Institute of Computer Science
Department of Mathematics and Computer Science

Freie Universität Berlin

Algorithms and Programming IV

Design and Implementation of Parallel Applications II

Summer Term 2023 | 07.06.2023

Barry Linnert

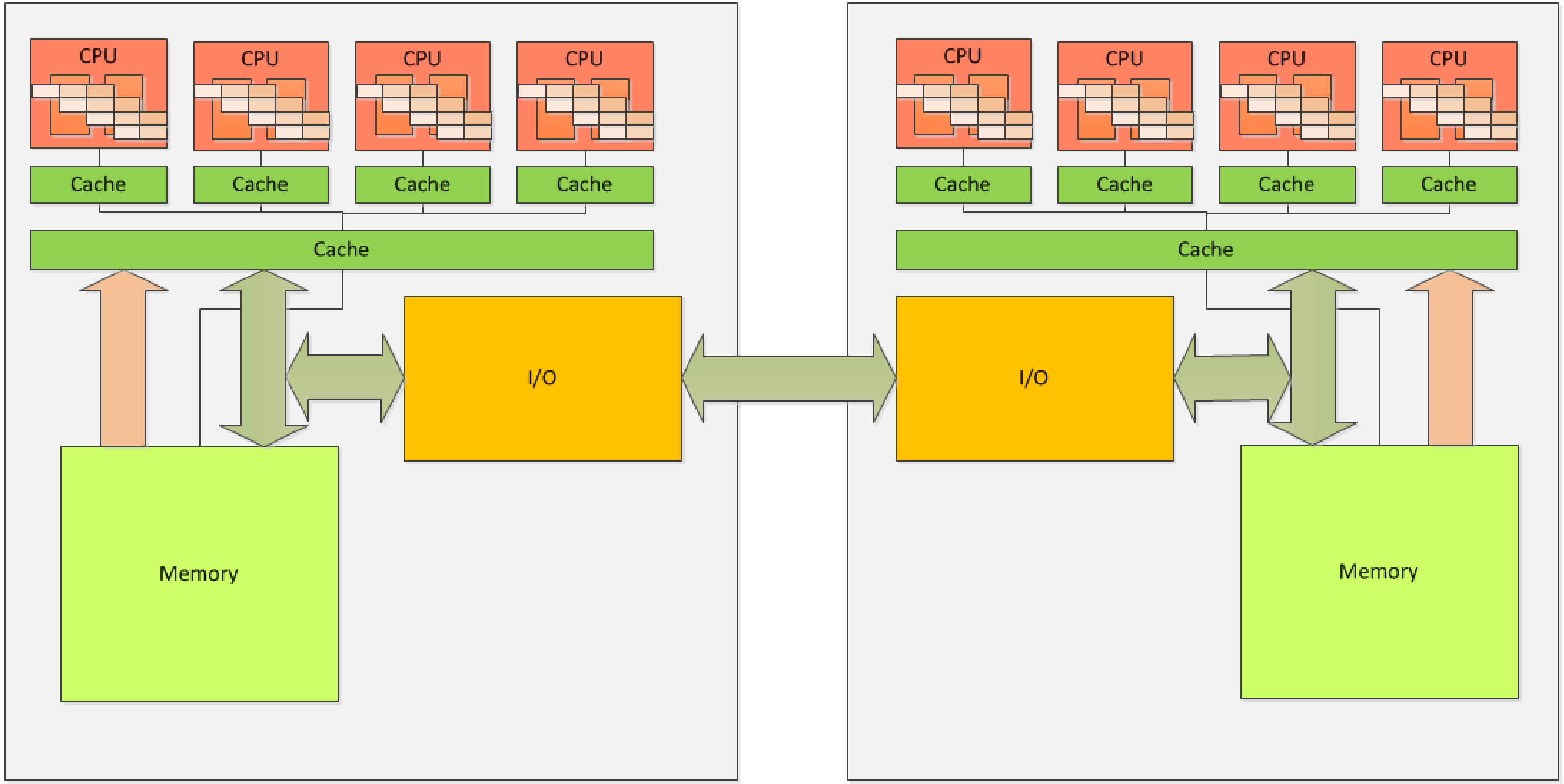
Objectives of Today‘s Lecture

- N-body simulation
- Sorting

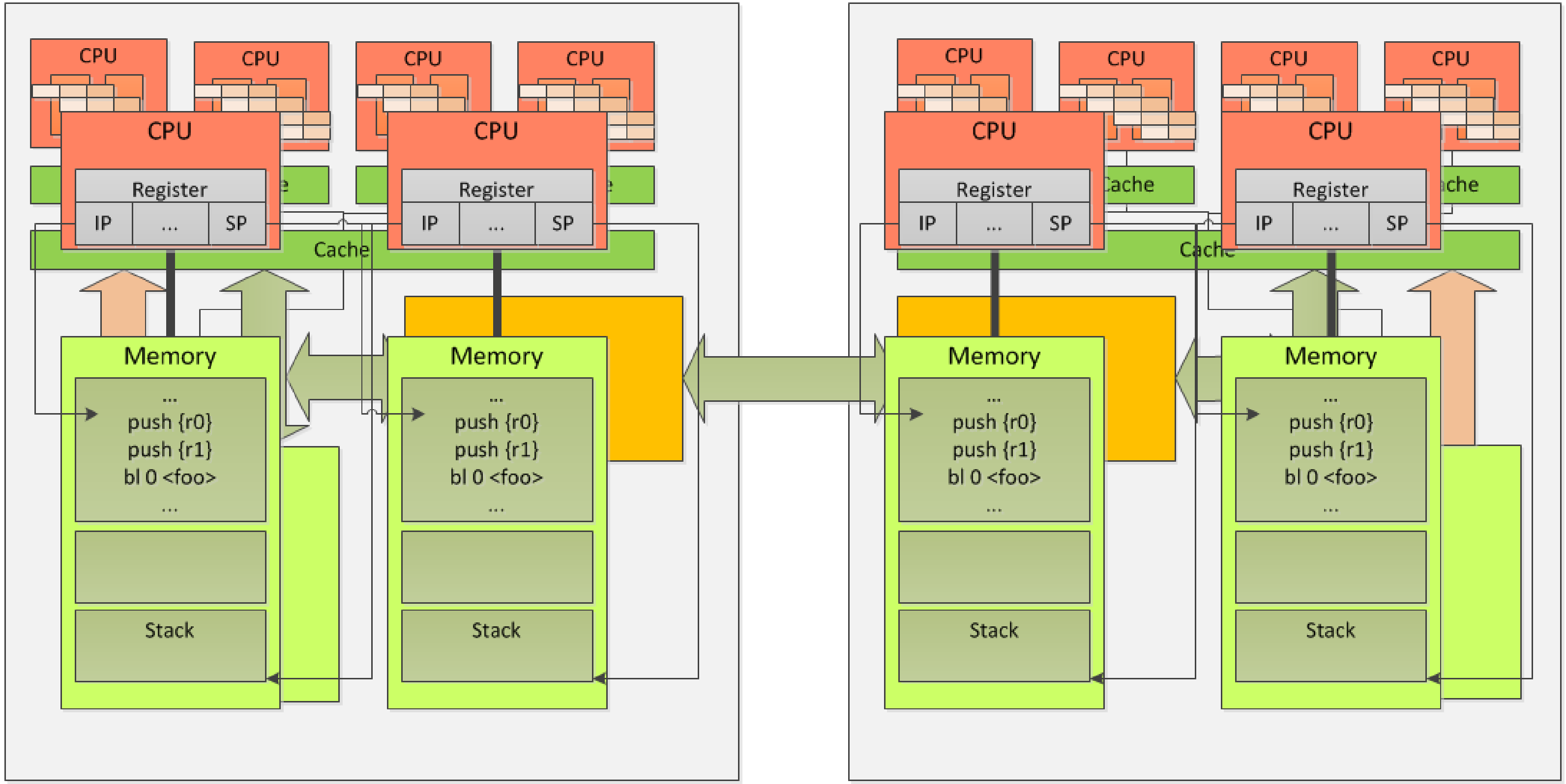
Concepts of Non-sequential and Distributed Programming

