Design of Parallel LDPC Interleaver Architecture: A Bipartite Edge Coloring Approach

Awais. H. SANI, P. Coussy, C Chavet, E. Martin
University Bretagne-Sud / Lab-STICC
awais-hussain.sani@univ-ubs.fr

December 14, 2010

Outline

- **Problem Formulation**
- **Modeling**
- **Algorithm**
- **Conclusion**

Outline

- **Problem Formulation**
- **Modeling**
- **Algorithm**
- **Conclusion**

Problem Formulation

- Set of K data elements $\{d_0,d_1,\ldots\ldots,d_{K-1}\}$
- Set of P processing elements $\{PE_0,PE_1,\ldots\ldots PE_{P-1}\}$
- Set of $B = P$ memory banks $\{b_0,b_1,\ldots\ldots,b_{P-1}\}$
- Size of each memory bank $M = K/P$
- Set of N time instances $\{t_1,t_2,\ldots\ldots t_N\}$ in which P processing elements process K data elements.

Problem Formulation

- Set of K data elements $\{d_0,d_1,\ldots\ldots,d_{K-1}\}$
- Set of P processing elements $\{PE_0,PE_1,\ldots\ldots PE_{P-1}\}$
- Set of $B = P$ memory banks $\{b_0,b_1,\ldots\ldots,b_{P-1}\}$
- Size of each memory bank $M = K/P$
- Set of N time instances $\{t_1,t_2,\ldots\ldots t_N\}$ in which P processing elements process K data elements.

Mapping problem: *Store K data elements in B memory banks in such a manner that P processing element can access B memory banks in parallel at each time instance for first reading and then writing B data elements without any conflict.*

Example

- Set of K data elements $\{d_0,d_1,\ldots\ldots,d_{K-1}\}$
- Set of P processing elements $\{PE_0,PE_1,\ldots\ldots PE_{P-1}\}$
- Set of $B = P$ memory banks $\{b_0,b_1,\ldots\ldots,b_{P-1}\}$
- Size of each memory bank $M = K/P$
- Set of N time instances $\{t_1,t_2,\ldots\ldots t_N\}$ in which P processing elements process K data elements.

K = 6,
P = B = 3
M = 2
N = 6

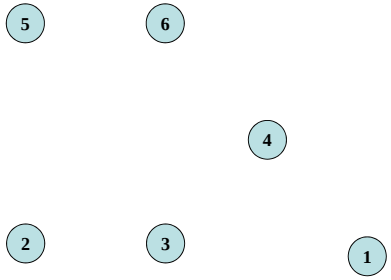
Parallelism

	t_1	t_2	t_3	t_4	t_5	t_6
PE_1	1	3	6	5	4	2
PE_2	2	5	1	6	3	1
PE_3	3	6	4	2	5	4

Time

Data Access Matrix

Traditional Approach

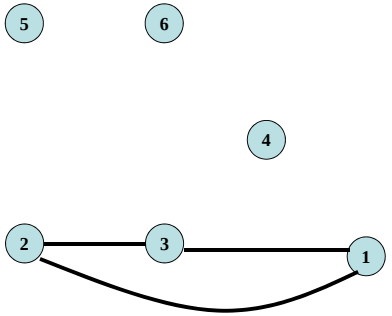


K = 6,
P = B = 3,
M = 2
N = 6

Conflict Access Graph

	t_1	t_2	t_3	t_4	t_5	t_6
PE_1	1	3	6	5	4	2
PE_2	2	5	1	6	3	1
PE_3	3	6	4	2	5	4

Traditional Approach

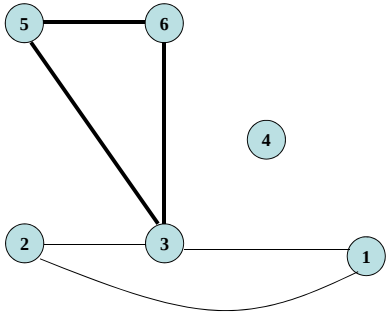


K = 6,
P = B = 3,
M = 2
N = 6

Conflict Access Graph

	t_1	t_2	t_3	t_4	t_5	t_6
PE_1	1	3	6	5	4	2
PE_2	2	5	1	6	3	1
PE_3	3	6	4	2	5	4

Traditional Approach

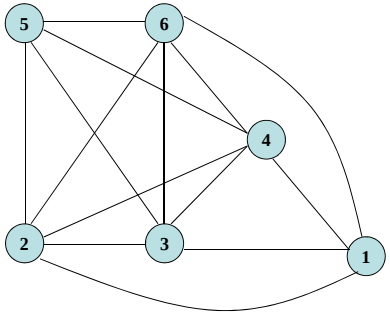


Conflict Access Graph

K = 6,
P = B = 3,
M = 2
N = 6

	t_1	t_2	t_3	t_4	t_5	t_6
PE_1	1	3	6	5	4	2
PE_2	2	5	1	6	3	1
PE_3	3	6	4	2	5	4

Traditional Approach

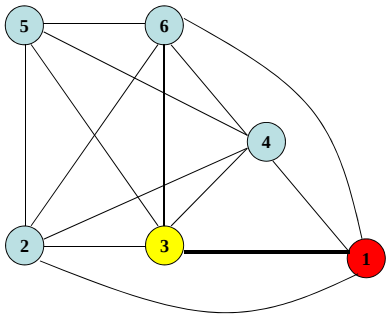


Conflict Access Graph

K = 6,
P = B = 3,
M = 2
N = 6

	t_1	t_2	t_3	t_4	t_5	t_6
PE_1	1	3	6	5	4	2
PE_2	2	5	1	6	3	1
PE_3	3	6	4	2	5	4

Traditional Approach



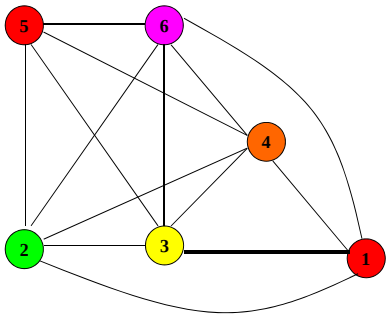
Conflict Access Graph

K = 6,
P = B = 3,
M = 2
N = 6

	t_1	t_2	t_3	t_4	t_5	t_6
PE_1	1	3	6	5	4	2
PE_2	2	5	1	6	3	1
PE_3	3	6	4	2	5	4

Nodes connected with the same edge should be of different color and each color represents memory bank.

