NORTHERN ARIZONA UNIVERSITY
School of Informatics, Computing, and Cyber Systems

Using OpenMP. Part III

CS450/CS550: Parallel Programming

Week 11

Functional parallelism

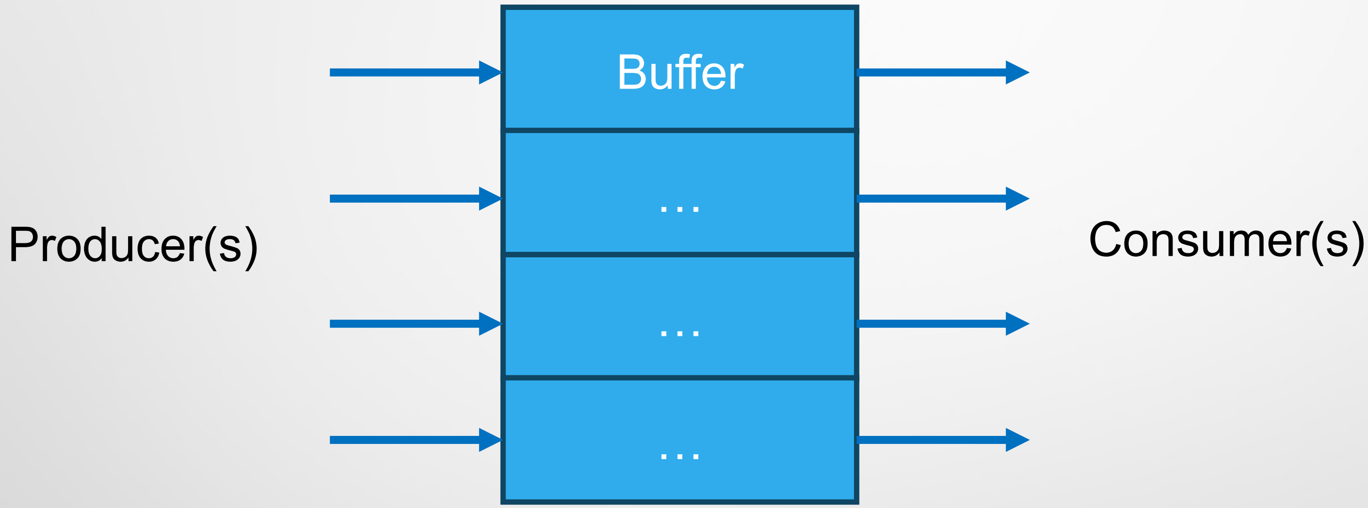
- Two types of parallelism:
  - data parallelism
  - functional parallelism
- To implement the functional parallelism, OpenMP uses the `parallel sections` directive:

```
#pragma omp parallel sections
{
    #pragma omp section // optional
    {…} // function 1
    #pragma omp section
    {…} // function 2
    #pragma omp section // optional
    {…} // function 3
}
```

Sections

- The OpenMP `sections` directive breaks work into separate sections
- Each section is executed by a corresponding thread
- If the number of available threads is more than the number of the defined sections, then some threads remain idle
- If the number of available threads is fewer than sections' number, then some sections are serialized
- The `section` directive may be omitted for the first and the last sections defined inside the `sections` block

Producer-Consumer Problem: example code



The `single` and `master` directives

- The `single` directive specifies a structured block that is executed by a single (arbitrary) thread:

```
#pragma omp single [clauses]
```

- The `master` directive is a specialization of the `single` directive, in which only the master thread executes the specified block:

```
#pragma omp master
```

- The single and master directives are useful for computing global data

The `nowait` clause

- The `nowait` clause syntax:

`#pragma omp parallel for [`*`clauses`*`]` **`nowait`**

- The `nowait` clause added to a `parallel for` directive tells the compiler to omit the barrier synchronization at the end of the loop
- As the result of using the `nowait` clause, threads are allowed to proceed the execution without waiting all other threads to complete the loop

The `barrier` directive

- The `barrier` directive syntax:

```
#pragma omp barrier
```

- On encountering the `barrier` directive, all threads wait until others have reached the barrier point

