University of Nizhni Novgorod
Faculty of Computational Mathematics & Cybernetics

Introduction to Parallel Programming

Section 10.

Parallel Methods for Sorting

Gergel V.P., Professor, D.Sc.,
Software Department

Contents

Problem Statement

Sorting is one of the typical problems of data processing and is usually considered as a problem of locating the elements of unregulated set of values

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

in the order of monotonous increase or decrease

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

Parallelizing Techniques…

- **The basic operation** – " *compare-exchange* "

```
// the basic sorting operation
if ( A[i] > A[j] ) {
  temp = A[i];
  A[i] = A[j];
  A[j] = temp;
}
```

  – The sequential use of the operation makes possible to sort the data,
  – The difference between the sorting algorithms reveals itself in the method of choosing pairs of values for comparison

Parallelizing Techniques…

- **Parallel generalizing of the basic operation when *p* = *n*** (each processor contains a data element):
  – To exchange the values available on the processors P_i and P_j (saving the original elements on the processors),
  – To compare the identical pairs of values (a_i, a_j) on each processor P_i and P_j; according to the results of the comparison to distribute the data between the processors: to locate the smaller element on one of the processors (for instance, P_i), and to store the greater value of the pair on the other processor (i.e. P_j)

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

Parallelizing Techniques…

- **Parallel generalization of the basic operation when *p* < *n*** (each processor contains a data block of n/p size):
  - To sort the block on each processor at the beginning of sorting,
  - To exchange the blocks between the processors P_i and P_{i+1} ,
  - To unite the blocks A_i and A_{i+1} on each processor into a sorted block with the help of merge operation,
  - To split the obtained double block into equal parts and to locate one of the parts (for instance, the one with smaller data values) on processor P_i , and the other part (with the greater data values) – on processor P_{i+1}

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

This procedure is usually referred to as *the “compare-split” operation*

Parallelizing Techniques

- **The result of parallel algorithm execution has to be as follows:**
  - The data available on the processors is sorted,
  - The order of data distribution among the processors corresponds to the linear enumeration order (i.e. the value of the last element on the processor P_i is less or equal to the value of the first element on the processor P_{i+1} for any $0 \le i < p-1$)

Bubble Sort: *Sequential Algorithm*

```
// Sequential algorithm of bubble sorting
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]);
}
```

- The complexity of the computations is $O(n^2)$,
- The algorithm is hard to parallelize in its direct form

Bubble sorting: *Algorithm of odd-even permutation…*

```
// Sequential algorithm of odd-even permutation
OddEvenSort(double A[], int n) {
  for (i=1; i<n; i++) {
    if ( i%2==1) // odd iteration
      for (j=0; j<n/2-1; j++)
        compare_exchange(A[2j+1],A[2j+2]);
     if (i%2==0) // even iteration
       for (j=1; j<n/2-1; j++)
         compare_exchange(A[2j],A[2j+1]);
  }
}
```

Different rules for carrying out odd and even iterations

$\Downarrow$

Suitability of parallelizing

Bubble sorting: *Algorithm of odd-even permutation…*

```
// Parallel algorithm of odd-even permutation
ParallelOddEvenSort ( double A[], int n ) {
  int id = GetProcId();  // process number
  int np = GetProcNum(); // number of processes
  for ( int i=0; i<np; i++ ) {
    if ( i%2 == 1 ) { // odd iteration
      if ( id%2 == 1 ) // odd process number
        compare_split_min(id+1); // compare-exchange to the right
      else compare_split_max(id-1); // compare-exchange to the left
    }
    if ( i%2 == 0 ) { // even iteration
      if( id%2 == 0 )  // even process number
        compare_split_min(id+1); // compare-exchange to the right
      else compare_split_max(id-1); // compare-exchange to the left
}
  }
}
```

Bubble sorting: *Algorithm of odd-even permutation…*

- **Efficiency analysis…**
  - The general estimation of efficiency and speedup characteristics:

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

Bubble sorting: *Algorithm of odd-even permutation…*

- **Efficiency analysis** (detailed estimates):

  - Time of parallel algorithm execution, that corresponds to the processor calculations:

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

  - The duration of data accumulation in case of the Hockney model is determined by means of the following equation:

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

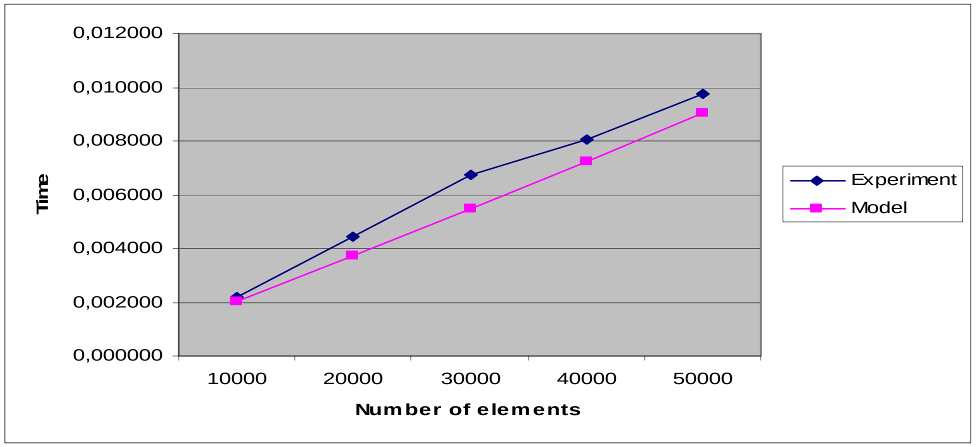
The total execution time for the parallel algorithm:

