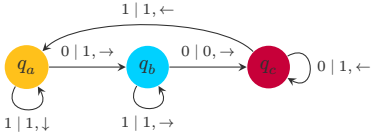
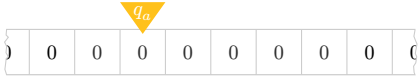
Fast computations in higher-dimensional tilings

Antonin Callard

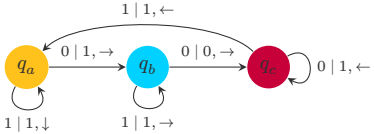
Journées SDA2 2025 – Montpellier

Turing machines

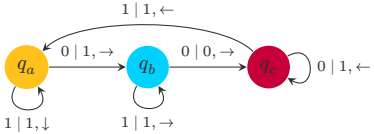


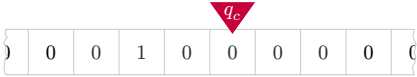


Turing machines

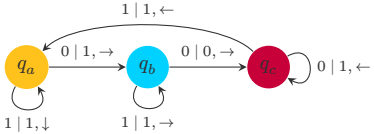


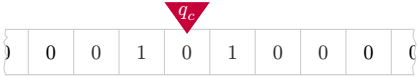
Turing machines



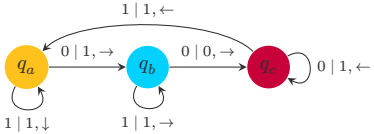


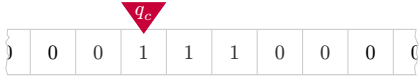
Turing machines



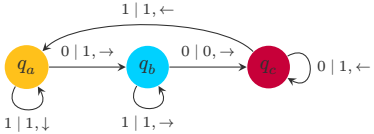


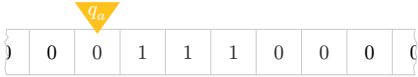
Turing machines



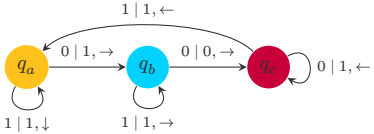


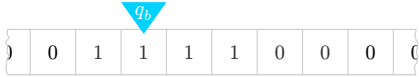
Turing machines



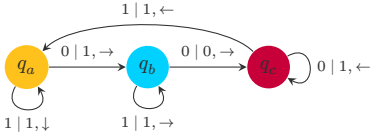


Turing machines

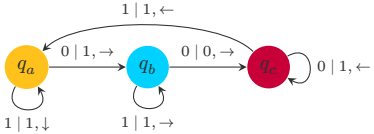




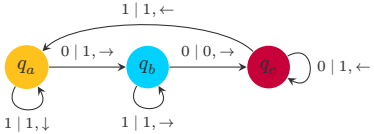
Turing machines

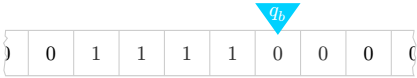


Turing machines

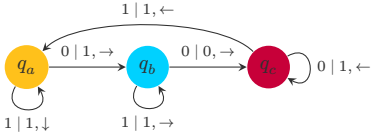


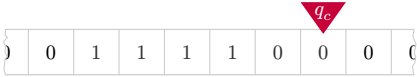
Turing machines



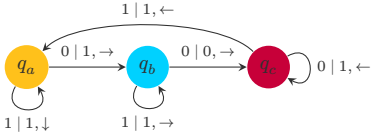


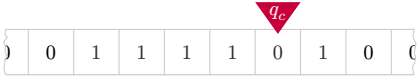
Turing machines



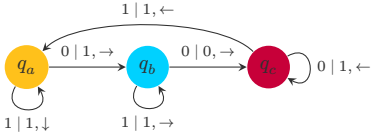


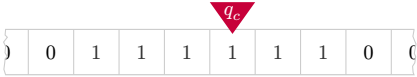
Turing machines



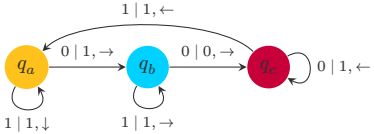


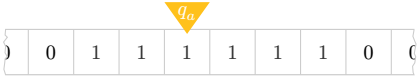
Turing machines



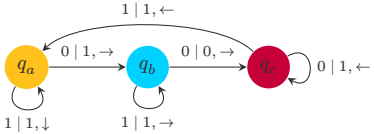


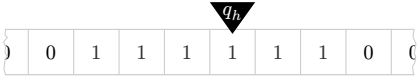
Turing machines



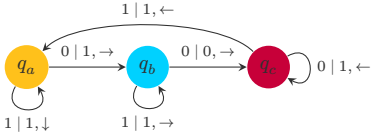


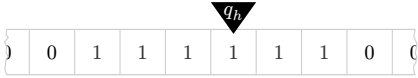
Turing machines



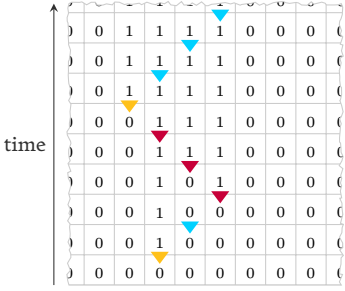


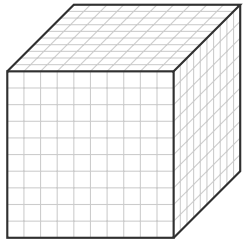
Turing machines





In 2-dimensional tilings, we can embed arbitrary computations by drawing *space-time diagrams*:





Main question

But what about higher dimensions? Space/time tradeoff?

Mesh-Connected MultiComputers

Mesh-Connected MultiComputers

Definition

A *Mesh-Connected MultiComputer* is an array of *processors*.

A processor:

- Contains finitely many *variables*;
- Can perform arithmetic operations on these variables;
- **Can communicate with its immediate neighbors.**

Mesh-Connected MultiComputers

Definition

A *Mesh-Connected MultiComputer* is an array of *processors*.

A processor:

- Contains finitely many *variables*;
- Can perform arithmetic operations on these variables;
- **Can communicate with its immediate neighbors.**

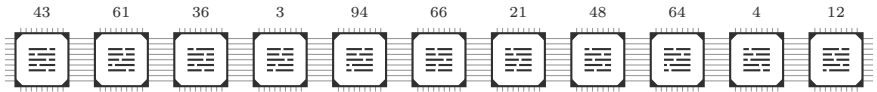
Consider the SORTING problem:

SORTING
Input: An array a of integers
Output: The array a sorted in increasing order

a =

43	61	36	3	94	66	21	48	64	4	12

Mesh-Connected MultiComputers



Definition

A *Mesh-Connected MultiComputer* is an array of *processors*.

A processor:

- Contains finitely many *variables*;
- Can perform arithmetic operations on these variables;
- **Can communicate with its immediate neighbors.**

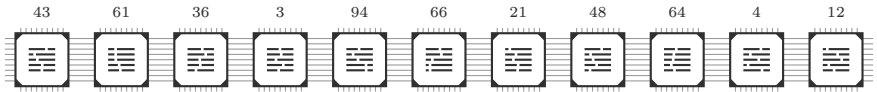
Consider the SORTING problem:

SORTING
Input: An array a of integers
Output: The array a sorted in increasing order

a =

43	61	36	3	94	66	21	48	64	4	12

Mesh-Connected MultiComputers



Definition

A *Mesh-Connected MultiComputer* is an array of *processors*.

A processor:

- Contains finitely many *variables*;
- Can perform arithmetic operations on these variables;
- **Can communicate with its immediate neighbors.**

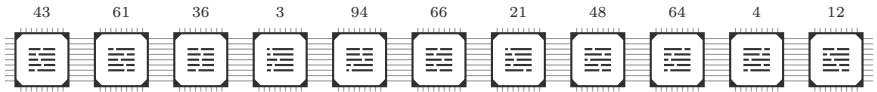
Consider the SORTING problem:

SORTING
Input: An array a of integers
Output: The array a sorted in increasing order

a =

43	61	36	3	94	66	21	48	64	4	12

Mesh-Connected MultiComputers



Definition

A *Mesh-Connected MultiComputer* is an array of *processors*.

