычислительных экспериментов

– Ускорение вычислений

Количество элементов	Последовательный алгоритм	Параллельный алгоритм			
		2 процессора		4 процессора	
		Время	Ускорение	Время	Ускорение
10 000	0,001422	0,001485	0,957576	0,002898	0,490683
20 000	0,002991	0,002180	1,372018	0,003770	0,793369
30 000	0,004612	0,003077	1,498863	0,004451	1,036172
40 000	0,006297	0,003859	1,631770	0,004721	1,333828
50 000	0,008014	0,005041	1,589764	0,005242	1,528806



Сортировка с использованием регулярного набора образцов: *параллельный алгоритм…*

Первый этап**:* упорядочивание блоков каждым процессором независимо друг от друга при помощи обычного алгоритма быстрой сортировки; далее каждый процессор формирует набор из элементов своих блоков с индексами ***0, m, 2m,…,(p-1)m, где ***$m=n/p^2$***,

Второй этап: все сформированные на процессорах наборы данных собираются на одном из процессоров системы и объединяются в ходе последовательного сливания в одно упорядоченное множество; из полученного множества значений из элементов с индексами

$$p+\lfloor p/2\rfloor-1,\quad 2p+\lfloor p/2\rfloor-1,\ldots,(p-1)p+\lfloor p/2\rfloor$$

формируется набор ведущих элементов, который передается всем процессорам; в завершение этапа каждый процессор выполняет разделение своего блока на ***p*** частей с использованием полученного набора ведущих значений,

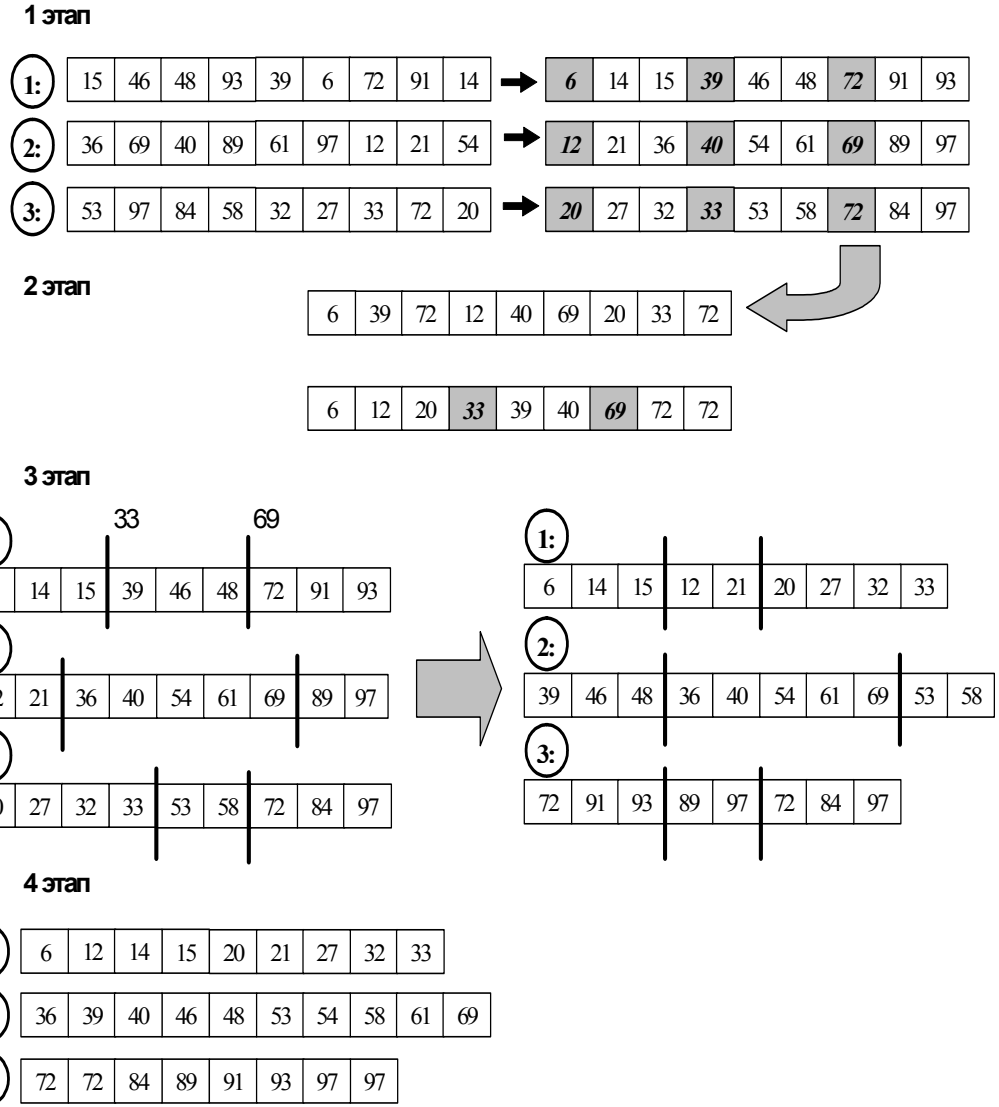
Третий этап: каждый процессор рассылает выделенные ранее части своего блока всем остальным процессорам системы в соответствии с порядком нумерации - часть ***j, $0\le j<p$***, каждого блока пересылается процессору с номером ***j***,