Advanced `reduction` operators

- The `reduction` clause is used in the `parallel` directive:

`#pragma omp parallel [`*`clauses`*`] reduction(`*`op`*`: `*`var`*`)`

- Starting from OMP3.0, the `min` and `max` reduction operators are available
- Starting from OMP4.0, reduction operators may be defined by users
- Starting from OMP4.5, the reduction is possible for the array elements:

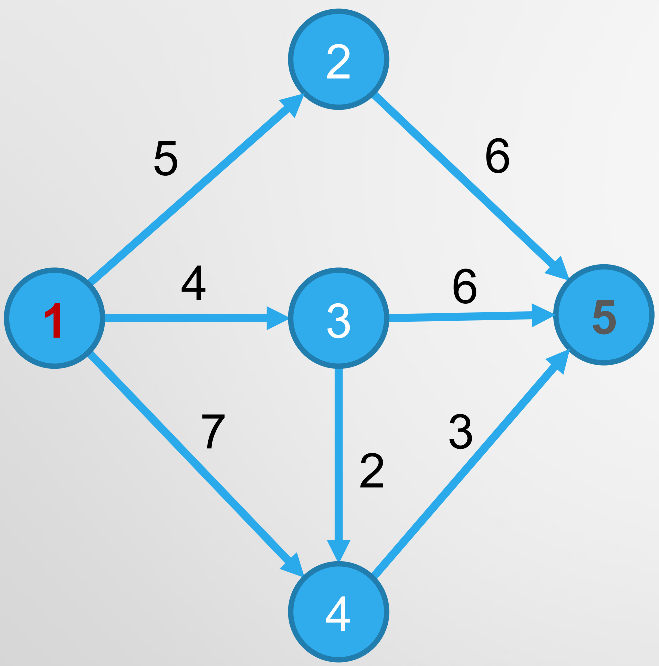
`#pragma omp parallel reduction(+: `*`ar`*`[:`*`size`*`])`

the array *`size`* should be specified for the number of the *`ar`* elements to be reduced

Dijkstra's algorithm

1. Assign an initial distance value to each node: set it to zero for the initial node and infinity for all other nodes
2. Make the starting node current. Mark all remaining nodes as unverified (unvisited). Create a set of all unvisited nodes
3. For the current node, calculate the distances for all unvisited neighbors. Compare the newly calculated distance to the current assigned value and assign a smaller one
4. Mark the current node as visited and remove it from the unvisited set
5. If the destination node is marked as visited, stop
6. Otherwise, select the unvisited node with the smallest distance, set it as the new "current node" and return to **step 3**

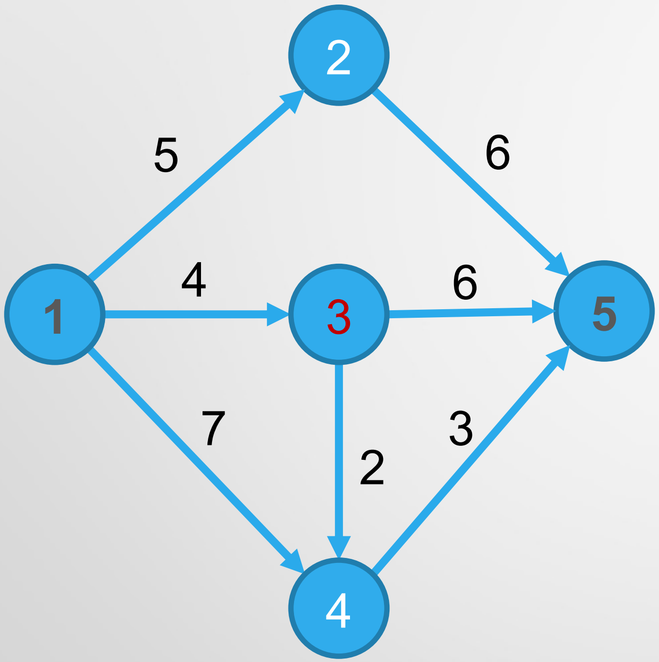
Dijkstra's algorithm example. Iteration #1