DESIGN AND IMPLEMENTATION OF PARALLEL APPLICATIONS

Machine Model

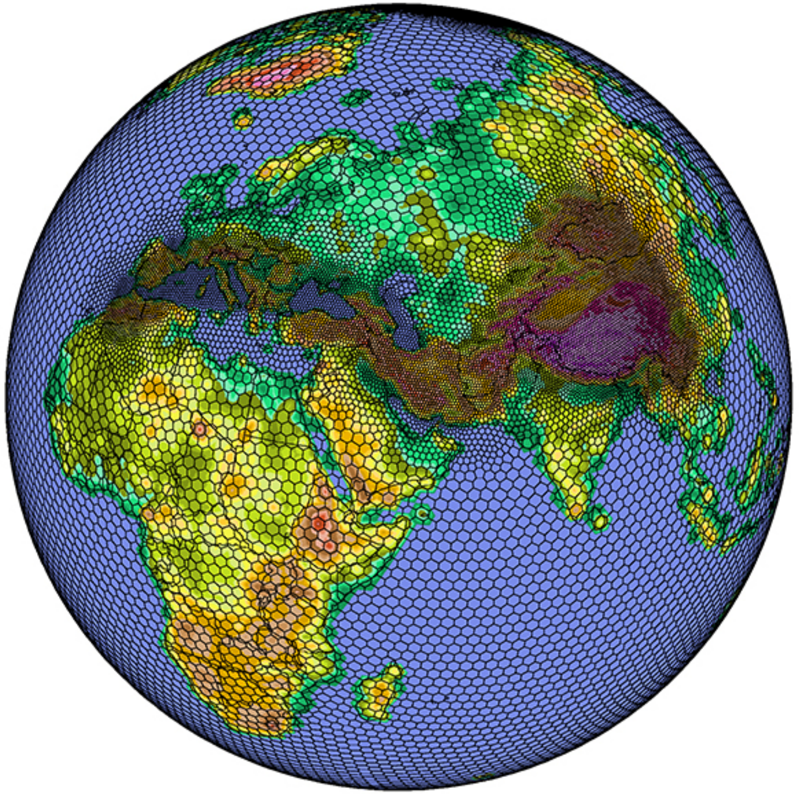


Machine and Execution Model

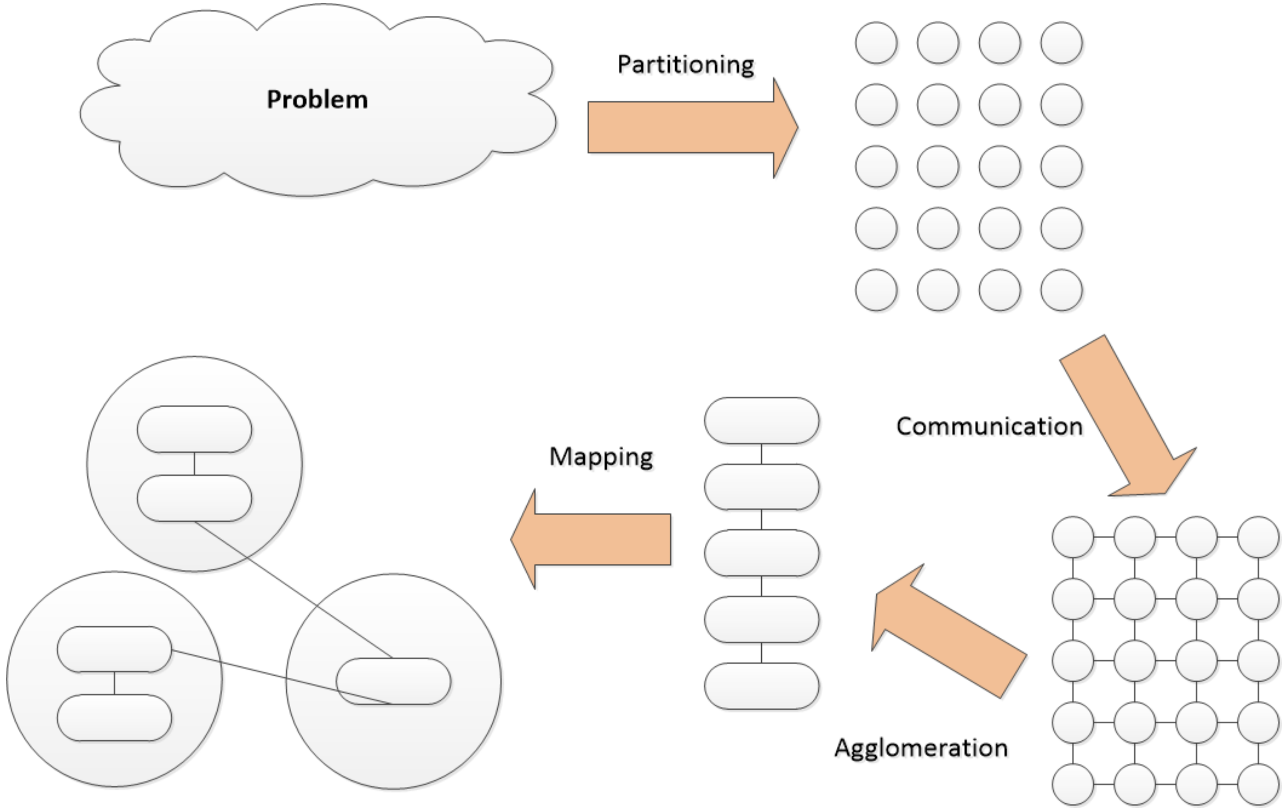


MPI and MPI-2

- With MPI and MPI-2 the foundation to design and implement parallel programs using message passing is given.
- Thus, the design methodology by Ian Foster is applicable.
- The functions coming with the MPI-2 standard extension can be used additionally to design parallel programs with dynamic runtime behavior, such as:
  - Ocean Land Atmosphere Model
    - http://olam-soil.org



Foster's Design Methodology



Task/Channel (Programming) Model

- The Task/Channel (Programming) Model serves as foundation for Forster‘s design methodology.
- A Task of the task/channel model is a part of the application with its own address space (process).
- Tasks can exchange data via messages using channels.
- A channel is a message queue connecting two specific tasks.
- If a task wants to receive a message, the task waits until the message is received (is blocked).
- Messages are sent immediately by the sender. The sender does not wait until the message is received.
- Thus, the task/channel model implements synchronous receive and asynchronous sending.

Concepts of Non-sequential and Distributed Programming

EXAMPLE: N-BODY SIMULATION

N-Body Simulation

- An N-body simulation is a simulation of physical or astronomical processes.
- It simulates mass points in space (our example is using a two dimension space only).
- The mass points have a mass but no size.
- The mass points have a position (current) and are moving and attract each other according to the gravitational constant.

- Extensions may include merging of colliding mass points, volume based masses (with diameter), attenuation of the space.

N-Body Simulation – Algorithm

- Each step of the simulation has to perform following tasks for every mass point:
- The motion vector for the mass point is calculated with respect to every other mass point (position and distance).
- The new position of the mass point is determined based on the motion vector and the time resolution of the simulation step.
- The result of the calculation is stored to be used in the next step of the simulation.

N-Body Simulation – Partitioning

- Which (preferably independent) tasks (elements of work) can be identified?

- The tasks of a single step of the simulation depend on the data of the mass points.
- Thus, function based decompositions will come with a reasonable amount of overhead.

