

# A scalable algorithm for homomorphic computing on multi-core clusters

Frédéric Gava & Léa Bayati

Laboratory of **A**lgorithms, **C**omplexity and **L**ogic (LACL)  
University of Paris-East



UNIVERSITÉ  
PARIS-EST CRÉTEIL  
VAL DE MARNE



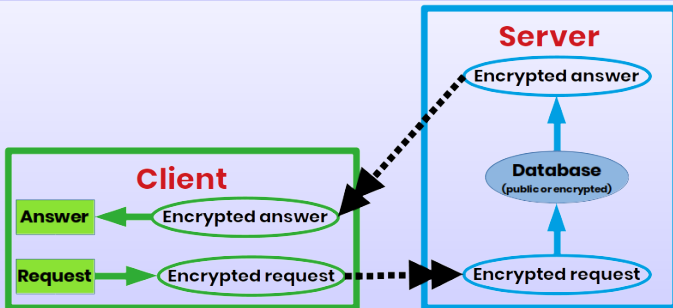
# Outline

- 1 Introduction
- 2 Homomorphic programming
- 3 BSP execution of boolean circuits
- 4 Conclusion

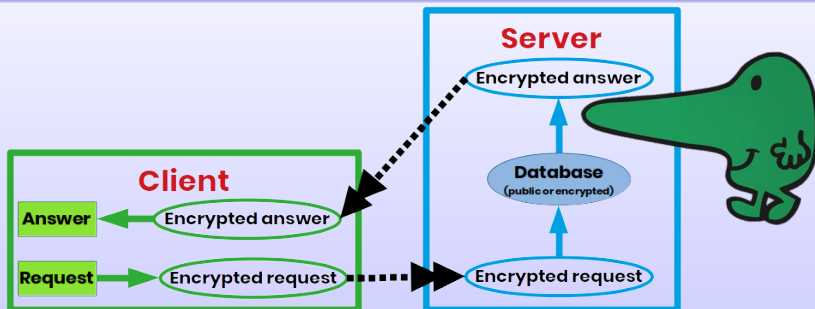
# Outline

- 1 Introduction
- 2 Homomorphic programming
- 3 BSP execution of boolean circuits
- 4 Conclusion

# Q: What is the context?



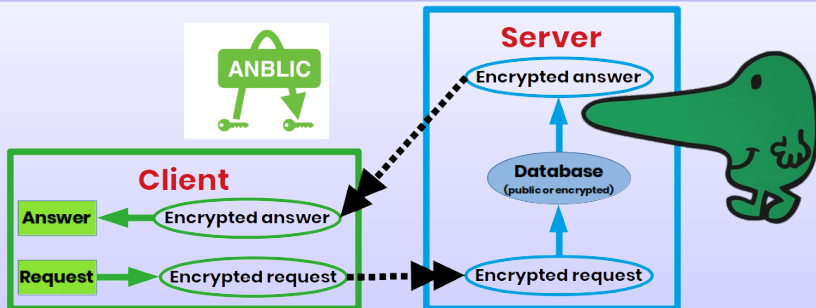
# Q: What is the context?



**Too-curious-but-honest** server (cloud provider):

- **Personal** data (e. g. medical; health pass)
- Company data (medical data or **secret formula**)

# Q: What is the context?

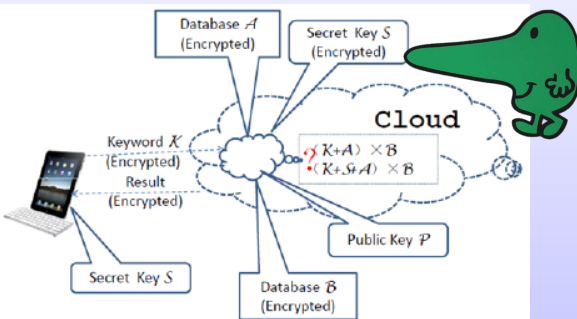


**Too-curious-but-honest** server (cloud provider):

- **Personal** data (e. g. medical; health pass)
- Compagny data (medical data or **secret formula**)

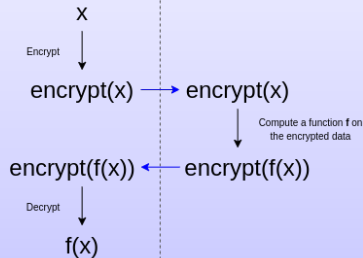
**ANBLIC** project (**AN**alysis in **BL**ind clouds), funded by FUI/BPI with ATOS, Capgemini, CoESSI, Wallix, CEA, ENS, UPEC

