Parallel Object Programming in POP-C++: a case study for sparse matrix vector multiplication

Clovis Dongmo Jiogo
Pierre Manneback
Faculté polytechnique de Mons

Pierre Kuonen
University of Fribourg

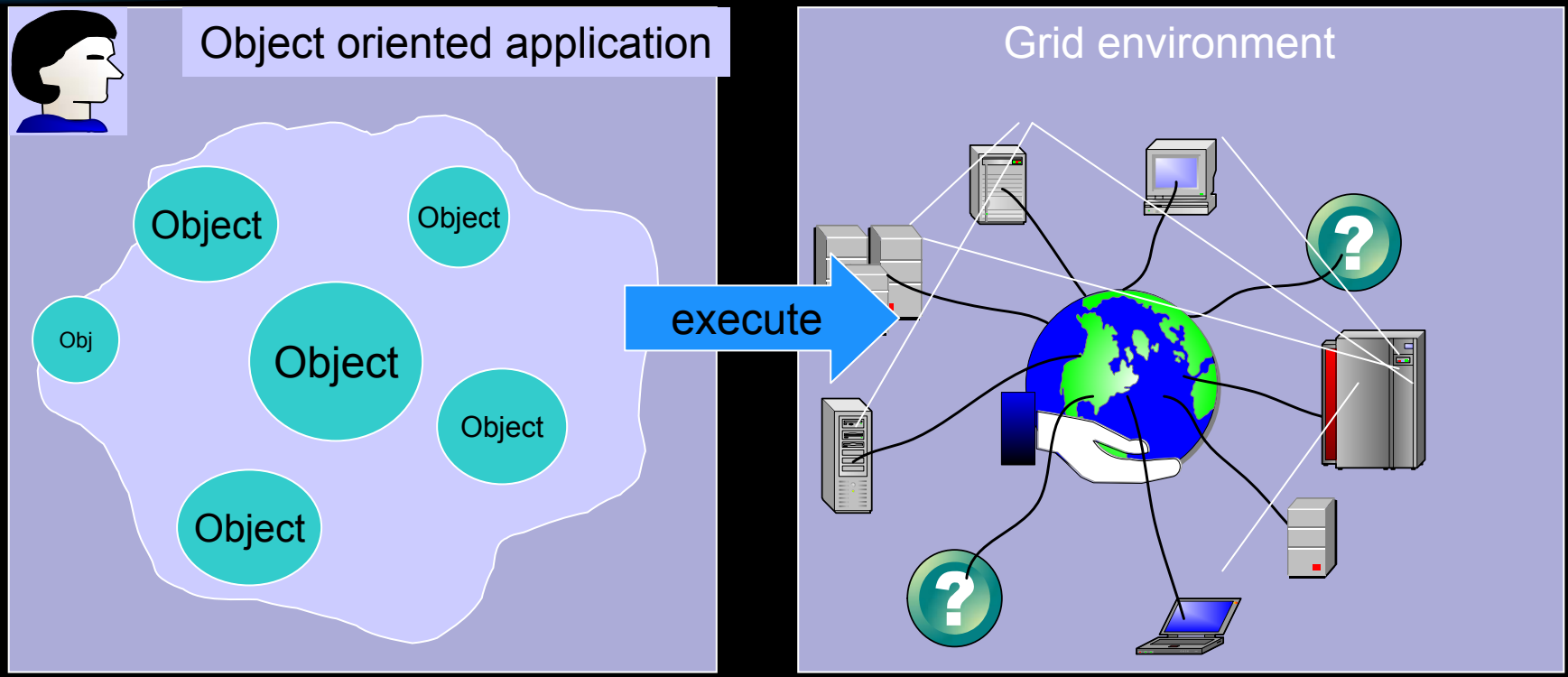
Purpose of this work

- Test Pop-C++ for some scientific computations on Grids
  - Present the parallel programming model POP-C++
  - Evaluate its performances in Grid environment
  - Show how POP-C++ can improve matrix computations

Agenda

- Overview of POP-C++
- Sparse Matrix/Vector product
- Programming in Pop-C++
- Experimental results
- Future work