A processor:

- Contains finitely many *variables*;
- Can perform arithmetic operations on these variables;
- **Can communicate with its immediate neighbors.**

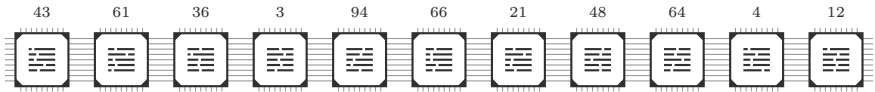
Consider the SORTING problem:

SORTING
Input: An array a of integers
Output: The array a sorted in increasing order

a =

43	61	36	3	94	66	21	48	64	4	12

Mesh-Connected MultiComputers



Definition

A *Mesh-Connected MultiComputer* is an array of *processors*.

A processor:

- Contains finitely many *variables*;
- Can perform arithmetic operations on these variables;
- **Can communicate with its immediate neighbors.**

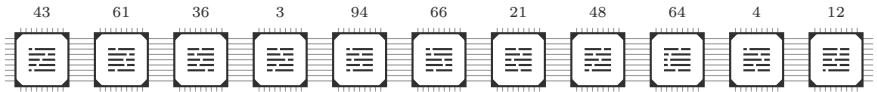
Consider the SORTING problem:

SORTING
Input: An array a of integers
Output: The array a sorted in increasing order

a =

43	61	36	3	94	66	21	48	64	4	12

Mesh-Connected MultiComputers



Definition

A *Mesh-Connected MultiComputer* is an array of *processors*.

A processor:

- Contains finitely many *variables*;
- Can perform arithmetic operations on these variables;
- **Can communicate with its immediate neighbors.**

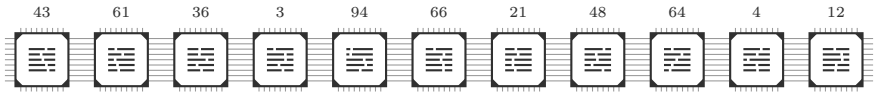
Consider the SORTING problem:

SORTING
Input: An array a of integers
Output: The array a sorted in increasing order

a =

43	61	36	3	94	66	21	48	64	4	12

Mesh-Connected MultiComputers



Definition

A *Mesh-Connected MultiComputer* is an array of *processors*.

A processor:

- Contains finitely many *variables*;
- Can perform arithmetic operations on these variables;
- **Can communicate with its immediate neighbors.**

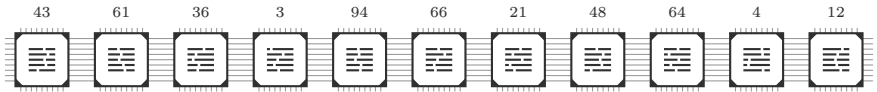
Consider the SORTING problem:

SORTING
Input: An array a of integers
Output: The array a sorted in increasing order

a =

43	61	36	3	94	66	21	48	64	4	12

Mesh-Connected MultiComputers



Definition

A *Mesh-Connected MultiComputer* is an array of *processors*.

A processor:

- Contains finitely many *variables*;
- Can perform arithmetic operations on these variables;
- **Can communicate with its immediate neighbors.**

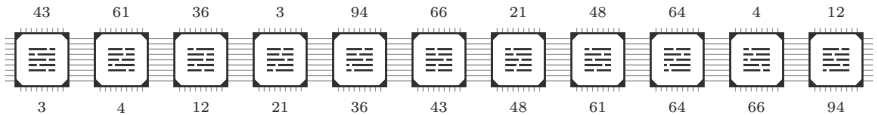
Consider the SORTING problem:

SORTING
Input: An array a of integers
Output: The array a sorted in increasing order

a =

43	61	36	3	94	66	21	48	64	4	12

Mesh-Connected MultiComputers



Definition

A *Mesh-Connected MultiComputer* is an array of *processors*.

A processor:

- Contains finitely many *variables*;
- Can perform arithmetic operations on these variables;
- **Can communicate with its immediate neighbors.**

Consider the SORTING problem:

SORTING
Input: An array a of integers
Output: The array a sorted in increasing order

a =

43	61	36	3	94	66	21	48	64	4	12

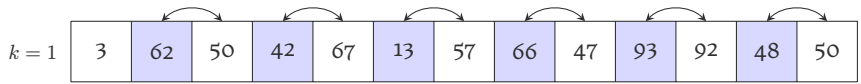
Sorting on a Mesh-Connected MultiComputer

function EVEN-ODD SORT(a) ▷ *a is an array of size n*
  for $k = 1, \ldots$ **do**
    if k is even **then**
      Compare (and swap if needed) every even position with its right neighbor;
    else (k is odd)
      Compare (and swap if needed) every odd position with its right neighbor.

3	62	50	42	67	13	57	66	47	93	92	48	50

Sorting on a Mesh-Connected MultiComputer

function EVEN-ODD SORT(a) ▷ *a is an array of size n*
  for $k = 1, \ldots$ **do**
    if k is even **then**
      Compare (and swap if needed) every even position with its right neighbor;
    else (k is odd)
      Compare (and swap if needed) every odd position with its right neighbor.



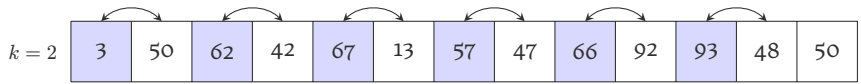
Sorting on a Mesh-Connected MultiComputer

function EVEN-ODD SORT(a) ▷ *a is an array of size n*
  for $k = 1, \ldots$ **do**
    if k is even **then**
      Compare (and swap if needed) every even position with its right neighbor;
    else (k is odd)
      Compare (and swap if needed) every odd position with its right neighbor.

$k = 1$	3	50	62	42	67	13	57	47	66	92	93	48	50

Sorting on a Mesh-Connected MultiComputer

function EVEN-ODD SORT(a) ▷ *a is an array of size n*
  for $k = 1, \ldots$ **do**
    if k is even **then**
      Compare (and swap if needed) every even position with its right neighbor;
    else (k is odd)
      Compare (and swap if needed) every odd position with its right neighbor.



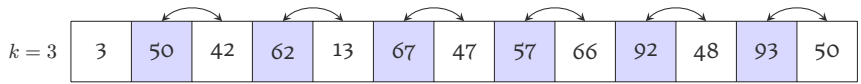
Sorting on a Mesh-Connected MultiComputer

```
function EVEN-ODD SORT(a)                                   ▷ a is an array of size n
    for k = 1, ... do
        if k is even then
            Compare (and swap if needed) every even position with its right neighbor;
        else (k is odd)
            Compare (and swap if needed) every odd position with its right neighbor.
```

$k = 2$	3	50	42	62	13	67	47	57	66	92	48	93	50

Sorting on a Mesh-Connected MultiComputer

```
function EVEN-ODD SORT(a)                                   ▷ a is an array of size n
    for k = 1, ... do
        if k is even then
            Compare (and swap if needed) every even position with its right neighbor;
        else (k is odd)
            Compare (and swap if needed) every odd position with its right neighbor.
```



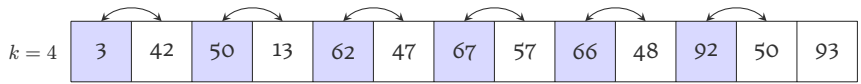
Sorting on a Mesh-Connected MultiComputer

```
function EVEN-ODD SORT(a)                                   ▷ a is an array of size n
    for k = 1, ... do
        if k is even then
            Compare (and swap if needed) every even position with its right neighbor;
        else (k is odd)
            Compare (and swap if needed) every odd position with its right neighbor.
```

$k = 3$	3	42	50	13	62	47	67	57	66	48	92	50	93

Sorting on a Mesh-Connected MultiComputer

function EVEN-ODD SORT(a) ▷ *a is an array of size n*
  for $k = 1, \ldots$ **do**
    if k is even **then**
      Compare (and swap if needed) every even position with its right neighbor;
    else (k is odd)
      Compare (and swap if needed) every odd position with its right neighbor.