# Q: More specifically?

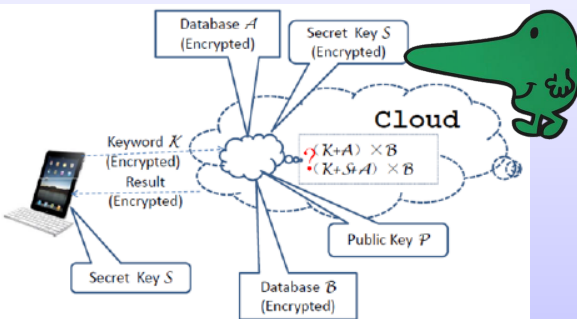


Client

Server

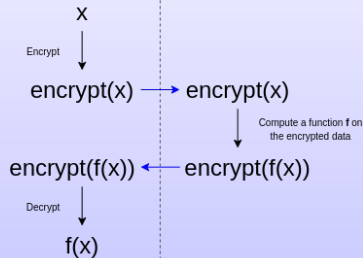


# Q: More specifically?



Client

Server



## Q: What is $f$ ?

Homomorphic programming is **computationally universal**  
But highly **inefficient**



## Q: What is homomorphic encryption?

A homomorphic operator  $(+, \oplus)$

- If  $Enc(m_i)=c_i$  and  $Dec(c_i)=m_i$  ( $i \in \{1, 2\}$ )
- Then  $Dec(c_1 \oplus c_2) = Dec(c_1) + Dec(c_2) = m_1 + m_2$

## Q: What is homomorphic encryption?

A homomorphic operator  $(+, \oplus)$

- If  $Enc(m_i)=c_i$  and  $Dec(c_i)=m_i$  ( $i \in \{1, 2\}$ )
- Then  $Dec(c_1 \oplus c_2) = Dec(c_1) + Dec(c_2) = m_1 + m_2$

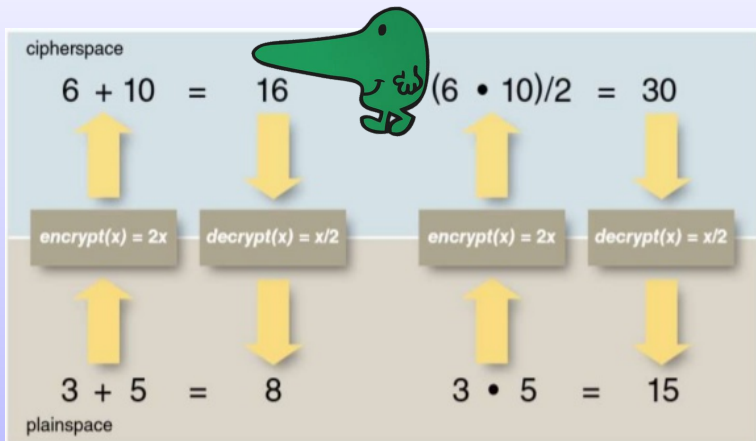
Fully homomorphic encryption (FHE)

FHE is when all the functions (Turing completeness) can be evaluated in a homomorphic way . Mainly, using boolean circuits (multiplicative depth problem).

# Q: Some examples?

Cloud

Client



## Q: Some examples?

cipherspace

$$6 + 10 = 16$$

$$(6 \cdot 10)/2 = 30$$

### Problem and additional solution

- Problem: very **inefficient** programs (an example later)
- Solution: have executed them on HPC **architectures** (parallel and **distributed** ones)

Client

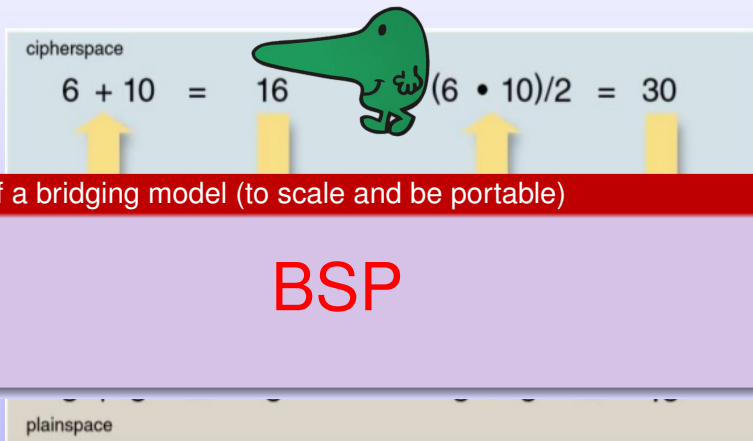
$$3 + 5 = 8$$

plaintext

$$3 \cdot 5 = 15$$

# Q: Some examples?

Cloud

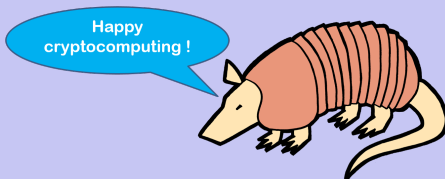


# Outline

- 1 Introduction
- 2 Homomorphic programming
- 3 BSP execution of boolean circuits
- 4 Conclusion

# Q: How FHE programming?

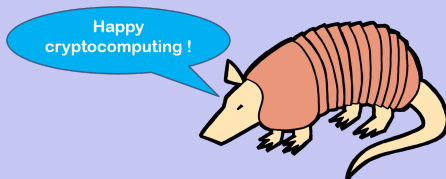
## Using Cingulata (CEA)



<https://github.com/CEA-LIST/Cingulata>

# Q: How FHE programming?

## Using Cingulata (CEA)



<https://github.com/CEA-LIST/Cingulata>

## Cingulata C++ environment

- Compilation chain (specific types)
- C++ operator overloading capabilities
- Generation of (encrypted) boolean circuits



**Q:** Can you show a simple example?

Sum of two integers !