POP: Parallel Object Programming



- Distributed objects
- Heterogeneous
- Dynamic

- Heterogeneous
- Large scale
- Unstructured
- Dynamic and unknown topology

Approach of POP-C++

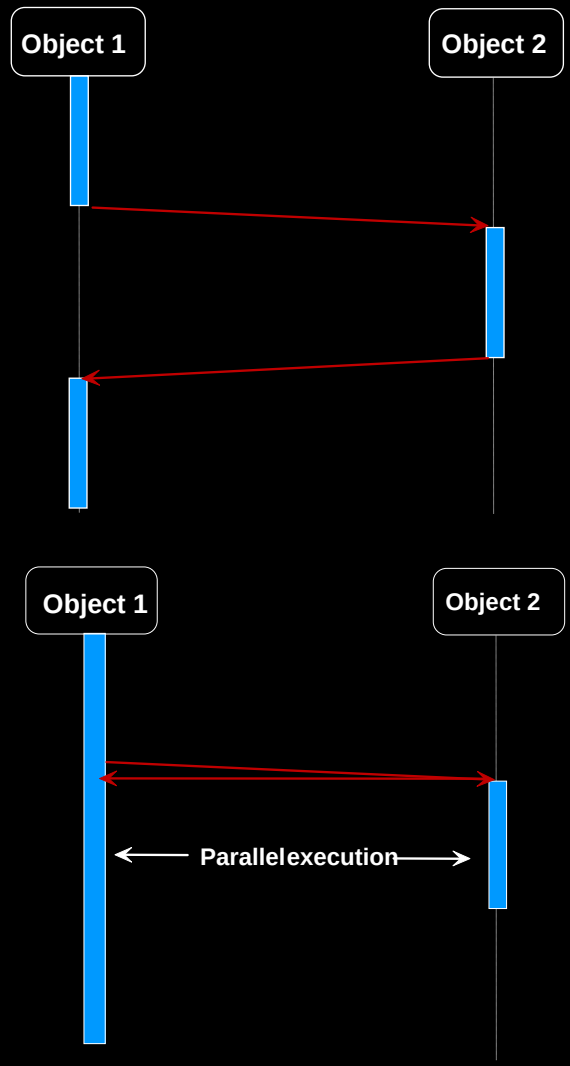
- Service oriented approach
- Resource allocation driven by object requirements
- Various invocations semantics
- Object-oriented parallel programming paradigm (parallel objects)
- Object-oriented Programming System

POP-C++ Programming Model

- Extension of C++ language
- Data transmission via shared object
- Two level of parallelism
  - Inter-object parallelism
  - Intra-object parallelism
- Transparent and dynamic object allocation guided by the object resources need
- Capacity to glue to Grid Toolkits

Semantic invocation : interface side

- Two ways to call a method
  - Synchronous
    - Method returns when the execution is finished
      - Same semantic than sequential invocation
  - Asynchronous
    - Method returns immediately
      - Allows parallelism but.. no returned value