Четвертый этап: каждый процессор выполняет слияние ***p*** полученных частей в один отсортированный блок.

Сортировка с использованием регулярного набора образцов: *параллельный алгоритм…*

Пример:

(p=3)



Сортировка с использованием регулярного набора образцов: *параллельный алгоритм…*

- **Анализ эффективности**
  - Общая оценка показателей ускорения и эффективности

$$S_p = \frac{p \log_2 n}{\log_2 (n/p) + 2p}, \qquad E_p = \frac{\log_2 n}{\log_2 (n/p) + 2p}$$

Сортировка с использованием регулярного набора образцов: *параллельный алгоритм…*

❑ Анализ эффективности (уточненные оценки)

- Время выполнения первого этапа параллельного алгоритма:

$$T_p^1 = (n/p)\log_2(n/p)\,\tau$$

- Время выполнения второго этапа параллельного алгоритма:

$$T_p^2 = [\alpha \log_2 p + wp(p-1)/\beta] + [p^2 \log_2 p\,\tau] + [p\tau] + [\log_2 p\,(\alpha + wp/\beta)]$$

- Время выполнения третьего этапа параллельного алгоритма:

$$T_p^3 = (n/p)\tau + \log_2 p(\alpha + w(n/2p)/\beta)$$

- Время выполнения четвертого этапа параллельного алгоритма:

$$T_p^4 = (n/p)\log_2 p\,\tau$$

Общее время выполнения параллельного алгоритма:

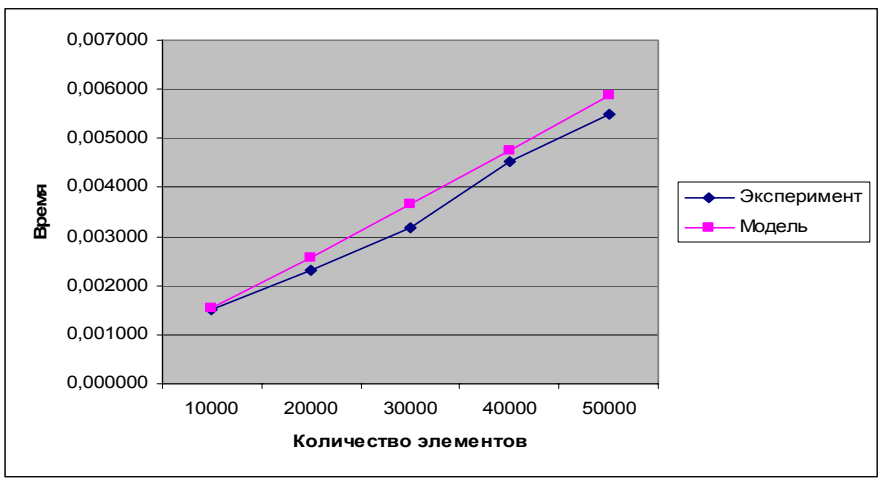
$$\begin{aligned} T_p \;=\; & (n/p)\log_2(n/p)\,\tau + (\alpha \log_2 p + wp(p-1)/\beta) + p^2 \log_2 p\,\tau + (n/p)\tau + \\ & + \log_2 p(\alpha + wp/\beta) + p\tau + \log_2 p(\alpha + w(n/2p)/\beta) + (n/p)\log_2 p\,\tau \end{aligned}$$