Traditional Approach



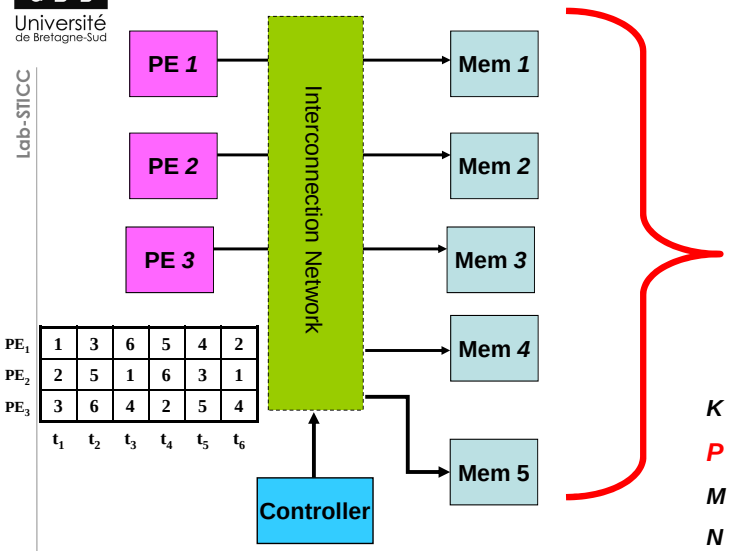
Conflict Access Graph

K = 6,
P = B = 3,
M = 2
N = 6

	t_1	t_2	t_3	t_4	t_5	t_6
PE_1	1	3	6	5	4	2
PE_2	2	5	1	6	3	1
PE_3	3	6	4	2	5	4

We need 5 memory banks to access 6 data elements.

Resultant Architecture Using Traditional Approach



	t_1	t_2	t_3	t_4	t_5	t_6
PE_1	1	3	6	5	4	2
PE_2	2	5	1	6	3	1
PE_3	3	6	4	2	5	4

K = 6,

P = 3 & B = 5

M = 2

N = 6

New Approach

To tackle this problem, we introduce the concept of multiple read and multiple write access Concept.

New Approach

	t_1	t_2	t_3	t_4	t_5	t_6
PE_1	1	3	6	5	4	2
PE_2	2	5	1	6	3	1
PE_3	3	6	4	2	5	4

Data Access Matrix

Replace

	R	W		R	W		R	W		R	W		R	W		R	W
1			3			6			5			4			2		
2			5			1			6			3			1		
3			6			4			2			5			4		
	t_1			t_2			t_3			t_4			t_5			t_6	

Mapping Matrix

Constraints

Mapping Constraints:

• At each time instance, all the memory banks in the read column (resp. in the write column) of the mapping matrix must be different.

	R	W		R	W		R	W		R	W		R	W		R	W
1			3			6			5			4			2		
2			5			1			6			3			1		
3			6			4			2			5			4		
	t_1			t_2			t_3			t_4			t_5			t_6	

Mapping Matrix

Constraints

Mapping Constraints:

- At each time instance, all the memory banks in the read column (resp. in the write column) of the mapping matrix must be different.
- The bank of the last write access to a data must be the same as the bank of its first read access.

	R	W		R	W		R	W		R	W		R	W		R	W
1			3			6			5			4			2		
2			5			1			6			3			1		
3			6			4			2			5			4		
	t_1			t_2			t_3			t_4			t_5			t_6	

Mapping Matrix

Constraints

Mapping Constraints:

- At each time instance, all the memory banks in the read column (resp. in the write column) of the mapping matrix must be different.
- The bank of the last write access to a data must be the same as the bank of its first read access.

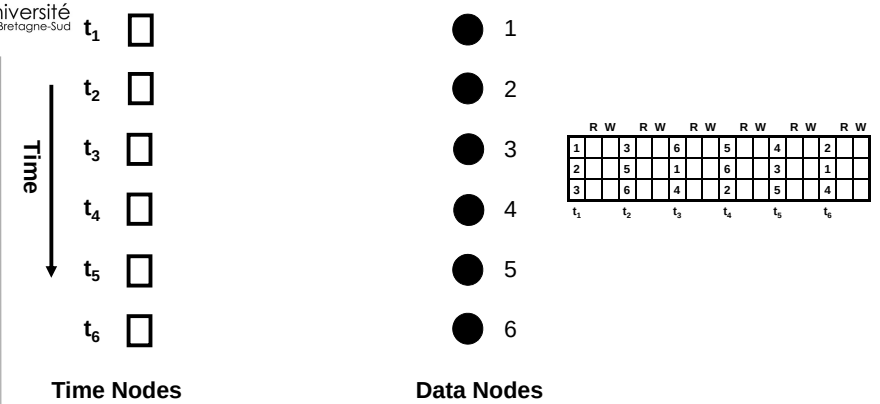
	R	W		R	W		R	W		R	W		R	W		R	W
1			3			6			5			4			2		
2			5			1			6			3			1		
3			6			4			2			5			4		
	t_1			t_2			t_3			t_4			t_5			t_6	

Mapping Matrix

Outline

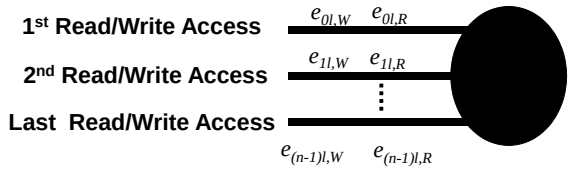
- **Problem Formulation**
- **Modeling**
  - **Preparing Bipartite Graph**
- **Algorithm**
- **Conclusion**

Preparing Bipartite Graph



Time Nodes **Data Nodes**

Preparing Bipartite Graph



Data Node Representation

Where

$e_{0l,R}$ and $e_{(n-1)l,R}$ represent first and last read access of the data l respectively.

$e_{0l,W}$ and $e_{(n-1)l,W}$ represent first and last write access of the data l respectively.

Preparing Bipartite Graph

Placement Property:

i_{th} *write access* = $modulo_{degree\ l}(i + 1)$ *read access*

This placement property is used to search edges during algorithm.

The read access of the i_{th} write access is called *direct edge* whereas the corresponding write access is called *induced edge* next in this presentation.