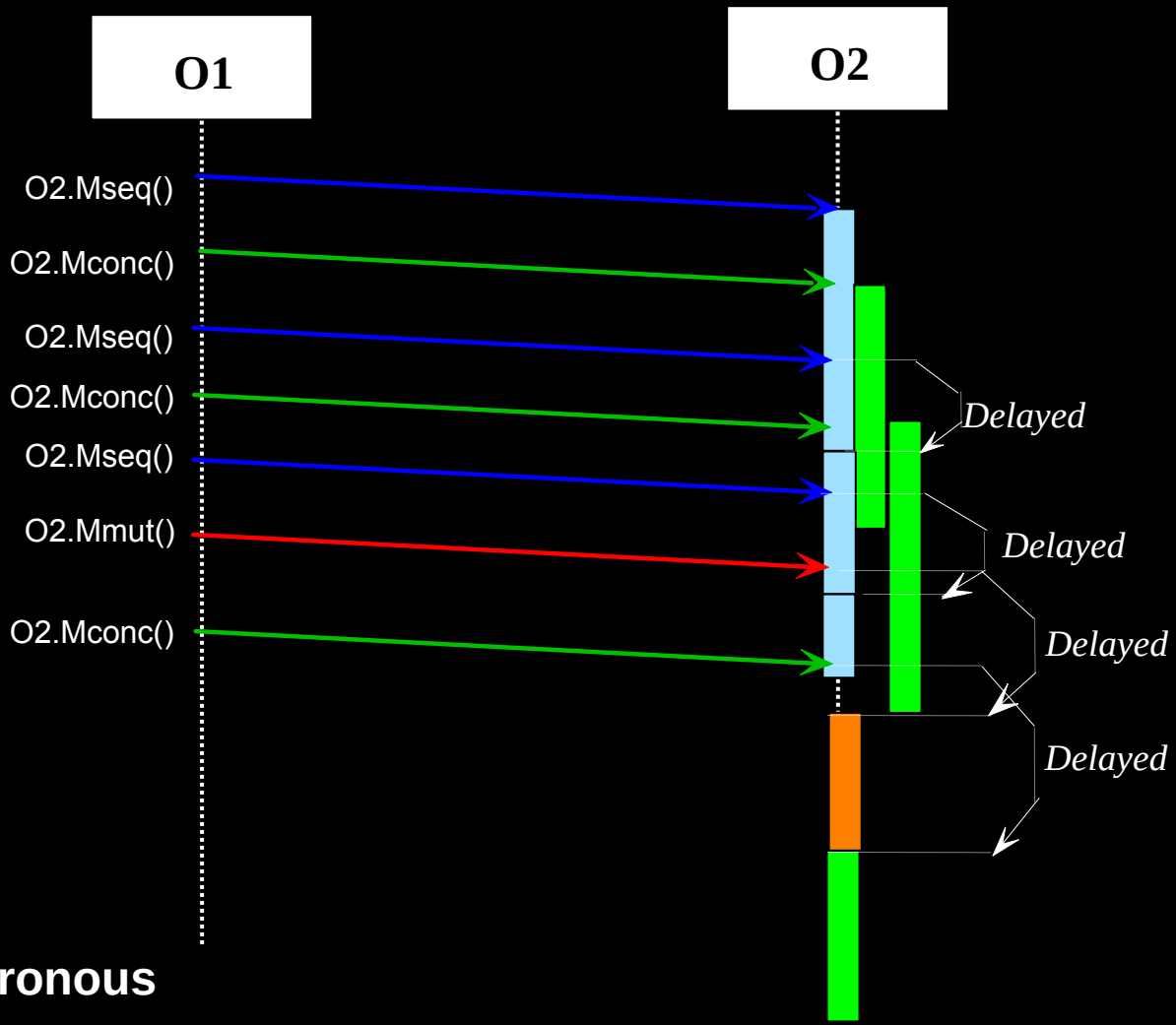
Method call semantics : definition

1 - An arriving concurrent call can be executed concurrently (time sharing) when it arrives, except if mutex calls are pending or executing. In the later case he is executed after completion of all mutex calls previously arrived.

2 - An arriving sequential call is executed after completion of all sequential and mutex calls previously arrived.

3 - An arriving mutex call is executed after completion of all calls previously arrived.

Method call semantics : example



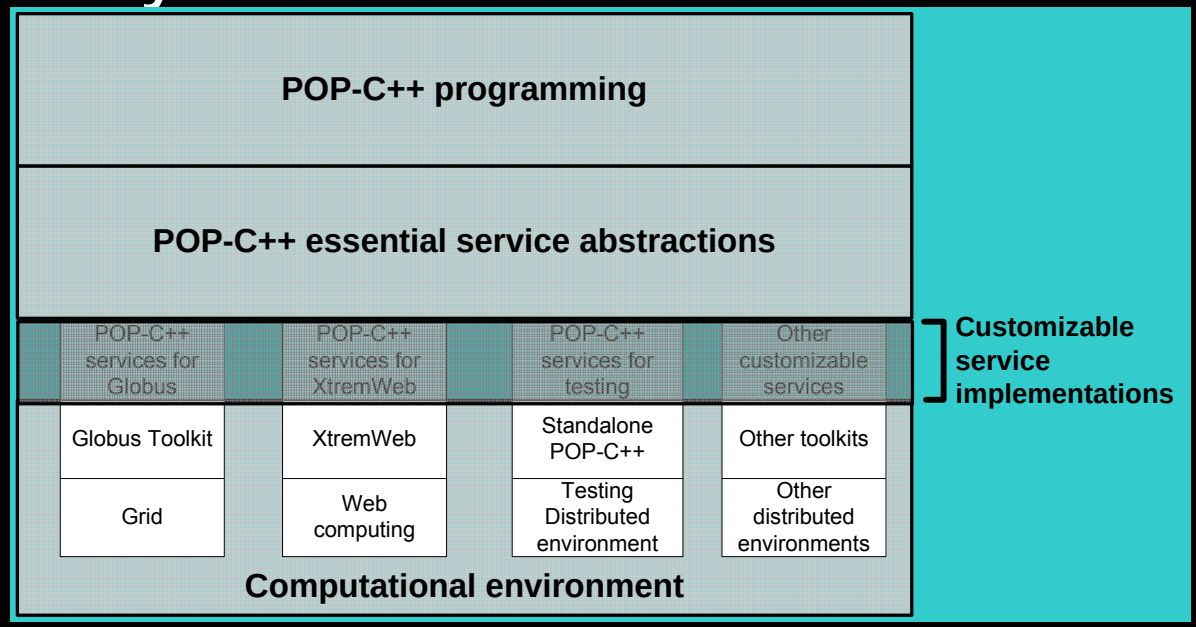
- All calls are asynchronous

POP-C++ Syntax

- POP-C++ is an implementation of the parallel object model as an extension of C++ with six new key words :
  - **`parclass`** : to declare a parallel class
  - **`async`** : asynchronous method call
  - **`sync`** : synchronous method call
  - **`conc`** : concurrent method execution
  - **`seq`** : sequential method execution
  - **`mutex`** : mutex method execution