Сортировка с использованием регулярного набора образцов: *параллельный алгоритм…*

❑ Результаты вычислительных экспериментов

– Сравнение теоретических оценок и экспериментальных данных T_4^*

Размер данных	Параллельный алгоритм			
	2 процессора		4 процессора	
10 000	0,001533	0,001513	0,001762	0,001166
20 000	0,002569	0,002307	0,002375	0,002081
30 000	0,003645	0,003168	0,003007	0,003099
40 000	0,004747	0,004542	0,003652	0,003819
50 000	0,005867	0,005503	0,004307	0,004370

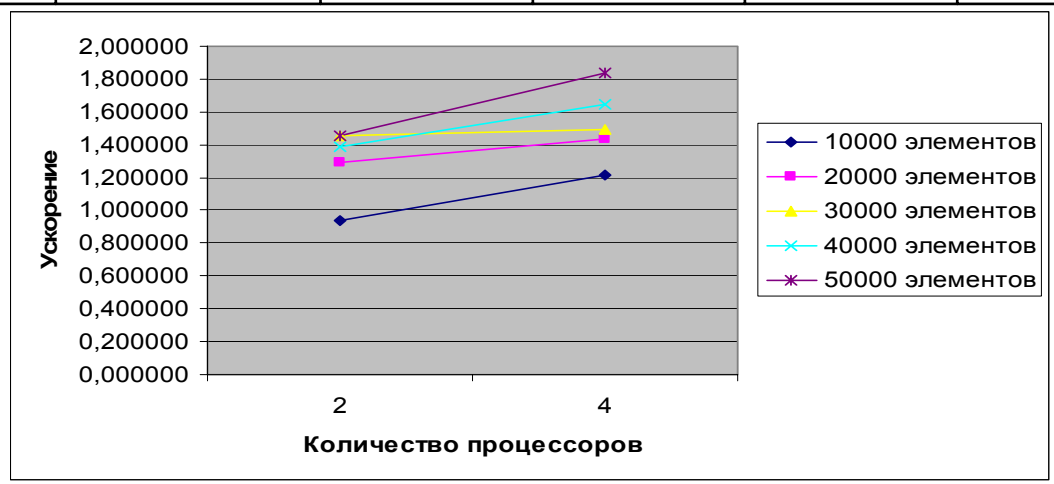


Сортировка с использованием регулярного набора образцов: *параллельный алгоритм*

❑ Результаты вычислительных экспериментов

– Ускорение вычислений

Количество элементов	Последовательный алгоритм	Параллельный алгоритм			
		2 процессора		4 процессора	
		Время	Ускорение	Время	Ускорение
10,000	0,001422	0,001513	0,939855	0,001166	1,219554
20,000	0,002991	0,002307	1,296489	0,002081	1,437290
30,000	0,004612	0,003168	1,455808	0,003099	1,488222
40,000	0,006297	0,004542	1,386394	0,003819	1,648861
50,000	0,008014	0,005503	1,456297	0,004370	1,833867



