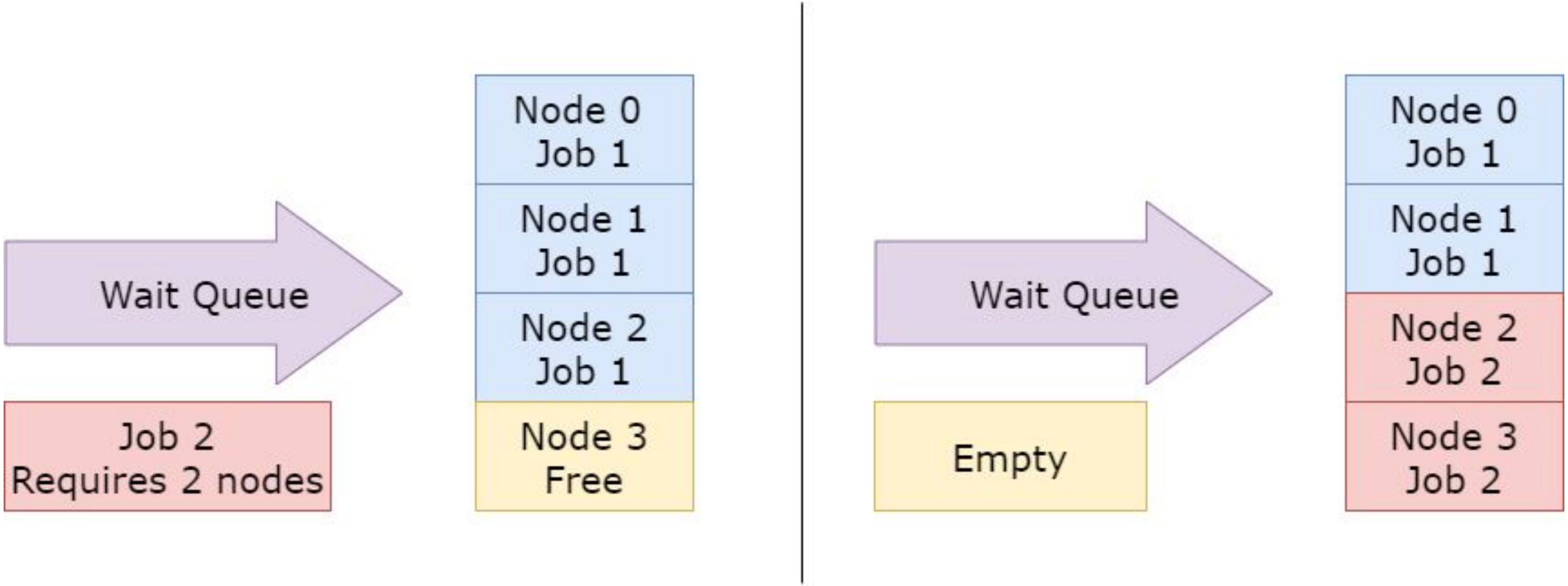
A synthetic tool for analysing adaptive workloads

Authors:
Iker Martín-Álvarez
José I. Aliaga
María Isabel Castillo
Sergio Iserte
Rafael Mayo

Outline:

- Motivation
- Synthetic application
  - Computation module
  - Reconfiguration module
  - Methric gatherer module
  - Configuration file parameters
- Results
  - Total execution times
  - Malleability times
  - Iteration times
- Conclusions
- Future work