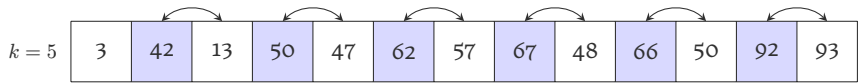
Sorting on a Mesh-Connected MultiComputer

function EVEN-ODD SORT(a) ▷ *a is an array of size n*
    for $k = 1, \ldots$ **do**
        if k is even **then**
            Compare (and swap if needed) every even position with its right neighbor;
        else (k is odd)
            Compare (and swap if needed) every odd position with its right neighbor.

$k = 4$	3	42	13	50	47	62	57	67	48	66	50	92	93

Sorting on a Mesh-Connected MultiComputer

function EVEN-ODD SORT(a) ▷ *a is an array of size n*
  for $k = 1, \ldots$ **do**
    if k is even **then**
      Compare (and swap if needed) every even position with its right neighbor;
    else (k is odd)
      Compare (and swap if needed) every odd position with its right neighbor.



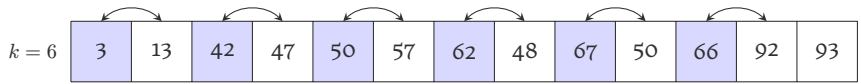
Sorting on a Mesh-Connected MultiComputer

```
function EVEN-ODD SORT(a)                                   ▷ a is an array of size n
    for k = 1, ... do
        if k is even then
            Compare (and swap if needed) every even position with its right neighbor;
        else (k is odd)
            Compare (and swap if needed) every odd position with its right neighbor.
```

$k = 5$	3	13	42	47	50	57	62	48	67	50	66	92	93

Sorting on a Mesh-Connected MultiComputer

function EVEN-ODD SORT(a) ▷ *a is an array of size n*
  for $k = 1, \ldots$ **do**
    if k is even **then**
      Compare (and swap if needed) every even position with its right neighbor;
    else (k is odd)
      Compare (and swap if needed) every odd position with its right neighbor.



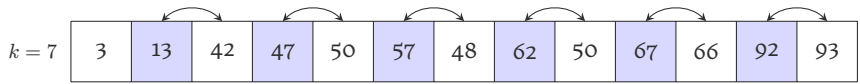
Sorting on a Mesh-Connected MultiComputer

function EVEN-ODD SORT(a) ▷ *a is an array of size n*
  for $k = 1, \ldots$ **do**
    if k is even **then**
      Compare (and swap if needed) every even position with its right neighbor;
    else (k is odd)
      Compare (and swap if needed) every odd position with its right neighbor.

$k = 6$	3	13	42	47	50	57	48	62	50	67	66	92	93

Sorting on a Mesh-Connected MultiComputer

function EVEN-ODD SORT(a) ▷ *a is an array of size n*
  for $k = 1, \ldots$ **do**
    if k is even **then**
      Compare (and swap if needed) every even position with its right neighbor;
    else (k is odd)
      Compare (and swap if needed) every odd position with its right neighbor.



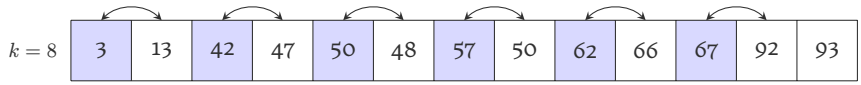
Sorting on a Mesh-Connected MultiComputer

function EVEN-ODD SORT(a) ▷ *a is an array of size n*
  for $k = 1, \ldots$ **do**
    if k is even **then**
      Compare (and swap if needed) every even position with its right neighbor;
    else (k is odd)
      Compare (and swap if needed) every odd position with its right neighbor.

$k = 7$	3	13	42	47	50	48	57	50	62	66	67	92	93

Sorting on a Mesh-Connected MultiComputer

function EVEN-ODD SORT(a) ▷ *a is an array of size n*
  for $k = 1, \ldots$ **do**
    if k is even **then**
      Compare (and swap if needed) every even position with its right neighbor;
    else (k is odd)
      Compare (and swap if needed) every odd position with its right neighbor.



Sorting on a Mesh-Connected MultiComputer

```
function EVEN-ODD SORT(a)                                   ▷ a is an array of size n
    for k = 1, ... do
        if k is even then
            Compare (and swap if needed) every even position with its right neighbor;
        else (k is odd)
            Compare (and swap if needed) every odd position with its right neighbor.
```

$k = 8$	3	13	42	47	48	50	50	57	62	66	67	92	93

Sorting on a Mesh-Connected MultiComputer

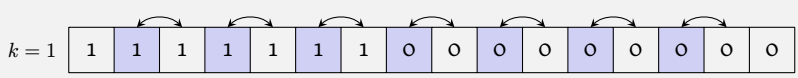
function EVEN-ODD SORT(a) ▷ *a is an array of size n*
  for $k = 1, \ldots$ **do**
    if k is even **then**
      Compare (and swap if needed) every even position with its right neighbor;
    else (k is odd)
      Compare (and swap if needed) every odd position with its right neighbor.

$k = 8$	3	13	42	47	48	50	50	57	62	66	67	92	93

Lemma

If the array a has size n, then it is sorted after the iteration $k = n$ of EVEN-ODD SORT.

Sketch of proof.



□

Sorting on a Mesh-Connected MultiComputer

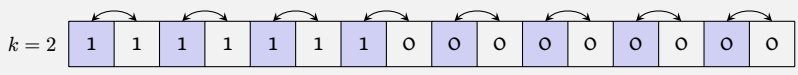
function EVEN-ODD SORT(a) ▷ *a is an array of size n*
  for $k = 1, \ldots$ **do**
    if k is even **then**
      Compare (and swap if needed) every even position with its right neighbor;
    else (k is odd)
      Compare (and swap if needed) every odd position with its right neighbor.

$k = 8$	3	13	42	47	48	50	50	57	62	66	67	92	93

Lemma

If the array a has size n, then it is sorted after the iteration $k = n$ of EVEN-ODD SORT.

Sketch of proof.



□

Sorting on a Mesh-Connected MultiComputer

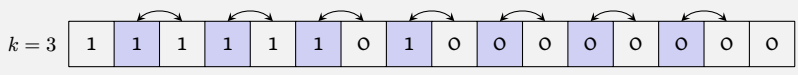
function EVEN-ODD SORT(a) ▷ *a is an array of size n*
  for $k = 1, \ldots$ **do**
    if k is even **then**
      Compare (and swap if needed) every even position with its right neighbor;
    else (k is odd)
      Compare (and swap if needed) every odd position with its right neighbor.

$k = 8$	3	13	42	47	48	50	50	57	62	66	67	92	93

Lemma

If the array a has size n, then it is sorted after the iteration $k = n$ of EVEN-ODD SORT.

Sketch of proof.



□

Sorting on a Mesh-Connected MultiComputer

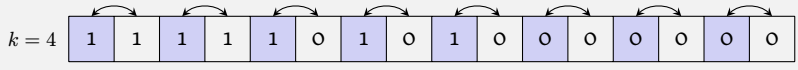
function EVEN-ODD SORT(a) ▷ *a is an array of size n*
  for $k = 1, \ldots$ **do**
    if k is even **then**
      Compare (and swap if needed) every even position with its right neighbor;
    else (k is odd)
      Compare (and swap if needed) every odd position with its right neighbor.

$k = 8$	3	13	42	47	48	50	50	57	62	66	67	92	93

Lemma

If the array a has size n, then it is sorted after the iteration $k = n$ of EVEN-ODD SORT.

Sketch of proof.



□

Sorting on a Mesh-Connected MultiComputer

function EVEN-ODD SORT(a) ▷ *a is an array of size n*
  for $k = 1, \ldots$ **do**
    if k is even **then**
      Compare (and swap if needed) every even position with its right neighbor;
    else (k is odd)
      Compare (and swap if needed) every odd position with its right neighbor.

$k = 8$	3	13	42	47	48	50	50	57	62	66	67	92	93

Lemma

If the array a has size n, then it is sorted after the iteration $k = n$ of EVEN-ODD SORT.

Sketch of proof.

$k = 5$	1	1	1	1	0	1	0	1	0	1	0	0	0	0	0	0

□

Sorting on a Mesh-Connected MultiComputer

function EVEN-ODD SORT(a) ▷ *a is an array of size n*
  for $k = 1, \ldots$ **do**
    if k is even **then**
      Compare (and swap if needed) every even position with its right neighbor;
    else (k is odd)
      Compare (and swap if needed) every odd position with its right neighbor.

