Computação de Alto Desempenho (High Performance Computing)

Slides adapted from “An Introduction to Parallel Programming”, Peter Pacheco

Parallel Hardware and Parallel Software (conclusion)

- Some background
- Modifications to the von Neumann model
  - Caching
  - Virtual memory
  - Instruction Level Parallelism
  - Hardware multithreading
- Parallel hardware
- Parallel software
- Input and output
- Performance
- Parallel program design

Last week:

- Serial systems
  - The standard model of computer hardware has been the von Neumann architecture.
  - Modifications to the von Neumann Model targeted at reducing the problem of the von Neumann bottleneck and making CPUs faster.

- Parallel hardware
  - Flynn's taxonomy.
- Parallel software
  - SPMD programs.
  - software for MIMD systems.

- Input and Output
  - We will write programs in which one process or thread can access stdin, and all processes can access stdout and stderr.
  - However, because of nondeterminism, except for debug output we will usually have a single process or thread accessing stdout.

- Performance
  - Speedup
  - Efficiency
  - Amdahl's law
  - Scalability
  - Gustafson's law
- Today: Parallel Program Design
  - Foster's methodology

PARALLEL PROGRAM DESIGN

Ian Foster's methodology
("Designing and Building Parallel Programs")

1. Partitioning: divide the computation to be performed and the data operated on by the computation into small tasks.

   The focus here should be on identifying tasks that can be executed in parallel.

Foster's methodology

2. Communication: determine what communication needs to be carried out among the tasks identified in the previous step.

Foster's methodology

3. Agglomeration or aggregation: combine tasks and communications identified in the first step into larger tasks.

   For example, if task A must be executed before task B can be executed, it may make sense to aggregate them into a single composite task.

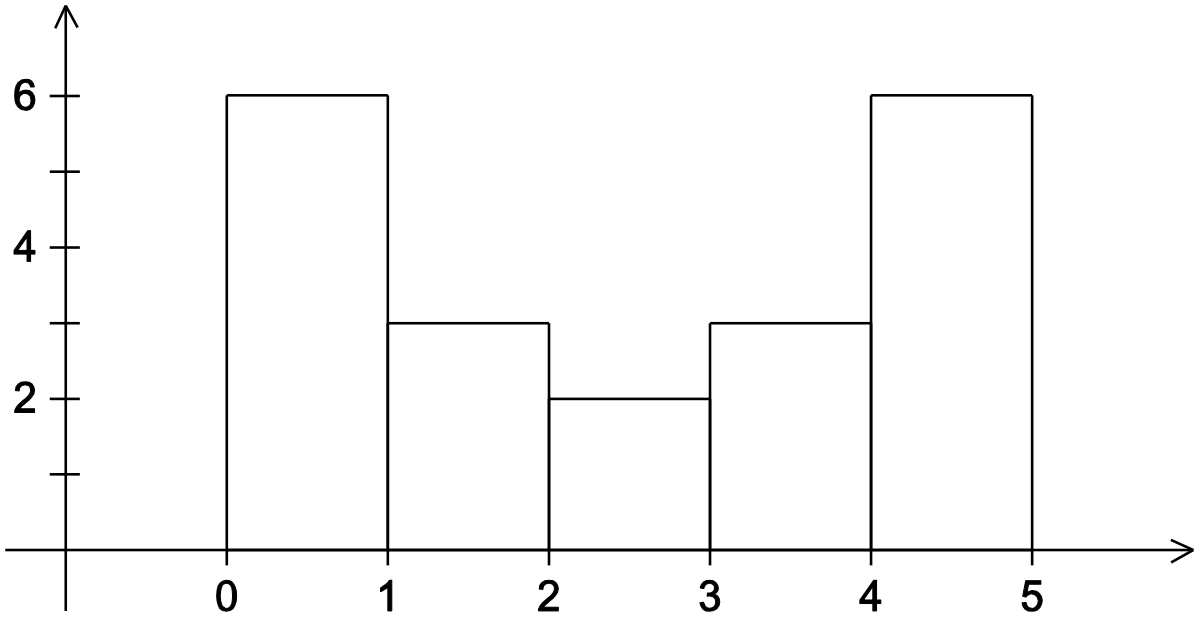
Foster's methodology

4. Mapping: assign the composite tasks identified in the previous step to processes/threads.

   This should be done so that communication is minimized, and each process/thread gets roughly the same amount of work.