Wait Queue

Job 2
Requires 2 nodes

Node 0
Job 1

Node 1
Job 1

Node 2
Job 1

Node 3
Free

Wait Queue

Empty

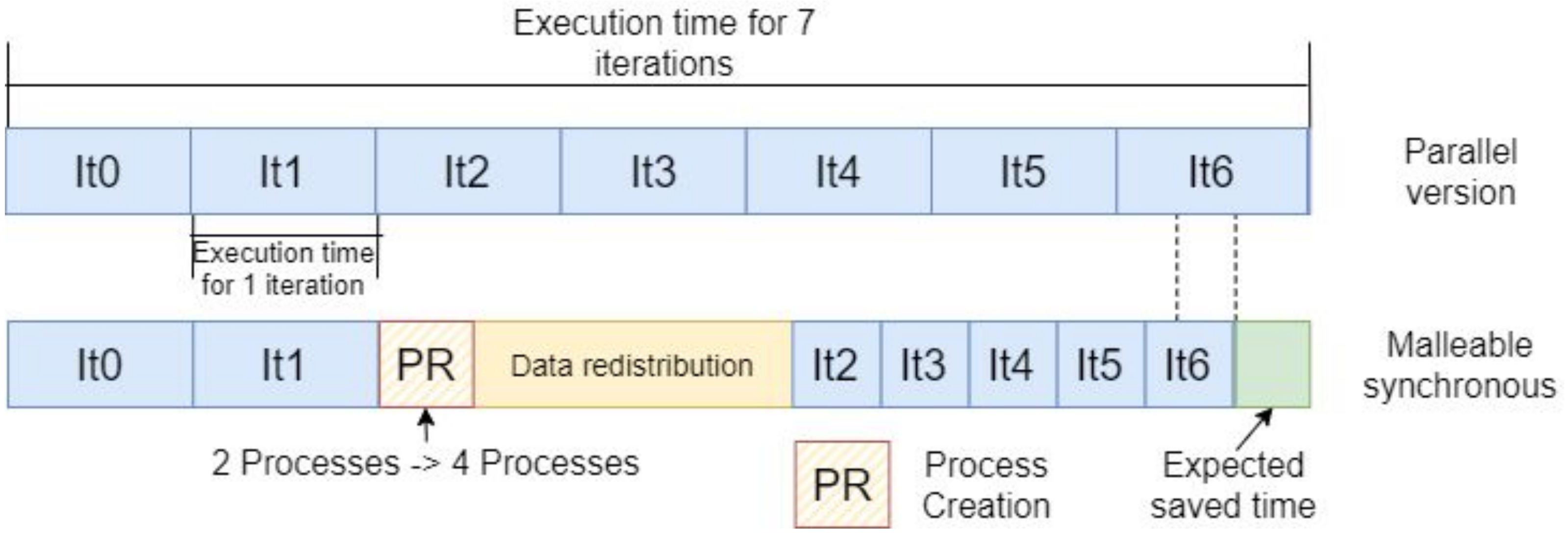
Node 0
Job 1

Node 1
Job 1

Node 2
Job 2

Node 3
Job 2

Motivation (I)



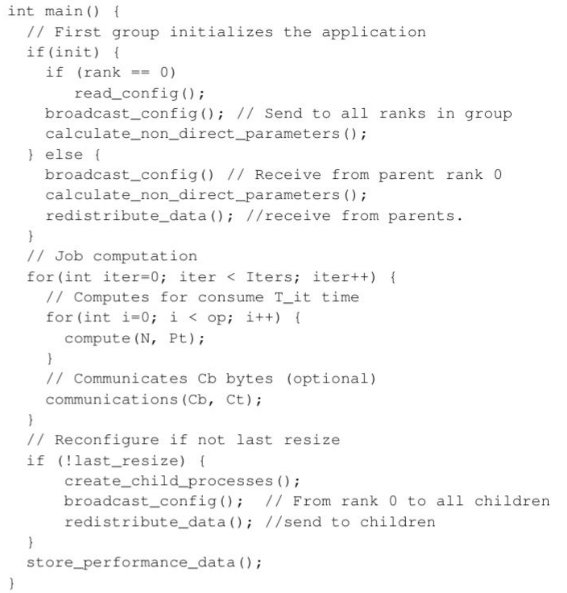
Motivation (II)