$k = 8$	3	13	42	47	48	50	50	57	62	66	67	92	93

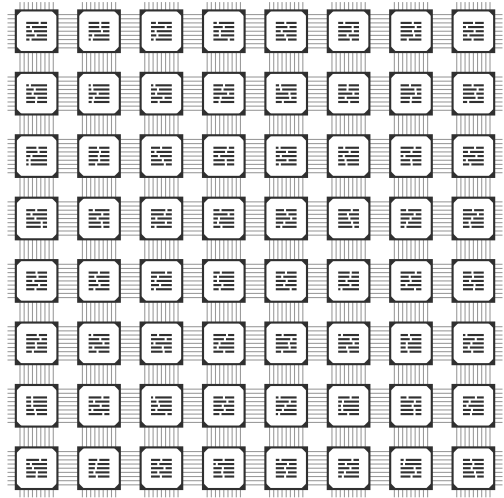
Lemma

If the array a has size n, then it is sorted after the iteration $k = n$ of EVEN-ODD SORT.

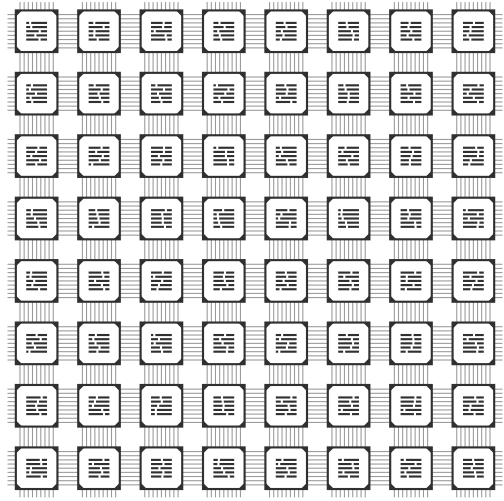
Corollary

The SORTING problem on MCMCs can be solved in time $O(n)$ on arrays of size n.

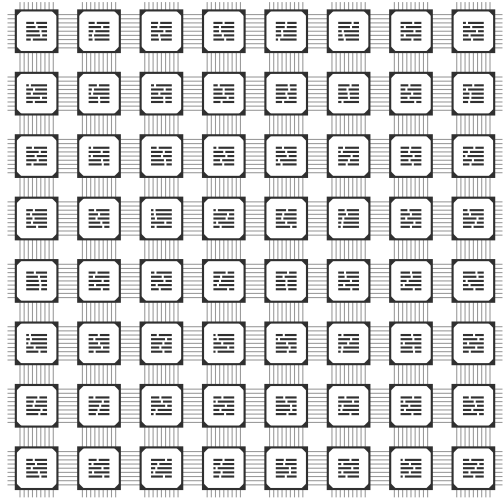
Sorting on a 2D Mesh-Connected MultiComputer



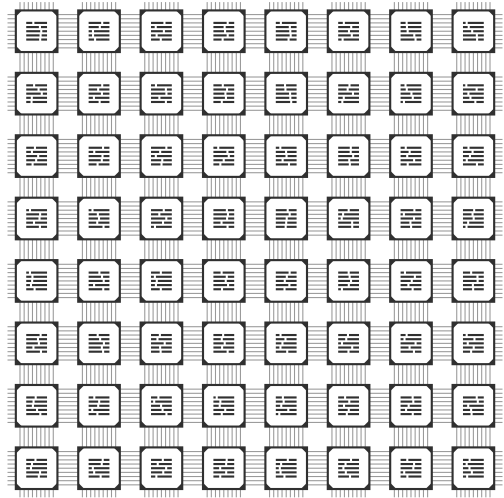
Sorting on a 2D Mesh-Connected MultiComputer



Sorting on a 2D Mesh-Connected MultiComputer



Sorting on a 2D Mesh-Connected MultiComputer



Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*

  for $k = 1, \ldots$ **do**

    Sort rows in increasing and decreasing order alternatively;

    Sort columns in increasing order.

11	80	47	13	33	5	9	99	77	24
91	5	24	80	85	43	39	53	60	67
55	8	9	50	13	94	77	50	24	28
32	97	20	43	45	60	79	23	71	89
75	16	59	88	72	4	90	53	3	3
48	39	24	18	21	50	37	21	12	30
5	11	82	99	11	14	47	89	43	1
20	56	61	58	95	68	45	79	78	12
92	48	50	13	92	96	18	50	14	53
3	62	50	42	67	13	57	66	47	93

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*

  for $k = 1, \ldots$ **do**

    Sort rows in increasing and decreasing order alternatively;

    Sort columns in increasing order.

$k = 1$

11	80	47	13	33	5	9	99	77	24
91	5	24	80	85	43	39	53	60	67
55	8	9	50	13	94	77	50	24	28
32	97	20	43	45	60	79	23	71	89
75	16	59	88	72	4	90	53	3	3
48	39	24	18	21	50	37	21	12	30
5	11	82	99	11	14	47	89	43	1
20	56	61	58	95	68	45	79	78	12
92	48	50	13	92	96	18	50	14	53
3	62	50	42	67	13	57	66	47	93

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*

  for $k = 1, \ldots$ **do**

    Sort rows in increasing and decreasing order alternatively;

    Sort columns in increasing order.

$k = 1$

5	9	11	13	24	33	47	77	80	99
91	5	24	80	85	43	39	53	60	67
55	8	9	50	13	94	77	50	24	28
32	97	20	43	45	60	79	23	71	89
75	16	59	88	72	4	90	53	3	3
48	39	24	18	21	50	37	21	12	30
5	11	82	99	11	14	47	89	43	1
20	56	61	58	95	68	45	79	78	12
92	48	50	13	92	96	18	50	14	53
3	62	50	42	67	13	57	66	47	93

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*

  for $k = 1, \ldots$ **do**

    Sort rows in increasing and decreasing order alternatively;

    Sort columns in increasing order.

$k = 1$

5	9	11	13	24	33	47	77	80	99
91	5	24	80	85	43	39	53	60	67
55	8	9	50	13	94	77	50	24	28
32	97	20	43	45	60	79	23	71	89
75	16	59	88	72	4	90	53	3	3
48	39	24	18	21	50	37	21	12	30
5	11	82	99	11	14	47	89	43	1
20	56	61	58	95	68	45	79	78	12
92	48	50	13	92	96	18	50	14	53
3	62	50	42	67	13	57	66	47	93

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*

  for $k = 1, \ldots$ **do**

    Sort rows in increasing and decreasing order alternatively;

    Sort columns in increasing order.

$k = 1$

5	9	11	13	24	33	47	77	80	99
91	85	80	67	60	53	43	39	24	5
55	8	9	50	13	94	77	50	24	28
32	97	20	43	45	60	79	23	71	89
75	16	59	88	72	4	90	53	3	3
48	39	24	18	21	50	37	21	12	30
5	11	82	99	11	14	47	89	43	1
20	56	61	58	95	68	45	79	78	12
92	48	50	13	92	96	18	50	14	53
3	62	50	42	67	13	57	66	47	93

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*

  for $k = 1, \ldots$ **do**

    Sort rows in increasing and decreasing order alternatively;

    Sort columns in increasing order.

$k = 1$

5	9	11	13	24	33	47	77	80	99
91	85	80	67	60	53	43	39	24	5
55	8	9	50	13	94	77	50	24	28
32	97	20	43	45	60	79	23	71	89
75	16	59	88	72	4	90	53	3	3
48	39	24	18	21	50	37	21	12	30
5	11	82	99	11	14	47	89	43	1
20	56	61	58	95	68	45	79	78	12
92	48	50	13	92	96	18	50	14	53
3	62	50	42	67	13	57	66	47	93

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*

  for $k = 1, \ldots$ **do**

    Sort rows in increasing and decreasing order alternatively;

    Sort columns in increasing order.