POP-C++ architecture

- A multi-layer architecture



  - Integration of new middleware into the system in a PnP flavor

Requirement-driven objects

- Each parallel object has a user-specified object description (OD)
- OD describes the requirements of parallel objects
- OD is used as a guideline for allocating resource and object migration
- OD can be expressed in terms of:
  - Maximum computing power (e.g. Mflops)
  - Communication bandwidth with its interface
  - Memory needed
- OD can be parameterized on each parallel object (based on the actual input)

Object description example

```
parclass Matrix
{Matrix (int n) @{
    od.power(300 , 100);
    od.memory(n*n*sizeof(double)/1E6)
    od.protocol("socket http")
...                                          }
}
```

The creation of an object for Matrix parallel class requires:

- A computing power of 300Mfps, but 100Mfps are acceptable
- A capacity memory of de n*n*sizeof(double)/1E6 Mbytes
- A protocol socket or http for the communication

Agenda

- Overview of POP-C++
- Sparse Matrix/Vector product
- Programming in Pop-C++
- Experimental results
- Future work

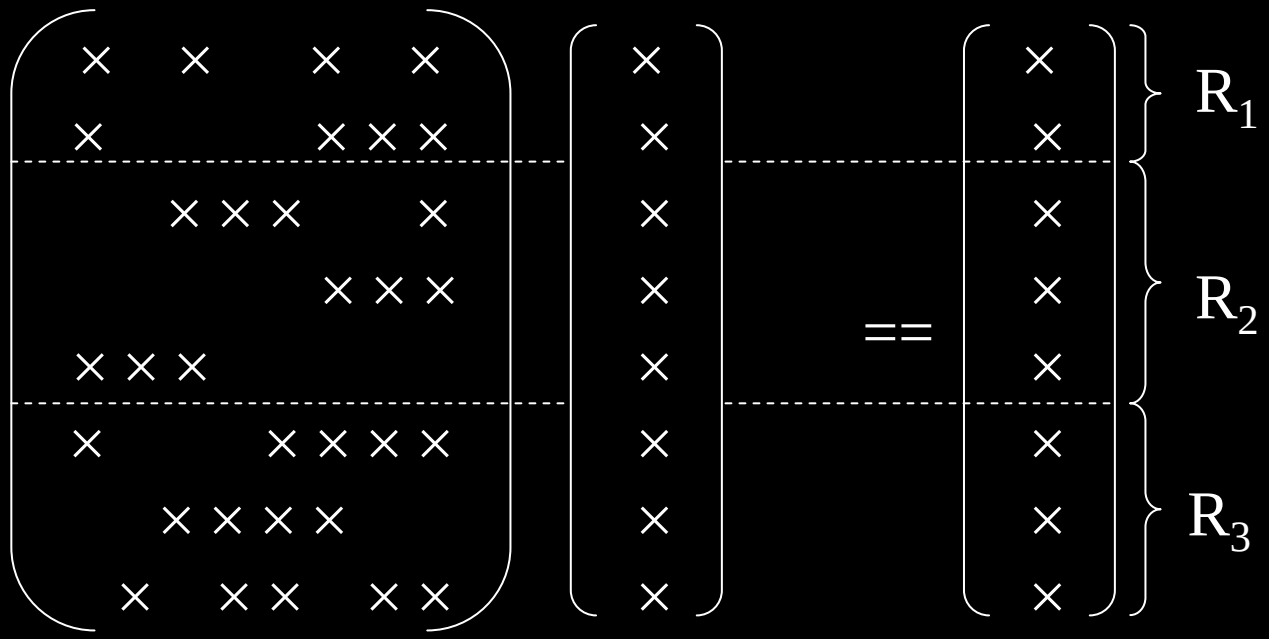
Sparse storage format : CRS

- CRS data structure use three vectors

$$\begin{pmatrix} 11 & 0 & 14 & 0 & 0 \\ 0 & 22 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 14 & 0 & 0 & 0 & 45 \\ 15 & 0 & 0 & 45 & 0 \end{pmatrix}$$

```
Row_ptr[*]  = [1; 3; 4; 6; 8]
Col_ind[*]  = [1; 3; 2; 1; 5; 1; 4]
Mat_val[*]  = [11; 14; 22; 14; 45; 15; 45]
```