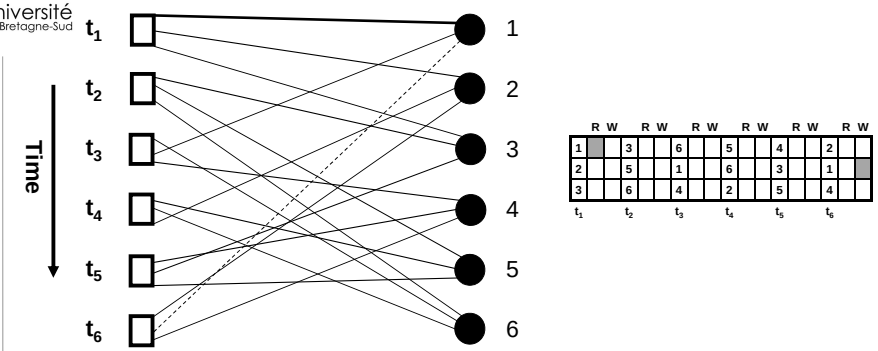
for *(degree l) = 3*

i_{th} *write access* = $modulo_3(i + 1)$*th read access*

$e_{0l,W} = e_{1l,R}$
$e_{1l,W} = e_{2l,R}$
$e_{2l,W} = e_{0l,R}$

Induced edges **Direct edges**

Preparing Bipartite Graph

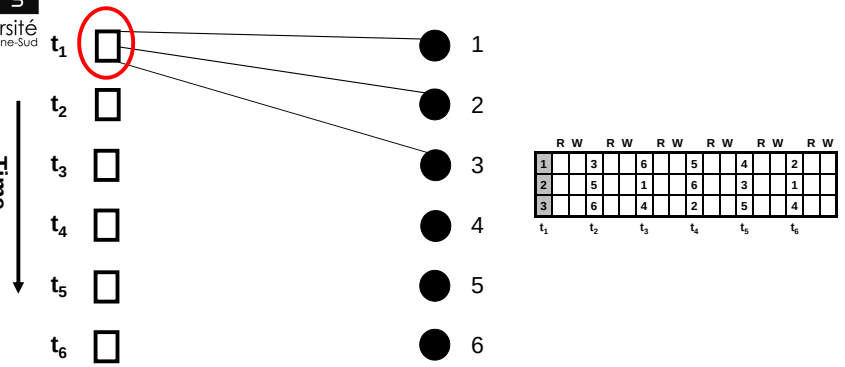


Time Nodes **Data Nodes**

$e_{0l,W} = e_{1l,R}$
$e_{1l,W} = e_{2l,R}$
$e_{2l,W} = e_{0l,R}$

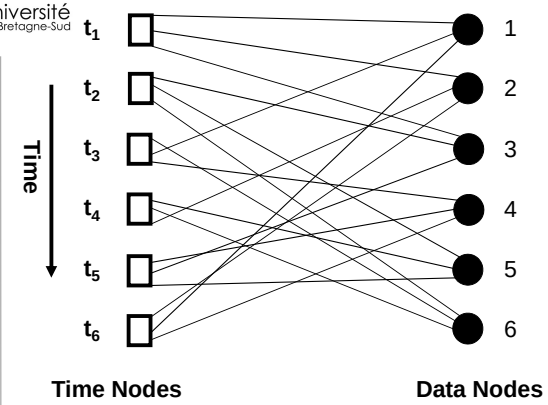
Dotted edges **Bold edges**

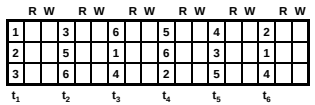
Preparing Bipartite Graph



Time Nodes **Data Nodes**

Preparing Bipartite Graph





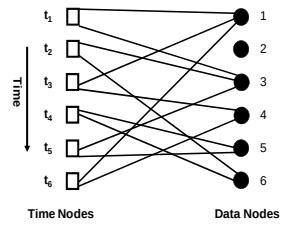
Outline

- **Problem Formulation**
- **Modeling**
  - **Preparing Bipartite Graph**
- **Algorithm**
  - **Partitioning**
  - **Coloring**
- **Conclusion**

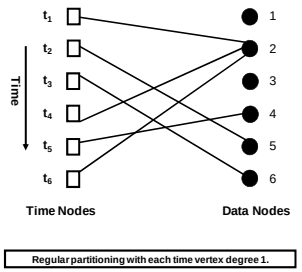
Partitioning Algorithm

Proper Partition:

Proper partition is the subgraph in which all the time vertices have degree 2.



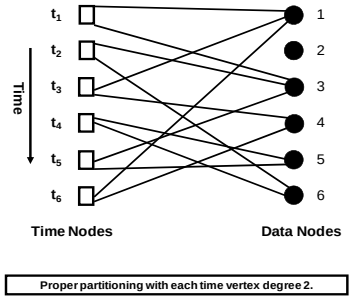
Proper partition is the subgraph in which all the time vertices have degree 1.

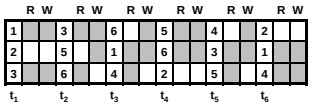


Partitioning Algorithm

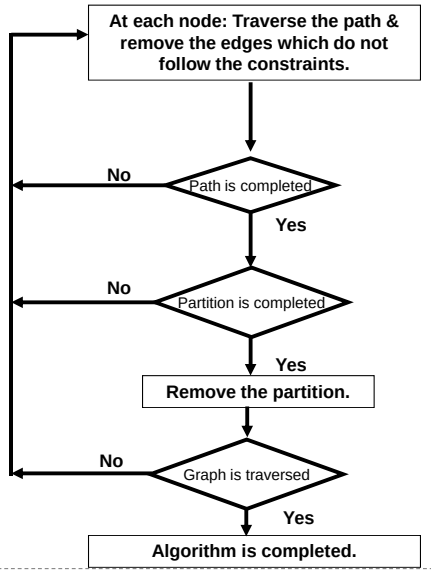
Partitioning Constraint:

No more than 2 read or write accesses have to be done at each time instance in a proper partition.





Partitioning Algorithm



Partitioning Algorithm.

Partitioning Algorithm

Partitioning Algorithm:

- Process of Traversal
- Process of Elimination

Partitioning Algorithm

Partitioning Algorithm:

- Process of Traversal
- Process of Elimination

Process of Traversal

1. Randomly selects the edge available at the current data and time vertex.
2. Records the induced edge of the selected edge.

Partitioning Algorithm

Partitioning Algorithm:

- Process of Traversal
- Process of Elimination

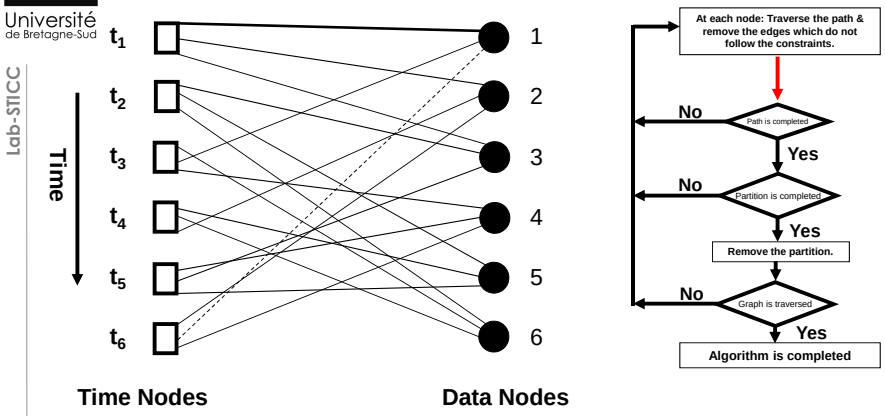
Process of Traversal

1. Randomly selects the edge available at the current data and time vertex.
2. Records the induced edge of the selected edge.

Process of Elimination

1. Removes the edges from current partition, the selection of which makes the construction of *proper partition* following "*partitioning constraint*" impossible.