$k = 1$

5	9	11	13	24	33	47	77	80	99
91	85	80	67	60	53	43	39	24	5
8	9	13	24	28	50	50	55	77	94
32	97	20	43	45	60	79	23	71	89
75	16	59	88	72	4	90	53	3	3
48	39	24	18	21	50	37	21	12	30
5	11	82	99	11	14	47	89	43	1
20	56	61	58	95	68	45	79	78	12
92	48	50	13	92	96	18	50	14	53
3	62	50	42	67	13	57	66	47	93

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*

  for $k = 1, \ldots$ **do**

    Sort rows in increasing and decreasing order alternatively;

    Sort columns in increasing order.

$k = 1$

5	9	11	13	24	33	47	77	80	99
91	85	80	67	60	53	43	39	24	5
8	9	13	24	28	50	50	55	77	94
32	97	20	43	45	60	79	23	71	89
75	16	59	88	72	4	90	53	3	3
48	39	24	18	21	50	37	21	12	30
5	11	82	99	11	14	47	89	43	1
20	56	61	58	95	68	45	79	78	12
92	48	50	13	92	96	18	50	14	53
3	62	50	42	67	13	57	66	47	93

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*

  for $k = 1, \ldots$ **do**

    Sort rows in increasing and decreasing order alternatively;

    Sort columns in increasing order.

$k = 1$

5	9	11	13	24	33	47	77	80	99
91	85	80	67	60	53	43	39	24	5
8	9	13	24	28	50	50	55	77	94
97	89	79	71	60	45	43	32	23	20
75	16	59	88	72	4	90	53	3	3
48	39	24	18	21	50	37	21	12	30
5	11	82	99	11	14	47	89	43	1
20	56	61	58	95	68	45	79	78	12
92	48	50	13	92	96	18	50	14	53
3	62	50	42	67	13	57	66	47	93

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*

  for $k = 1, \ldots$ **do**

    Sort rows in increasing and decreasing order alternatively;

    Sort columns in increasing order.

$k = 1$

5	9	11	13	24	33	47	77	80	99
91	85	80	67	60	53	43	39	24	5
8	9	13	24	28	50	50	55	77	94
97	89	79	71	60	45	43	32	23	20
3	3	4	16	53	59	72	75	88	90
50	48	39	37	30	24	21	21	18	12
1	5	11	11	14	43	47	82	89	99
95	79	78	68	61	58	56	45	20	12
13	14	18	48	50	50	53	92	92	96
93	67	66	62	57	50	47	42	13	3

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*

  for $k = 1, \ldots$ **do**

    Sort rows in increasing and decreasing order alternatively;

    Sort columns in increasing order.

$k = 1$

5	9	11	13	24	33	47	77	80	99
91	85	80	67	60	53	43	39	24	5
8	9	13	24	28	50	50	55	77	94
97	89	79	71	60	45	43	32	23	20
3	3	4	16	53	59	72	75	88	90
50	48	39	37	30	24	21	21	18	12
1	5	11	11	14	43	47	82	89	99
95	79	78	68	61	58	56	45	20	12
13	14	18	48	50	50	53	92	92	96
93	67	66	62	57	50	47	42	13	3

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*

  for $k = 1, \ldots$ **do**

    Sort rows in increasing and decreasing order alternatively;

    Sort columns in increasing order.

$k = 1$

1	9	11	13	24	33	47	77	80	99
3	85	80	67	60	53	43	39	24	5
5	9	13	24	28	50	50	55	77	94
8	89	79	71	60	45	43	32	23	20
13	3	4	16	53	59	72	75	88	90
50	48	39	37	30	24	21	21	18	12
91	5	11	11	14	43	47	82	89	99
93	79	78	68	61	58	56	45	20	12
95	14	18	48	50	50	53	92	92	96
97	67	66	62	57	50	47	42	13	3

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*

  for $k = 1, \ldots$ **do**

    Sort rows in increasing and decreasing order alternatively;

    Sort columns in increasing order.

$k = 1$

1	3	4	11	14	24	21	21	13	3
3	5	11	13	24	33	43	32	18	5
5	9	11	16	28	43	43	39	20	12
8	9	13	24	30	45	47	42	23	12
13	14	18	37	50	50	47	45	24	20
50	48	39	48	53	50	47	55	77	90
91	67	66	62	57	50	50	75	80	94
93	79	78	67	60	53	53	77	88	96
95	85	79	68	60	58	56	82	89	99
97	89	80	71	61	59	72	92	92	99

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*

  for $k = 1, \ldots$ **do**

    Sort rows in increasing and decreasing order alternatively;

    Sort columns in increasing order.

$k = 2$

1	3	3	4	11	13	14	21	21	24
43	33	32	24	18	13	11	5	5	3
5	9	11	12	16	20	28	39	43	43
47	45	42	30	24	23	13	12	9	8
13	14	18	20	24	37	45	47	50	50
90	77	55	53	50	50	48	48	47	39
50	50	57	62	66	67	75	80	91	94
96	93	88	79	78	77	67	60	53	53
56	58	60	68	79	82	85	89	95	99
99	97	92	92	89	80	72	71	61	59

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*

  for $k = 1, \ldots$ **do**

    Sort rows in increasing and decreasing order alternatively;

    Sort columns in increasing order.

$k = 2$

1	3	3	4	11	13	11	5	5	3
5	9	11	12	16	13	13	12	9	8
13	14	18	20	18	20	14	21	21	24
43	33	32	24	24	23	28	39	43	39
47	45	42	30	24	37	45	47	47	43
50	50	55	53	50	50	48	48	50	50
56	58	57	62	66	67	67	60	53	53
90	77	60	68	78	77	72	71	61	59
96	93	88	79	79	80	75	80	91	94
99	97	92	92	89	82	85	89	95	99

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*

  for $k = 1, \ldots$ **do**

    Sort rows in increasing and decreasing order alternatively;

    Sort columns in increasing order.

$k = 3$

1	3	3	3	4	5	5	11	11	13
16	13	13	12	12	11	9	9	8	5
13	14	14	18	18	20	20	21	21	24
43	43	39	39	33	32	28	24	24	23
24	30	37	42	43	45	45	47	47	47
55	53	50	50	50	50	50	50	48	48
53	53	56	57	58	60	62	66	67	67
90	78	77	77	72	71	68	61	60	59
75	79	79	80	80	88	91	93	94	96
99	99	97	95	92	92	89	89	85	82

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*

  for $k = 1, \ldots$ **do**

    Sort rows in increasing and decreasing order alternatively;

    Sort columns in increasing order.

$k = 3$

1	3	3	3	4	5	5	9	8	5
13	13	13	12	12	11	9	11	11	13
16	14	14	18	18	20	20	21	21	23
24	30	37	39	33	32	28	24	24	24
43	43	39	42	43	45	45	47	47	47
53	53	50	50	50	50	50	50	48	48
55	53	56	57	58	60	62	61	60	59
75	78	77	77	72	71	68	66	67	67
90	79	79	80	80	88	89	89	85	82
99	99	97	95	92	92	91	93	94	96

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*

  for $k = 1, \ldots$ **do**

    Sort rows in increasing and decreasing order alternatively;

    Sort columns in increasing order.

$k = 4$

1	3	3	3	4	5	5	5	8	9
13	13	13	13	12	12	11	11	11	9
14	14	16	18	18	20	20	21	21	23
39	37	33	32	30	28	24	24	24	24
39	42	43	43	43	45	45	47	47	47
53	53	50	50	50	50	50	50	48	48
53	55	56	57	58	59	60	60	61	62
78	77	77	75	72	71	68	67	67	66
79	79	80	80	82	85	88	89	89	90
99	99	97	96	95	94	93	92	92	91

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*

  for $k = 1, \ldots$ **do**

    Sort rows in increasing and decreasing order alternatively;

    Sort columns in increasing order.

$k = 4$

1	3	3	3	4	5	5	5	8	9
13	13	13	13	12	12	11	11	11	9
14	14	16	18	18	20	20	21	21	23
39	37	33	32	30	28	24	24	24	24
39	42	43	43	43	45	45	47	47	47
53	53	50	50	50	50	50	50	48	48
53	55	56	57	58	59	60	60	61	62
78	77	77	75	72	71	68	67	67	66
79	79	80	80	82	85	88	89	89	90
99	99	97	96	95	94	93	92	92	91

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*

  for $k = 1, \ldots$ **do**

    Sort rows in increasing and decreasing order alternatively;

    Sort columns in increasing order.