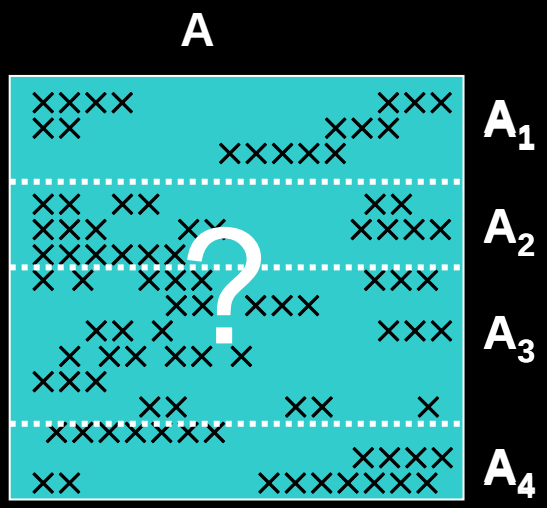
Sparse Matrix/vector partitioning

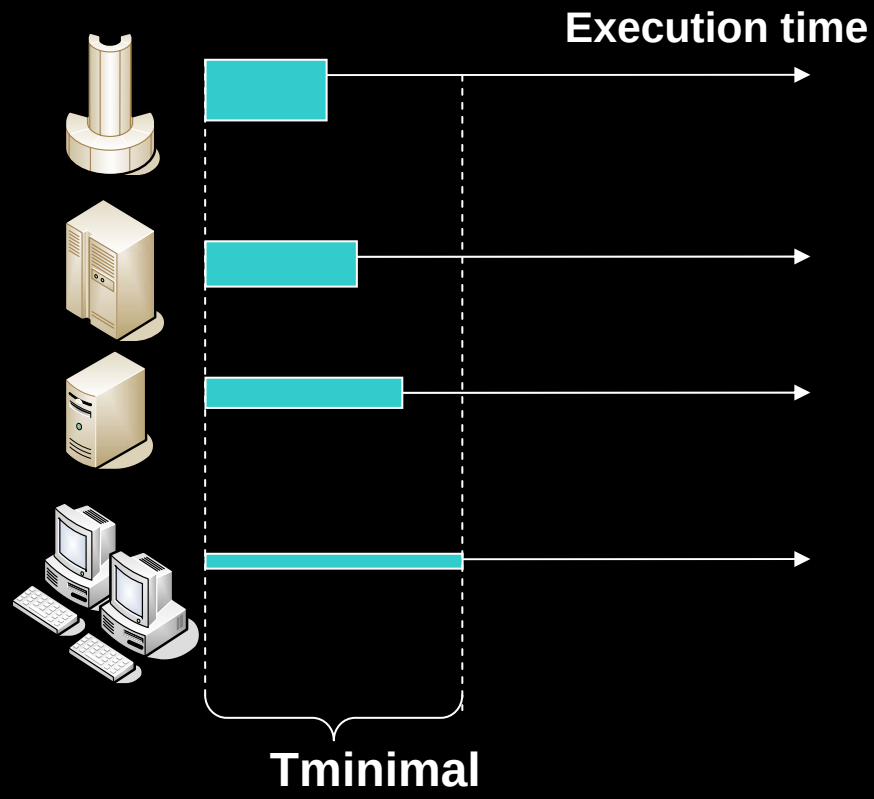


Sparse matrix is partitioned according to the resource power

Distribution model

Find a matrix partitioning which minimizes the total execution time?





Balancing Heuristic

Objectives:

- Load balancing: $$W_i \approx \frac{kp_i}{p} W_{avg}(1+\varepsilon) = \frac{p_i}{p}(1+\varepsilon)\sum_i W_i$$
- Fast : linear computing time
- Efficient : $\varepsilon << 1$

Agenda

- Overview of POP-C++
- Sparse Matrix/Vector product
- Programming in Pop-C++
- Experimental results
- Future work