## Q: Can you show a simple example?

```
1  #include <*.hxx>
2  int main() {
3      CiContext::set_config(...); // Set context
4
5      Cilnt a{Cilnt::u8}; // create from unsigned 8-bit template
6      Cilnt b{Cilnt::u8};
7      Cilnt c{Cilnt::u8};
8
9      a.read("a");          // read variable a and set name
10     b.read("b");          // read variable a and set name
11     c = a + b ;
12     c.write("c");         // set name and write variable
13
14     /* Export to file the "tracked" circuit */
15     CiContext::get_bit_exec_t<BitTracker>()->export_blif(blif_name, "hello")
16 }
```

**Q:** Can you show a real example?

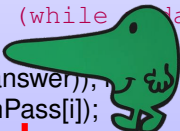
Membership in an  
encrypted data-base !

## Q: Can you show a real example?

```
1  int main() {
2      Cilnt PassTarget{Cilnt::s32};
3      vector<Cilnt> HeathPass(list_size, Cilnt::s32);
4      PassTarget.read("a");
5      // Initialising the Data Base (reading) as an array
6      for (int i = 0; i < list_size; i++) {
7          HeathPass[i].read("b_" + to_string(i));
8      }
9      // Comparing all the data (while updating the answer)
10     bool answer = false;
11     for (int i = 0; (i < list_size) or (not(answer)); i++) {
12         answer = (PassTarget == HeathPass[i]);
13     }
14     // Finally writing the answer
15     answer.write("s");
16 }
```

## Q: Can you show a real example?

```
1 int main() {  
2     Cilnt PassTarget{Cilnt::s32};  
3     vector<Cilnt> HeathPass(list_size, Cilnt::s32);  
4     PassTarget.read("a");  
5     // Initialising the Data Base (reading) as an array  
6     for (int i = 0; i < list_size; i++) {  
7         HeathPass[i].read("b_" + to_string(i));  
8     }  
9     // Comparing all the data (while calculating the answer)  
10    bool answer = false;  
11    for (int i = 0; (i < list_size) or (not(answer)),  
12        answer = (PassTarget == HeathPass[i]);  
13    }  
14    // Finally writing the answer  
15    answer.write("s");  
16 }
```

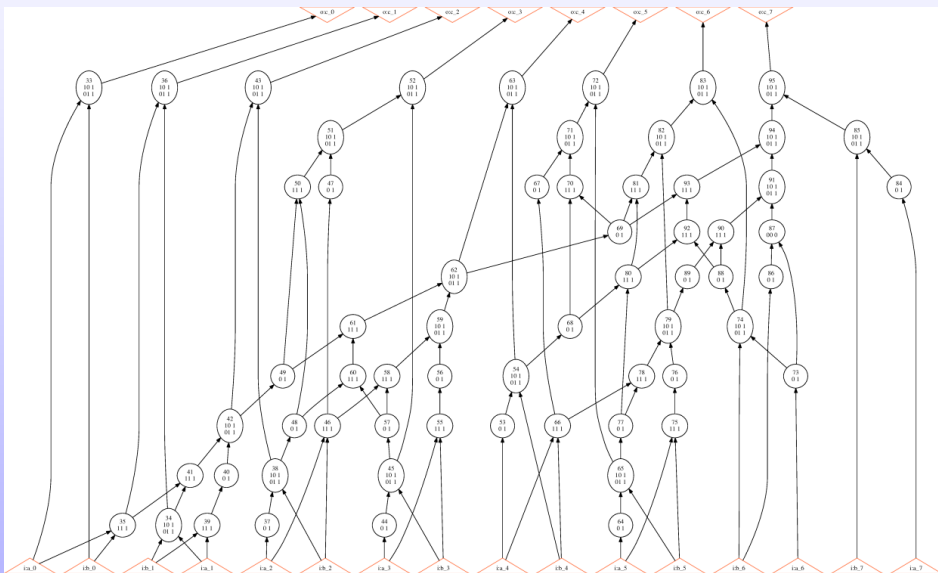


Ho, got i !