- The domain-/data-based decomposition is much more promising:
  - The amount of tasks scales with the amount of mass points.
  - Usually there are much more mass points to be processed as processors to be used.

- Thus, a tasks combines the functions to be executed on a single mass point in order to calculate and store the new position.

N-Body Simulation – Communication

- What is the data that has to be transferred (from which task to which other task)?
- Which transfer of data depends on which other activities (calculation, communication)?

- Every task has to provided with the position and mass of all (other) mass points.
- The new position of the specific mass point has to be transmitted to all other tasks to be used in the next step of simulation.

N-Body Simulation – Agglomeration

- Which tasks can be combined as long as the number of available processors is (much) smaller than the number of combined tasks (now processes)?
- How can the aggregation of the tasks be used to reduce communication between the processes?

- As the communication is uniformly distributed (all tasks send the new position of the mass point to all other tasks), the distribution of load should be crucial for the combination of the tasks.
- The mass points should be uniformly distributed between the processes to be formed.
- The combination of tasks will reduce the communication costs as well.

N-Body Simulation – Mapping

- Which combined tasks (processes) should be assigned to which processor?

- How can communication costs between the processes can be reduced?
  - As there is an all-to-all communication the mapping of the specific process in relation to others will not have an impact on the communication cost.
  - (Thus, the nodes of the parallel computer running the N-body simulation should be in the neighborhood regarding the network topology. For this and more about management of parallel computers you may take the course Cluster Computing.)
- Are there any requirements coming with the distribution of (calculation) load?
  - If the agglomeration produces as many combined tasks as there are processors available and the corresponding processes all do the same work (effort of marking the multiples), the load is balanced.

N-Body Simulation – Programming Model

- The approach of the sequential algorithm is to be extended to provide every task with the new data about the mass points for the next step of simulation.
- Thus, there is a combination of calculating the new coordinates for the mass point and the communication of these new coordinates.
- The approach follows the BSP (bulk synchronous parallel) programming model.

Concepts of Non-sequential and Distributed Programming

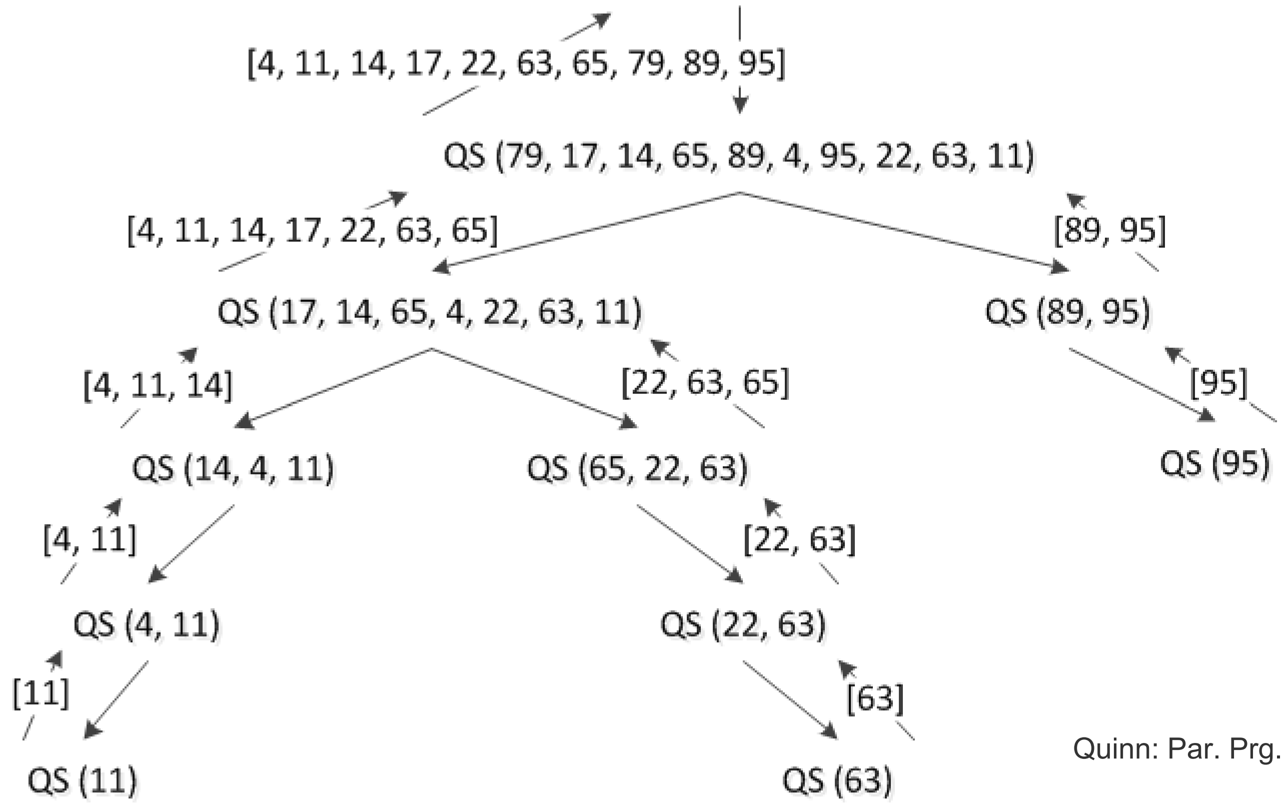
EXAMPLE: SORTING

Sorting

- Quicksort is a well known sorting algorithm and was introduced by C. A. R. Hoare in 1962.
- Quicksort is a recursive algorithm and follows the divide-and-conquer approach.