The parallel class SparseMatrix

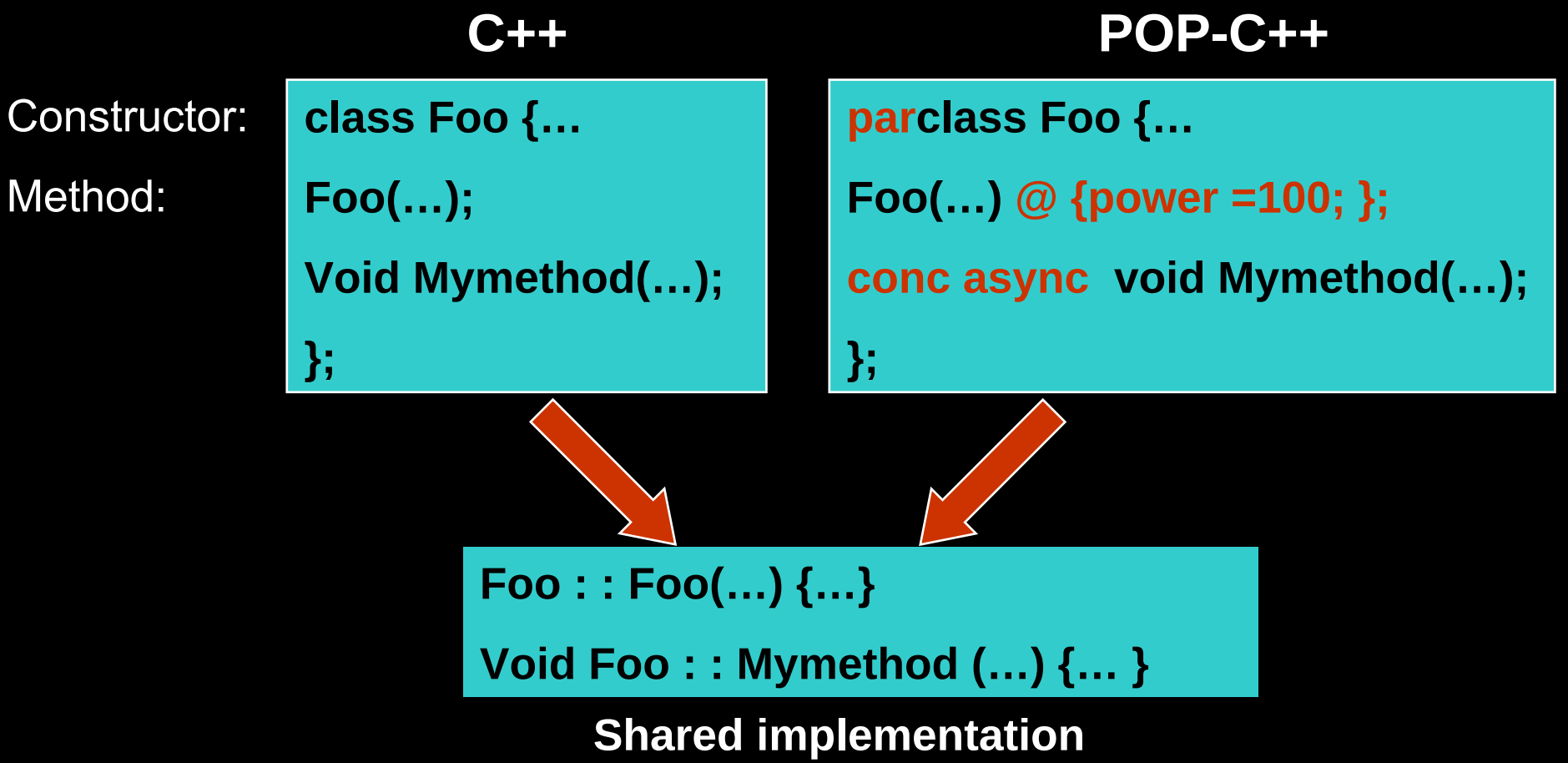
```
parclass SparseMatrix{

public :
SparseMatrix(int wanted, int min)@od.power(wanted, min) ;
seq async void Init( [in, size=n+1] double *rom_ptr, int n, …) ;
seq async void MvMultiply( [in, size=n] double *vector, int n) ;
mutex sync int GetResult( [out, size=m] double *V, int m) ;

private :
double *mat_val , *vect_res;
int *col_ind, *row_ptr;
…
}
```