Example – histogram (to get some feel of the distribution of data in an array)

- 1.3,2.9,0.4,0.3,1.3,4.4,1.7,0.4,3.2,0.3,4.9,2.4,3.1,4.4,3.9,0.4,4.2,4.5,4.9,0.9



-- divide the range of the data up into equal sized subintervals, or bins;
-- determine the number of measurements in each bin; and
-- plot a bar graph showing the relative sizes of the bins.

Serial program - input

1. The number of measurements: data_count
2. An array of data_count floats: data
3. The minimum value for the bin containing the smallest values: min_meas
4. The maximum value for the bin containing the largest values: max_meas
5. The number of bins: bin_count

Serial program - output

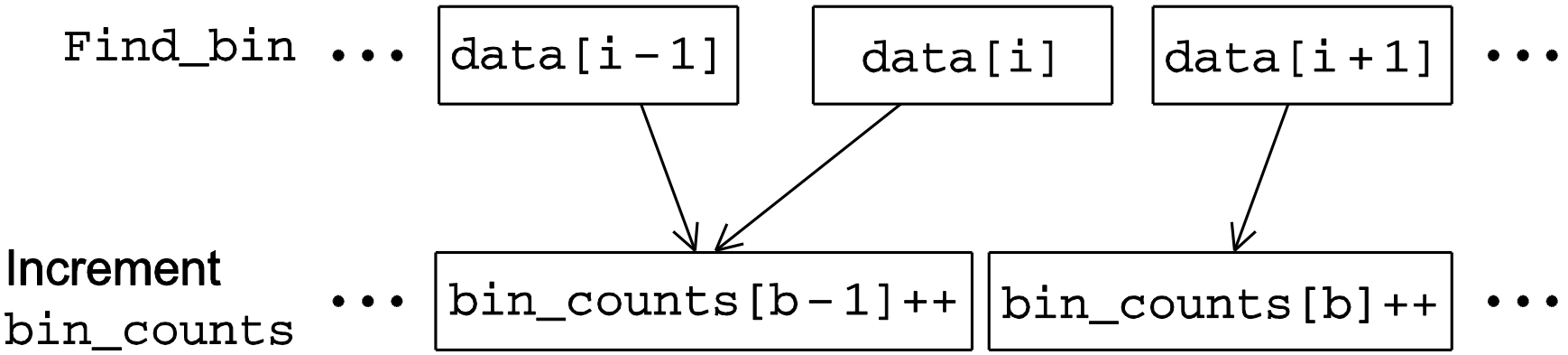
1. bin_maxes : an array of bin_count floats

   stores the upper bound for each bin

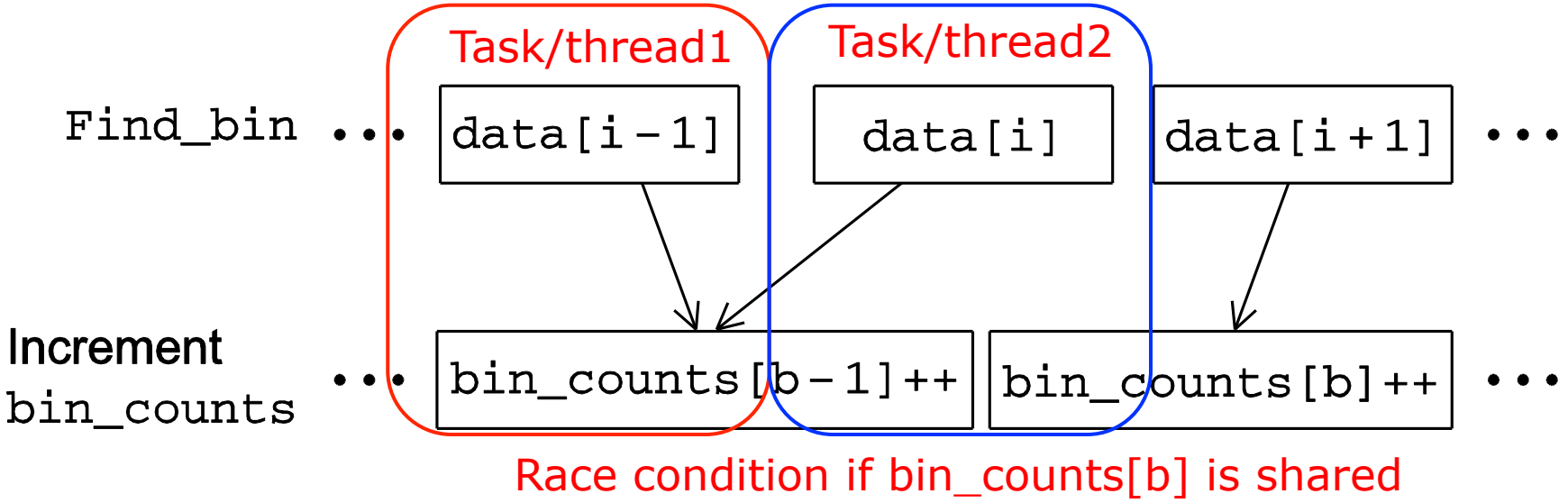
2. bin_counts : an array of bin_count ints