- **Visited nodes:** $\mathbf{P} = \{\emptyset\}$
- **Unvisited nodes:** $\mathbf{Q} = \{1, 2, 3, 4, 5\}$
- **Initialization:** $d(1) = 0$, $d(2) = \infty$, $d(3) = \infty$, $d(4) = \infty$, $d(5) = \infty$ $p(1) = \emptyset$, $p(2) = \emptyset$, $p(3) = \emptyset$, $p(4) = \emptyset$, $p(5) = \emptyset$
- **Current node is 1:** $d(1) = 0$, $p(1) = \emptyset$
  $d(2) = \min\{d(2); d(1) + l(1,2)\} =$
  $= \min\{\infty; 0 + 5\} = 5$, $p(2) = 1$
  $d(3) = \min\{d(3); d(1) + l(1,3)\} =$
  $= \min\{\infty; 0 + 4\} = 4$, $p(3) = 1$
  $d(4) = \min\{d(4); d(1) + l(1,4)\} =$
  $= \min\{\infty; 0 + 7\} = 7$, $p(4) = 1$

NAU NORTHERN ARIZONA UNIVERSITY
School of Informatics, Computing, and Cyber Systems

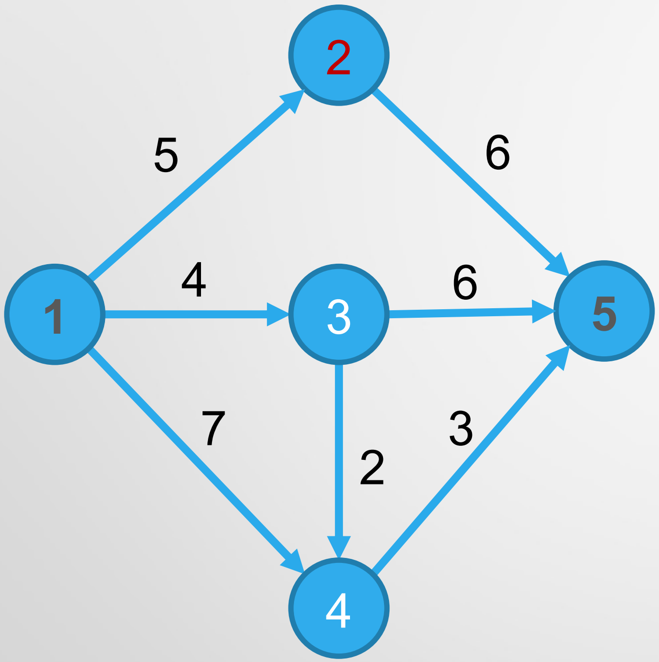
Dijkstra's algorithm example. Iteration #2



- **Visited nodes:** $\mathbf{P} = \{1\}$
- **Unvisited nodes:** $\mathbf{Q} = \{2, 3, 4, 5\}$
- **Current node is 3:** $d(3) = 4, p(2) = 1$

$$d(4) = \min\{d(4); d(3) + l(3,4)\} = $$
$$= \min\{7; 4 + 2\} = 6, p(4) = 3$$
$$d(5) = \min\{d(5); d(3) + l(3,5)\} = $$
$$= \min\{\infty; 4 + 6\} = 10, p(5) = 3$$