- Algorithm:
  - An element (pivot) from the sequence of elements is selected.
  - All elements of the sequence are compared with the pivot element and reorganized into two areas during the comparison step (partitioning). The first area contains the elements that are smaller than the pivot element and the second area contains all elements that are larger or equal. At the end of the step, the pivot element is positioned between the two areas.
  - After each partitioning, the Quicksort algorithm is executed recursively with both subareas (as long as the subareas contain more than one element).

Sorting – Quicksort



Quinn: Par. Prg. in C with MPI + OMP

Quicksort – Partitioning

- Which (preferably independent) tasks (elements of work) can be identified?

- A function-based decomposition is not applicable here, as every step of the recursive algorithm depends on the other (at least of the determination of the pivot element).
- A static domain- or data-based decomposition is not possible, as the parts of the sequence of elements to be sorted are based on the determination of the pivot element at runtime.
- Thus, a combination of function-based and data-based decomposition is to be used.

Quicksort – Partitioning II

- For every recursion, the sub-lists (areas) of the sequence of elements are represented by different tasks.
- The sorting of the sub-lists at the lowest recursion level can be performed independently.
  - The merging of the sub-lists requires synchronization again.

Quicksort – Communication

- What is the data that has to be transferred (from which task to which other task)?
- Which transfer of data depends on which other activities (calculation, communication)?

- The sub-lists must be transferred to the tasks (processes) that have been created.
- After sorting, the sub-lists must be received and the combined (sub)list must be returned to the higher level.

Quicksort – Agglomeration

- Which tasks can be combined as long as the number of available processors is (much) smaller than the number of combined tasks (now processes)?
- How can the aggregation of the tasks be used to reduce communication between the processes?

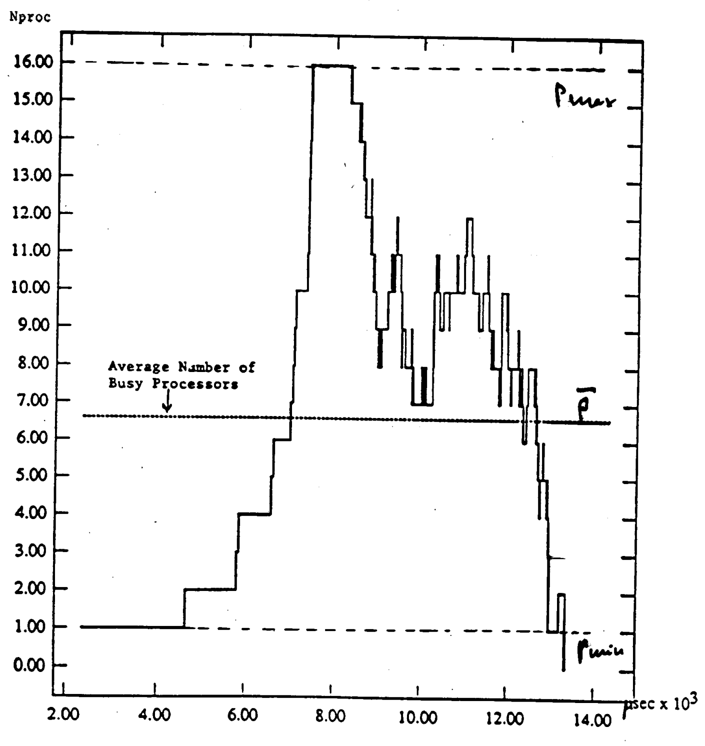
- The sorting can be performed independently by the tasks, but the combination of tasks working on different areas of the list of elements would come with additional communication costs.
- Thus, the combined tasks should work on the same sub-list (at higher level).
- As the amount of sub-lists generated recursively depends on the choice of the pivot element a uniform load distribution cannot be guaranteed.

Quicksort – Mapping

- Which combined tasks (processes) should be assigned to which processor?

- As processes may be created in order to process a sub-list these processes should be mapped near to the creating process to reduce communication costs for the transfer of the list to the new processes and for the transfer of the results back to the process dealing with higher level of recursion.

Example: Quicksort on 16 Processors



Example: Hyperquicksort

- B. Wagar in 1987 introduced a parallel Quicksort algorithm aimed at Hypercube architectures.
- The Hyperquicksort approach bases on a combination of partitioning and agglomeration with respect to load balancing.
  - What about communication costs?

- Algorithm:
  - Splitting the data in pieces corresponding to the number of processes to be used.
  - Each process sorts the sub-list assigned to.
  - Exchange of parts of the sorted sub-list between processes.
  - Sorting the new sub-list.

