Concepts and Models of Parallel and Data-centric Programming

BSP II

Lecture, Summer 2020

Dr. Christian Terboven <terboven@itc.rwth-aachen.de>

i12 High Performance Computing | RWTH AACHEN UNIVERSITY

Outline

0. Organization
1. Foundations
2. Shared Memory
3. GPU Programming
4. **Bulk-Synchronous Parallelism**
5. Message Passing
6. Distributed Shared Memory
7. Parallel Algorithms
8. Parallel I/O
9. MapReduce
10. Apache Spark

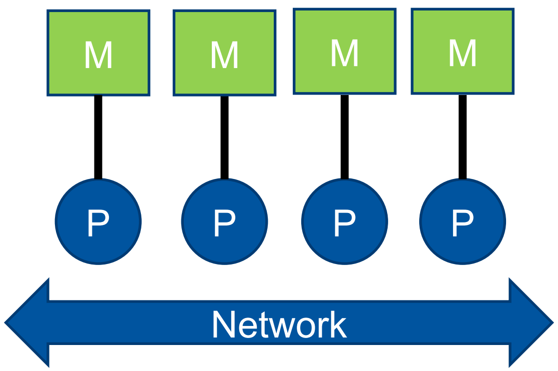
a. Motivation
b. BSP Computer
c. BSP Programming Model
d. BSP Cost Model
e. Bulk Library

BSP Programming Model

BSP Model Components

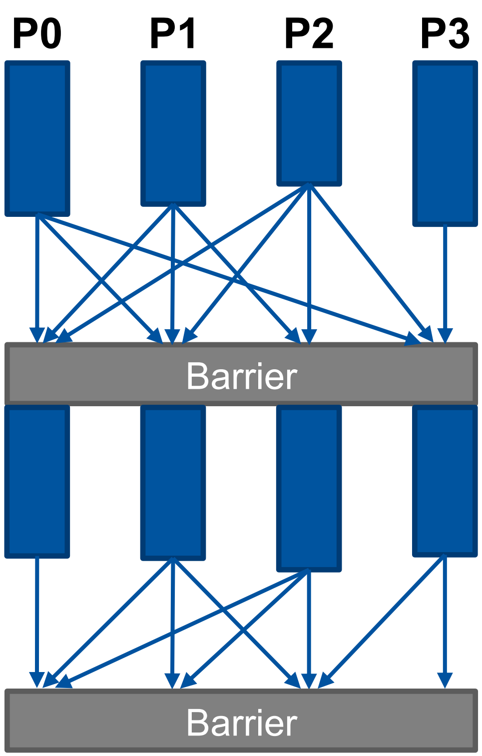
BSP Computer

(Distributed Memory Computer)

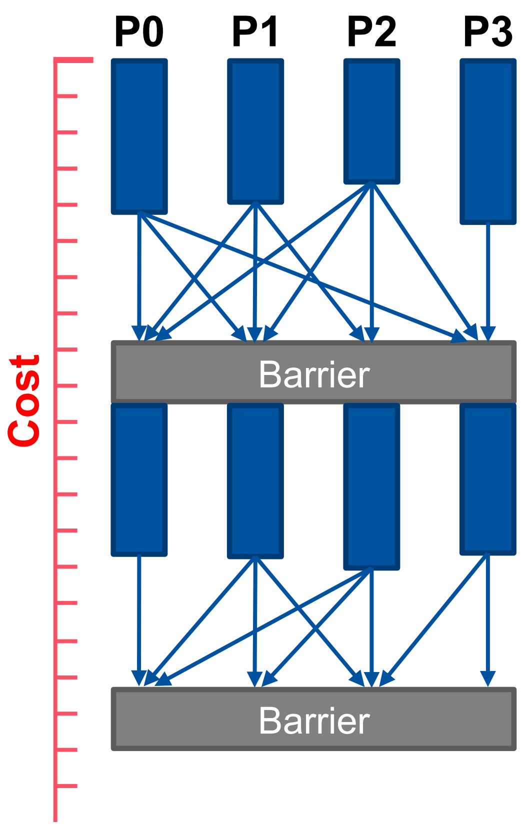


Programming Model

(Algorithmic Framework)