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

Bubble sorting: *Algorithm of odd-even permutation…*

- **Results of computational experiments…**
  – Comparison of theoretical estimations and experimental data T_2^*

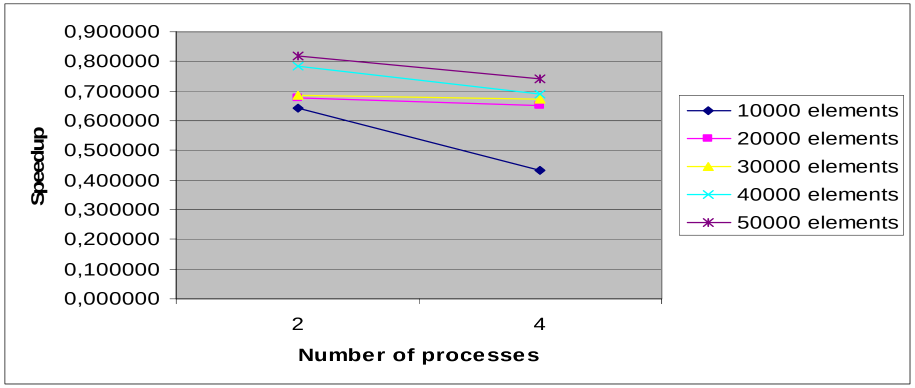
Data size	Parallel algorithm			
	2 processors		4 processors	
10,000	0.002003	0.002210	0.002057	0.003270
20,000	0.003709	0.004428	0.003366	0.004596
30,000	0.005455	0.006745	0.004694	0.006873
40,000	0.007227	0.008033	0.006035	0.009107
50,000	0.009018	0.009770	0.007386	0.010840



Bubble sorting: *Algorithm of odd-even permutation…*

- **Results of computational experiments:**
  - Speedup

Number of elements	Sequential algorithm	Parallel algorithm			
		2 processors		4 processors	
		Time	Speedup	Time	Speedup
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



Bubble sorting: *algorithm of odd-even permutation*

- The parallel variant of the algorithm operates more slowly than the original sequential bubble sorting method:
  - The amount of the data transmitted among the processors is quite big and can be compared to the number of the executed computational operations,
  - This misbalance of the amount of computations and the complexity of the data communication operations rises as the number of processors increases

Shell sort: *Sequential algorithm…*

- The general concept of the Shell sort is the comparison of the pairs of values located rather far from each other in the set of values to be ordered at the initial stages of sorting (sorting such pairs requires, as a rule, a big number of permutation operations, if only neighboring elements are compared):
  - At the first step of the algorithm the elements $n/2$ pairs $(a_i, a_{n/2+i})$ for $1 \leq i \leq n/2$ are sorted,
  - At the second step the elements in $n/4$ groups of four elements $(a_i, a_{n/4+1}, a_{n/2+1}, a_{3n/4+1})$ for $1 \leq i \leq n/4$ are sorted etc.,
  - At the last step the elements of all the array $(a_1, a_2, \ldots, a_n)$ are sorted.
- The total number of the Shell algorithm iterations is equal to $\log_2 n$

Shell sort: *Sequential algorithm*

```
// Sequential algorithm of Shell sort
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;
  }
}
```

- The complexity of the computations is $O(n\log_2 n)$

Shell sort: *Parallel algorithm…*

Let the communication network topology be an ***N***-dimensional hypercube (i.e. the number of processors is equal to $p=2^N$).

The algorithm operates as follows:

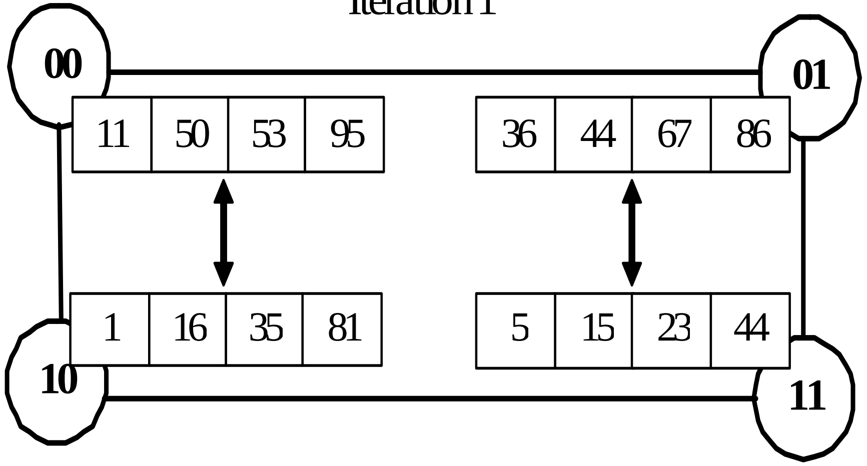
– *The first stage (**N** iterations)*: the operation “compare-split” is executed for each pair of processors in the hypercube. The pairs of processors are formed according to the rule: at each iteration ***i, 0 ≤ i < N***, the processors whose numbers differ in their bit presentation only in position ***N-I*** form a pair,

– *The second stage*: the execution of parallel algorithm iterations of odd-even permutation. The iterations are carried out until the actual change of the sorted set is terminated. Their total number ***L*** may vary from ***2*** to ***p***.