Synthetic application

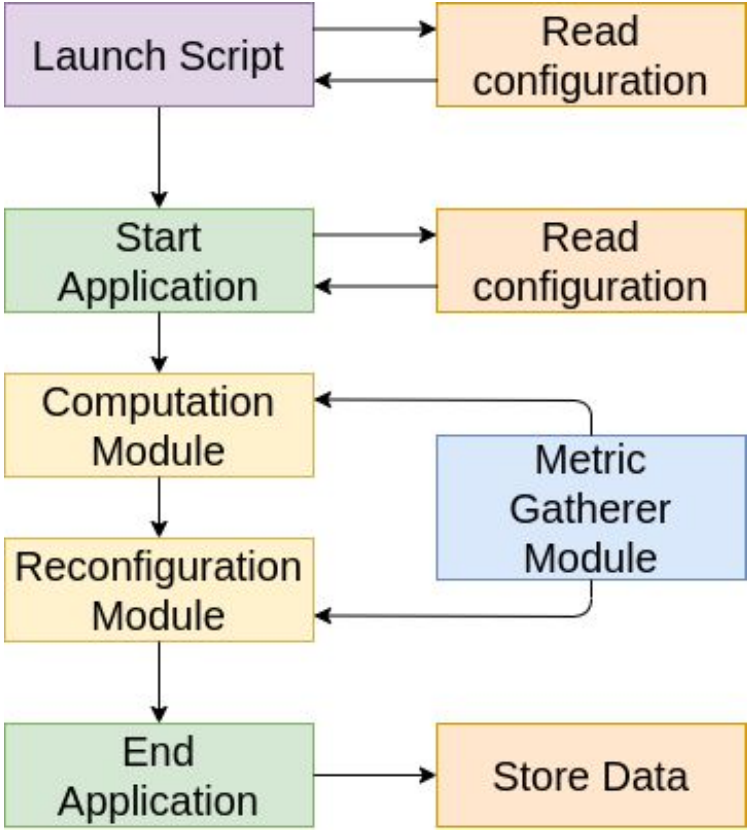
Application composed of three modules:

- Computation module
- Reconfiguration module
- Metric gatherer module

and a configuration file



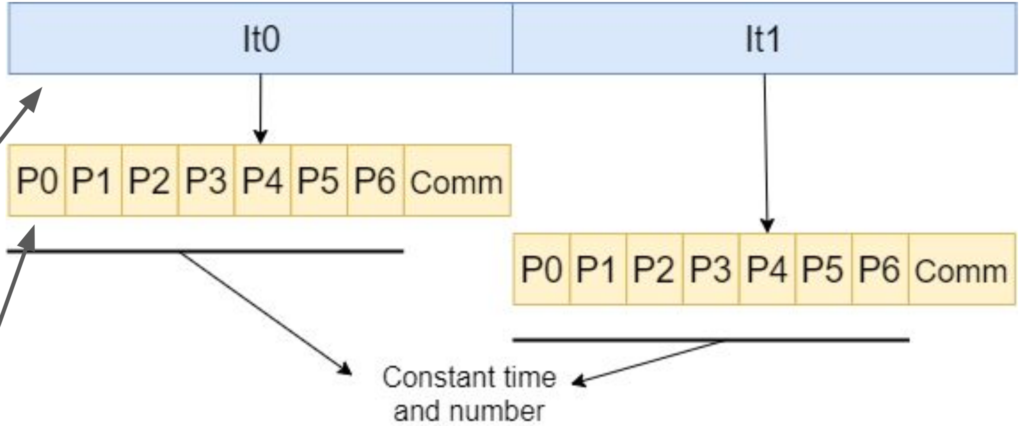
```
int main() {
  // First group initializes the application
  if(init) {
    if (rank == 0)
       read_config();
    broadcast_config(); // Send to all ranks in group
    calculate_non_direct_parameters();
  } else {
    broadcast_config() // Receive from parent rank 0
    calculate_non_direct_parameters();
    redistribute_data(); //receive from parents.
  }
  // Job computation
  for(int iter=0; iter < Iters; iter++) {
    // Computes for consume T_it time
    for(int i=0; i < op; i++) {
      compute(N, Pt);
    }
    // Communicates Cb bytes (optional)
    communications(Cb, Ct);
  }
  // Reconfigure if not last resize
  if (!last_resize) {
      create_child_processes();
      broadcast_config();  // From rank 0 to all children
      redistribute_data(); //send to children
  }
  store_performance_data();
}
```



Synthetic application

Computation module

Computation module - Computation procedures



Iteration time (T_it)

Procedure time (T_op)