## Q: Can you show a real example?

```
1 int main() {
2     Cilnt PassTarget{Cilnt::s32};
3     vector<Cilnt> HeathPass(list_size, Cilnt::s32);
4     PassTarget.read("a");
5     // Initialising the Data Base (reading) as an array
6     for (int i = 0; i < list_size; i++) {
7         HeathPass[i].read("b_" + to_string(i));
8     }
9     // Comparing all the data (while updating the answer)
10    CiBit answer(0);
11    for (int i = 0; i < list_size; i++) {
12        answer = answer xor (PassTarget == HeathPass[i]);
13    }
14    // Finally writing the answer
15    answer.write("s");
16 }
```

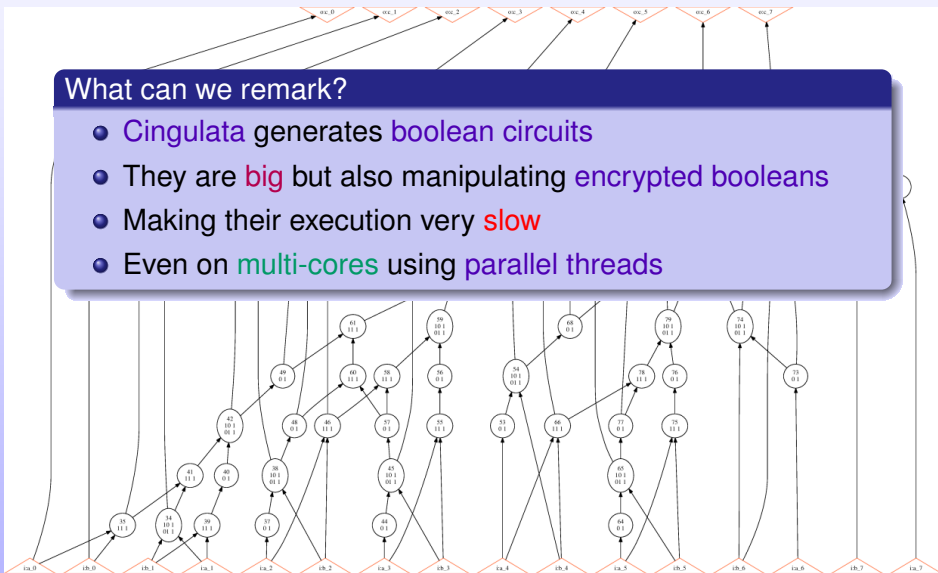
Q: So?



# Q: So?

## What can we remark?

- Cingulata generates **boolean circuits**
- They are **big** but also manipulating **encrypted booleans**
- Making their execution very **slow**
- Even on **multi-cores** using **parallel threads**





# Q: So?

## What can we remark?

- Cingulata generates boolean circuits
- They are big but also manipulating encrypted booleans
- Making their execution very slow
- Even on multi-cores using parallel threads

## What can we do?

- Distributed execution using the BSP model
- Finding a distribution of the gates
- Benchmarking

## Q: So?

## Why not using?

- Dynamic **scheduling** (workflow): too long to redistribute data
- **Master-slave**: same
- **Complex** algorithm (model) = loss of confidence

# Outline

- 1 Introduction
- 2 Homomorphic programming
- 3 BSP execution of boolean circuits**
- 4 Conclusion

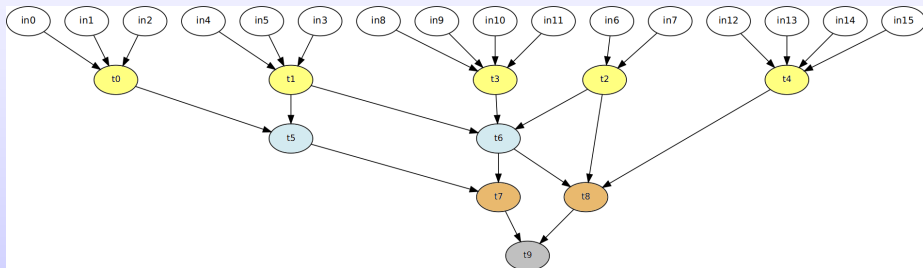
## Q: How to manage the gates?

**Levels** ( $v$  is a gate of  $G$ )

**level**( $G, v$ ) is inductively defined by :

- 0 if  $v \in V_{in}$  (input)
- $\max(n_i) + 1$  if  $\forall v_i$  such that  $v \in \mathbf{edges}(G, v_i)$  and **level**( $G, v_i$ ) =  $n_i$  (otherwise)

# Q: How to manage the gates?



## Example

<i>Niveau<sub>0</sub></i>	$[in_0, in_1, \dots, in_{15}]$
<i>Niveau<sub>1</sub></i>	$[t_0, t_1, t_3, t_2, t_4]$
<i>Niveau<sub>2</sub></i>	$[t_5, t_6]$
<i>Niveau<sub>3</sub></i>	$[t_7, t_8]$
<i>Niveau<sub>4</sub></i>	$[t_9]$

# Q: What is the BSP execution?

## Algorithm: BSP execution of boolean circuits

```
1 input : A boolean circuit  $G$  and encrypted input bits
2 output: Encrypted output bits
3  $(G, levels, N) \leftarrow \text{build}(G)$ ;
4  $\{G_i\} \leftarrow \text{select}(G, levels, N, \text{pid})$ ;
5  $bits \leftarrow \text{read}(V_{in})$ ;
6 for  $l \in \{1 \dots N\}$  do
7    $sub \leftarrow G_l$  where  $G_l \in \{G_i\}$ ;
8    $toSend \leftarrow \text{run}(sub, bits)$ ;
9    $rcv \leftarrow \text{CommBSP}(toSend)$ ;
10   $bits \leftarrow \text{update}(rcv)$ ;
11 end
12  $\text{write}(V_{out}, bits)$ ;
```

