Exercise 3: Distributed Memory

Concepts and Models of Parallel and Data-centric Programming

Lecture, Summer 2019

Dr. Christian Terboven

i12 High Performance Computing | RWTH AACHEN UNIVERSITY

Task 1

General Questions

Task 1.1 (1)

Task 1.1: We discussed the BSP model with the Bulk implementation and the PGAS model with the DASH implementation. What are the most important conceptual differences between them?

Solution:

- Programming structure
  - BSP: Supersteps in an algorithmic framework
  - PGAS: No predefined algorithmic structure of the program
- Communication
  - BSP: Only performed in communication phases of a superstep
  - PGAS: Can be done everywhere in the program

Task 1.1 (2)

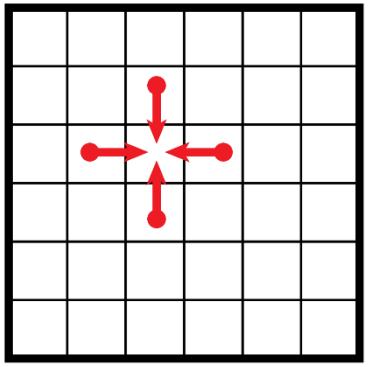
- Memory model
  - BSP: Distributed memory visible to the user
    - Communication explicitly defines the target rank
    - Example: Access element i of array a at rank t: $a(t)[i]$
    - Pro: User is aware of distributed memory transfers (local and global accesses)
  - PGAS / DASH: Distributed memory abstracted to a shared memory
    - Target rank in communication is implicitly determined by DASH
    - User can directly access all array elements as in shared memory
      - DASH translates array accesses to distributed memory transfers in runtime
    - Pro: User does not have to care about target rank calculations
- Cost model
  - BSP: Part of the programming model specification
  - PGAS: No specific cost model (due to missing algorithmic framework)

Task 1.2

Task 1.2: Using distributed memory computers also means partitioning data. There are different data distribution schemes available in Bulk and DASH. Explain with examples for which kinds of problems a cyclic distribution is better suited than a block distribution and vice versa.

Solution:

- Block distribution: Any stencil code (see Task 3)
  - Splitting data in blocks is better than cyclic distribution, because each cell needs values of neighboring cells for computation
  - Example 2D stencil: Splitting matrix in blocks is better than distributing neighboring elements to different processors
  - Cyclic distribution would require a lot of communication

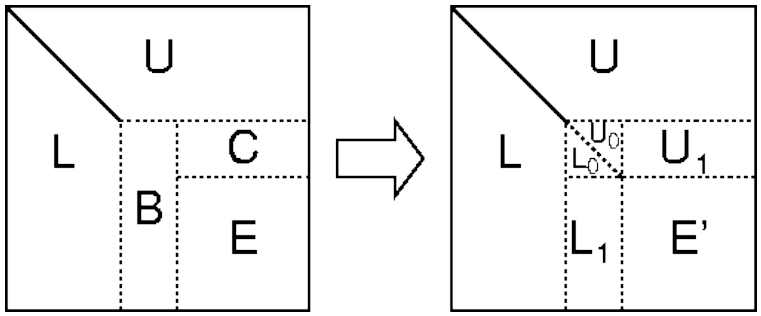


2D von Neumann stencil
Source: Wikipedia
https://en.wikipedia.org/wiki/Stencil_code

Task 1.2

Solution:

- Cyclic distribution: Problems with different amounts of work on the data
- Example: Matrix factorization codes like LU factorization
  - Works only on portions of the matrix, getting smaller in each iteration
  - Problem: Block distribution would lead to small number of processors that have to do all the computation (load imbalance)
  - Cyclic distribution: Better load balance, but more communication
  - Tradeoff: Blockcyclic distribution (see next task)



Source: http://www.netlib.org/utk/papers/siam-review93/node13.html

Task 1.3

Beside cyclic and block distributions, a combination of both patterns, a *blockcyclic* distribution is possible (see DASH slide 33). How would a distribution function look like for a 1-dimensional array? Give a function definition that matches an array of size n to p processes in a blockcyclic distribution with the following signature:

$$distr\colon \{0, \dots, n-1\} \rightarrow \{0, \dots, p-1\}$$

Task 1.3

$$distr\colon \{0, \dots, n-1\} \to \{0, \dots, p-1\}$$

Solution:

- Block distribution: $\mathrm{i} \mapsto \left\lfloor \frac{i}{b} \right\rfloor$, for $0 \leq i < n$ where $b := \left\lceil \frac{n}{p} \right\rceil$ is the block size
- Cyclic distribution: $\mathrm{i} \mapsto i \bmod p$, for $0 \leq i < n$
- Blockcyclic distribution: $\mathrm{i} \mapsto \left\lfloor \frac{i}{b} \right\rfloor \bmod p$ where $1 \leq b \leq n$ is the block size

Processors: $p = 4$
Vector length: $n = 16$
Block size: $b = 2$

- Observation: Cyclic distribution ($b = 1$) and block distribution ($b = \left\lceil \frac{n}{p} \right\rceil$) are special cases of blockcyclic distributions.