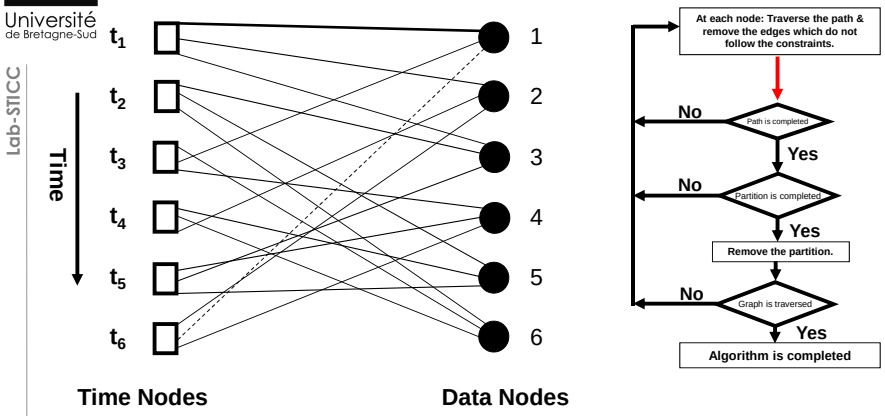
Partitioning Algorithm



Algorithm invokes the process of traversal to add the edge (*1*, t_1) into the path p_1 as shown with bold line. Induced edge (*1*, t_6) of the (*1*, t_1) is also recorded with dotted line.

$p_1 = \{(1,t_1)\}$

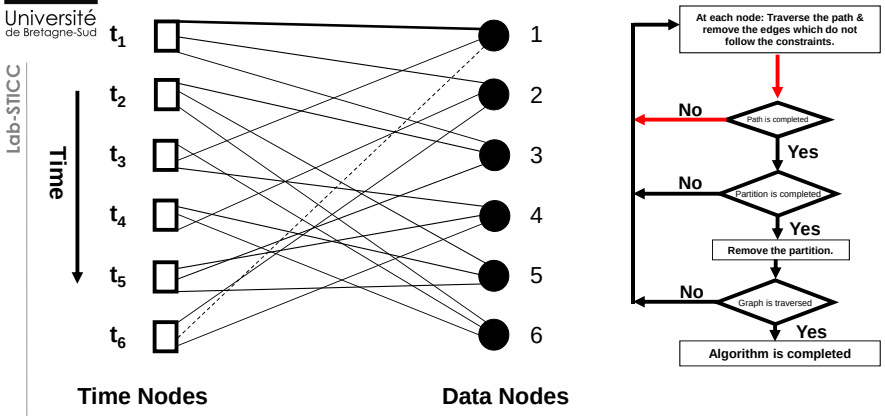
Partitioning Algorithm



Process of elimination is invoked but do not find any edge to be deleted.

$p_1 = \{(1,t_1)\}$

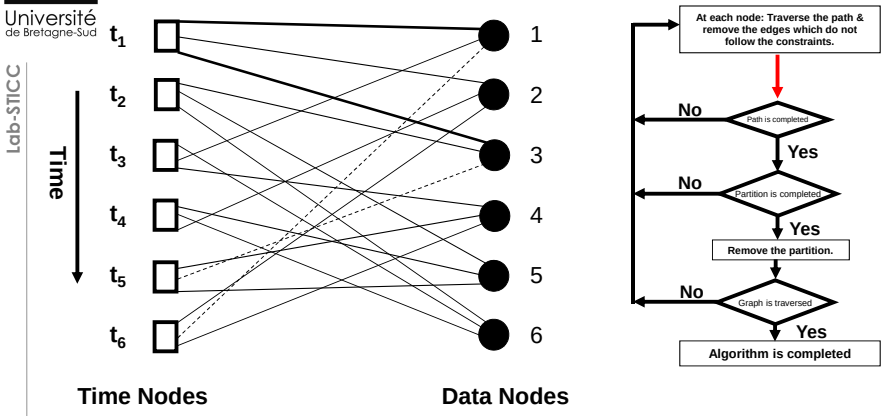
Partitioning Algorithm



Path is not completed. Start adding other edges to the path.

$p_1 = \{(1,t_1)\}$

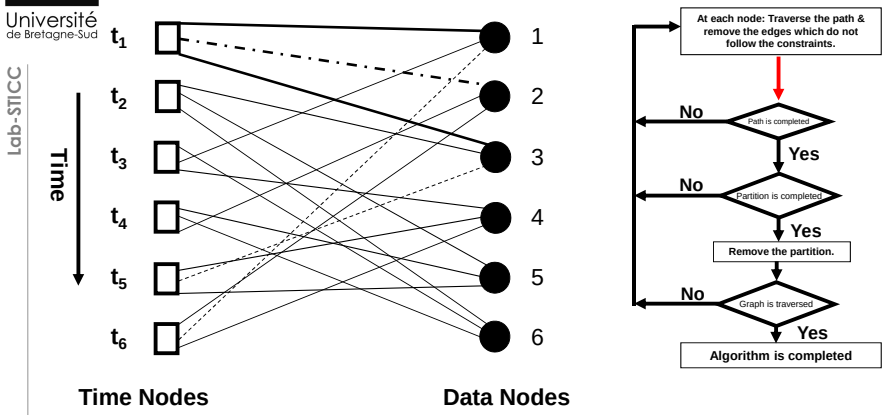
Partitioning Algorithm



Algorithm invokes the process of traversal to add the edge (*t_1*,*3*) into the path p_1 as shown with bold line. Induced edge (*1*, t_6) of the (*3*, t_5) is also recorded with dotted line.

$p_1 = \{(1,t_1), (t_1,3)\}$

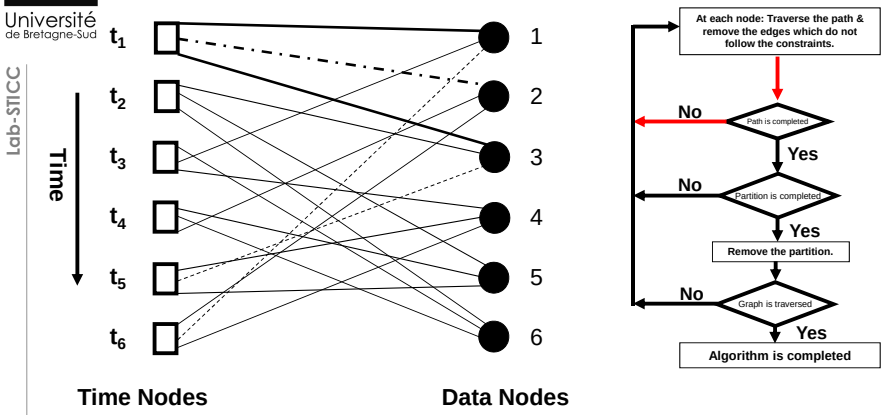
Partitioning Algorithm



Process of elimination is invoked and found that two read accesses are done at t_1, other read accesses at t_1 is deleted as shown with large and small dotted line.