The object requirements are defined by the constructor

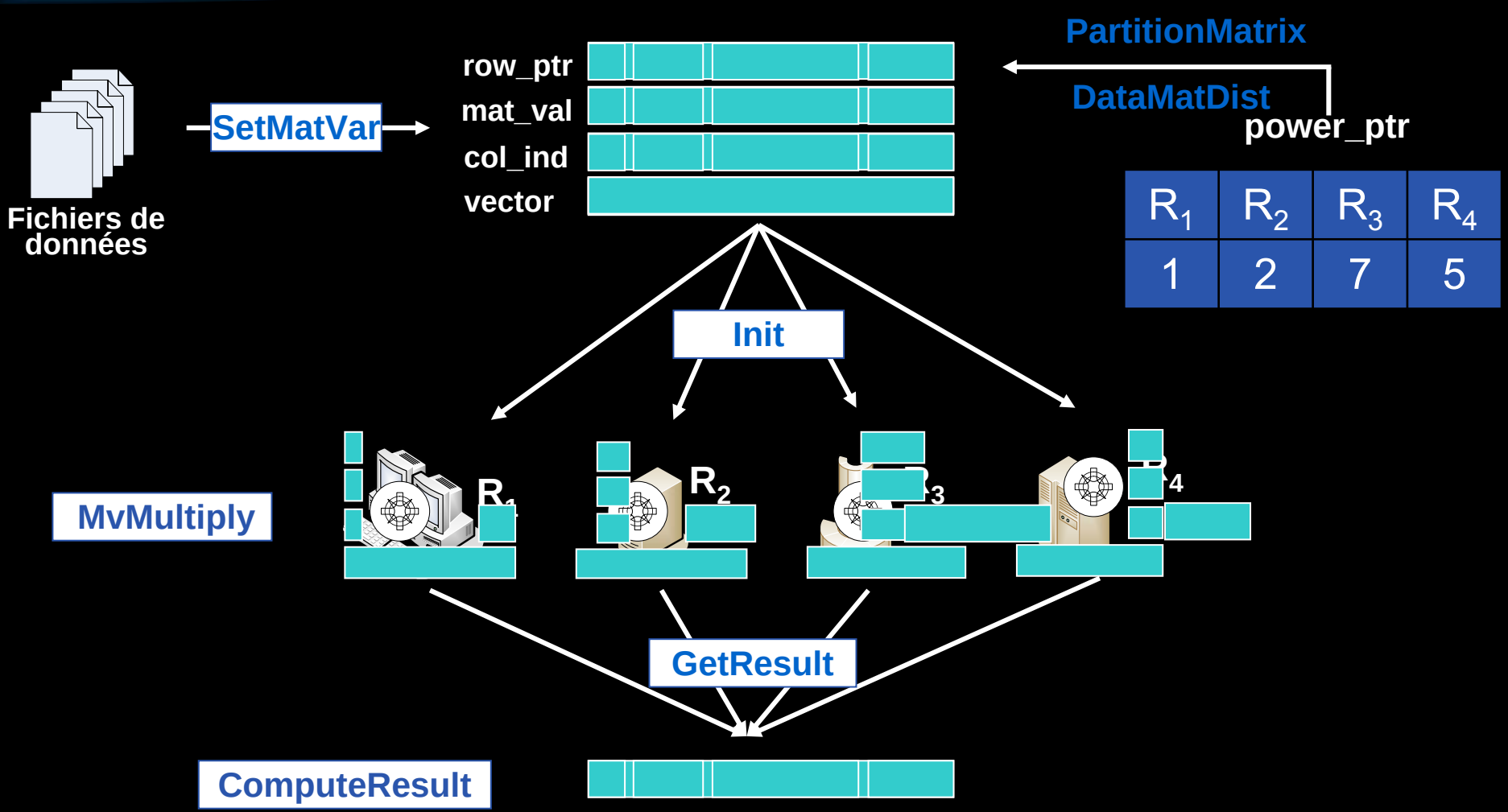
Minimal extension of C++



Methods are implemented in C++

```
…
void SparseMatrix : :MvMultiply ( double *vector, int n) {
        for (int i = 0 ; i < n ; i++){
        vect res[i] = 0.0 ;
        for (int j=row ptr[i] ; j<row ptr[i+1] ; j++)
                vect res[i] += mat val[j] * vector[col ind[j]] ;
        }
}
…
```

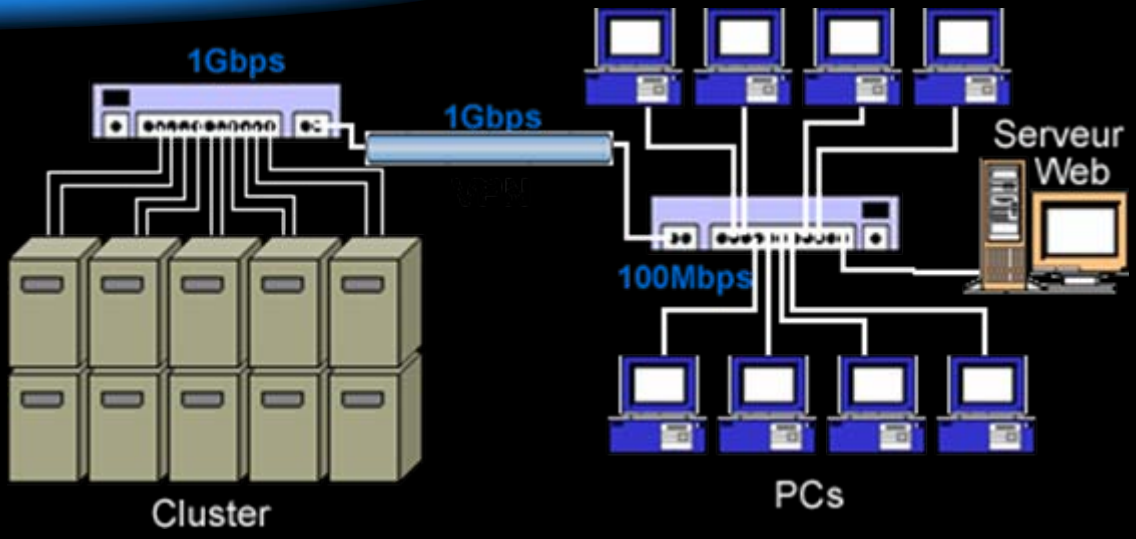
Execution steps



Agenda

- Overview of POP-C++
- Sparse Matrix/Vector product
- Programming in Pop-C++
- Experimental results
- Future work