# Q: What is the BSP execution?

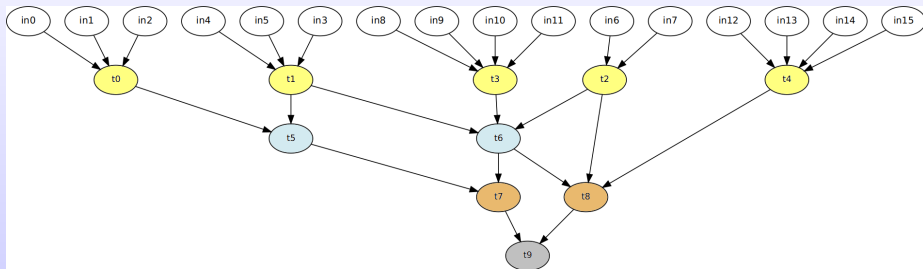
## Algorithm: BSP execution of boolean circuits

```
1 input : A boolean circuit  $G$  and encrypted input bits
2 output: Encrypted output bits
3  $(G, levels, N) \leftarrow \mathbf{build}(G)$ ;
4  $\{G_i\} \leftarrow \mathbf{select}(G, levels, N, \mathbf{pid})$ ;
5  $\mathbf{bits} \leftarrow \mathbf{read}(V_{in})$ ;
6 for  $l \in \{1 \dots N\}$  do
7    $\mathbf{sub} \leftarrow G_l$  where  $G_l \in \{G_i\}$ ;
8    $\mathbf{toSend} \leftarrow \mathbf{run}(\mathbf{sub}, \mathbf{bits})$ ;
9    $\mathbf{rcv} \leftarrow \mathbf{CommBSP}(\mathbf{toSend})$ ;
10   $\mathbf{bits} \leftarrow \mathbf{update}(\mathbf{rcv})$ ;
11 end
12  $\mathbf{write}(V_{out}, \mathbf{bits})$ ;
```

## Distribution of the gates

Hashing the gates modulo  $p$

# Q: What is the BSP execution?



## Example of distribution (BSP machine with $p = 2$ )

<i>Niveau<sub>1</sub></i>	$[t_0 \rightarrow 0, t_1 \rightarrow 1, t_3 \rightarrow 0, t_2 \rightarrow 1, t_4 \rightarrow 0]$
<i>Niveau<sub>2</sub></i>	$[t_5 \rightarrow 0, t_6 \rightarrow 1]$
<i>Niveau<sub>3</sub></i>	$[t_7 \rightarrow 0, t_8 \rightarrow 1]$
<i>Niveau<sub>4</sub></i>	$[t_9 \rightarrow 0]$



# Q: What is your distributed architecture?

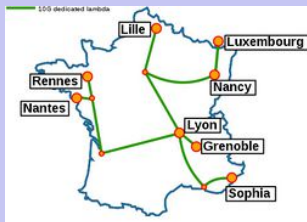
Benchmarks on Grid'5k (Grid'5000)



8 places, 38 clusters, 761  
nodes, 15696 CPU cores,  
101.5 TiB RAM

# Q: What is your distributed architecture?

## Benchmarks on Grid'5k (Grid'5000)



8 places, 38 clusters, 761  
nodes, 15696 CPU cores,  
101.5 TiB RAM

# Q: What is your distributed architecture?

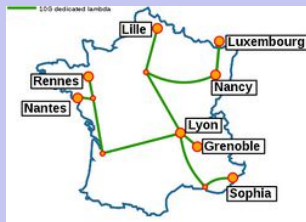
## Benchmarks on Grid'5k (Grid'5000)



8 places, 38 clusters, 761  
nodes, 15696 CPU cores,  
101.5 TiB RAM

# Q: What is your distributed architecture?

## Benchmarks on Grid'5k (Grid'5000)



8 places, 38 clusters, 761  
nodes, 15696 CPU cores,  
101.5 TiB RAM

## Machines

- Lille/chetemi; 15 nodes of 2×Intel Xeon E5-2630 of 10 cores (with 256 GiB RAM) and a 10 Gbps (SR-IOV) network
- Nantes/ecotype; 48 nodes of 2×Intel Xeon E5-2630 of 10 cores (with 128 GiB) and a 10 Gbps (SR-IOV) network

# Q: For which performances?

## Benchmarks of membership

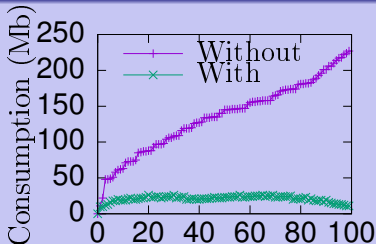
#size	#nodes	#procs	#time(s)	speedup
80	4	2	22	3.6
80	16	2	12	6.7
160	4	2	66	2.6
160	16	2	22	7.8
320	4	2	97	-
320	16	2	41	-