$p_1 = \{(1,t_1), (t_1, 3)\}$

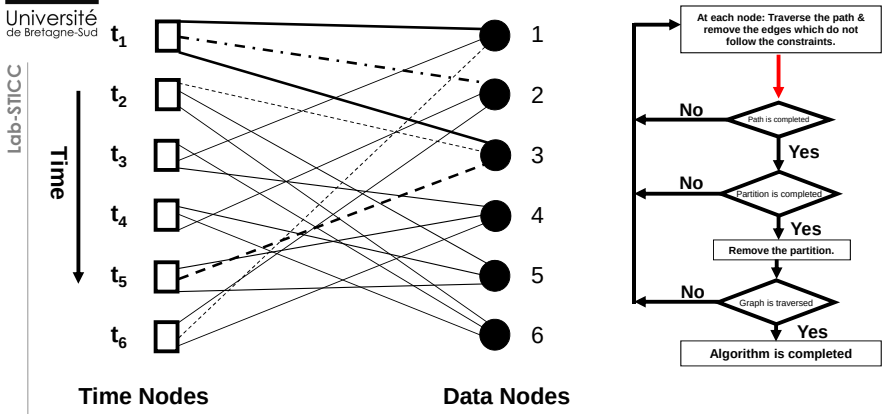
Partitioning Algorithm



Path is not completed. Start adding other edges to the path.

$p_1 = \{(1,t_1), (t_1, 3)\}$

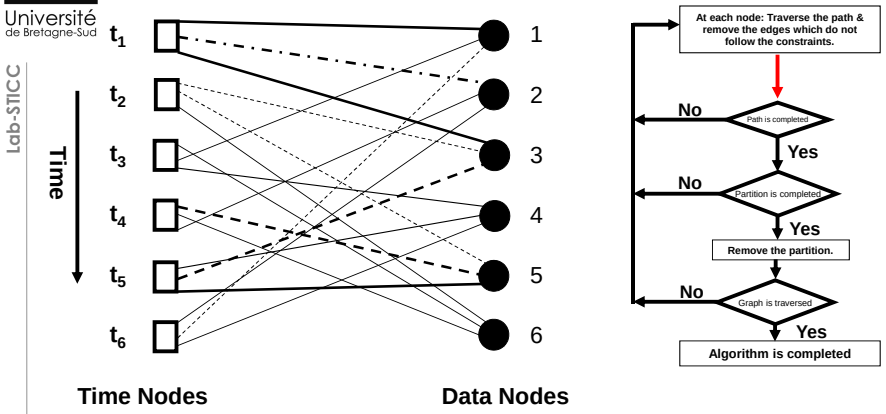
Partitioning Algorithm



Algorithm invokes the process of traversal to add the edge $(3, t_5)$ into the path p_1. Since $(3, t_5)$ is already a induced edge so this edge is now bold and dotted.

$p_1 = \{(1,t_1), (t_1, 3), (3, t_5)\}$

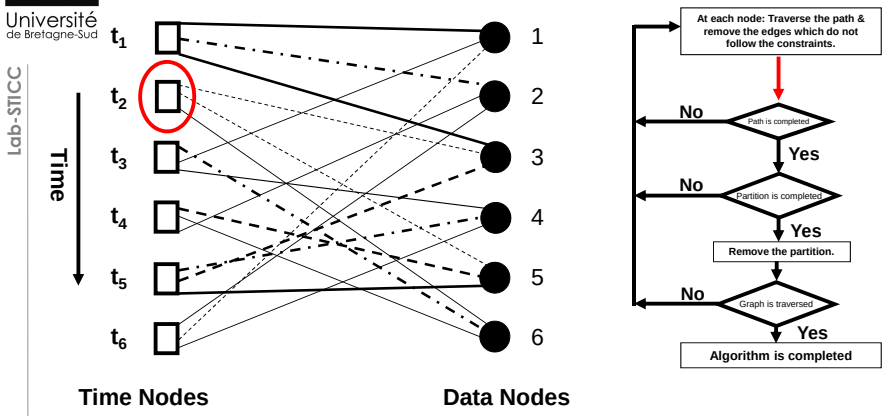
Partitioning Algorithm



Traversal continues until, p_1 reaches at t_4.

$p_1 = \{(1, t_1), (t_1, 3), (3, t_5), (t_5, 5), (5, t_4)\}$

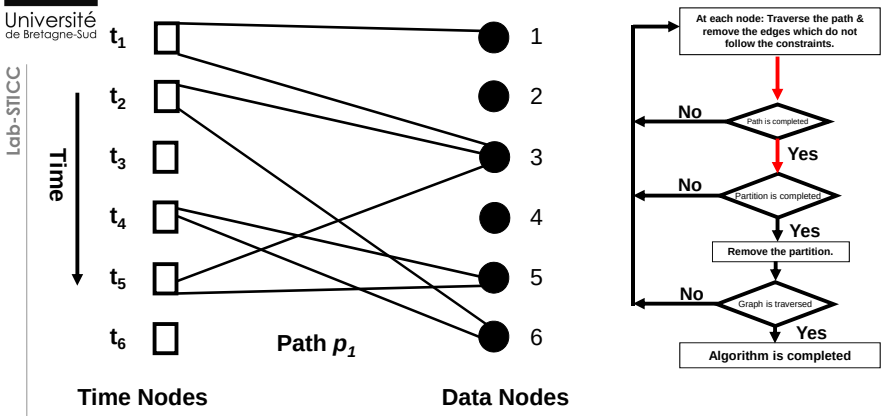
Partitioning Algorithm



Process of elimination finds that 2 write accesses is done at t_2 as shown through induced edges. So other edges, (t_3 , 6) in this case, which have write accesses at t_2 is deleted.

$p_1 = \{(1, t_1), (t_1, 3), (3, t_5), (t_5, 5), (5, t_4)\}$

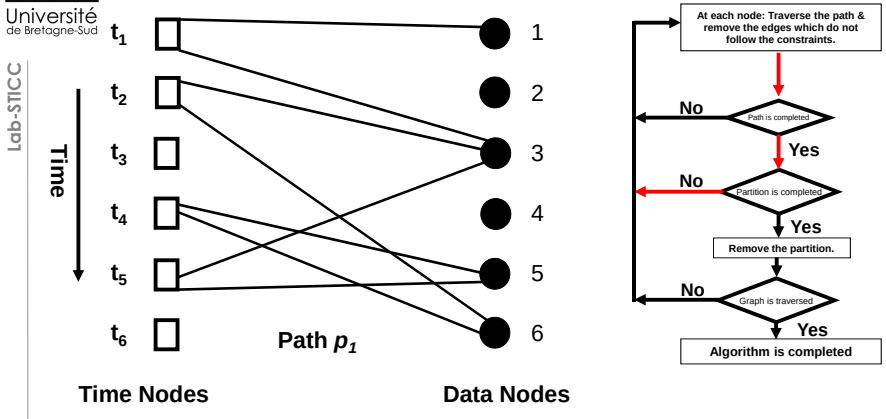
Partitioning Algorithm



Traversal continues until path p_1 is completed as shown with bold lines.