1	3	3	3	4	5	5	5	8	9
13	13	13	13	12	12	11	11	11	9
14	14	16	18	18	20	20	21	21	23
39	37	33	32	30	28	24	24	24	24
39	42	43	43	43	45	45	47	47	47
53	53	50	50	50	50	50	50	48	48
53	55	56	57	58	59	60	60	61	62
78	77	77	75	72	71	68	67	67	66
79	79	80	80	82	85	88	89	89	90
94	99	97	96	95	94	93	92	92	91

Lemma

An array of size $\sqrt{n} \times \sqrt{n}$ is sorted after the iteration $k = \log_2(\sqrt{n}) + 1$.

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*
  for $k = 1, \dots$ **do**
    Sort rows in increasing and decreasing order alternatively;
    Sort columns in increasing order.

Lemma

An array of size $\sqrt{n} \times \sqrt{n}$ is sorted after the iteration $k = \log_2(\sqrt{n}) + 1$.

Sketch of proof.

0	0	0	0	1	1	1	1	1	1
1	1	0	0	0	0	0	0	0	0

□

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*
  for $k = 1, \dots$ **do**
    Sort rows in increasing and decreasing order alternatively;
    Sort columns in increasing order.

Lemma

An array of size $\sqrt{n} \times \sqrt{n}$ is sorted after the iteration $k = \log_2(\sqrt{n}) + 1$.

Sketch of proof.

0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	1	1	1

□

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*
  for $k = 1, \dots$ **do**
    Sort rows in increasing and decreasing order alternatively;
    Sort columns in increasing order.

Lemma

An array of size $\sqrt{n} \times \sqrt{n}$ is sorted after the iteration $k = \log_2(\sqrt{n}) + 1$.

Sketch of proof.

0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	1	1	1

0	0	0	0	1	1	1	1	1	1
1	1	1	1	1	1	1	0	0	0

□

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*

  for $k = 1, \dots$ **do**

    Sort rows in increasing and decreasing order alternatively;

    Sort columns in increasing order.

Lemma

An array of size $\sqrt{n} \times \sqrt{n}$ is sorted after the iteration $k = \log_2(\sqrt{n}) + 1$.

Sketch of proof.

0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	1	1	1

0	0	0	0	1	1	1	0	0	0
1	1	1	1	1	1	1	1	1	1

□

Sorting on a 2D Mesh-Connected MultiComputer

function SNAKE SORT(a) ▷ *a is an array of size $\sqrt{n} \times \sqrt{n}$*
  for $k = 1, \ldots$ **do**
    Sort rows in increasing and decreasing order alternatively;
    Sort columns in increasing order.

Lemma

An array of size $\sqrt{n} \times \sqrt{n}$ is sorted after the iteration $k = \log_2(\sqrt{n}) + 1$.

Sketch of proof.

0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	1	1	1

0	0	0	0	1	1	1	0	0	0
1	1	1	1	1	1	1	1	1	1

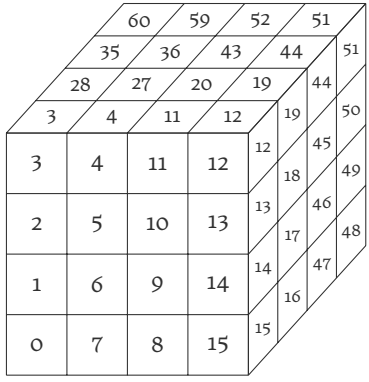
The number of "mixed" rows is halved after each iteration, hence $\log_2(\sqrt{n})$ iterations. □

Corollary

The SORTING problem on MCMCs can be solved in time $O(\sqrt{n} \cdot \log n)$ on arrays of size $\sqrt{n} \times \sqrt{n}$.

Sorting on a d-dimensional Mesh-Connected MultiComputer

Snake ordering generalizes to higher dimensions:

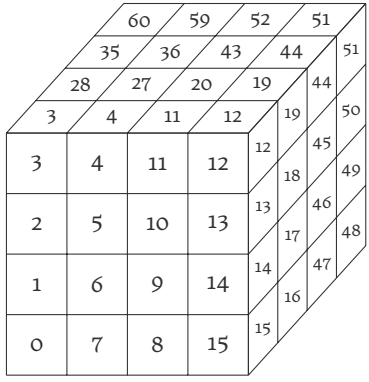


Theorem ([Corbett & Scherson, 1992])

The SORTING problem on MCMCs can be solved in time $O(\sqrt[d]{n} \cdot \log n)$ on arrays of size $\underbrace{\sqrt[d]{n} \times \cdots \times \sqrt[d]{n}}_{d \text{ times}}$.

Sorting on a d-dimensional Mesh-Connected MultiComputer

Snake ordering generalizes to higher dimensions:



Theorem (~~[Corbett & Scherson, 1992]~~ [Nassimi & Sahni, 1979])

The SORTING problem on MCMCs can be solved in time $O(\sqrt[d]{n} \cdot \sout{\log n})$ on arrays of size $\underbrace{\sqrt[d]{n} \times \cdots \times \sqrt[d]{n}}_{d \text{ times}}$.

Fast computing with tilings

The RAM model

Definition

A *Random Access Machine* is composed of:

- A finite sequence of *instructions* called a *program* (and an *instruction pointer* addressing the instruction currently being executed);
- A finite set of variables $\texttt{var}_1, \dots, \texttt{var}_k$, each containing an integer;
- A *memory array* M composed of infinitely many *memory cells* $(M_i)_{i \in \mathbb{N}}$, each containing an integer.

Instructions are of two types:

- Arithmetic instructions on variables, *e.g.* $\texttt{var}_i \leftarrow \texttt{var}_j + \texttt{var}_k$;
- Memory instructions, either reading $\texttt{var}_i \leftarrow M_j$ or writing $M_i \leftarrow \texttt{var}_j$.

The RAM model

Definition

A *Random Access Machine* is composed of:

- A finite sequence of *instructions* called a *program* (and an *instruction pointer* addressing the instruction currently being executed);
- A finite set of variables $\texttt{var}_1, \dots, \texttt{var}_k$, each containing an integer;
- A *memory array* M composed of infinitely many *memory cells* $(M_i)_{i \in \mathbb{N}}$, each containing an integer.

Instructions are of two types:

- Arithmetic instructions on variables, *e.g.* $\texttt{var}_i \leftarrow \texttt{var}_j + \texttt{var}_k$;
- Memory instructions, either reading $\texttt{var}_i \leftarrow M_j$ or writing $M_i \leftarrow \texttt{var}_j$.

Lemma

The RAM model is Turing-complete.

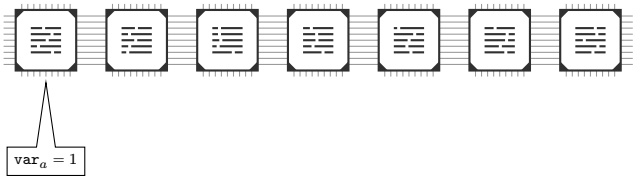
Simulating RAM computations with Mesh-Connected MultiComputers

Fix $n \in \mathbb{N}$. We simulate n computations steps of a RAM program p in an MCMC of size n:

Simulating RAM computations with Mesh-Connected MultiComputers

Fix $n \in \mathbb{N}$. We simulate n computations steps of a RAM program p in an MCMC of size n:

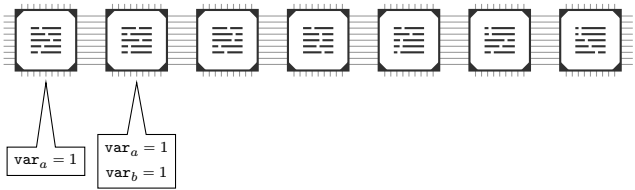
function FIBONACCI
- $\texttt{var}_a \leftarrow 1$;
- $\texttt{var}_b \leftarrow 1$;
- **for** $i = 1, \dots, 10$ **do**
  - $\texttt{var}_c \leftarrow \texttt{var}_a + \texttt{var}_b$;
  - $\texttt{var}_a \leftarrow \texttt{var}_b$;
  - $\texttt{var}_b \leftarrow \texttt{var}_c$;
  - $M_i \leftarrow \texttt{var}_b$;
- $\texttt{var}_c \leftarrow M_{10}$;
- $\texttt{var}_c \leftarrow \texttt{var}_c + 4$;