Task 2

Inner Product Revisited

Task 2

- Inner product algorithm lecture: Each processor computes local product and broadcasts local intermediate result to all other processors
- Adaptation: Use a single root process which computes the final inner product and broadcasts the result to all other processes.
- How does this adaptation affect the BSP costs?

Task 2.1: Adapt the pseudocode algorithm on slide 25 (BSP slides) such that the intermediate results are sent to a designated root process which computes the final inner product and broadcasts the result back to all other processes.

Task 2.1 – Original Algorithm

Input: x, y: vector of length n

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

Output: $\alpha = x^T y$, **Algorithm for processor** $s \in \{0, \ldots, p-1\}$:

```
α_s := 0;
for (i := s; i < n; i += p) do
    α_s := α_s + x_i y_i;            // compute local product

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

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

Comp. Phase

Comm. Phase

Superstep 0

Comp. Phase

Superstep 1 (omitted Comm.)

Task 2.1 – Adapted Version

Input: x, y: vector of length n

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

Output: $\alpha = x^T y$, **Algorithm for processor** $\boldsymbol{s} \in \{\boldsymbol{0}, \ldots, \boldsymbol{p-1}\}$:

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

put αs in P(0);                // send result to 0
barrier();

if (s == 0) then
   α := 0;
   for (t :=0; t < p; t++) do
      α := α + αt;             // sum up local and received α values
   for (t :=0; t < p ; t++) do
      put α in P(t);           // broadcast final result to all processors t
fi
barrier();
```

Comp. Phase

Comm. Phase

Superstep 0

Comp. Phase

Comm. Phase

Superstep 1

Task 2.2 (1)

Task 2.2: Provide the number of FLOPS $w_i^{(s)}$ of processor s in superstep i in your algorithm. Assume that process 0 has been chosen to be the root process.

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

put αs in P(0);                 // send result to 0
barrier();

if (s == 0) then
    α := 0;
    for (t :=0; t < p; t++) do
        α := α + αt;            // sum up local and received α values
    for (t :=0; t < p ; t++) do
        put α in P(t);          // broadcast final result to all processors t
fi
barrier();
```

Comp. Phase

$$w_0^{(s)} = 2\lceil\frac{n}{p}\rceil$$

Comp. Phase

$$w_1^{(s)} = \begin{cases} p & s = 0 \\ 0 & s \neq 0 \end{cases}$$

Task 2.3 (2)

Lecture Algorithm	Adapted Algorithm
Superstep 0: $w_0^{(s)} = 2\lceil\frac{n}{p}\rceil \quad \forall\, s \in \{0, \dots, n-1\}$	**Superstep 0:** $w_0^{(s)} = 2\lceil\frac{n}{p}\rceil \quad \forall\, s \in \{0, \dots, n-1\}$
Superstep 1: $w_1^{(s)} = p \qquad \forall\, s \in \{1, \dots, n-1\}$	**Superstep 1:** $w_1^{(0)} = p$ $w_1^{(s)} = 0 \qquad \forall\, s \in \{1, \dots, n-1\}$

Task 2.3 (1)

Task 2.3: Provide the number of data elements $r_i^{(s)}$ / $t_i^{(s)}$ received / transmitted by process s in superstep i in your adapted algorithm. In addition, determine the h-relation for each superstep. Assume that process 0 has been chosen to be the root process.

```
αs := 0;
for (i := s; i < n; i += p) do
    αs := αs + xiyi

put αs in P(0);
barrier();