Заключение

- Рассмотрены способы параллельного выполнения трех широко известных метода упорядочения данных:
  - Алгоритм пузырьковой сортировки,
  - Сортировка Шелла,
  - Быстрая сортировка
- Для алгоритма быстрой сортировки приведены две дополнительные модификации:
  - Обобщенная быстрая сортировка,
  - Сортировка с использованием регулярного набора образцов
- Представлена программная реализация метода обобщенной быстрой сортировки
- Использованный порядок изложения параллельных методов сортировки можно рассматривать как пример процесса последовательного совершенствования параллельных вычислений с целью улучшения показателей ускорения и эффективности

Вопросы для обсуждения

- В чем состоит параллельное обобщение базовой операции сортировки?
- Какие оценки трудоемкости последовательных вычислений следует использовать при определении показателей ускорения и эффективности?
- Какой из рассмотренных алгоритмов обладает наилучшими показателями ускорения и эффективности?
- Какие схемы выбора ведущих элементов могут быть предложены для алгоритма быстрой сортировки?
- Какие операции передачи данных необходимы в параллельных алгоритмах сортировки?

Темы заданий для самостоятельной работы

- Выполните реализацию параллельного варианта алгоритма пузырьковой сортировки.
- Выполните реализацию сортировки Шелла.
- Разработайте параллельный вариант для алгоритма сортировки слиянием. Постройте теоретические оценки времени работы алгоритма.
- Проведите вычислительные эксперименты. Сравните результаты реальных экспериментов с полученными теоретическими оценками

Литература

- **Akl,** S. G. (1985). Parallel Sorting Algorithms. – Orlando, FL: Academic Press
- **Knuth,** D.E. (1973). The Art of Computer Programming: Sorting and Searching. – Reading, MA: Addison-Wesley.
- **Кормен** Т., **Лейзерсон** Ч., **Ривест** Р. (1999). Алгоритмы: построение и анализ. – М.: МЦНТО.
- **Kumar** V., Grama, A., Gupta, A., Karypis, G. (1994). Introduction to Parallel Computing. - The Benjamin/Cummings Publishing Company, Inc. (2nd edn., 2003)
- **Quinn**, M. J. (2004). Parallel Programming in C with MPI and OpenMP. – New York, NY: McGraw-Hill.

Следующая тема

- **Параллельные методы работы с графами**

Авторский коллектив

Гергель В.П., профессор, д.т.н., руководитель

Гришагин В.А., доцент, к.ф.м.н.

Сысоев А.В., ассистент (раздел 1)

Лабутин Д.Ю., ассистент (система ПараЛаб)

Абросимова О.Н., ассистент (раздел 10)

Гергель А.В., аспирант (раздел 12)

Лабутина А.А., магистр (разделы 7,8,9, система ПараЛаб)

Сенин А.В. (раздел 11)

О проекте

Целью проекта является создание образовательного комплекса "Многопроцессорные вычислительные системы и параллельное программирование", обеспечивающий рассмотрение вопросов параллельных вычислений, предусматриваемых рекомендациями Computing Curricula 2001 Международных организаций IEEE-CS и ACM. Данный образовательный комплекс может быть использован для обучения на начальном этапе подготовки специалистов в области информатики, вычислительной техники и информационных технологий.

Образовательный комплекс включает **учебный курс "Введение в методы параллельного программирования" и лабораторный практикум "Методы и технологии разработки параллельных программ"**, что позволяет органично сочетать фундаментальное образование в области программирования и практическое обучение методам разработки масштабного программного обеспечения для решения сложных вычислительно-трудоемких задач на высокопроизводительных вычислительных системах.

Проект выполнялся в Нижегородском государственном университете им. Н.И. Лобачевского на кафедре математического обеспечения ЭВМ факультета вычислительной математики и кибернетики (http://www.software.unn.ac.ru). Выполнение проекта осуществлялось при поддержке компании Microsoft.