Simulating RAM computations with Mesh-Connected MultiComputers

Fix $n \in \mathbb{N}$. We simulate n computations steps of a RAM program p in an MCMC of size n:

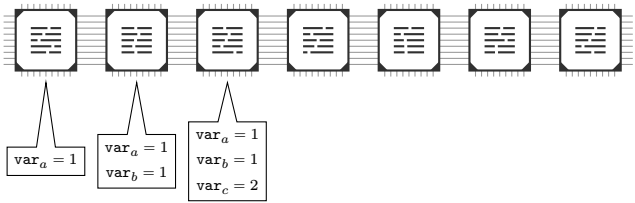
```
function FIBONACCI
    var_a ← 1;
    var_b ← 1;
    for i = 1, ..., 10 do
        var_c ← var_a + var_b;
        var_a ← var_b;
        var_b ← var_c;
        M_i ← var_b;
    var_c ← M_10;
    var_c ← var_c + 4;
```



Simulating RAM computations with Mesh-Connected MultiComputers

Fix $n \in \mathbb{N}$. We simulate n computations steps of a RAM program p in an MCMC of size n:

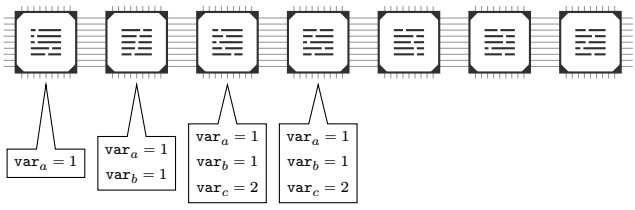
function FIBONACCI
- $\texttt{var}_a \leftarrow 1$;
- $\texttt{var}_b \leftarrow 1$;
- **for** $i = 1, \dots, 10$ **do**
  - $\texttt{var}_c \leftarrow \texttt{var}_a + \texttt{var}_b$;
  - $\texttt{var}_a \leftarrow \texttt{var}_b$;
  - $\texttt{var}_b \leftarrow \texttt{var}_c$;
  - $M_i \leftarrow \texttt{var}_b$;
- $\texttt{var}_c \leftarrow M_{10}$;
- $\texttt{var}_c \leftarrow \texttt{var}_c + 4$;



Simulating RAM computations with Mesh-Connected MultiComputers

Fix $n \in \mathbb{N}$. We simulate n computations steps of a RAM program p in an MCMC of size n:

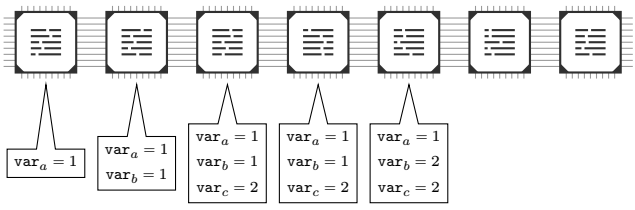
```
function FIBONACCI
    var_a ← 1;
    var_b ← 1;
    for i = 1, ..., 10 do
        var_c ← var_a + var_b;
        var_a ← var_b;
        var_b ← var_c;
        M_i ← var_b;
    var_c ← M_10;
    var_c ← var_c + 4;
```



Simulating RAM computations with Mesh-Connected MultiComputers

Fix $n \in \mathbb{N}$. We simulate n computations steps of a RAM program p in an MCMC of size n:

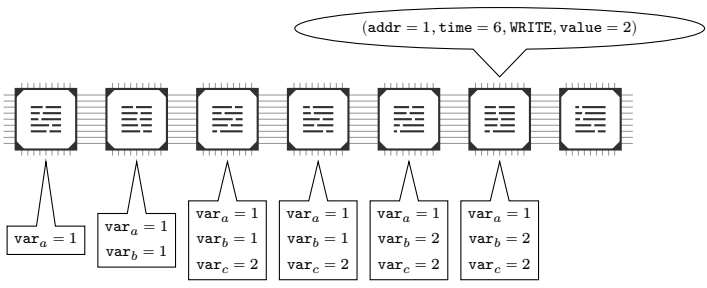
function FIBONACCI
- $\texttt{var}_a \leftarrow 1$;
- $\texttt{var}_b \leftarrow 1$;
- **for** $i = 1, \ldots, 10$ **do**
  - $\texttt{var}_c \leftarrow \texttt{var}_a + \texttt{var}_b$;
  - $\texttt{var}_a \leftarrow \texttt{var}_b$;
  - $\texttt{var}_b \leftarrow \texttt{var}_c$;
  - $M_i \leftarrow \texttt{var}_b$;
- $\texttt{var}_c \leftarrow M_{10}$;
- $\texttt{var}_c \leftarrow \texttt{var}_c + 4$;



Simulating RAM computations with Mesh-Connected MultiComputers

Fix $n \in \mathbb{N}$. We simulate n computations steps of a RAM program p in an MCMC of size n:

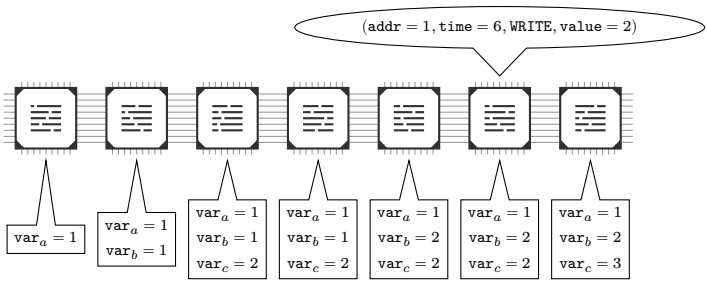
function FIBONACCI
- $\texttt{var}_a \leftarrow 1$;
- $\texttt{var}_b \leftarrow 1$;
- **for** $i = 1, \dots, 10$ **do**
  - $\texttt{var}_c \leftarrow \texttt{var}_a + \texttt{var}_b$;
  - $\texttt{var}_a \leftarrow \texttt{var}_b$;
  - $\texttt{var}_b \leftarrow \texttt{var}_c$;
  - $M_i \leftarrow \texttt{var}_b$;
- $\texttt{var}_c \leftarrow M_{10}$;
- $\texttt{var}_c \leftarrow \texttt{var}_c + 4$;



Simulating RAM computations with Mesh-Connected MultiComputers

Fix $n \in \mathbb{N}$. We simulate n computations steps of a RAM program p in an MCMC of size n:

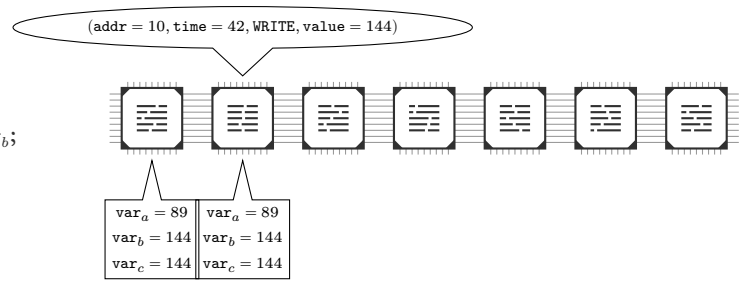
function FIBONACCI
- $\texttt{var}_a \leftarrow 1$;
- $\texttt{var}_b \leftarrow 1$;
- **for** $i = 1, \dots, 10$ **do**
  - $\texttt{var}_c \leftarrow \texttt{var}_a + \texttt{var}_b$;
  - $\texttt{var}_a \leftarrow \texttt{var}_b$;
  - $\texttt{var}_b \leftarrow \texttt{var}_c$;
  - $M_i \leftarrow \texttt{var}_b$;
- $\texttt{var}_c \leftarrow M_{10}$;
- $\texttt{var}_c \leftarrow \texttt{var}_c + 4$;



Simulating RAM computations with Mesh-Connected MultiComputers

Fix $n \in \mathbb{N}$. We simulate n computations steps of a RAM program p in an MCMC of size n:

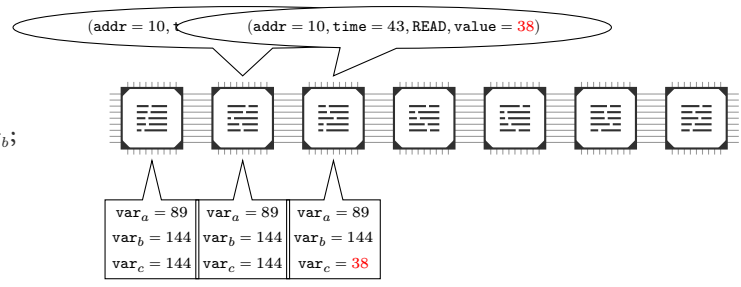
function FIBONACCI
- $\texttt{var}_a \leftarrow 1$;
- $\texttt{var}_b \leftarrow 1$;
- **for** $i = 1, \dots, 10$ **do**
  - $\texttt{var}_c \leftarrow \texttt{var}_a + \texttt{var}_b$;
  - $\texttt{var}_a \leftarrow \texttt{var}_b$;
  - $\texttt{var}_b \leftarrow \texttt{var}_c$;
  - $M_i \leftarrow \texttt{var}_b$;
- $\texttt{var}_c \leftarrow M_{10}$;
- $\texttt{var}_c \leftarrow \texttt{var}_c + 4$;



Simulating RAM computations with Mesh-Connected MultiComputers

Fix $n \in \mathbb{N}$. We simulate n computations steps of a RAM program p in an MCMC of size n:

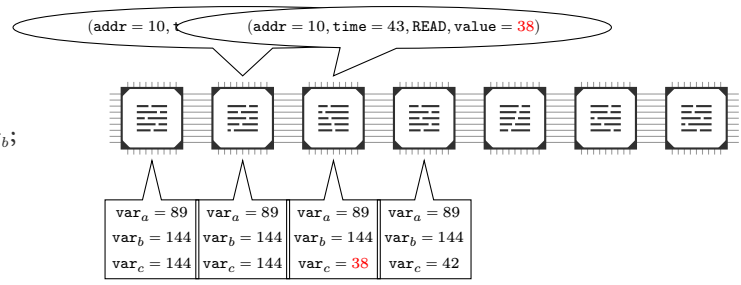
function FIBONACCI
- $\texttt{var}_a \leftarrow 1$;
- $\texttt{var}_b \leftarrow 1$;
- **for** $i = 1, \dots, 10$ **do**
  - $\texttt{var}_c \leftarrow \texttt{var}_a + \texttt{var}_b$;
  - $\texttt{var}_a \leftarrow \texttt{var}_b$;
  - $\texttt{var}_b \leftarrow \texttt{var}_c$;
  - $M_i \leftarrow \texttt{var}_b$;
- $\texttt{var}_c \leftarrow M_{10}$;
- $\texttt{var}_c \leftarrow \texttt{var}_c + 4$;



Simulating RAM computations with Mesh-Connected MultiComputers

Fix $n \in \mathbb{N}$. We simulate n computations steps of a RAM program p in an MCMC of size n:

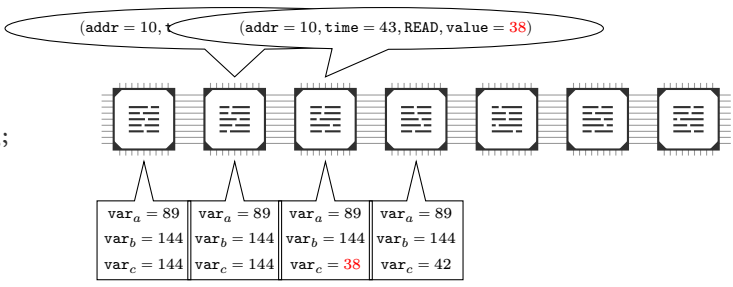
function FIBONACCI
- $\texttt{var}_a \leftarrow 1$;
- $\texttt{var}_b \leftarrow 1$;
- **for** $i = 1, \dots, 10$ **do**
  - $\texttt{var}_c \leftarrow \texttt{var}_a + \texttt{var}_b$;
  - $\texttt{var}_a \leftarrow \texttt{var}_b$;
  - $\texttt{var}_b \leftarrow \texttt{var}_c$;
  - $M_i \leftarrow \texttt{var}_b$;
- $\texttt{var}_c \leftarrow M_{10}$;
- $\texttt{var}_c \leftarrow \texttt{var}_c + 4$;



Simulating RAM computations with Mesh-Connected MultiComputers

Fix $n \in \mathbb{N}$. We simulate n computations steps of a RAM program p in an MCMC of size n:

function FIBONACCI
- $\texttt{var}_a \leftarrow 1$;
- $\texttt{var}_b \leftarrow 1$;
- **for** $i = 1, \dots, 10$ **do**
  - $\texttt{var}_c \leftarrow \texttt{var}_a + \texttt{var}_b$;
  - $\texttt{var}_a \leftarrow \texttt{var}_b$;
  - $\texttt{var}_b \leftarrow \texttt{var}_c$;
  - $M_i \leftarrow \texttt{var}_b$;
- $\texttt{var}_c \leftarrow M_{10}$;
- $\texttt{var}_c \leftarrow \texttt{var}_c + 4$;



At the end of the procedure, we have a list of records of the form:

- Writing records: $(\texttt{addr}, \texttt{time}, \texttt{WRITE}, \texttt{value})$;
- Reading records: $(\texttt{addr}, \texttt{time}, \texttt{READ}, \texttt{value})$;

By sorting this list in lexicographic order, we can check in time $O(1)$ the consistency of the memory guesses!

Simulating RAM computations with Mesh-Connected MultiComputers

Fix $n \in \mathbb{N}$. We simulate n computations steps of a RAM program p in an MCMC of size n:

procedure

- **for** $i = 1, \dots, n$ **do** ▷ *Parallel time* $O(1)$.
  - The i^{th} processor performs the i^{th} step of the program p.
  - If this computation step involves a *reading a memory cell* (resp. writing), the processor stores a *memory record* $(\texttt{addr}, \texttt{time}, \texttt{READ}, \texttt{value})$ (resp. $(\texttt{addr}, \texttt{time}, \texttt{WRITE}, \texttt{value})$).
- Sort the memory records lexicographically. ▷ *Parallel time* $O(\sqrt[4]{n})$.
- Check the consistency of the memory guesses. ▷ *Parallel time* $O(1)$.

Simulating RAM computations with Mesh-Connected MultiComputers

Fix $n \in \mathbb{N}$. We simulate n computations steps of a RAM program p in an MCMC of size n:

procedure
- **for** $i = 1, \dots, n$ **do** ▷ *Parallel time* $O(1)$.
  - The i^{th} processor performs the i^{th} step of the program p.
  - If this computation step involves a *reading a memory cell* (resp. writing), the processor stores a *memory record* (`addr`, `time`, `READ`, `value`) (resp. (`addr`, `time`, `WRITE`, `value`)).
- Sort the memory records lexicographically. ▷ *Parallel time* $O(\sqrt[4]{n})$.
- Check the consistency of the memory guesses. ▷ *Parallel time* $O(1)$.

Theorem

A cubic d-dimensional Mesh-Connected MultiComputer of edge length n can simulate n^d steps of RAM computations in time $O(n)$.

Conclusion

Take-home message

A 3-dimensional cube of size $n \times n \times n$ can embed $O(n^2)$ steps of arbitrary RAM computations.

Actually, technical details:

- Integers appearing in the RAM computations should be "small enough" to fit in the memory of a single MCMC processor (*e.g.* $O(\log n)$ bits): $\implies$ **word-RAM model**;
- When implementing with Wang tiles, the colors should at least contain the state of an MCMC processor (*e.g.* $O(\log n)$ bits): **non-constant!**

$\implies$ Mesh-Connected MultiComputers are a natural computation model for the "fixpoint construction" (Durand, Romaschenko and Shen).

Save the date!

What can we do with this? $\implies$ Prove the soficity of many multidimensional subshifts!

(June, 24th in Caen)


That's all Folks!