Experimental Platform



Cluster properties

- Cluster Sun Fire V20
- 10 bi-opteron nodes
- 1.8 Ghz
- 1Gb of Ram
- GigaBit Ethernet

PCs properties

- AMD Athelon
- 2 Ghz
- 256Mb of Ram
- Fast Ethernet

Test matrices

Nom matrice	Domaine d'application	Taille(n)	NZ(m)
(a) fidap	Finite element modeling	16614	1091362
(b) poisson3Db	Finite element modeling	85623	2374949
(c) Stanford-web	Web crawling	281903	2382912
(d) Stanford-w.b.	Web crawling	685230	8006115

Matrix Markets Format

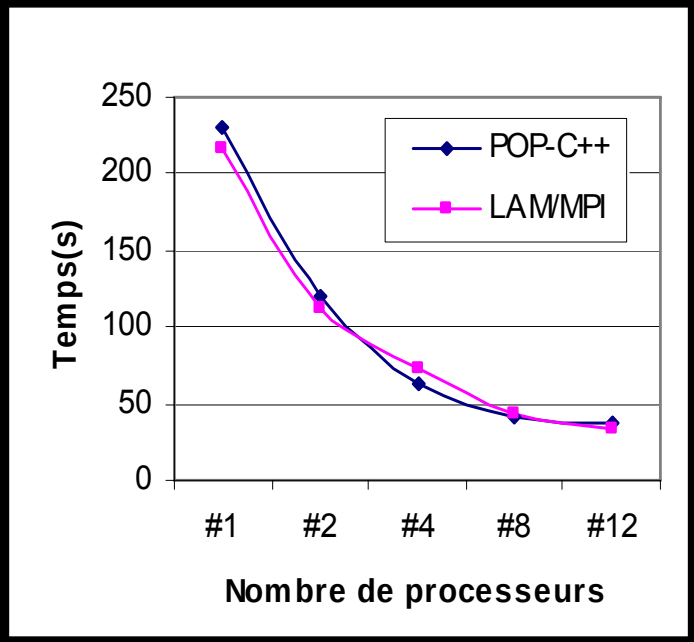
<i>	<j>	<Aij>

Experimental results

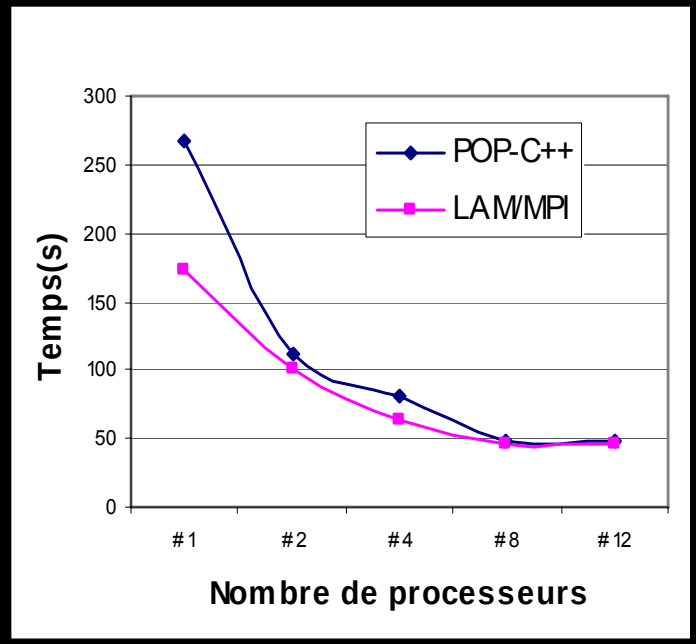
# Proc. / Matrice	Type	#1	#2	#4	#8	#12
(b)	POP-C++	108.2	62.8	31.4	22.9	22.7
	LAM/MPI	96.5	52.6	39.2	20.7	16.9
(c)	POP-C++	230.3	120.0	63.3	41.4	36.4
	LAM/MPI	215.6	111.6	73.8	43.2	33.6
(d)	POP-C++	267.7	112.4	80.5	49.2	48.4
	LAM/MPI	173.5	101.3	64.5	46.2	46.8

Total execution time for 1000 iterations

Experimental results



Matrice (b)



Matrice (d)

Agenda

- Overview of POP-C++
- Sparse Matrix/Vector product
- Programming in Pop-C++
- Experimental results
- Future work

Future work

- **Improve the performance by coupling POP-C++ with MPI**
- **Setting up a Scheduler for tasks assignment**
- **Implement iterative methods in grid environment based on heuristic for load balancing**
- **Evaluate POP-C++ performance in Globus environment**