if (s == 0) then
    α := 0;
    for (t :=0; t < p; t++) do
        α := α + αt;
    for (t :=0; t < p ; t++) do
        put α in P(t);
fi
barrier();
```

Comm. Phase

$$t_0^{(s)} = \begin{cases} 0 & s = 0 \\ 1 & s \neq 0 \end{cases} \qquad r_0^{(s)} = \begin{cases} p-1 & s = 0 \\ 0 & s \neq 0 \end{cases}$$

Comm. Phase

$$t_1^{(s)} = \begin{cases} p-1 & s = 0 \\ 0 & s \neq 0 \end{cases} \qquad r_1^{(s)} = \begin{cases} 0 & s = 0 \\ 1 & s \neq 0 \end{cases}$$

Task 2.3 (2)

Lecture Algorithm	Adapted Algorithm
Superstep 0: $w_0^{(s)} = 2\left\lceil \frac{n}{p} \right\rceil \quad \forall\, s \in \{0, \dots, n-1\}$ $t_0^{(s)} = p-1, \quad r_0^{(s)} = p-1$ $\forall\, s \in \{0, \dots, n-1\}$	**Superstep 0:** $w_0^{(s)} = 2\left\lceil \frac{n}{p} \right\rceil \quad \forall\, s \in \{0, \dots, n-1\}$ $t_0^{(0)} = 0, \quad r_0^{(0)} = p-1$ $t_0^{(s)} = 1, \quad r_0^{(s)} = 0 \;\forall\, s \in \{1, \dots, n-1\}$
Superstep 1: $w_1^{(s)} = p \quad \forall\, s \in \{1, \dots, n-1\}$ $t_1^{(s)} = 0, \quad r_1^{(s)} = 0$ $\forall\, s \in \{0, \dots, n-1\}$	**Superstep 1:** $w_1^{(0)} = p$ $w_1^{(s)} = 0 \quad \forall\, s \in \{1, \dots, n-1\}$ $t_1^{(0)} = p-1, \quad r_1^{(0)} = 0$ $t_1^{(s)} = 0, \quad r_1^{(s)} = 1 \;\forall\, s \in \{1, \dots, n-1\}$

Task 2.3 (2)

Lecture Algorithm	Adapted Algorithm
Superstep 0: $w_0^{(s)} = 2\left\lceil \frac{n}{p} \right\rceil \quad \forall\, s \in \{0, \dots, n-1\}$ $h_0 = p - 1$	**Superstep 0:** $w_0^{(s)} = 2\left\lceil \frac{n}{p} \right\rceil \quad \forall\, s \in \{0, \dots, n-1\}$ $h_0 = p - 1$
Superstep 1: $w_1^{(s)} = p \quad \forall\, s \in \{1, \dots, n-1\}$ $h_1 = 0$	**Superstep 1:** $w_1^{(0)} = p$ $w_1^{(s)} = 0 \quad \forall\, s \in \{1, \dots, n-1\}$ $h_1 = p - 1$

Task 2.4 (1)

Task 2.4: Use the BSP cost formula (slide 33) to calculate the final costs of your adapted algorithm. Compare it to the costs of the lecture. What is remarkable? Explain the result.

Costs per superstep i:

$$T_i = \frac{1}{r} \cdot \max_{0 \le s < p} w_i^{(s)} + g h_i + l$$

Costs of adapted algorithm:

$$T_{0,adapted} = \frac{2}{r} \cdot \left\lceil \frac{n}{p} \right\rceil + g(p-1) + l$$

$$T_{1,adapted} = \frac{p}{r} \cdot + g(p-1) + l$$

$$T_{adapted} = \frac{2}{r} \left\lceil \frac{n}{p} \right\rceil + \frac{p}{r} + 2g(p-1) + 2l$$

Adapted Algorithm

Superstep 0:

$w_0^{(s)} = 2 \left\lceil \frac{n}{p} \right\rceil \quad \forall\, s \in \{0, \dots, n-1\}$

$h_0 = p - 1$

Superstep 1:

$w_1^{(0)} = p$

$w_1^{(s)} = 0 \qquad \forall\, s \in \{1, \dots, n-1\}$

$h_1 = p - 1$

Task 2.4 (2)

$$T_{adapted} = \frac{2}{r}\left\lceil\frac{n}{p}\right\rceil + \frac{p}{r} + 2g(p-1) + 2l \qquad T_{lecture} = \frac{2}{r}\left\lceil\frac{n}{p}\right\rceil + \frac{p}{r} + g(p-1) + 2l$$

- BSP cost model tells us that the adapted algorithm is more expensive than the lecture algorithm
  - Cost model always takes the time of the process whose computation / communication takes the longest time
  - No difference between all processors sending / receiving to all other processes and using a single processor, even worse: Adapted algorithm requires additional communication phase ($g(p-1)$) for broadcast.

Task 2.5 (1)

Task 2.5: Provide the *total* number of FLOPS W and the total number of data elements received / transmitted R / T for both algorithms and explain the results.

Lecture Algorithm:

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

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

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

$W_0 = 2n$

$T_0 = p \cdot (p-1)$
$R_0 = p \cdot (p-1)$

$W_1 = p^2$

Task 2.5 (2)

Adapted Algorithm:

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

put αs in P(0);                // send result to 0
barrier();

if (s == 0) then
   α := 0;
   for (t :=0; t < p; t++) do
      α := α + αt;             // sum up local and received α values
   for (t :=0; t < p ; t++) do
      put α in P(t);           // broadcast final result to all processors t
fi
barrier();
```

$W_0 = 2n$

$T_0 = p - 1$
$R_0 = p - 1$

$W_1 = p$

$T_1 = p - 1$
$R_1 = p - 1$

Task 2.5 (3)

Lecture Algorithm:

$W = W_0 + W_1 = 2n + p^2$

$T = T_0 = p \cdot (p-1) = p^2 - p$

$R = R_0 = p \cdot (p-1) = p^2 - p$

Adapted Algorithm:

$W = W_0 + W_1 = 2n + p$

$T = T_0 + T_1 = 2(p-1) = 2p - 2$

$R = R_0 + R_1 = 2(p-1) = 2p - 2$

- In the lecture algorithm, all processes compute the final result locally, so the total number of FLOPS in the second superstep is p^2.
- All-to-all communication in lecture algorithm leads to higher number of transmitted / received data.