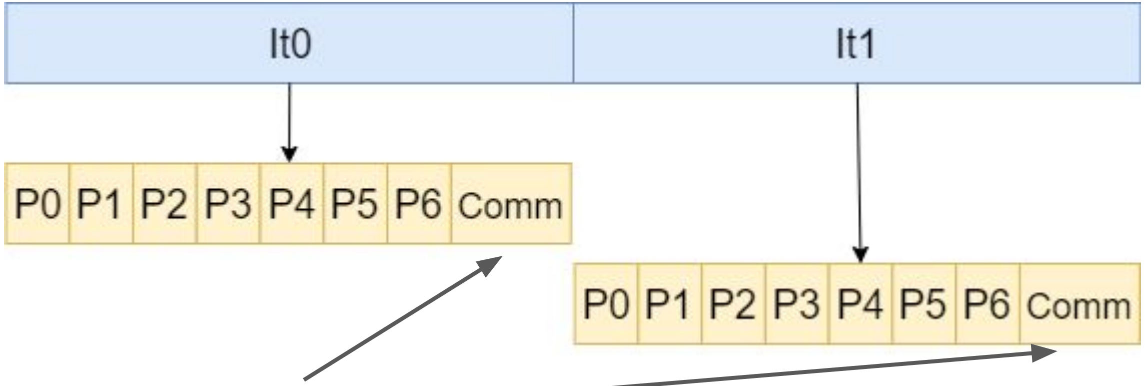
Procedures per iteration (T_it)/(T_op)

Computation procedures:

- Compute-bound: Montecarlo Pi estimation
- Memory-bound: Matrix-vector multiplication
- Process number does not affect procedure final time (T_op)

Communication procedures:

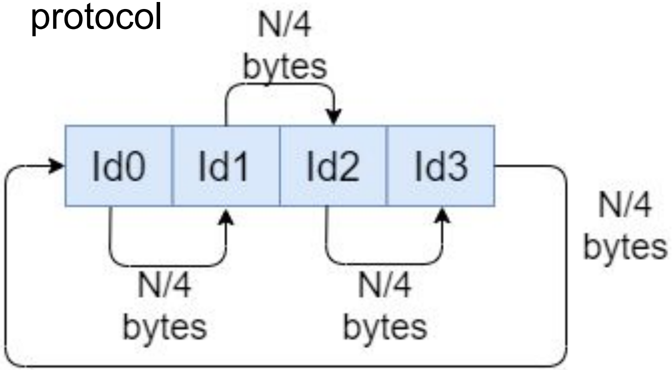
- Point to point (MPI_Send/Recv)
- Collective one-to-all (MPI_Bcast)
- Collective all-to-all (MPI_Alltoall)
- Reduction (MPI_Reduce)



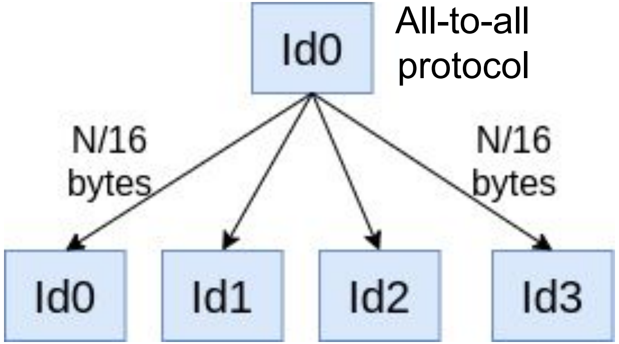
Communications cost:

- *N* bytes

Point to point protocol

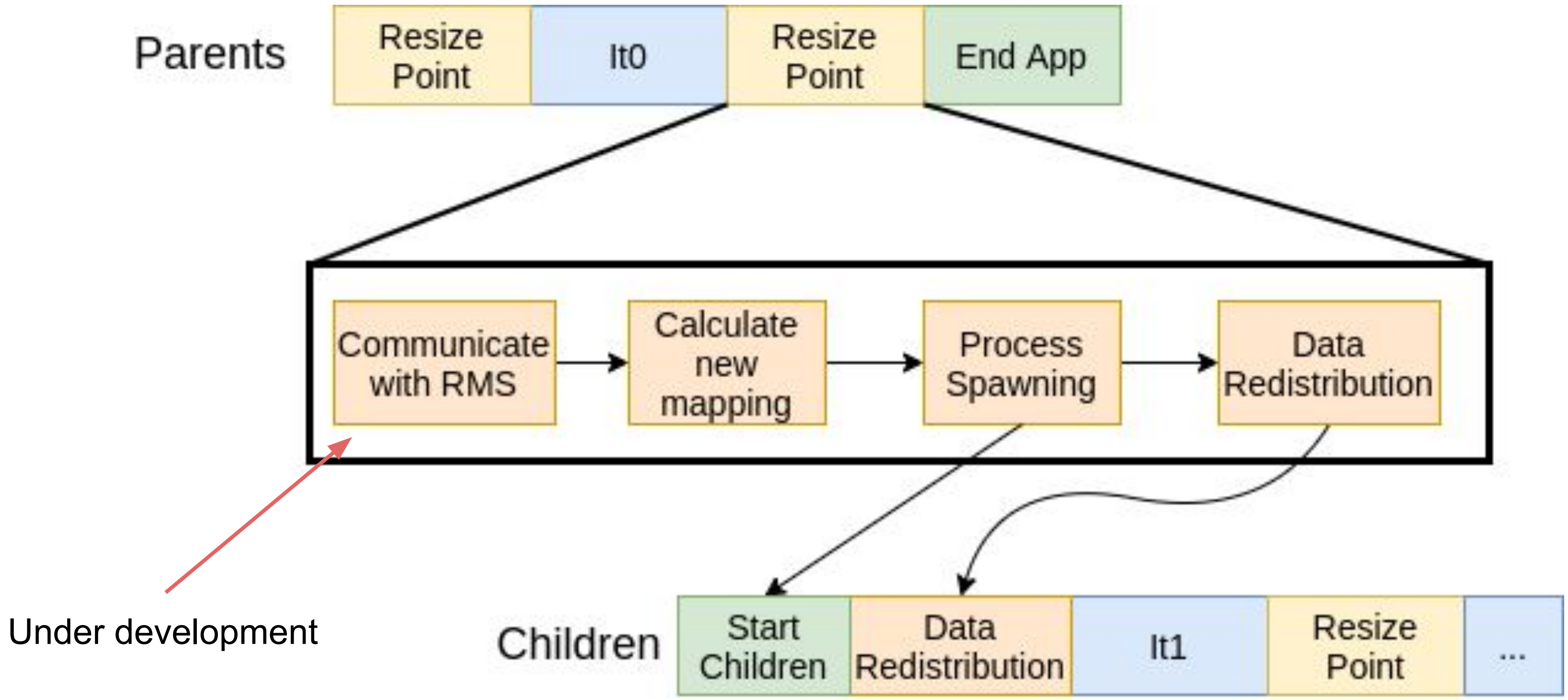


All-to-all protocol



Computation module - Communication procedures

Reconfiguration module



Parents

Resize Point

It0

Resize Point

End App

Communicate with RMS

Calculate new mapping

Process Spawning

Data Redistribution

Under development

Children

Start Children

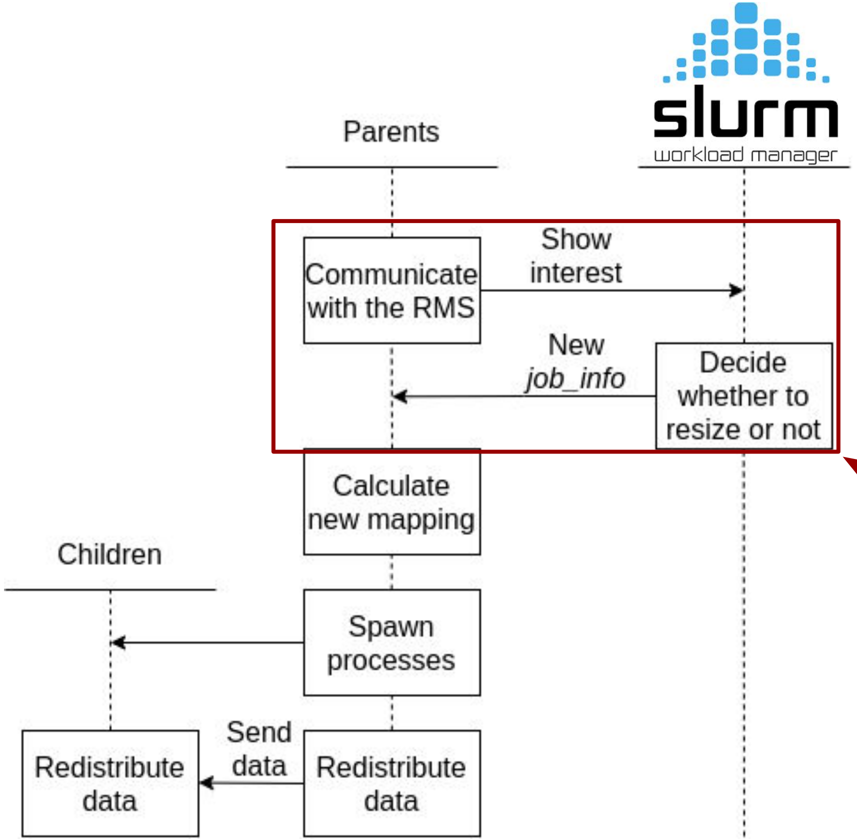
Data Redistribution

It1

Resize Point

...