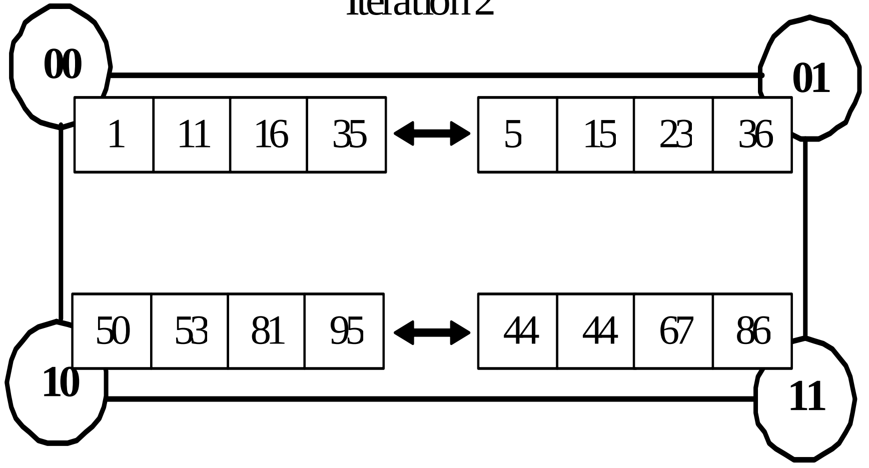
Shell sort: *Parallel algorithm…*

Example:

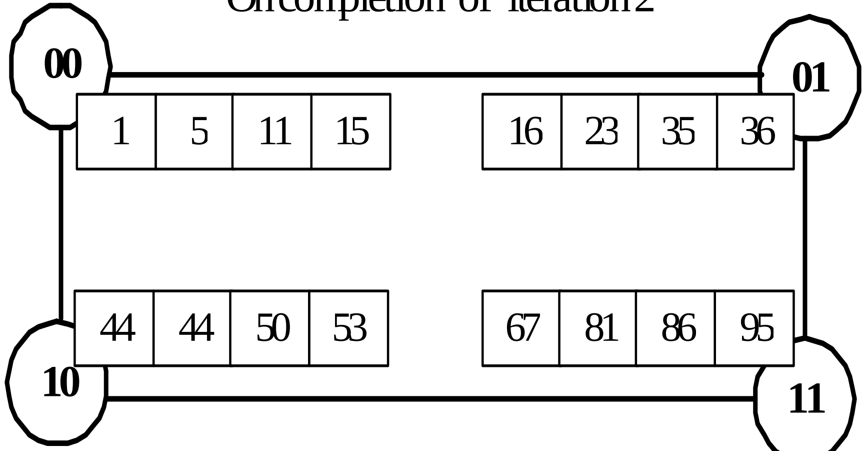
Iteration 1



Iteration 2



On completion of iteration 2



Shell sort: *Parallel algorithm…*

- **Efficiency analysis…**
  - The general estimation of speedup and efficiency:

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

Shell sort: *Parallel algorithm…*

- **Efficiency analysis** (detailed estimates):

  - Time of parallel algorithm execution, that corresponds to the processor calculations:

$$T_p(calc) = \left((n/p)\log_2(n/p) + \left(\log_2 p + L\right)\cdot\left(2n/p\right)\right)\tau,$$

  - The duration of data accumulation in case of the Hockney model is determined by the following equation:

$$T_p(comm) = (\log_2 p + L)\cdot\left(\alpha + w\cdot\left(n/p\right)/\beta\right)$$

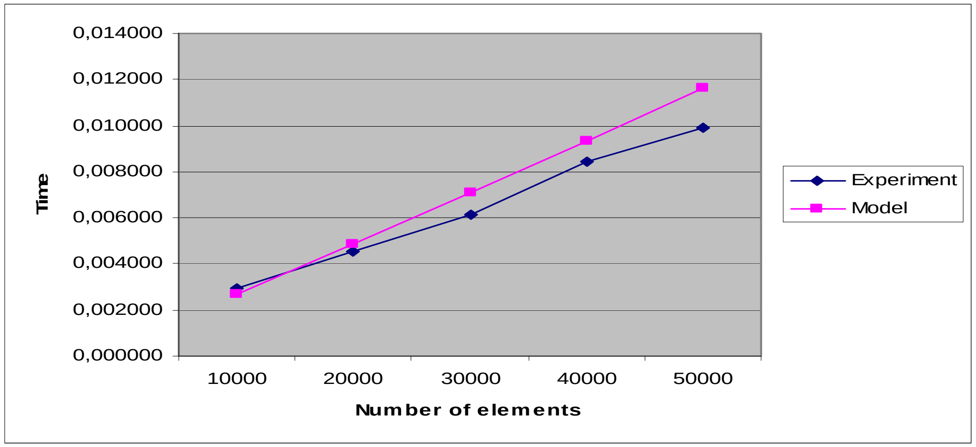
The total execution time of the parallel algorithm:

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

Shell sort: *Parallel algorithm…*

- **Results of computational experiments…**
  - Comparison of theoretical estimations and experimental data

Number of elements	Parallel algorithm			
	2 processors		4 processors	
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

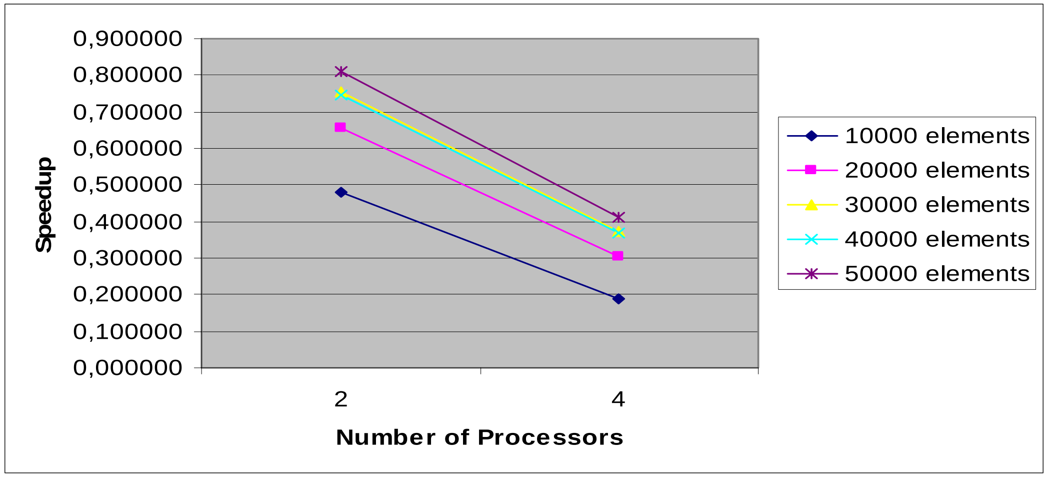


Shell sort: *Parallel algorithm*

❑ Results of computational experiments:

– Speedup

Number of elements	Sequential algorithm	Parallel algorithm			
		2 processors		4 processors	
		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



Quick sort: *Sequential algorithm…*

- *The quick sort algorithm* proposed by *Hoare C.A.R.* is based on sequential partitioning of the data set being sorted into blocks of smaller sizes so that order relations are provided between the values of different blocks (for any pair of blocks all the values of one of them do not exceed the values of the other one):
  - At the first iteration of the method the initial data set is split into first two parts; a certain *pivot element* is selected to arrange this splitting, all the set values, which are less than the pivot element, are transferred to the first block being formed, all the other values create the second block of the set,
  - At the second iteration the above described rules are applied recursively to both blocks, which have been formed, etc.
- If the choice of the pivot elements is optimal, the initial data array appears to be sorted after executing $\log_2 n$ iterations

Quick sort: *Sequential algorithm*

```
// Sequential algorithm of quick sort
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);
  }
}
```

The average number of operations is $1.4n\log_2 n$

Quick sort: *Parallel algorithm*…

Let the communication network topology be an ***N***-dimensional hypercube (i.e. the number of processors is equal to $p=2^N$).

The algorithm operates as follows:

– To choose the pivot element and send it to all the processes (for instance, the arithmetic mean of the elements located on the pivot processor can be chosen as the pivot element),

– To split the block available on each processor into two parts using the obtained pivot element,

to be continued

Quick sort: *Parallel algorithm*…

– To create the pairs of processors, for which the bit presentations of their numbers differ only in position ***N***, and exchange the data between the processors; as a result of these data communication operations, the parts of blocks with the values less than the pivot element must appear on the processors, for which the number of bit position N is equal to 0 in the bit presentation of the numbers; the processors with the numbers, where bit N is equal to 1 must gather correspondingly all the data values, which exceed the value of the pivot element,

– To pass over to the subhypercube of smaller dimension and repeat of the above described procedure.

Quick sort: *Parallel algorithm*…

- **Efficiency analysis…**
  - The general estimation of speedup and efficiency:

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

$$E_p = \frac{n\log_2 n}{p\cdot((n/p)\log_2(n/p) + \log_2 p \cdot (2n/p))}$$

Quick sort: *Parallel algorithm*…

- **Efficiency analysis** (detailed estimates):
  - Time of parallel algorithm execution, that corresponds to the processor calculations:

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

  - The duration of data accumulation in case of the Hockney model is determined by the following equation:

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

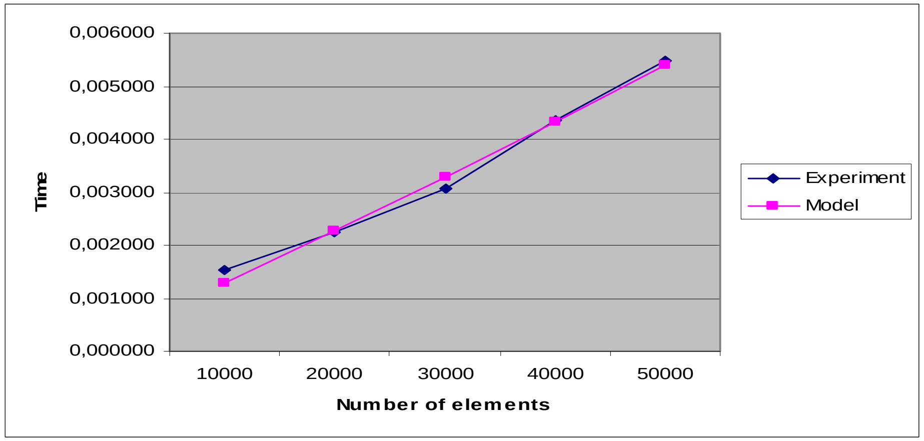
The total execution time of the parallel algorithm:

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

Quick sort: *Parallel algorithm*…

- **Results of computational experiments…**
  - Comparison of theoretical estimations and experimental data

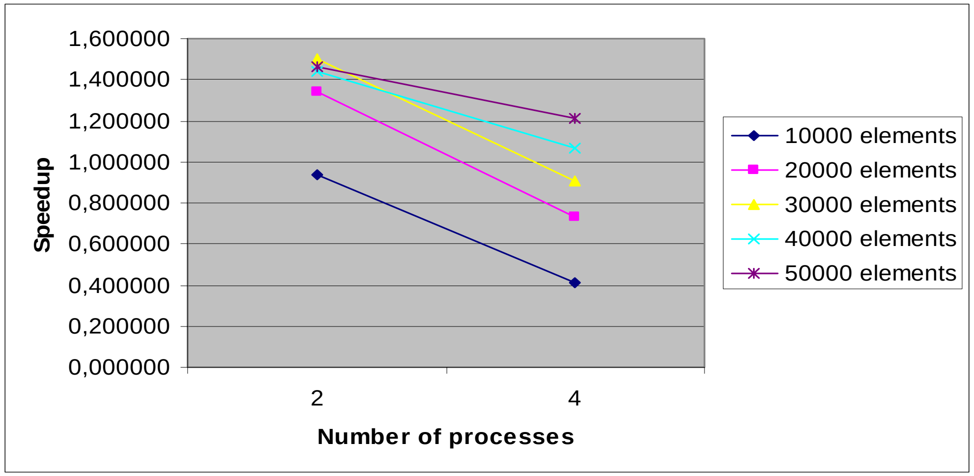
Data size	Parallel algorithm			
	2 processors		4 processors	
10,000	0.001280	0.001521	0.001735	0.003434
20,000	0.002265	0.002234	0.002321	0.004094
30,000	0.003289	0.003080	0.002928	0.005088
40,000	0.004338	0.004363	0.003547	0.005906
50,000	0.005407	0.005486	0.004175	0.006635



Quick sort: *Parallel algorithm*

- **Results of computational experiments:**
  - Speedup

Number of elements	Sequential algorithm	Parallel algorithm			
		2 processors		4 processors	
		Time	Speedup	Time	Speedup
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



HyperQuick sort: *Parallel algorithm…*

- The main difference between this algorithm and the previous one consists in the method of choosing the pivot element
- The block sorting is executed at the very beginning of computations. The average element of some block is chosen as the pivot element (for instance, on the first processor of the computer system). The pivot element selected in such a way appears in some cases to be closer to the real mean value of the sorted set than any other arbitrarily chosen value
- To keep ordering the values in the course of computations, the processors carry out the operation of merging the parts of blocks obtained after splitting

HyperQuick sort: *Parallel algorithm…*

- **Efficiency analysis…**
  - The general estimation of speedup and efficiency:

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

$$E_p = \frac{n\log_2 n}{p\cdot((n/p)\log_2(n/p) + \log_2 p \cdot (2n/p))}$$

HyperQuick sort: *Parallel algorithm…*

- **Efficiency analysis** **(**detailed estimations**):**
  - Time of parallel algorithm execution, that corresponds to the processor calculations:

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

  - The duration of data accumulation in case of the Hockney model is determined by the following equation:

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

The total execution time of the parallel algorithm:

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

HyperQuick sort: *Parallel algorithm…*

- **Description of the parallel program sample…**
  - **First stage**: Data initialization and data distribution among the processors:
    - Obtaining the size of the sorted array,
    - Determining the initial data for the sorted array,
    - Distributing the initial data among the processors is presented by the function *DataDistribution*.