$p_1 = \{(1, t_1), (t_1, 3), (3, t_5), (t_5, 5), (5, t_4), (t_4, 6), (6, t_2), (t_2, 3)\}$

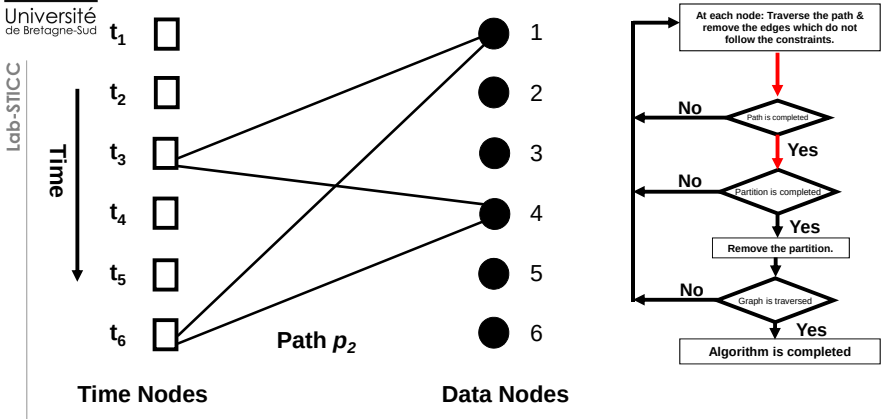
Partitioning Algorithm



But partition sg_1 is not completed.

$p_1 = \{(1, t_1), (t_1, 3), (3, t_5), (t_5, 5), (5, t_4), (t_4, 6), (6, t_2), (t_2, 3)\}$

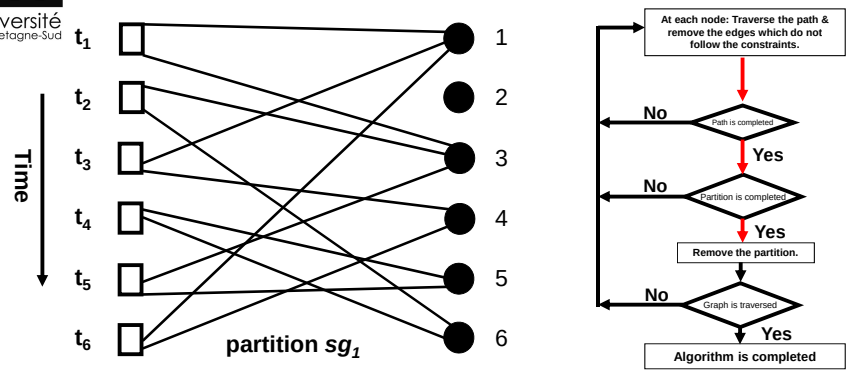
Partitioning Algorithm



Algorithm traverses another path p_2 *to complete* sg_1.

$p_2 = \{(1, t_3), (t_3, 4), (4, t_6), (t_6, 1)\}$

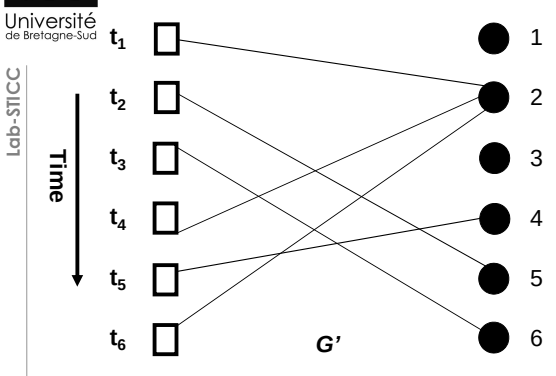
Partitioning Algorithm

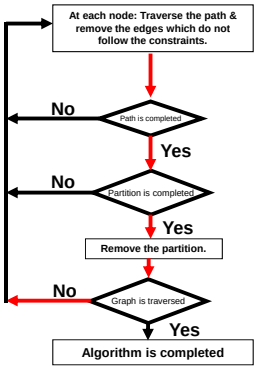


partition sg_1 is completed.

$sg_1 = p_1 + p_2$

Partitioning Algorithm

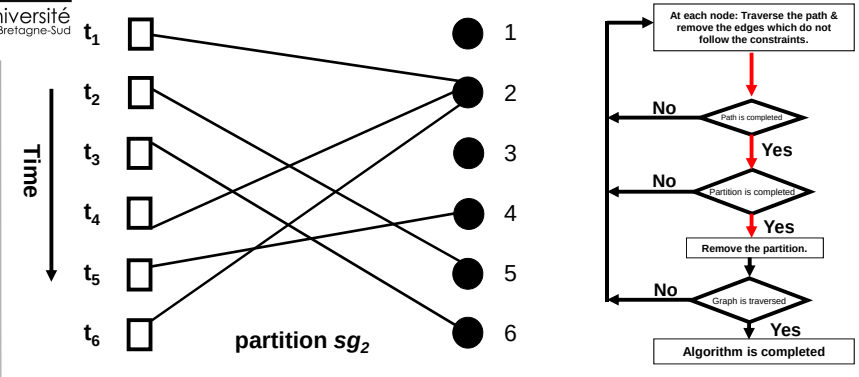




Remove sg_1 from graph *G*.

$G' = G - sg_1$

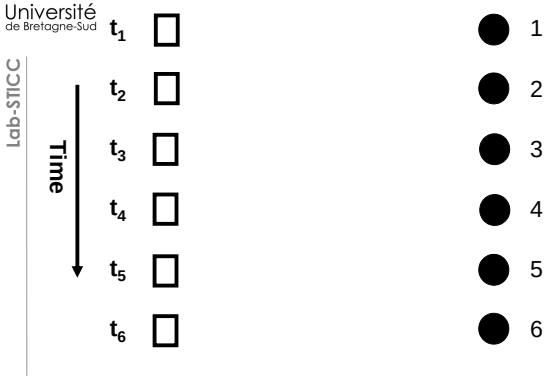
Partitioning Algorithm

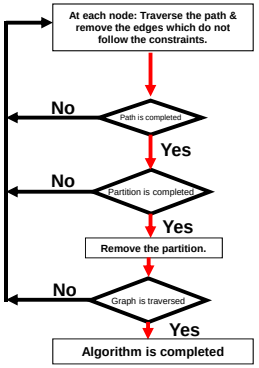


$sg_2 = p'_1 + p'_2 + p'_3 + p'_4 + p'_5 + p'_6$

$p'_1 = \{(2 , t_1)\},\ p'_2 = \{(2 , t_4)\},\ p'_3 = \{(2 , t_6)\},$
$p'_4 = \{(4 , t_5)\}, p'_5 = \{(5 , t_2)\}, p'_6 = \{(6 , t_3)\}$

Partitioning Algorithm





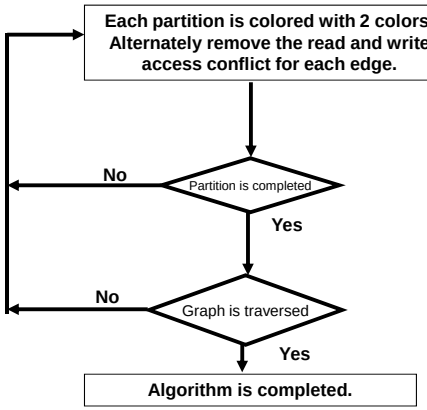
Remove sg_2 from graph *G'*.
Graph is completely traversed.
Partitioning algorithm is completed.