# Q: For which performances?

## Benchmarks of membership

#size	#nodes	#procs	#time(s)	speedup
80	4	2	22	3.6
80	16	2	12	6.7
160	4	2	66	2.6
160	16	2	22	7.8
320	4	2	97	-
320	16	2	41	-

## Memory consumption (with/without managment)



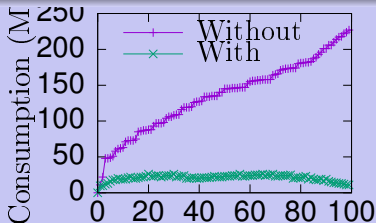
## Q: For which performances?

### Benchmarks of membership

#size	#nodes	#procs	#time(s)	speedup
80	4	2	22	3.6
80	16	2	12	6.7
160	4	2	66	2.6

### Not enough !

- Too much and **badly balanced** communication
- Badly balanced computations
- We need **better distributions** !

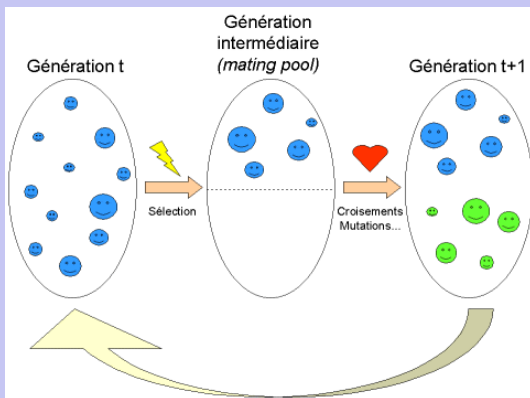


# Q: How to find such distributions?

NP-Hard problem

So, need of an heuristic

Generic genetic algorithm





# Q: How configuring such an algorithm?

## Generic to Instantiation

- Individuals  $\equiv$  association gates  $\Rightarrow$  processors (per level)
- Selection  $\equiv$  abstract (cost) BSP execution of the circuit for a particular machine with its BSP parameters
- Crossover  $\equiv$  merging associations
- Mutation  $\equiv$  moving a gate in the association

and more...

# Q: How configuring such an algorithm?

## Generic to Instantiation

- Individuals  $\equiv$  association gates  $\Rightarrow$  processors (per level)
- Selection  $\equiv$  abstract (cost) BSP execution of the circuit for a particular machine with its BSP parameters
- Crossover  $\equiv$  merging associations
- Mutation  $\equiv$  moving a gate in the association

and more

## Q: How configuring such an algorithm?

### Generic to Instantiation

- **Individuals**  $\equiv$  **association** gates  $\Rightarrow$  processors (per level)
- **Selection**  $\equiv$  abstract (cost) BSP **execution** of the circuit for a particular machine with its BSP **parameters**
- **Crossover**  $\equiv$  **merging** associations
- **Mutation**  $\equiv$  **moving** a gate in the association

### Pro and cons

• Scalability

• Expressiveness

# Q: How configuring such an algorithm?

## Generic to Instantiation

- **Individuals**  $\equiv$  **association** gates  $\Rightarrow$  processors (per level)
- **Selection**  $\equiv$  abstract (cost) BSP **execution** of the circuit for a particular machine with its BSP **parameters**
- **Crossover**  $\equiv$  **merging** associations
- **Mutation**  $\equiv$  **moving** a gate in the association

## Pro and cons

- **Cons**
  - Complexity of the algorithm
  - Complexity of the circuit
  - Complexity of the machine
  - Complexity of the parameters
- **Pros**
  - Simplicity of the algorithm
  - Simplicity of the circuit
  - Simplicity of the machine
  - Simplicity of the parameters

## Q: How configuring such an algorithm?

### Generic to Instantiation

- **Individuals**  $\equiv$  **association** gates  $\Rightarrow$  processors (per level)
- **Selection**  $\equiv$  abstract (cost) BSP **execution** of the circuit for a particular machine with its BSP **parameters**
- **Crossover**  $\equiv$  **merging** associations
- **Mutation**  $\equiv$  **moving** a gate in the association

### Pro and cons

- Easy to implement and local optimum
- Partial results can be reused

## Q: How configuring such an algorithm?

### Generic to Instantiation

- Individuals  $\equiv$  association gates  $\Rightarrow$  processors (per level)
- Selection  $\equiv$  abstract (cost) BSP execution of the circuit for a particular machine with its BSP parameters
- Crossover  $\equiv$  merging associations
- Mutation  $\equiv$  moving a gate in the association

### Pro and cons

- Easy to implement and local optimum
- Partial results can be reused
- But working only for a single machine for each circuit
- But "slow" so only for the biggest circuits