Task 2.6

Task 2.6: After solving the previous tasks, think about the expressiveness of the BSP cost model and write down at least one resulting weakness.

Solution:

- *Overall traffic* performed in the network is not considered, only isolated view for single processes
  - Possible congestion and saturation of data links not modeled
- *Overall CPU work* performed is not considered, only isolated view for single processes
  - If all processes do exactly the same with the same data: Waste of energy and computing capacity, but no difference in BSP costs

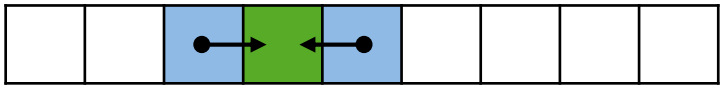
Exercise Evaluation

Task 3

Stencil Codes in BSP

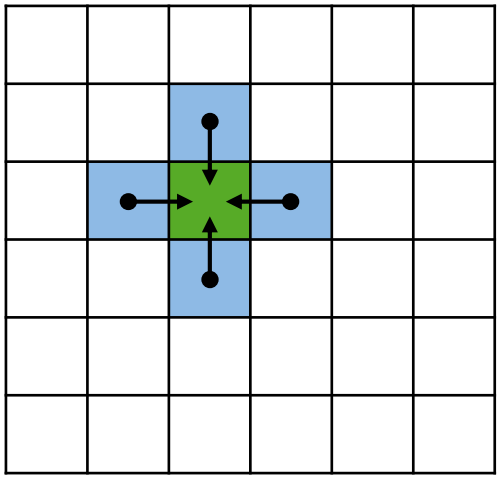
Task 3 (1)

- Stencil codes: Typical class of iterative kernels used in various scenarios (image processing, partial differential equations, ...).
- General idea: Perform a seqeuence of timesteps on a given n-dimensional data set.
  - In one timestep, for each data element called “cell” a new value is computed using the data of the neighboring cells.



1D stencil

$$x_i^{k+1} = 0.5 \cdot (x_{i-1}^k + x_{i+1}^k)$$



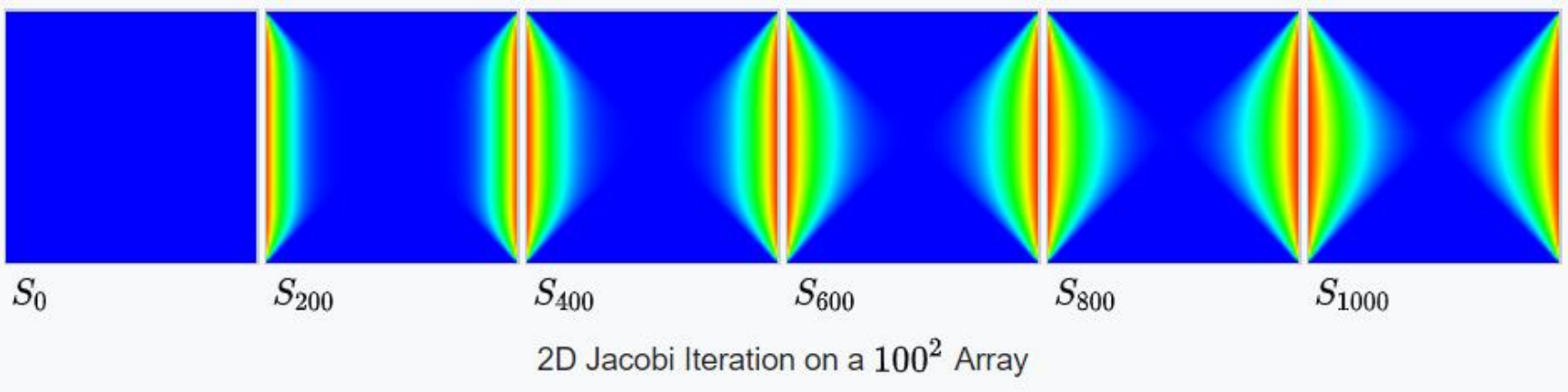
2D stencil

$$a_{ij}^{k+1} = 0.25 \cdot (a_{i-1j}^k + a_{ij+1}^k + a_{i+1j}^k + a_{ij-1}^k)$$

Task 3 (2)

- Example: Leftmost and rightmost column of matrix initialized with 1, other elements initialized with 0
- Computation rule for timestep $k + 1$ as defined before:

$$a_{ij}^{k+1} = 0.25 \cdot (a_{i-1j}^{k} + a_{ij+1}^{k} + a_{i+1j}^{k} + a_{ij-1}^{k})$$



Source: Wikipedia
https://en.wikipedia.org/wiki/Stencil_code

Task 3.1

Task 3.1: Start with the 1D stencil. Decide between a block and circular data distribution for the array x. Give the distribution function $distr$. Justify your answer.