Dijkstra’s algorithm example. Iteration #3

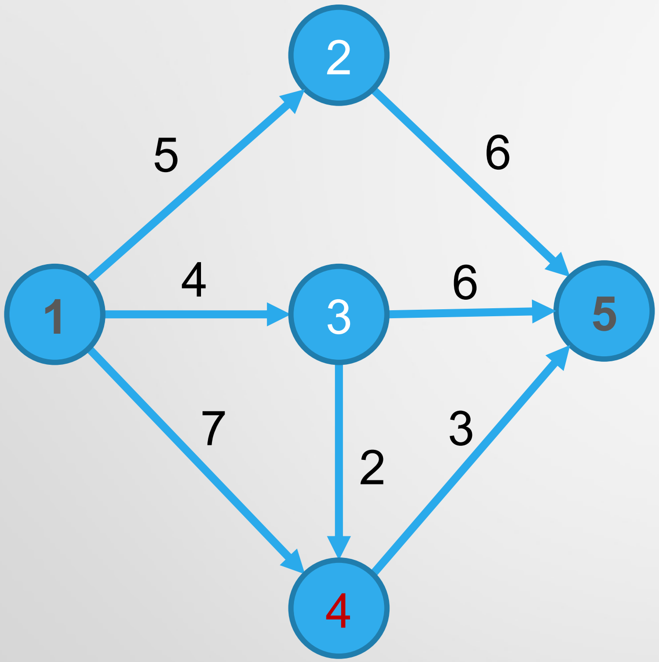


- **Visited nodes:** $\mathbf{P} = \{1, 3\}$
- **Unvisited nodes:** $\mathbf{Q} = \{2, 4, 5\}$
- **Current node is 2:** $d(2) = 5, p(2) = 1$

$$d(5) = \min\{d(5); d(2) + l(2,5)\} = \\ = \min\{10; 5 + 6\} = 10, p(5) = 3$$

NAU NORTHERN ARIZONA UNIVERSITY
School of Informatics, Computing, and Cyber Systems

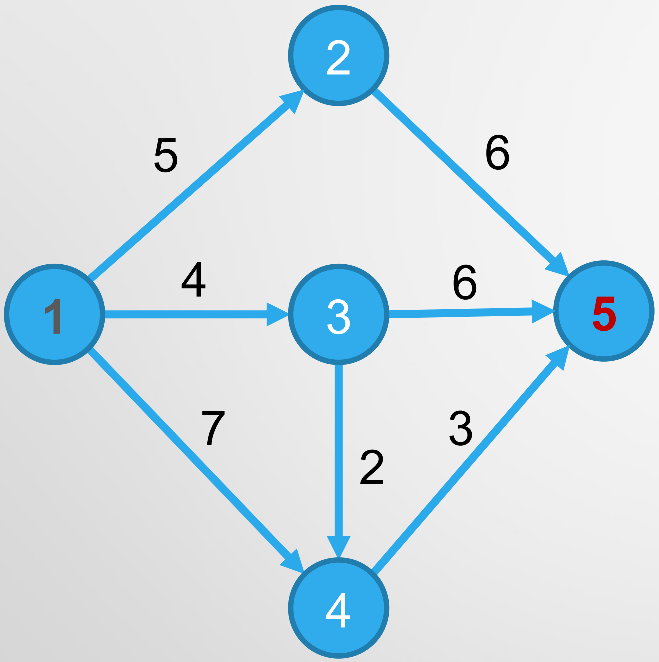
Dijkstra's algorithm example. Iteration #4



- **Visited nodes:** $\mathbf{P} = \{1, 3, 2\}$
- **Unvisited nodes:** $\mathbf{Q} = \{4, 5\}$
- **Current node is 4:** $d(4) = 6, p(4) = 3$

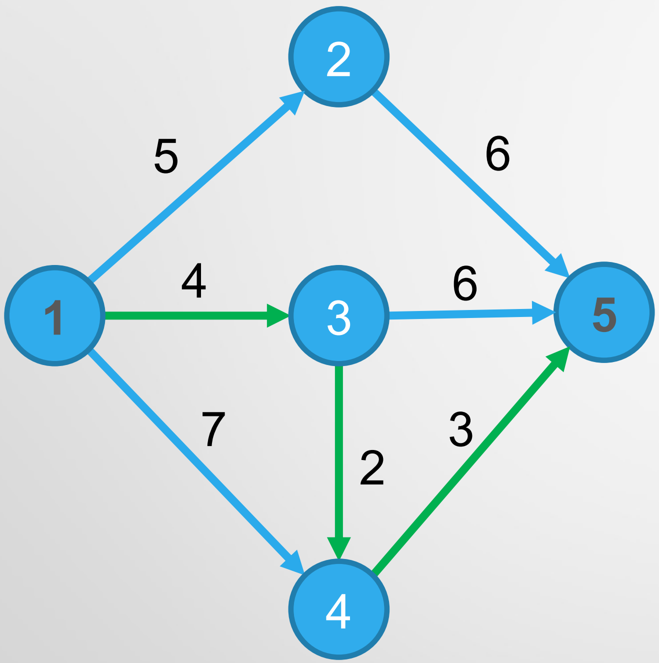
$$d(5) = \min\{d(5); d(4) + l(4,5)\} =$$
$$= \min\{10; 6 + 3\} = 9, p(5) = 4.$$