   stores the number of data elements in each bin

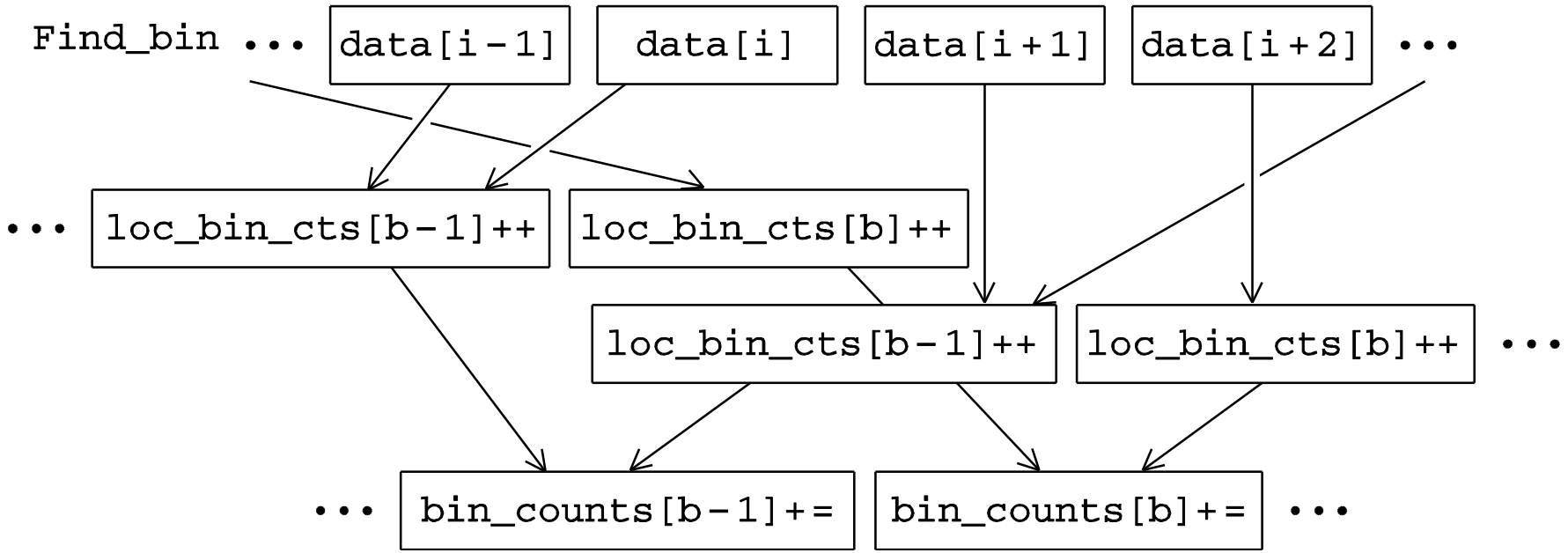
First two stages of Foster's Methodology