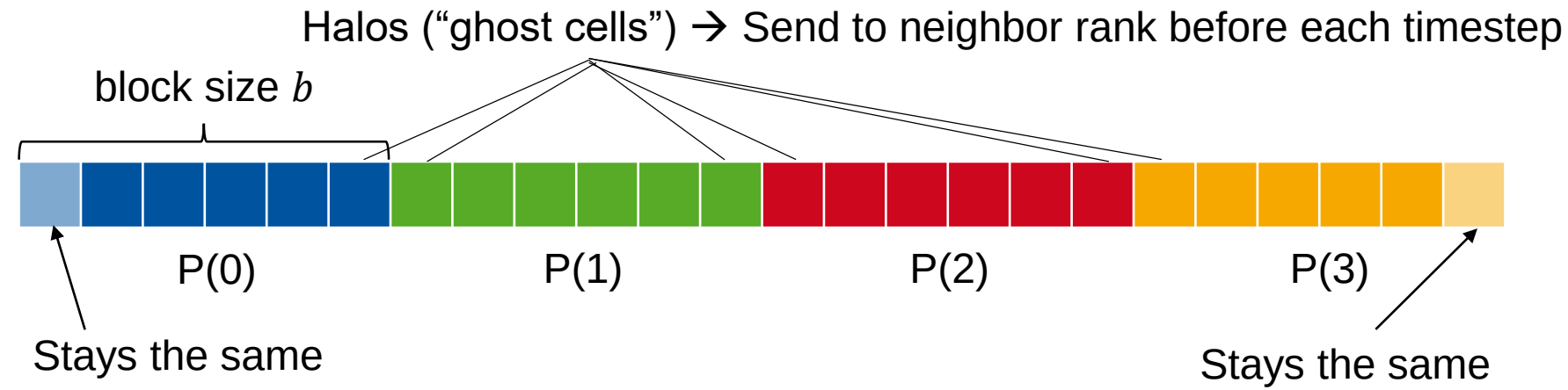
Solution:

- Stencil codes process neighboring elements and each cell is recomputed with the same operations (no load imbalances) → Block distribution

$distr: \{0, \dots, n-1\} \rightarrow \{0, \dots, p-1\}$

Element x_i: i $\mapsto \left\lfloor \frac{i}{b} \right\rfloor$, for $0 \leq i < n$ where $b := \left\lceil \frac{n}{p} \right\rceil$ is the block size

Task 3.2 (1)



Data elements P(1) has to send:

Generalized to processor s:

- s sends its leftmost element x_{b*s} to processor $s-1$
- s sends its rightmost element $x_{b*(s+1)-1}$ to $s+1$

```
if (s > 0) then
   put x_{b*s} in P(s-1);            // Send leftmost element
if (s < p-1) then
   put x_{b*(s+1)-1} in P(s+1); // Send rightmost element
```

Task 3.2 (2)

Input: k: # timesteps, x: vector of length n, block distributed $b := \lceil \frac{n}{p} \rceil$

Output: x after k timesteps

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

```
for (iter := 0; iter < k; iter++) do
  // Send elements required by neighbor
  if (s > 0) then
    put x_{b*s} in P(s-1);          // Send leftmost element
  if (s < p-1) then
    put x_{b*(s+1)-1} in P(s+1); // Send rightmost element
  barrier();

  // All required data available, do stencil operation locally
  for (i := b * s; i < min(b*(s+1), n); i++) do
    if (i != 0 && i != n-1) // exclude outer elements
      x_i = 0.5 · (x_{i-1} + x_{i+1})
```

Comm. Phase

Comp. Phase

Superstep

Task 3.3 – BSP Costs (1)

```
for (iter := 0; iter < k; iter++) do
  // Send elements required by neighbor
  if (s > 0) then
    put x_{b*s} in P(s-1);          // Send leftmost element
  if (s < p-1) then
    put x_{b*(s+1)-1} in P(s+1); // Send rightmost element
  barrier();
```

- All BSP costs are the same for all iterations, so we omit the superstep index.
- $t^{(i)} = r^{(i)} = \begin{cases} 2 & i \in \{1, \dots, p-2\} \\ 1 & i = 0,\ i = p-1 \end{cases}$ (send / receive right and left neighbor)
- $h = 2$ (for all supersteps)

Task 3.3 – BSP Costs (2)