Dijkstra's algorithm example. Iteration #5



- **Visited nodes:** $\mathbf{P} = \{1, 3, 2, 4\}$
- **Unvisited nodes:** $\mathbf{Q} = \{5\}$
- **Current node is 5:** $d(5) = 9, p(5) = 4$

Dijkstra's algorithm example. Retrieve path



- **Visited nodes:** $\mathbf{P} = \{1, 3, 2, 4, 5\}$
- **Unvisited nodes:** $\mathbf{Q} = \{\emptyset\}$
- **The shortest path length is:** $d(5) = 9$
- **The shortest path is:**

$$path = \{\ldots; 5\}$$
$$p(5) = 4 \Rightarrow path = \{\ldots; 4; 5\}$$
$$p(4) = 3 \Rightarrow path = \{\ldots; 3; 4; 5\}$$
$$p(3) = 1 \Rightarrow path = \{\ldots 1; 3; 4; 5\}$$
$$p(1) = \emptyset \Rightarrow path = \{1; 3; 4; 5\}$$

Dijkstra’s algorithm: example code

School of Informatics, Computing, and Cyber Systems

Assignment #5

- Prepare the parallelized version of the provided sequential code: all the functions may be parallelized (including the fragments of the `main` function)
- Maximize the speedup of your parallelized version (you may justify your decisions in commentaries to your code)
- The solution must satisfy the following conditions:
  - The program must not contain race conditions
  - The parallelized version must be as fast as possible (certainly faster than the sequential version)
  - The result returned by the parallelized version must be correct (the sum of empirical frequencies must be equal the sample size)

School of Informatics, Computing, and Cyber Systems

Generating the normally distributed variable

Notation: $(N: \mu, \sigma)$

Parameters:

- *location parameter* μ – expected value, $\mu = \bar{x} = \frac{1}{N} \cdot \sum_{i=1}^{N} x_i$
- *scale parameter* σ – standard deviation, $\sigma = \sqrt{\frac{1}{N} \cdot \sum_{i=1}^{N} (x_i - \bar{x})^2}$

Generating based on $(R: 0{,}1)$ **:**

$$(N: \mu, \sigma) \sim \mu + \sigma \cdot \frac{2 \cdot \sqrt{3}}{\sqrt{k}} \cdot \left(-\frac{k}{2} + \sum_{i=1}^{k} R_i\right)$$

$$k = 12: \ (N: \mu, \sigma) \sim \mu + \sigma \cdot \left(-6 + \sum_{i=1}^{12} R_i\right)$$

Normalizing the random variable values

- The normalized (standardized) value of a random variable:

$$\hat{x}_i = \frac{x_i - \bar{x}}{\sigma}$$

where $\bar{x}$ is a sample average

σ is a sample standard deviation

School of Informatics, Computing, and Cyber Systems

Calculating the empirical frequencies

- Divide the range of possible values into k ranges (bins):

$r_1 = [x_{min};\ x_{min} + h]$

$r_i = (x_{min} + h \cdot i;\ x_{min} + h \cdot (i + 1)],\ \ i = 2 \dots k$

where x_{min} and x_{max} are minimal and maximal values in a sample,

h is the range width:

$h = \frac{x_{max} - x_{min}}{k}$

- Calculate the empirical frequencies – the number of appearances of random values in each of the ranges