Code

HyperQuick sort: *Parallel algorithm…*

- **Description of the parallel program sample…**

  – **<u>Second stage</u>**: the execution of the HyperQuick sorting iteration is implemented in the function ***ParallelHyperQuickSort***

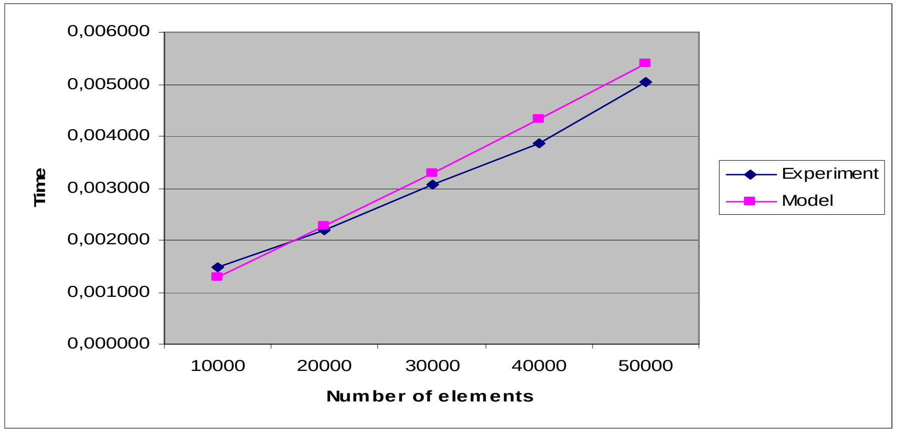
Code

HyperQuick sort: *Parallel algorithm…*

❑ Results of computational experiments…

– Comparison of theoretical estimations and experimental data T_2^*

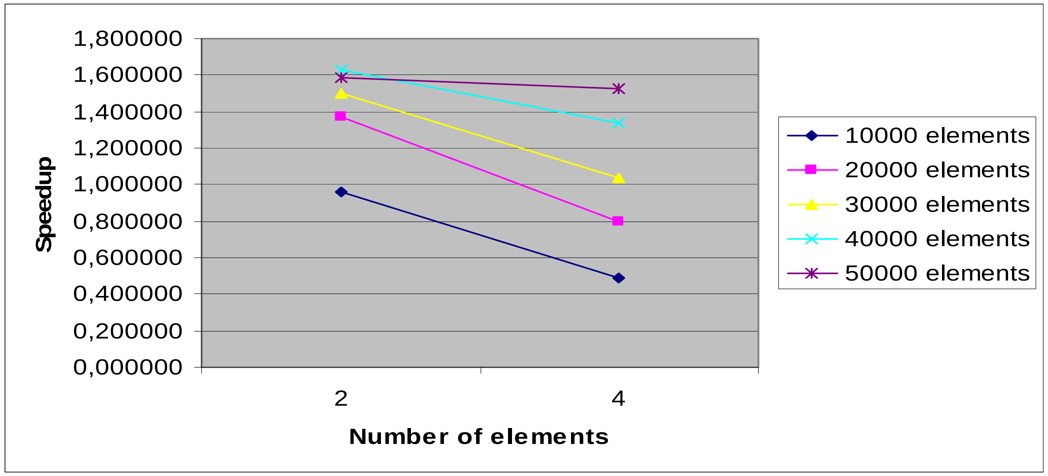
Data size	Parallel algorithm			
	2 processors		4 processors	
10,000	0.001281	0.001485	0.001735	0.002898
20,000	0.002265	0.002180	0.002322	0.003770
30,000	0.003289	0.003077	0.002928	0.004451
40,000	0.004338	0.003859	0.003547	0.004721
50,000	0.005407	0.005041	0.004176	0.005242



HyperQuick sort: *Parallel algorithm*

- **Results of computational experiments:**
  - Speedup

Number of elements	Sequential algorithm	Parallel algorithm			
		2 processors		4 processors	
		Time	Speedup	Time	Speedup
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



Sorting by Regular Sampling: *Parallel Algorithm…*

The first stage: the blocks are sorted on each processor independently by means of the conventional quick sorting algorithm; each processor further forms a set of the elements of its blocks with the indices ***0, m, 2m,…,(p-1)m***, where ***m=n/p²***,

The second stage: all the data sets created on the processors are gathered on one of the system processors and united in the course of sequential merge into a sorted set; the obtained set of values of the elements with the indices

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

is the basis for forming the set of the pivot elements, which is transmitted to all the processors; at the end of this stage, each processor splits its block into ***p*** parts using the obtained set of the pivot values,