Example: Hyperquicksort I

97	48	16	8	66	96	17	49	58	76	54	39	82	47	65	51	11	50	53	95	36	67	86	44	35	16	81	1	44	23	15	5

Example: Hyperquicksort II

- Splitting data regarding the amount of processes:

97	48	16	8	66	96	17	49	58	76	54	39	82	47	65	51	11	50	53	95	36	67	86	44	35	16	81	1	44	23	15	5

Example: Hyperquicksort III

- Distribution of data to the processes:

11	50	53	95	36	67	86	44

35	16	81	1	44	23	15	5

97	48	16	8	66	96	17	49

58	76	54	39	82	47	65	51

Example: Hyperquicksort IV

- Sorting of local data of the processes:

11	36	44	50	53	67	86	95

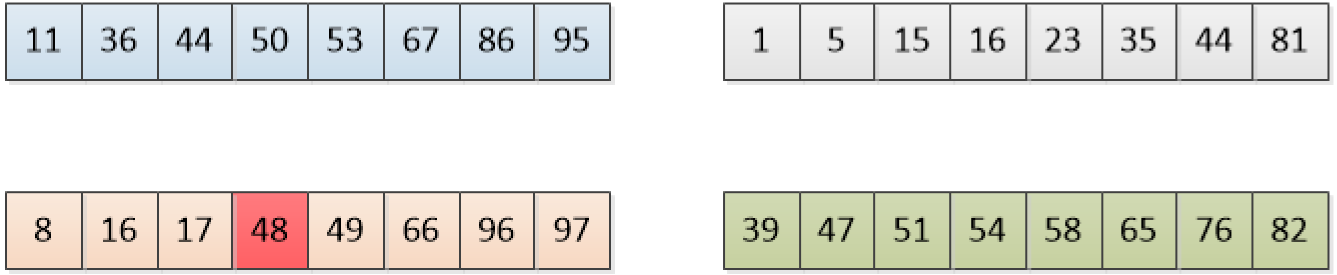
1	5	15	16	23	35	44	81

8	16	17	48	49	66	96	97

39	47	51	54	58	65	76	82

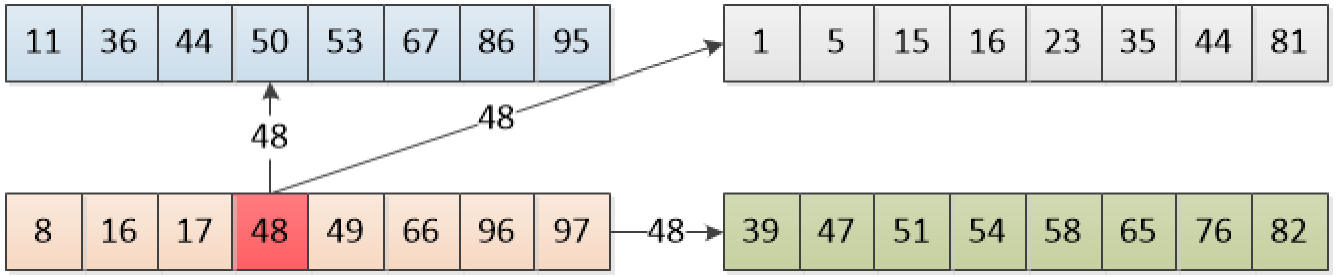
Example: Hyperquicksort V

- Determination of a pivot element:



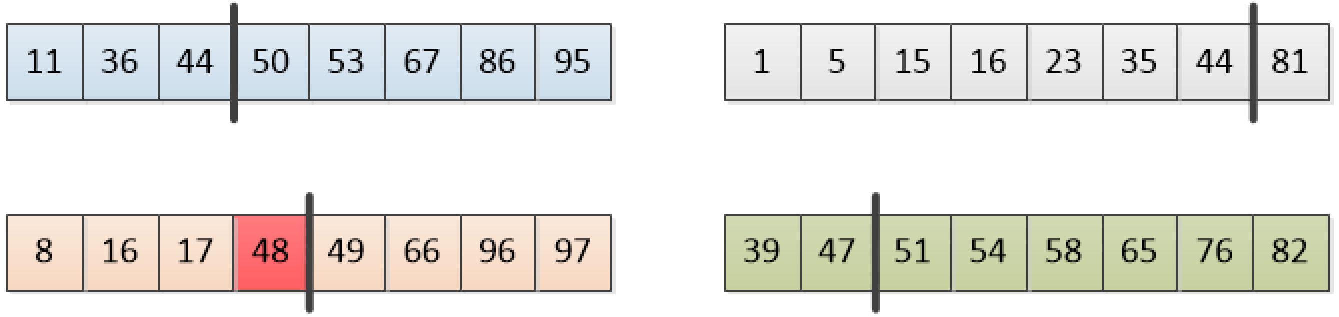
Example: Hyperquicksort VI

- Communication of the pivot element to all other processes:



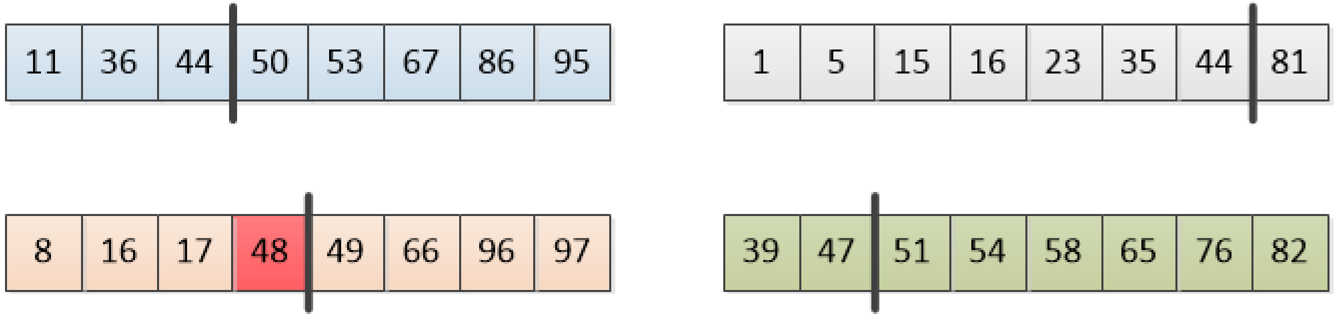
Example: Hyperquicksort VII

- Using the pivot element to split the local data into pieces:



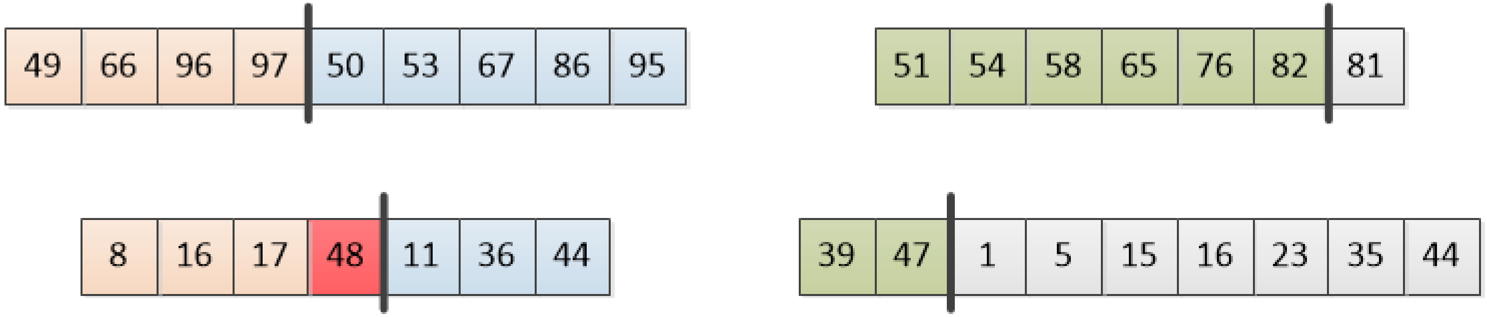
Example: Hyperquicksort VIII

- Selection of sub-lists that can be exchanged:



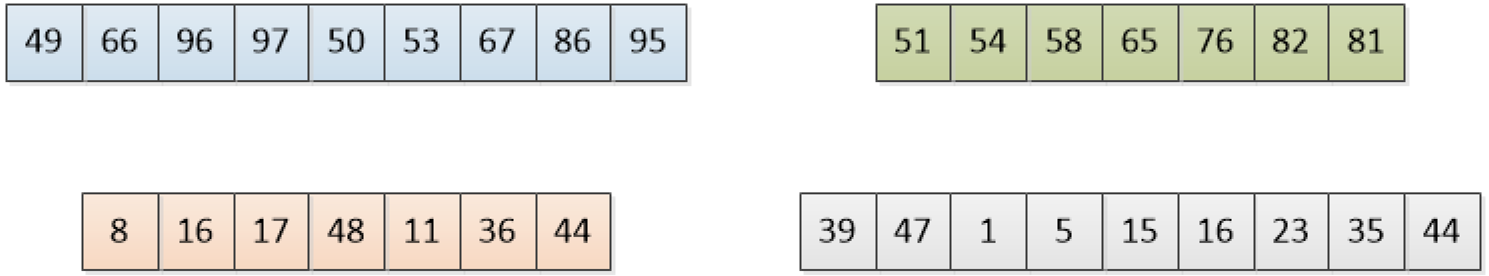
Example: Hyperquicksort IX

- Exchange of sub-lists:



Example: Hyperquicksort X

- Merging the sub-lists:



Example: Hyperquicksort XI

- Sorting the local lists:

49	50	53	66	67	86	95	96	97

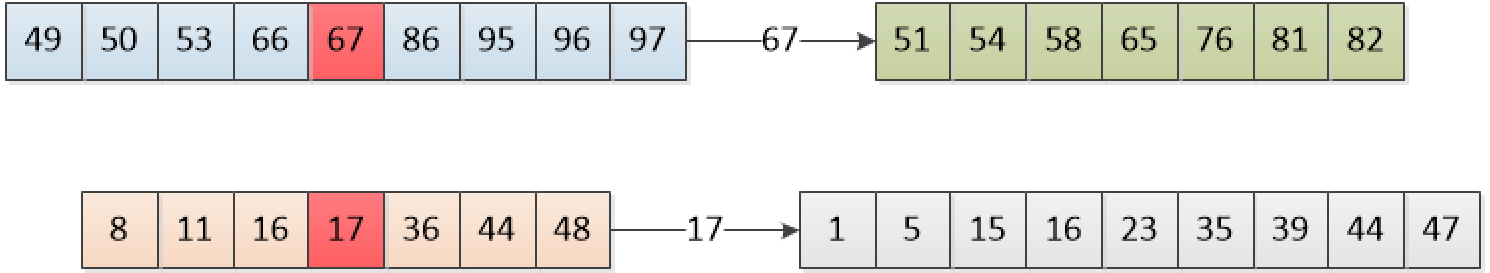
51	54	58	65	76	81	82

8	11	16	17	36	44	48

1	5	15	16	23	35	39	44	47

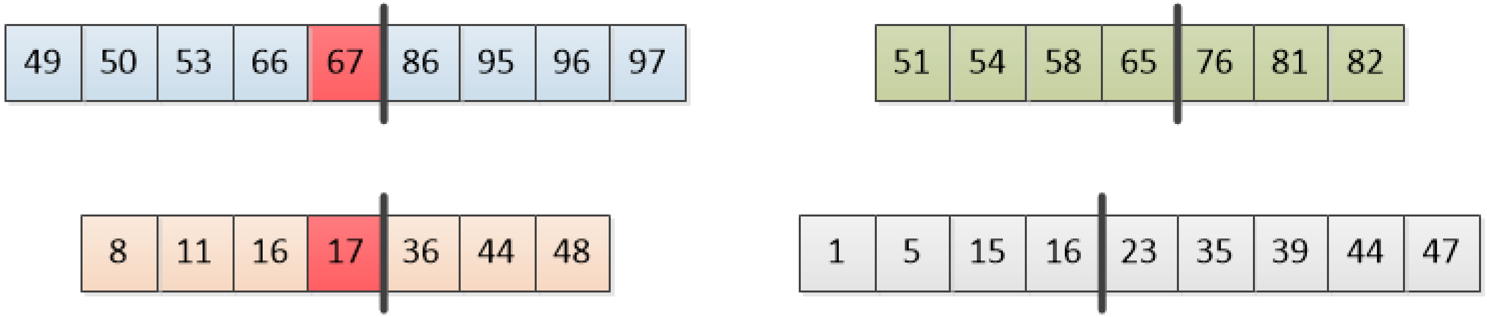
Example: Hyperquicksort XII

- Determination and communication of new pivot elements:



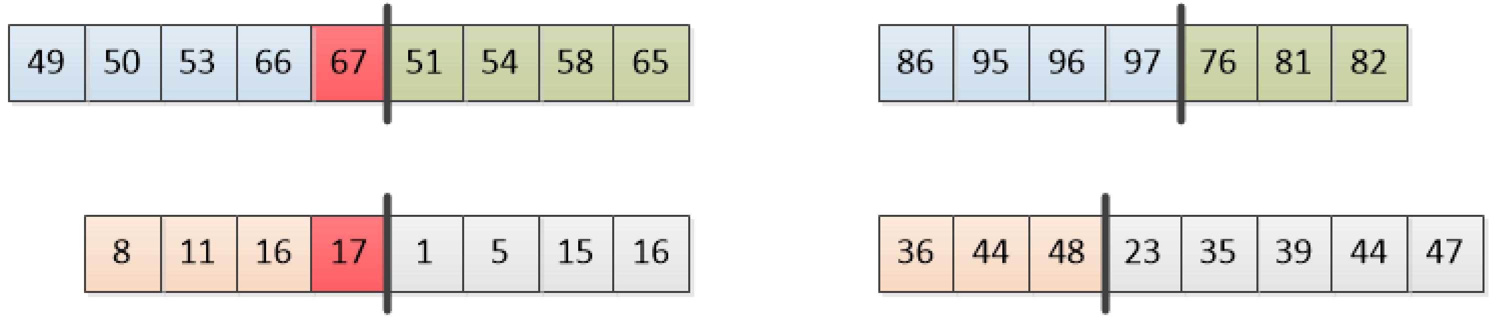
Example: Hyperquicksort XIII

- Splitting into sub-lists based on new pivot elements:



Example: Hyperquicksort XIV

- Exchange of sub-lists with other processes:



Example: Hyperquicksort XV

- Merging the sub-lists:

49	50	53	66	67	51	54	58	65

86	95	96	97	76	81	82

8	11	16	17	1	5	15	16

36	44	48	23	35	39	44	47

Example: Hyperquicksort XVI

- Sorting the lists:

49	50	51	53	54	58	65	66	67

76	81	82	86	95	96	97

1	5	8	11	15	16	16	17

23	35	36	39	44	44	47	48

Example: Hyperquicksort – Summary

- Partitioning and agglomeration
  - Partitioning is performed regarding the amount of processes to be used (agglomeration).

- Communication
  - Communication costs are reduced due to a high amount of local sorting.
  - Communication effort: log p exchange operations

- Mapping
  - The exchange of sub-lists is performed between separate groups of processes, reducing the communication costs by message congestion on the network.
  - In the end there is a all-to-all communication. Thus, the overall-distance of the nodes used by the processes should be small.

Concepts of Non-sequential and Distributed Programming

NEXT LECTURE

Institute of Computer Science
Department of Mathematics and Computer Science

Freie Universität Berlin

Design and Implementation of Parallel Applications III

APL IV: Concepts of Non-sequential and Distributed Programming (Summer Term 2023)