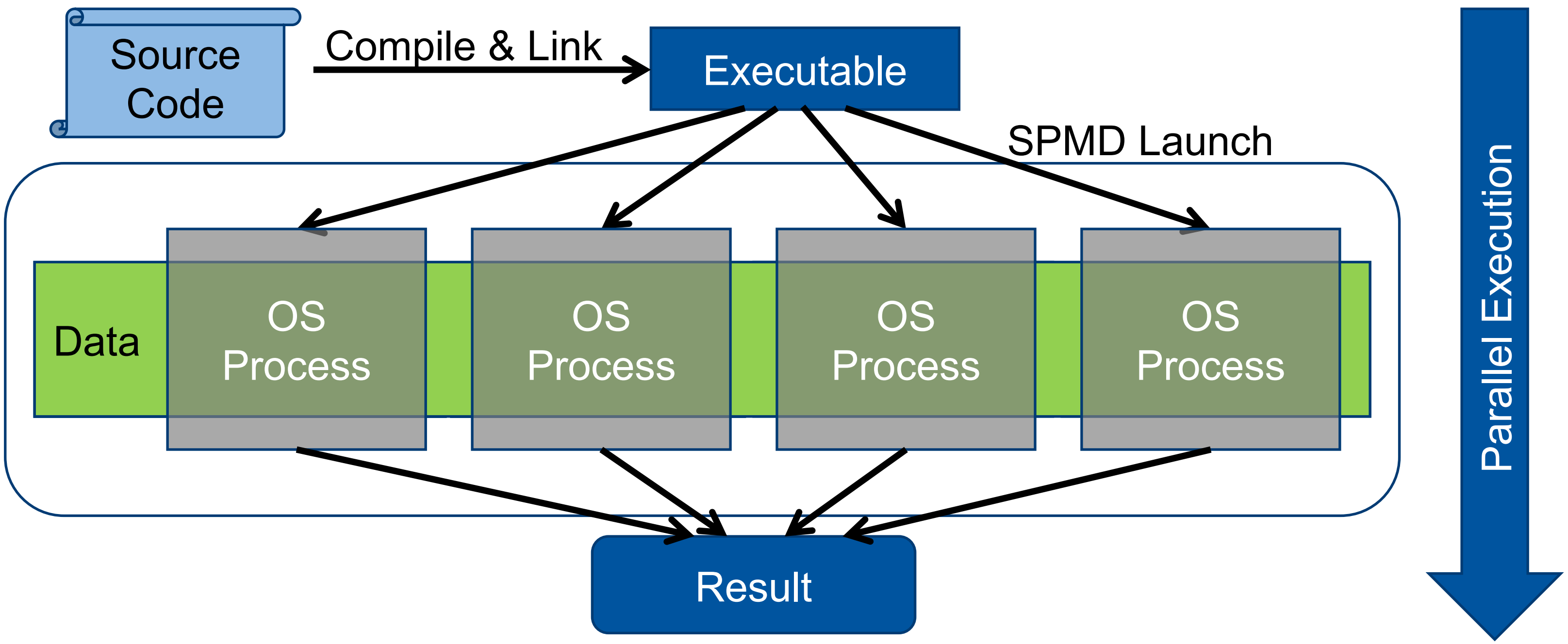


Cost Model



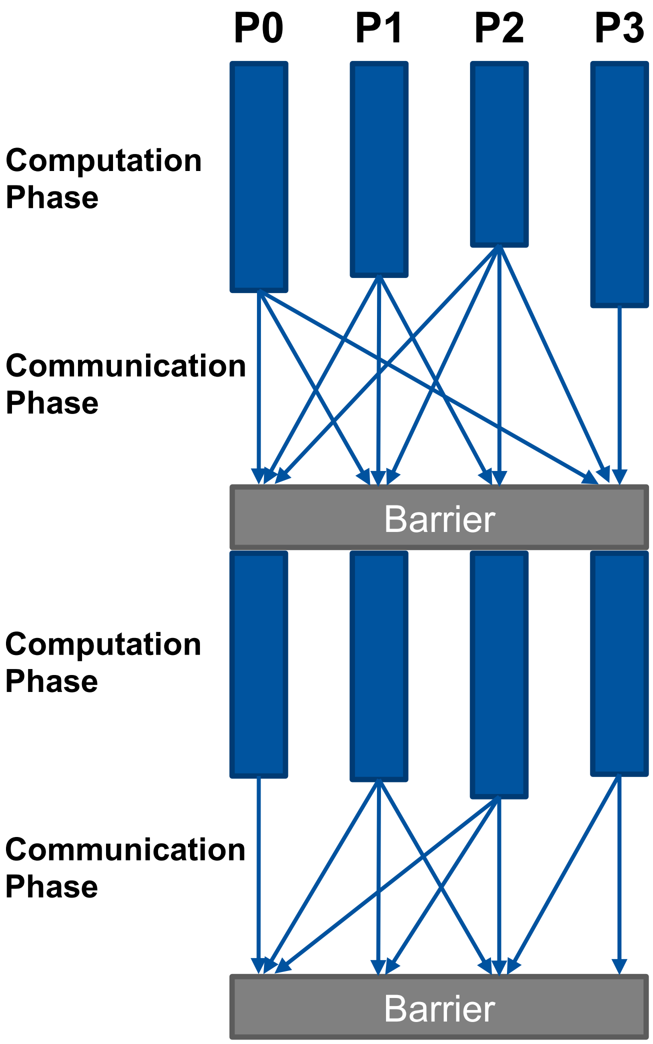
SPMD Model

- BSP programming model follows SPMD Program Lifecycle
  - Multiple processes run the same program with different data (Single Program Multiple Data)
- Program uses processor identity to differentiate processors (MPI: rank)



Programming Model: Supersteps

- BSP algorithm: Series of supersteps
- Superstep: Computation and communication
  - *Computation phase:* Perform local calculations with available data (e.g., FP operations)
  - *Communication phase:* Transfer data (e.g., results) between the different processors
- End of each superstep: Barrier
  - Each processor has to wait until all other processors have reached the barrier.
    → Bulk synchronization
  - Ensures that communication between processors has finished



Simple BSP Example: Parallel Inner Product Computation

- **Given:** Two vectors $x = (x_0, \dots, x_{n-1})^T$ and $y = (y_0, \dots, y_{n-1})^T$
- The *inner product* of x and y is defined as

$$\alpha = x^T y = \sum_{i=0}^{n-1} x_i y_i \,.$$

- **Goal:** Parallelize problem on a BSP computer with p processes
  - Result should be available at *all* processors in the end
- First step: Data distribution
- Second step: BSP algorithm design (supersteps)

First Step: Data Distribution

- **Given:** Vector $x = (x_0, \dots, x_{n-1})^T$, distribute to $p > 0$ processors
- **Distribution function:** $\text{distr}: \{0, \dots, \text{n} - 1\} \rightarrow \{0, \dots, p - 1\}$
- **Block distribution:** $\text{i} \mapsto \left\lfloor \frac{i}{b} \right\rfloor$, for $0 \leq i < n$ (x_i distributed to process $P(\left\lfloor \frac{i}{b} \right\rfloor)$)
  where $b \coloneqq \lceil \frac{n}{p} \rceil$ is the block size

  \# Processors: $p = 4$
  Vector length: $n = 16$
  Block size: $b = 4$

  If we have many data exchange between two cells then the block distribution can be better option

- **Cyclic distribution:** $\text{i} \mapsto i \bmod p$, for $0 \leq i < n$

  \# Processors: $p = 4$
  Vector length: $n = 16$

Second Step: BSP Algorithm Design (1)