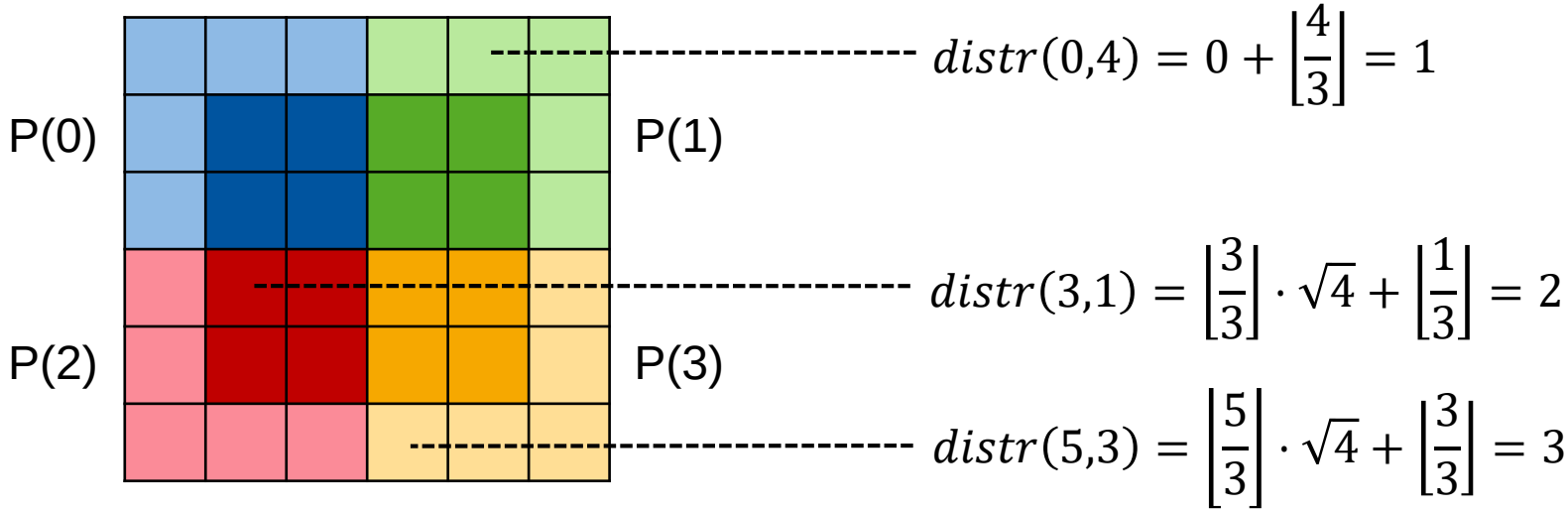
```
// All required data available, do stencil operation locally
for (i := b * s; i < min(b*(s+1), n); i++) do
  if (i != 0 && i != n-1) // exclude outer elements
    x_i = 0.5 · (x_{i-1} + x_{i+1})
```

- Remember: $b := \left\lceil \frac{n}{p} \right\rceil$
- $w^{(i)} = \begin{cases} 2b & i \in \{1, \dots, p-2\} \\ 2b-1 & i = 0 \\ 2b-1-(pb-n) & i = p-1 \end{cases}$
- $T_i = \frac{1}{r} \cdot \max_{0 \le s < p} w_i^{(s)} + gh_i + l = \frac{1}{r} \cdot 2b + 2g + l$ (same for each step)
- BSP costs for k supersteps: $T(k) = k \cdot \left(\frac{1}{r} \cdot 2b + 2g + l\right) \underbrace{+ l}$