$p'_1 = \{(2 , t_1)\},\ p'_2 = \{(2 , t_4)\},\ p'_3 = \{(2 , t_6)\},$
$p'_4 = \{(4 , t_5)\}, p'_5 = \{(5 , t_2)\}, p'_6 = \{(6 , t_3)\}$

Outline

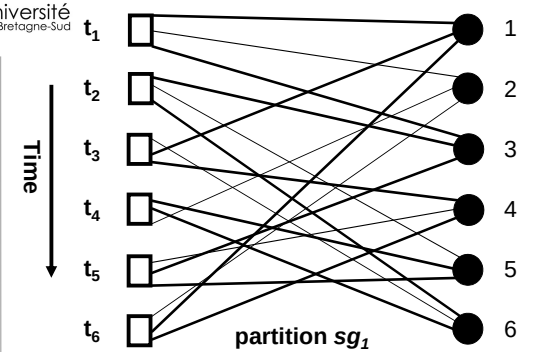
- **Problem Formulation**
- **Modeling**
  - **Preparing Bipartite Graph**
- **Algorithm**
  - **Partitioning**
  - **Coloring**
- **Conclusion**

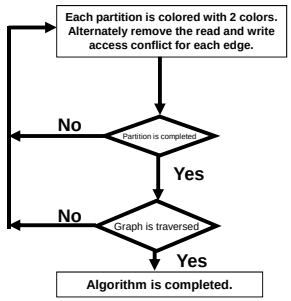
Coloring Algorithm



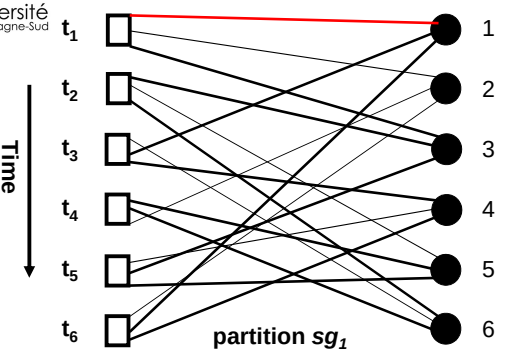
Coloring Algorithm.

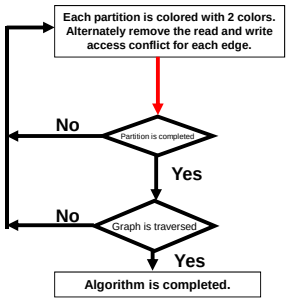
Coloring Algorithm





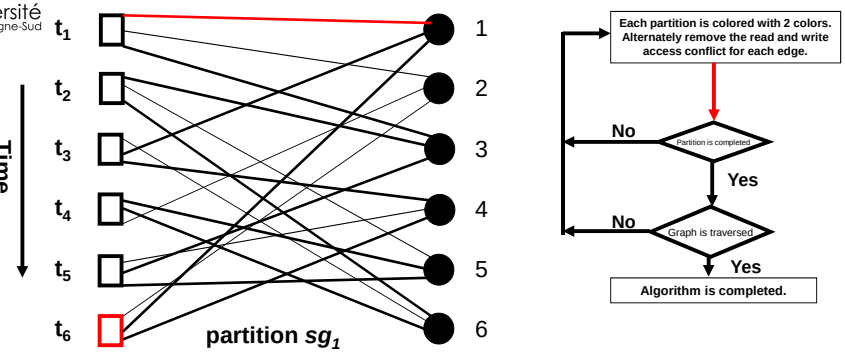
Coloring Algorithm





Color one edge.

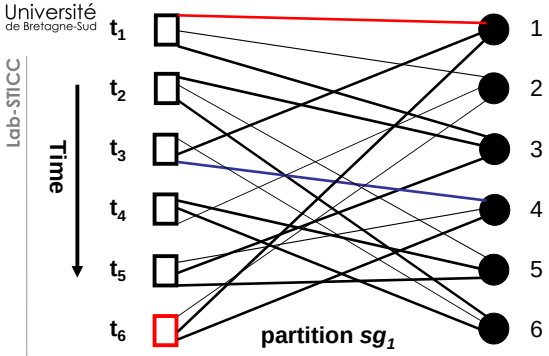
Coloring Algorithm

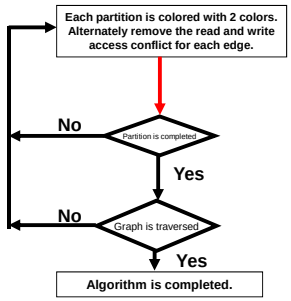


Search for the time node in which induced edge of the current colored edge exits. t_6 is the required node.

$$e_{0l,W} = e_{1l,R}$$
$$e_{1l,W} = e_{2l,R}$$
$$e_{2l,W} = e_{0l,R}$$

Coloring Algorithm

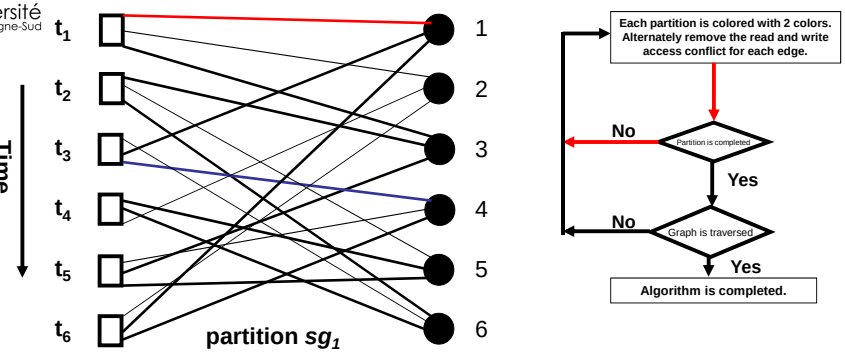




Search for the other edge which has induced edge at t_6 and give it 2nd color. In that way, we remove the write access conflict.

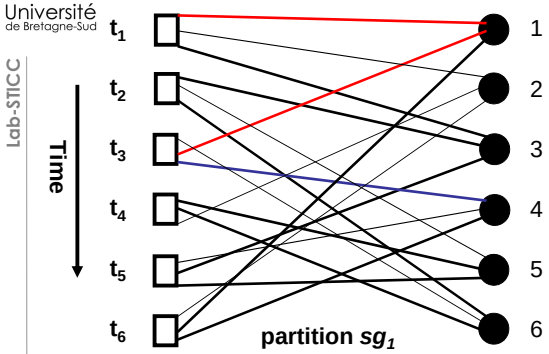
$$e_{0l,W} = e_{1l,R}$$
$$e_{1l,W} = e_{2l,R}$$
$$e_{2l,W} = e_{0l,R}$$

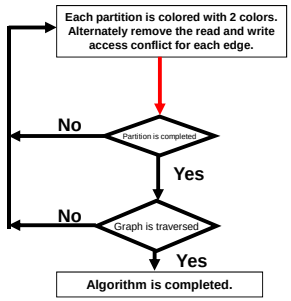
Coloring Algorithm



Partition is not completely colored. Go on coloring other edges.