## Q: How configuring such an algorithm?

### Generic to Instantiation

- Individuals  $\equiv$  association gates  $\Rightarrow$  processors (per level)
- Selection  $\equiv$  abstract (cost) BSP execution of the circuit for a particular machine with its BSP parameters
- Crossover  $\equiv$  merging associations
- Mutation  $\equiv$  moving a gate in the association

### Pro and cons

- Easy to implement and local optimum
- Partial results can be reused
- But working only for a single machine for each circuit
- But “slow” so only for the biggest circuits

## Q: How configuring such an algorithm?

### Generic to Instantiation

- Individuals  $\equiv$  association gates  $\Rightarrow$  processors (per level)
- Selection  $\equiv$  abstract (cost) BSP execution of the circuit for a particular machine with its BSP parameters
- Crossover  $\equiv$  merging associations
- Mutation  $\equiv$  moving a gate in the association

### Pro and cons

- Easy to implement and local optimum
- Partial results can be reused
- But working only for a single machine for each circuit
- But “slow” so only for the biggest circuits



## Q: How configuring such an algorithm?

### Generic to Instantiation

- Individuals  $\equiv$  association gates  $\Rightarrow$  processors (per level)
- Selection  $\equiv$  abstract (cost) BSP execution of the circuit for a particular machine with its BSP parameters
- Crossover  $\equiv$  merging associations
- Mutation  $\equiv$  moving a gate in the association

### Pro and cons

- Easy to implement and local optimum
- Partial results can be reused
- But working only for a single machine for each circuit
- But “slow” so only for the biggest circuits

## Q: How configuring such an algorithm?

### Generic to Instantiation

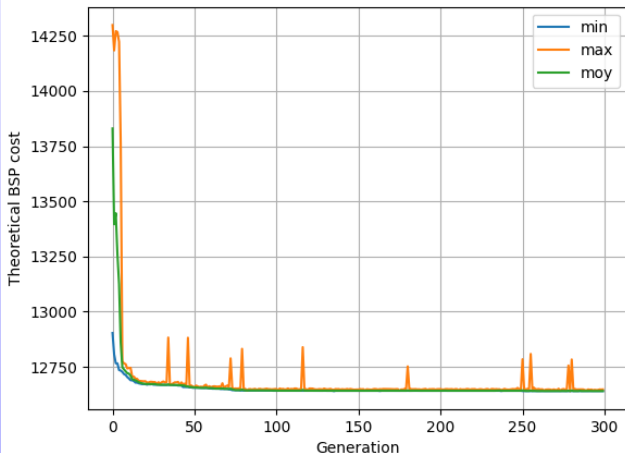
- Individuals  $\equiv$  association gates  $\Rightarrow$  processors (per level)
- Selection  $\equiv$  abstract (cost) BSP execution of the circuit for a particular machine with its BSP parameters
- Crossover  $\equiv$  merging associations
- Mutation  $\equiv$  moving a gate in the association

### Pro and cons

- Easy to implement and local optimum
- Partial results can be reused
- But working only for a single machine for each circuit
- But “slow” so only for the biggest circuits

# Q: How much time to find such local optimums?

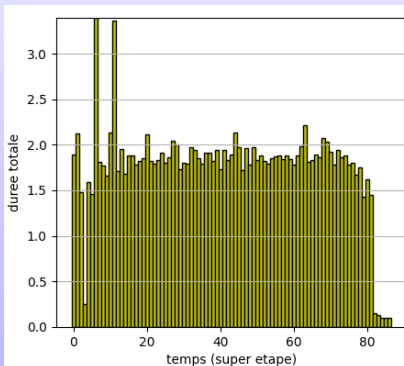
Membership (80 items) for a  $2 \times 4$  processors machine:



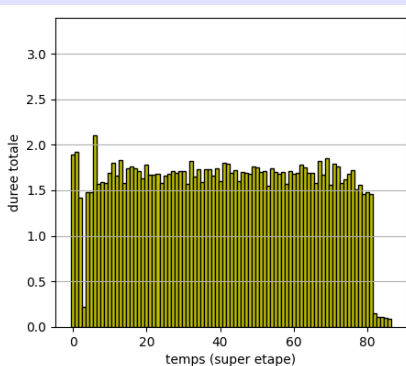
# Q: Do we gain in load-balancing?

For each super-step (membership 40 items):

“Random” distribution



Genetic distribution



Q: Have you another example?

## Q: Have you another example?

From the ANBLIC project

Public **medical database** and **encrypted** requests (scoring):

$$\text{score}(q, d) = \text{coord}(q, d) \times \sum_{t \in q} \text{tf}(t, d) \times \text{idf}(t)^2$$

Could be rewrite as:

$$\text{score}(q, d) = \sum_{t \in T} x_t^{(q)} \cdot \chi(t, d) \times \sum_{t \in T} x_t^{(q)} \cdot \alpha(t, d)$$

## Q: Have you another example?

From the ANBLIC project

Public **medical database** and **encrypted** requests (scoring):

$$\text{score}(q, d) = \text{coord}(q, d) \times \sum_{t \in q} \text{tf}(t, d) \times \text{idf}(t)^2$$