First stages of Foster's Methodology

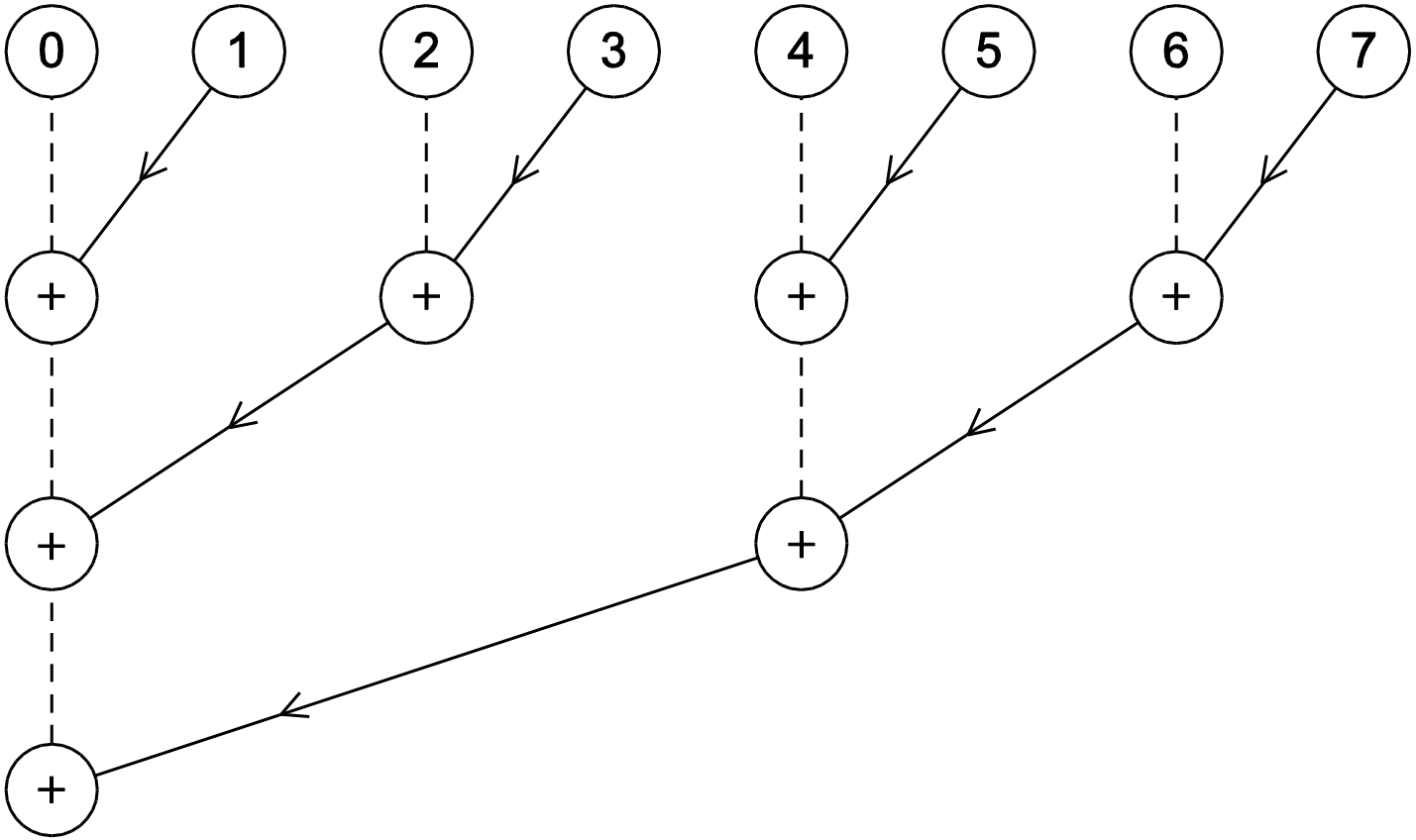


Alternative definition of tasks and communication



Alternative is to store multiple "local" copies of bin counts and add the values in the local copies after all the calls to Find bin.

Adding the local arrays



Foster's methodology

1. *Partitioning*

   Divide the computation to be performed and the data operated on by the computation into small tasks. The focus here should be on identifying tasks that can be executed in parallel.

2. *Communication*

   Determine what communication needs to be carried out among the tasks identified in the previous step.

3. *Agglomeration or aggregation*

   Combine tasks and communications identified in the first step into larger tasks.

4. *Mapping*

   Assign the composite tasks identified in the previous step to processes/ threads.