Coloring Algorithm

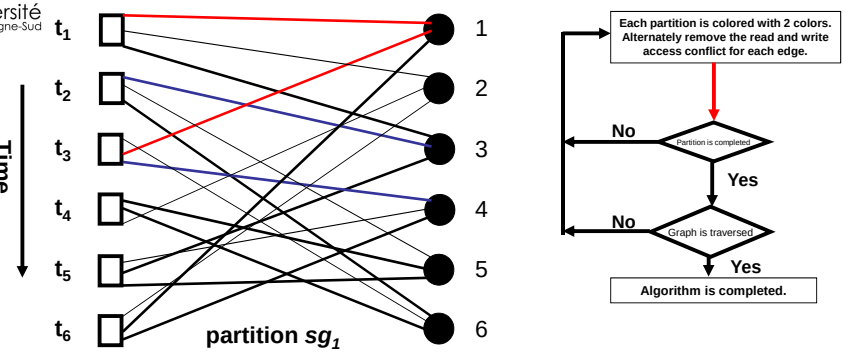




Remove the read access conflict by giving different color to the edge at t_3.

$$e_{0l,W} = e_{1l,R}$$
$$e_{1l,W} = e_{2l,R}$$
$$e_{2l,W} = e_{0l,R}$$

Coloring Algorithm

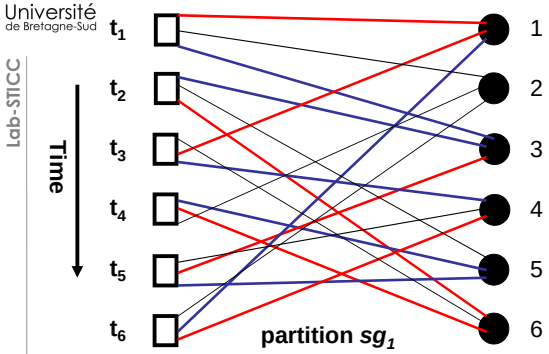


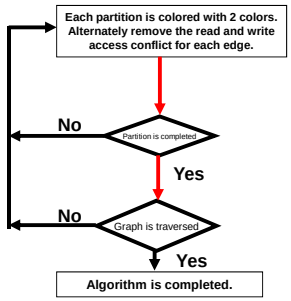
Time Nodes **Data Nodes**

Remove the write access conflict by searching for induced edges.

$$e_{0l,W} = e_{1l,R}$$
$$e_{1l,W} = e_{2l,R}$$
$$e_{2l,W} = e_{0l,R}$$

Coloring Algorithm

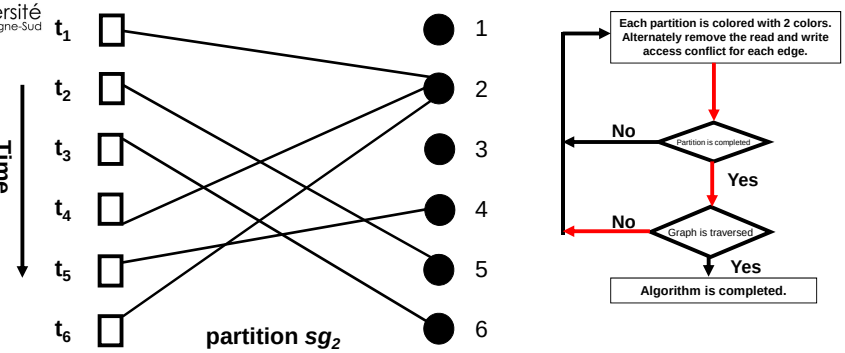




Time Nodes **Data Nodes**

Process continues until we completely colored the partition sg_1.

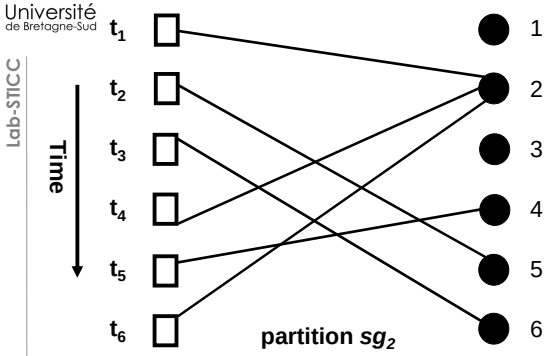
Coloring Algorithm

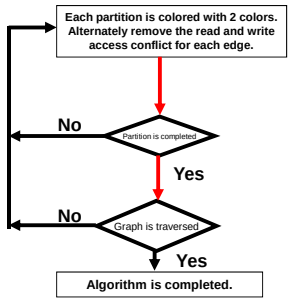


Time Nodes

Graph is not completely Colored.

Coloring Algorithm

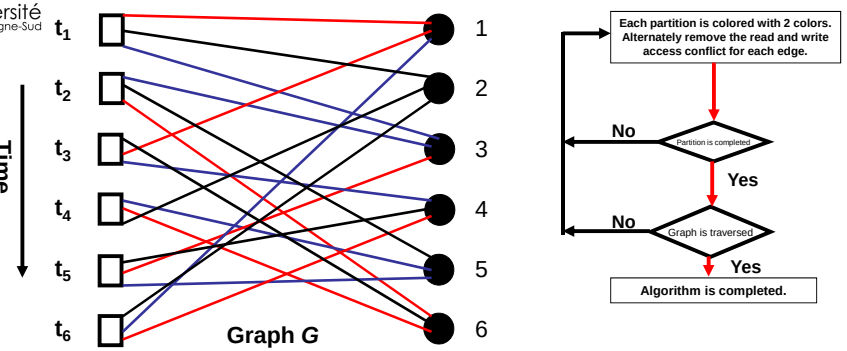




Time Nodes

Give one color to partition sg_2.

Coloring Algorithm



Graph is completely colored. Coloring algorithm is completed.

Complete Memory Mapping

	R	W		R	W		R	W		R	W		R	W		R	W
1			3			6			5			4			2		
2			5			1			6			3			1		
3			6			4			2			5			4		
t_1			t_2			t_3			t_4			t_5			t_6		

Complete Mapping

b_0
b_1
b_2

Outline

- **Problem Formulation**
- **Modeling**
  - **Preparing Bipartite Graph**
- **Algorithm**
  - **Partitioning**
  - **Coloring**
- **Conclusion**

Conclusion

- Concept of multiple read/write approach is presented.
- Modified bipartite edge coloring approach is introduced to find conflict free memory mapping for any type of parallel iterative decoding for LDPC.

Future Perspectives

Lab-STICC

- Additional constraints will be added in order to find mapping following the targeted interconnection network.
- Complexity of the interconnection network is considered in the future development of the mapping algorithm.

Questions

Lab-STICC