to be continued

Sorting by Regular Sampling: *Parallel Algorithm…*

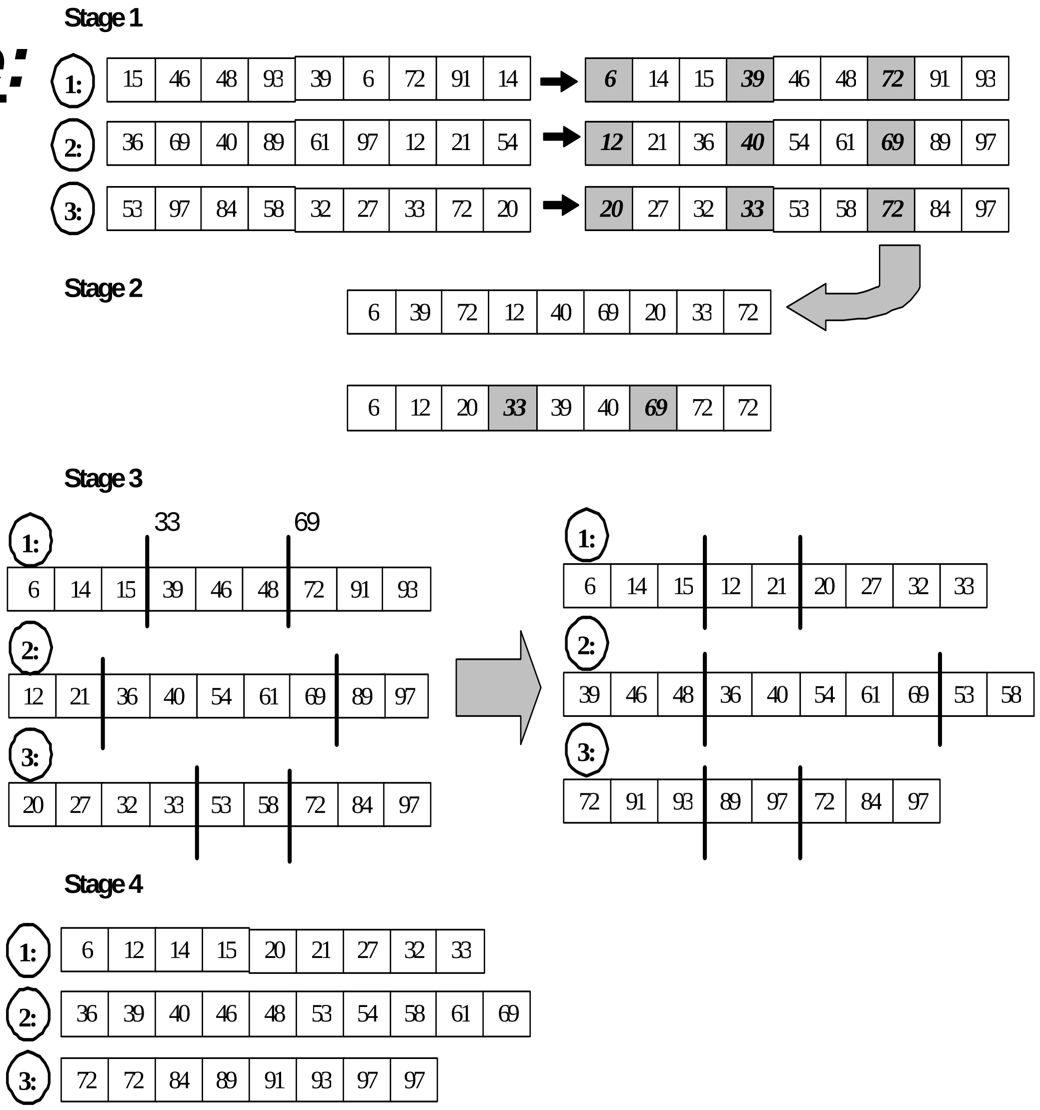
The third stage: each processor broadcasts the parts of its block to the other processors in accordance with the enumeration order - part ***j***, $0 \le j < p$, of each block is transmitted to the processor number ***j***,

The fourth stage: each processor performs merging ***p*** obtained parts into a sorted block.

Sorting by Regular Sampling: *Parallel Algorithm…*

Example:

(p=3)



Sorting by Regular Sampling: *Parallel Algorithm…*

Efficiency analysis (detailed estimations):

- The execution time of the first parallel algorithm stage:

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

- The execution time of the second parallel algorithm stage:

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

- The execution time of the third parallel algorithm stage:

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

- The execution time of the fourth parallel algorithm stage:

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

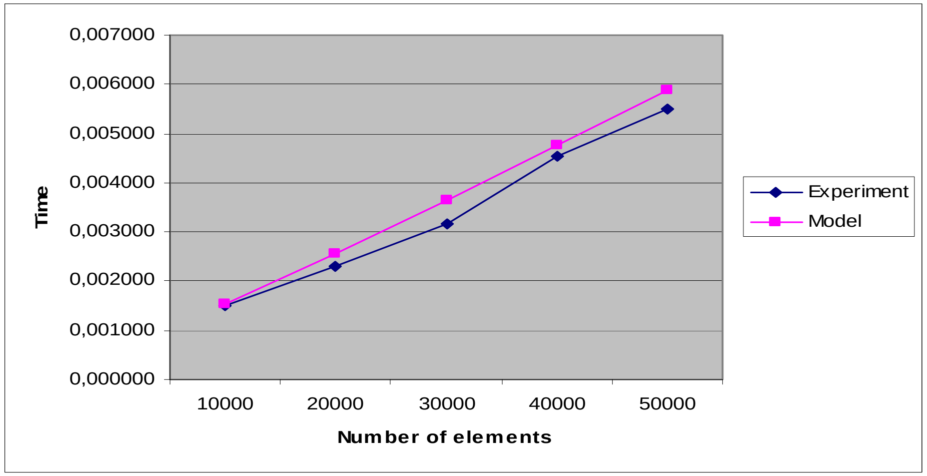
Total time of parallel algorithm execution:

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

Sorting by Regular Sampling: *Parallel Algorithm…*

- **Results of computational experiments…**
  - Comparison of theoretical estimations the experimental data T_2^*