Could be rewrite as:

$$\text{score}(q, d) = \sum_{t \in T} x_t^{(q)} \cdot \chi(t, d) \times \sum_{t \in T} x_t^{(q)} \cdot \alpha(t, d)$$

```
1 // Public database (ALPHA and CHI)
2 // NUM_OF_TERMS, number of terms in the dictionary
3 // NUM_OF_DOCS, number of documents in the database
4 for(int d=0;d<NUM_OF_DOCS;d++) {
5     coord=acc=0;
6     for(int t=0;t<NUM_OF_TERMS;t++) {
7         acc = acc + q[t]*ALPHA[t][d];
8         coord = coord + q[t]*CHI[t][d];
9     }
10 }
```

# Q: Have you benchmarks?

## Benchmarks of scoring (request in a medical database)

#nodes	#procs	#cores	#BSP-naive	#BSP-gen	speedup
1	1	16	-	5769	-
4	1	16	5408	3603	1.6
4	2	8	2485	1866	3.1
8	1	16	3432	1958	3.0
8	2	8	1390	1148	5.0
16	1	16	1310	1095	5.3
16	2	8	716	651	8.9



## Q: Have you benchmarks?

### Benchmarks of scoring (request in a medical database)

#nodes	#procs	#cores	#BSP-naive	#BSP-gen	speedup
1	1	16	-	5769	-
4	1	16	5408	3603	1.6
4	2	8	2485	1866	3.1
8	1	16	3432	1958	3.0
8	2	8	1390	1148	5.0
16	1	16	1310	1095	5.3
16	2	8	716	651	8.9

### Not enough !

- Better performances using the “genetic” distribution
- Better performances when using both multi-core and BSP
- But not a linear acceleration

# Outline

- 1 Introduction
- 2 Homomorphic programming
- 3 BSP execution of boolean circuits
- 4 Conclusion

# Conclusion

## Main results

- **BSP** execution of boolean circuits (from Cingulata)
- A genetic algorithm/heuristic for the gates distribution
- **Benchmarks** on the Grid'5k environment

## Perspectives (Ongoing Future Work)

# Conclusion

## Main results

- **BSP** execution of boolean circuits (from **Cingulata**)
- A **genetic** algorithm/heuristic for the **gates distribution**
- **Benchmarks** on the **Grid'5k** environment

## Perspectives (Ongoing/Future Work)

# Conclusion

## Main results

- **BSP** execution of boolean circuits (from **Cingulata**)
- A **genetic** algorithm/heuristic for the **gates distribution**
- **Benchmarks** on the **Grid'5k** environment

## Perspectives (Ongoing/Future Work)

- **Optimization** of the **gates distribution**
- **Integration** of the **BSP** execution with the **gates distribution**
- **Integration** of the **BSP** execution with the **gates distribution**
- **Integration** of the **BSP** execution with the **gates distribution**
- **Integration** of the **BSP** execution with the **gates distribution**

# Conclusion

## Main results

- **BSP** execution of boolean circuits (from **Cingulata**)
- A **genetic** algorithm/heuristic for the **gates distribution**
- **Benchmarks** on the **Grid'5k** environment

## Perspectives (Ongoing/Future Work)

- (Multi-)BSP execution
- Merging of sub-graphs (sub-distribution) to avoid some synchronisations and having better performances
- Comparison with other approaches

# Conclusion

## Main results

- **BSP** execution of boolean circuits (from **Cingulata**)
- A **genetic** algorithm/heuristic for the **gates distribution**
- **Benchmarks** on the **Grid'5k** environment

## Perspectives (Ongoing/Future Work)

- (Multi-)BSP execution
- **Merging** of **sub-graphs** (sub-distribution) to avoid some synchronisations and having better performances
- More examples and benchmarks

# Conclusion

## Main results

- **BSP** execution of boolean circuits (from **Cingulata**)
- A **genetic** algorithm/heuristic for the **gates distribution**
- **Benchmarks** on the **Grid'5k** environment

## Perspectives (Ongoing/Future Work)

- (Multi-)BSP execution
- **Merging** of **sub-graphs** (sub-distribution) to avoid some synchronisations and having better performances
- More examples and benchmarks



# Conclusion

## Main results

- **BSP** execution of boolean circuits (from **Cingulata**)
- A **genetic** algorithm/heuristic for the **gates distribution**
- **Benchmarks** on the **Grid'5k** environment

## Perspectives (Ongoing/Future Work)

- (Multi-)BSP execution
- **Merging** of **sub-graphs** (sub-distribution) to avoid some synchronisations and having better performances
- More examples and benchmarks

# Conclusion

## Main results

- **BSP** execution of boolean circuits (from **Cingulata**)
- A **genetic** algorithm/heuristic for the **gates distribution**
- **Benchmarks** on the **Grid'5k** environment

## Perspectives (Ongoing/Future Work)

- (Multi-)BSP execution
- **Merging** of **sub-graphs** (sub-distribution) to avoid some synchronisations and having better performances
- More examples and benchmarks

**Merci !**