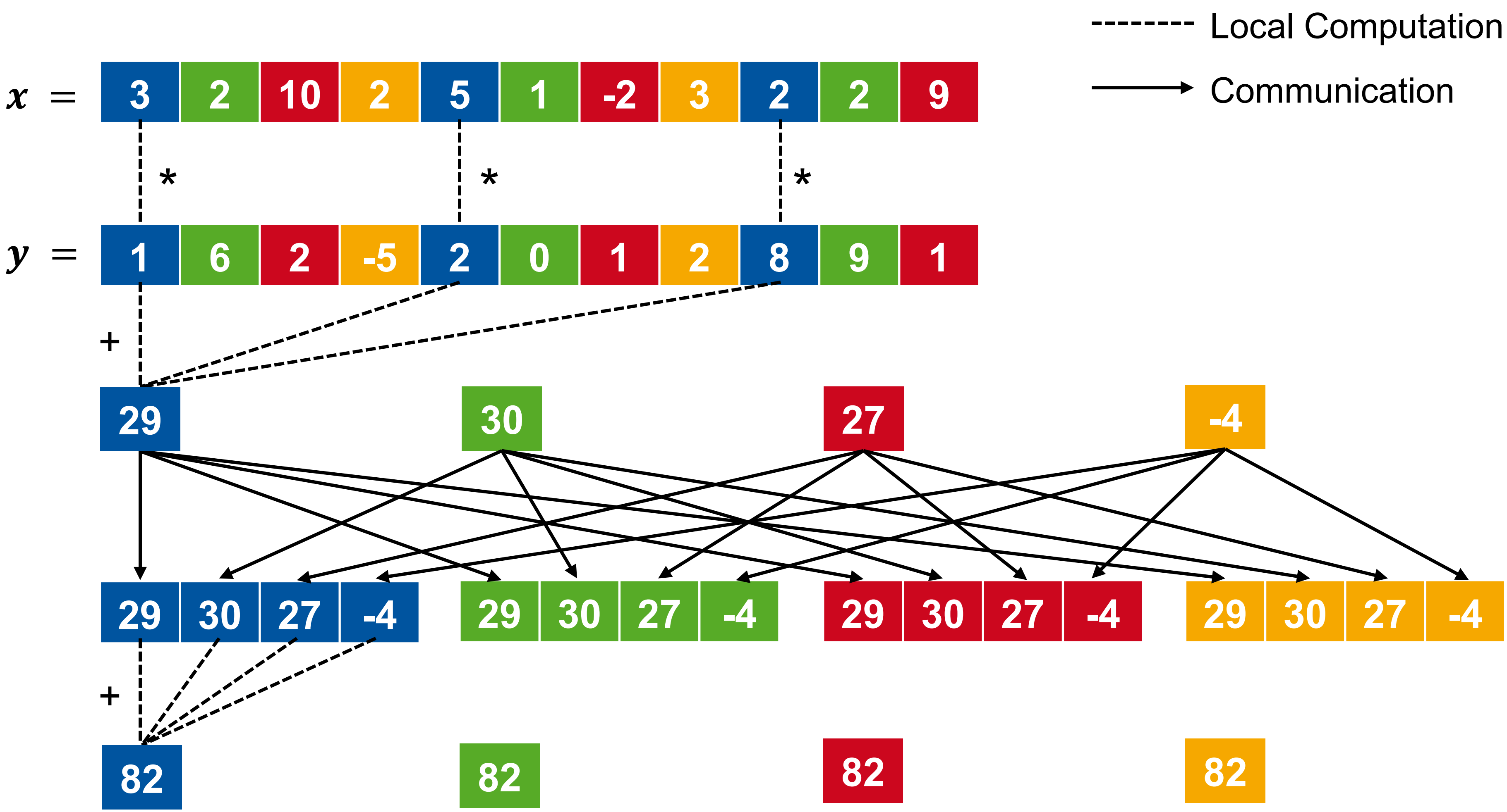
- Inner product: $\alpha = x^T y = \sum_{i=0}^{n-1} x_i y_i$
- Obvious: x_i and y_i should be on the same processor for each $0 \leq i < n$
  - Compute $x_i \cdot y_i$ locally
- Distribution should spread vector components **evenly** on the processors
  - Both block and cyclic distribution fulfill this requirement
- Choose cyclic distribution for x_i and y_i here

Second Step: BSP Algorithm Design (2)

Idea

- Superstep 1
  - Computation: Perform multiplications of locally available x_i and y_i, do a sum reduction on the products locally to get an intermediate result
  - Communication: Broadcast result to other processors
  - End of superstep: Barrier
- Superstep 2
  - Computation: Each processor performs a sum reduction on the received results
  - Communication: None, problem is solved. Result available on **all** processors.
  - End of superstep: Barrier (but not needed here)

Example for $n = 11$ and $p = 4$ (cyclic distribution)



BSP Algorithm: Parallel Inner Product (Cyclic Distribution)

Input: x, y: vector of length n

$distr(x) = distr(y) = d$ with $d(i) = i \bmod p$ for $0 \le i < n$ (cyclic)

Output: $\alpha = x^T y$

every processor has its own rank
by this rank, each processor compute local product
these are distributed by processor

Algorithm for processor $s \in \{0, \dots, p-1\}$:

Superstep 0

Comp. Phase

```
αs := 0;
for (i := s; i < n; i += p) do
   αs := αs + xi yi;          // compute local product
```

Comm. Phase

```
for (t :=0; t < p ; t++) do
   put αs in P(t);            // broadcast to all processors t
barrier();
```

Superstep 1 (omitted Comm.)

Comp. Phase

```
α := 0;
for (t :=0; t < p; t++) do
   α := α + αt;               // sum up local and received α values
```