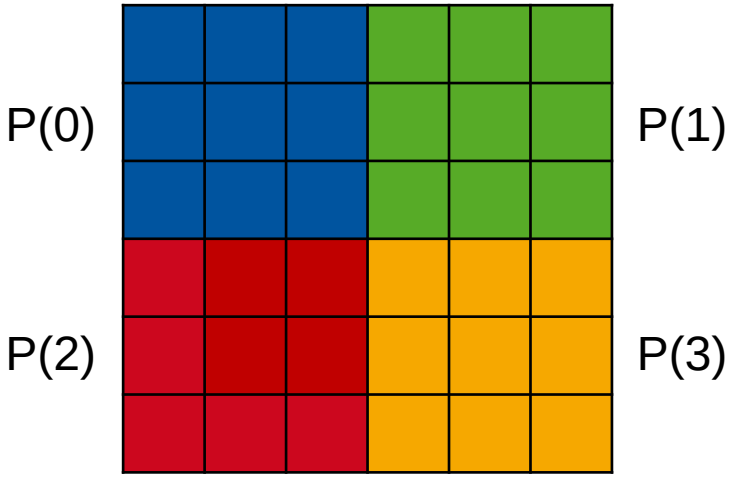
Implicit barrier after last iteration

Task 3.4 – Generalization to 2D von Neumann Stencil

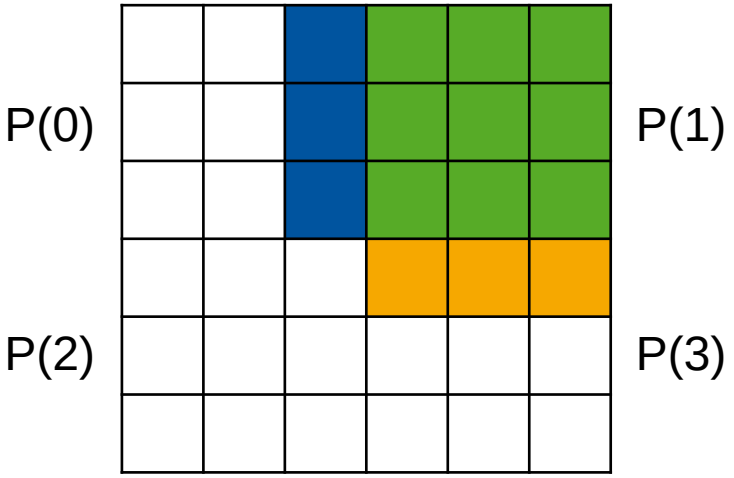
- $\mathrm{n} \times n$ matrix, $n = 6$
- $p = 4$, $\sqrt{p} = 2$, submatrix size $n^2/p = 9$, column / row size $\mathrm{b} := n/\sqrt{p} = 3$
- $distr\colon \{0, \dots, n-1\} \times \{0, \dots, n-1\} \to \{0, \dots, p-1\}$
- $distr(i,j) = \left\lfloor \frac{i}{b} \right\rfloor \cdot \sqrt{p} + \left\lfloor \frac{j}{b} \right\rfloor$ for $i, j \in \{0, \dots, n-1\}$ where $b := \frac{n}{\sqrt{p}} = 3$

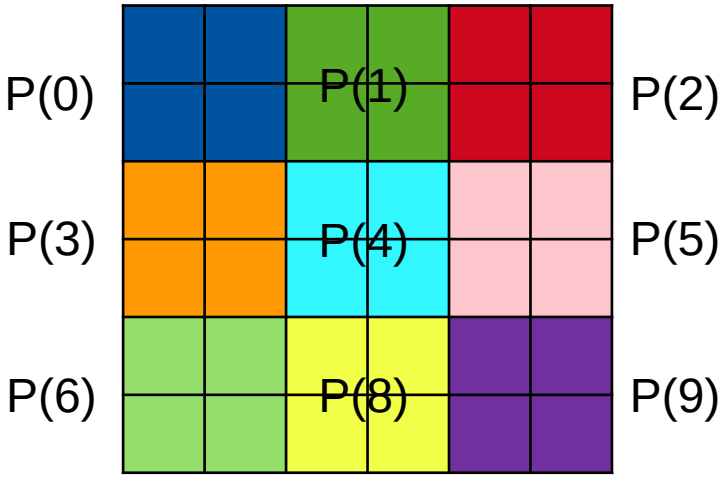


Task 3.4 – Generalization to 2D von Neumann Stencil

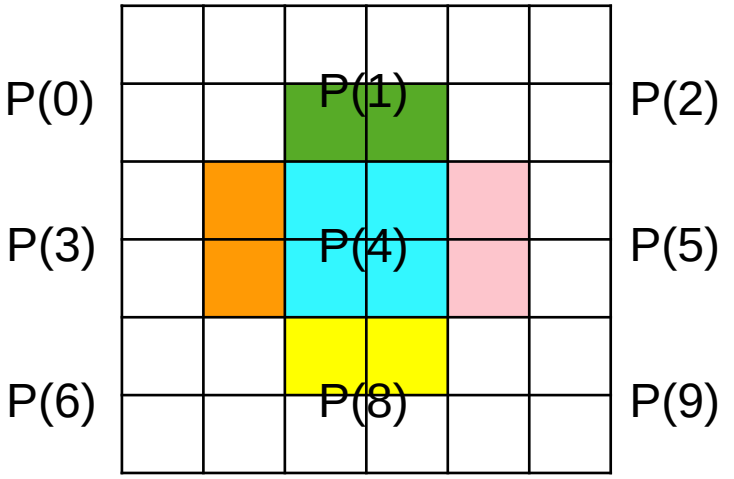


Data exchange P(1):