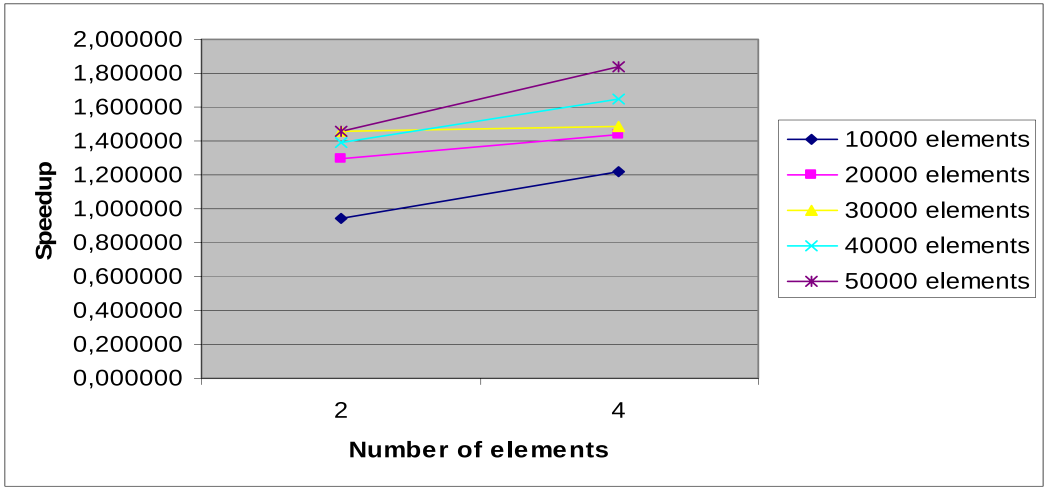
Data size	Parallel algorithm			
	2 processors		4 processors	
10,000	0.001533	0.001513	0.001762	0.001166
20,000	0.002569	0.002307	0.002375	0.002081
30,000	0.003645	0.003168	0.003007	0.003099
40,000	0.004747	0.004542	0.003652	0.003819
50,000	0.005867	0.005503	0.004307	0.004370



Sorting by Regular Sampling: *Parallel Algorithm*

- **Results of computational experiments:**
  - Speedup

Number of elements	Sequential algorithm	Parallel algorithm			
		2 processors		4 processors	
		Time	Speedup	Time	Speedup
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



Summary

- The following methods of parallel data sorting are described:
  - Bubble sort,
  - Shell sort,
  - Quick sort
- The two additional variants are described for the quick sorting algorithm:
  - HyperQuick sort,
  - Sorting by regular sampling
- Software implementation of the HyperQuick sorting is presented
- The order of adducing the parallel sorting methods can be considered as an example of step-by-step modifications of parallel methods aimed at improving the speedup and efficiency characteristics

Discussions

- What is the essence of the parallel generalization of the basic sorting operation?
- What complexity estimations of the sequential computations should be used for determining the speedup and efficiency characteristics?
- Which of the above described algorithms possesses the highest speedup and efficiency?
- What schemes of choosing the pivot elements may be suggested for the quick sorting algorithm?
- What data communication operations are required in parallel sorting algorithms?

Exercises

- Develop the implementation of the parallel variant of the bubble sorting algorithm.
- Develop the implementation of the Shell sort.
- Develop the parallel variant for the algorithm of sorting by merging. Formulate the theoretical estimations of the algorithm execution time.
- Carry out computational experiments. Compare the theoretical estimations to the experimental data.

References

- **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.
- **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.

Next Section

- Parallel Methods of Graph Calculations

Author’s Team

Gergel V.P., Professor, Doctor of Science in Engineering, Course Author

Grishagin V.A., Associate Professor, Candidate of Science in Mathematics

Abrosimova O.N., Assistant Professor (chapter 10)

Kurylev A.L., Assistant Professor (learning labs 4,5)

Labutin D.Y., Assistant Professor (ParaLab system)

Sysoev A.V., Assistant Professor (chapter 1)

Gergel A.V., Post-Graduate Student (chapter 12, learning lab 6)

Labutina A.A., Post-Graduate Student (chapters 7,8,9, learning labs 1,2,3, ParaLab system)

Senin A.V., Post-Graduate Student (chapter 11, learning labs on Microsoft Compute Cluster)

Liverko S.V., Student (ParaLab system)

About the project

The purpose of the project is to develop the set of educational materials for the teaching course “Multiprocessor computational systems and parallel programming”. This course is designed for the consideration of the parallel computation problems, which are stipulated in the recommendations of IEEE-CS and ACM Computing Curricula 2001. The educational materials can be used for teaching/training specialists in the fields of informatics, computer engineering and information technologies. The curriculum consists of **the training course “Introduction to the methods of parallel programming”** and **the computer laboratory training “The methods and technologies of parallel program development”**. Such educational materials makes possible to seamlessly combine both the fundamental education in computer science and the practical training in the methods of developing the software for solving complicated time-consuming computational problems using the high performance computational systems.

The project was carried out in Nizhny Novgorod State University, the Software Department of the Computing Mathematics and Cybernetics Faculty (http://www.software.unn.ac.ru). The project was implemented with the support of Microsoft Corporation.