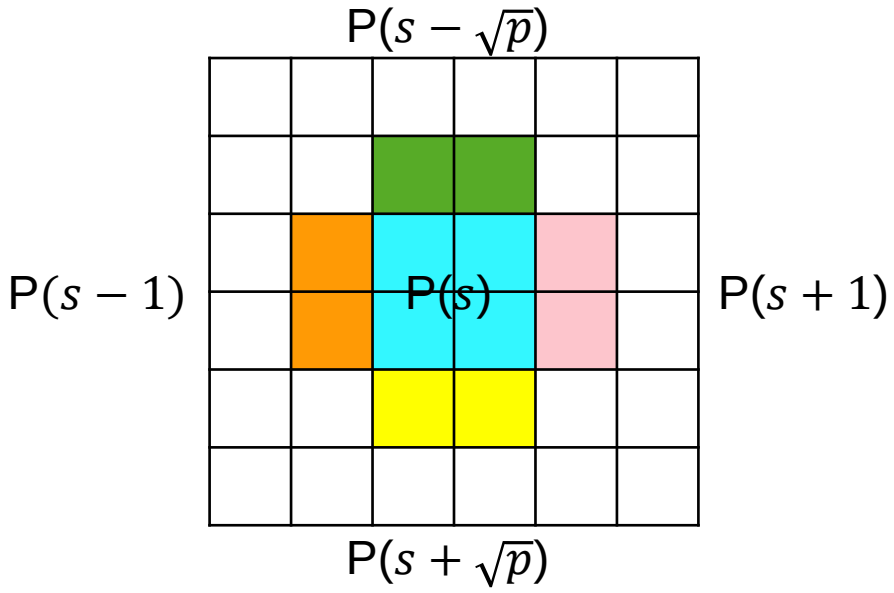




Data exchange P(4):



Task 3.4



- s sends its leftmost column $[a_{0,0}, a_{1,0}, \ldots a_{b-1,0}]$ to $s - 1$
- s sends its rightmost column $[a_{0,b-1}, a_{1,b-1}, \ldots a_{b-1,b-1}]$ to $s + 1$
- s sends its bottom row $[a_{b-1,0}, a_{b-1,1}, \ldots a_{b-1,b-1}]$ to $s + \sqrt{p}$
- s sends its top row $[a_{0,0}, a_{0,1}, \ldots a_{0,b-1}]$ to $s - \sqrt{p}$
- Worst case: Process has to exchange neighbors with 4 processes
- Note: We use local addressing here for simplicity.

Task 3.4

Input: k: # timesteps, a: matrix of size $n \times n$, block distributed

Output: a after k timesteps

```
for (iter := 0; iter < k; iter++) do
  // Send elements required by neighbor
  if (s mod √p != 0) then            // Current rank is not outer left
    for (int i=0; i < b; i++)        // Send leftmost column
      put a_{i,0} in P(s-1);
  if (s mod √p != √p-1) then         // Current rank is not outer right
    for (int i=0; i < b; i++)        // Send rightmost column
      put a_{i,b-1} in P(s+1);
  if (⌊s/√p⌋ != 0) then              // Current rank is not outer top
    for (int i=0; i < b; i++)        // Send top row
      put a_{0,i} in P(s-√p);
  if (⌊s/√p⌋ != √p-1) then           // Current rank is not outer bottom
    for (int i=0; i < b; i++)        // Send bottom row
      put a_{b-1,i} in P(s+√p);
  barrier();
```

Comm. Phase

Task 3.4

Input: k: # timesteps, a: matrix of size $n \times n$, block distributed

Output: a after k timesteps

```
// All required data available, do stencil operation locally
for (i := ⌊s/√p⌋ · b; i < ⌊s/√p⌋ · (b + 1); i++) do
  for (j := (s · b) mod n; j < (s · (b + 1)) mod n; j++) do
    if (i != 0 && i != n-1 && j != 0 && j != n-1) // exclude outer elements
      a_ij = 0.25 · (a_(i-1)j + a_ij+1 + a_i+1j + a_ij-1)
```

Comp. Phase

Task 3.4 – BSP Costs

- $n \times n$ matrix, p submatrices of size n^2/p, column / row size $n/\sqrt{p}$
- $\max\limits_{0 \le s < p} w_i^{(s)} = 4 \cdot \frac{n^2}{p}$ (4 FLOPs per cell, $\frac{n^2}{p}$ cells computed in the worst case)
- $h_i = 4 \cdot \frac{n}{\sqrt{p}}$ (Processor has to send 4 rows / columns in the worst case)
- $T_i = \frac{1}{r} \cdot \max\limits_{0 \le s < p} w_i^{(s)} + g h_i + l = 4 \cdot \frac{n^2}{rp} \cdot +4 \cdot \frac{ng}{\sqrt{p}} + l$ (same for each step)
- BSP costs for k supersteps: $T(k) = k \cdot \left(4 \cdot \frac{n^2}{rp} \cdot +4 \cdot \frac{ng}{\sqrt{p}} + \underbrace{l}\right) + l$

Implicit barrier after last iteration

Task 4

K-means in Bulk

Task 4

Live Demo

(Solution available in Moodle course room)