Reconfiguration module - Steps (I)

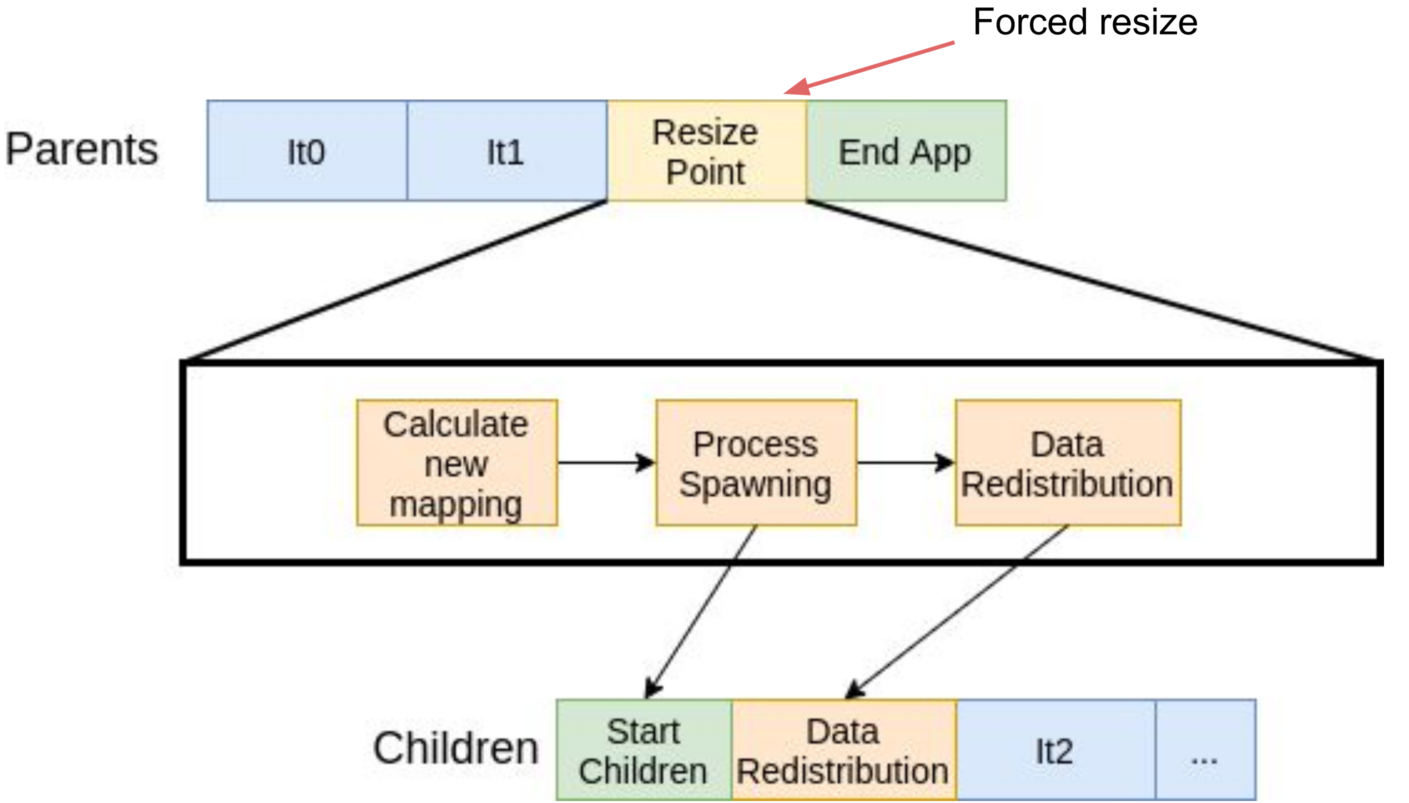


job_info:

- Node number
- Core number
- Node names

Under development

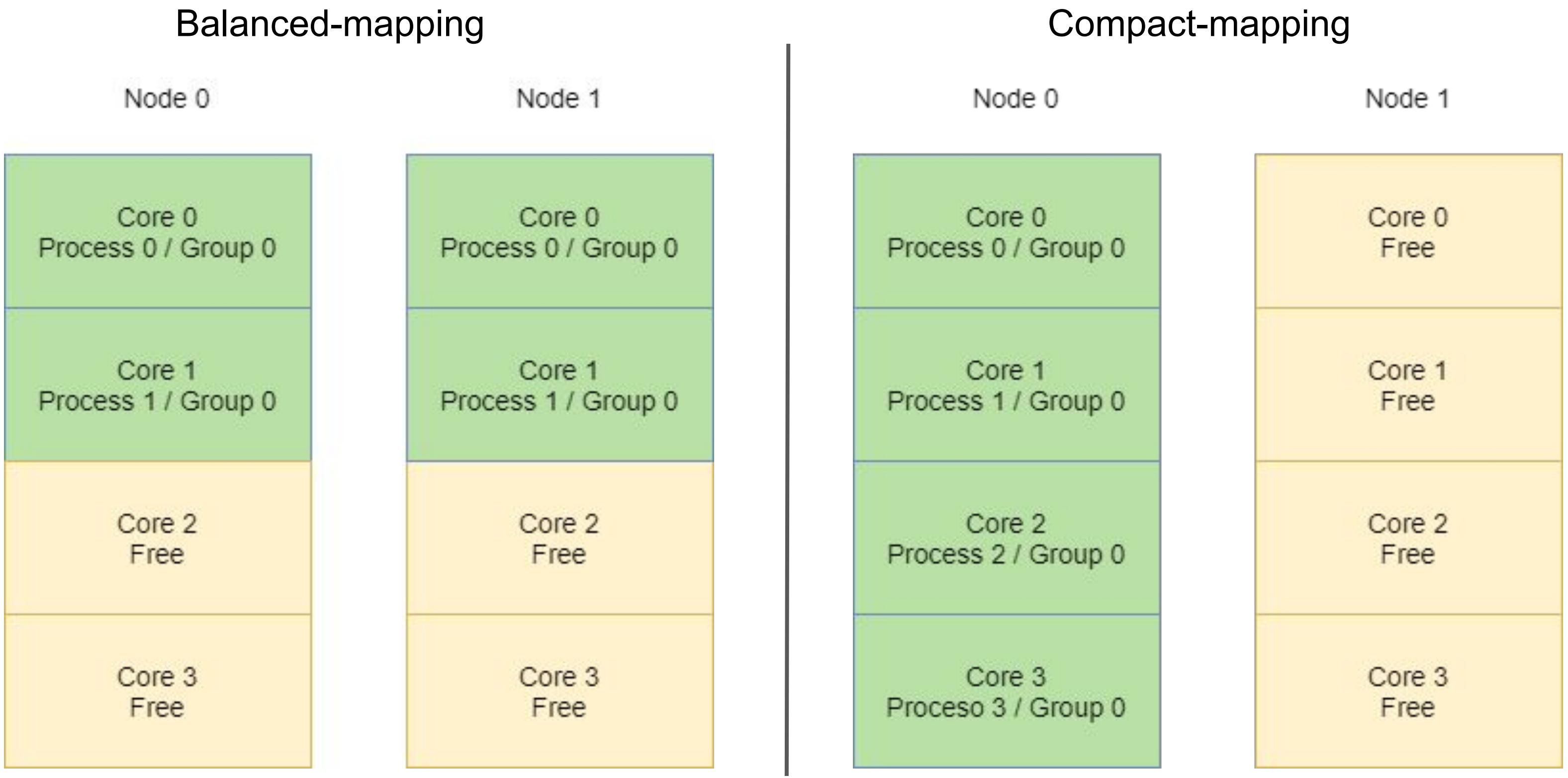
Reconfiguration module - Steps (II)



UNIVERSITAT JAUME I

Reconfiguration module - Steps (III)

Reconfiguration module - Physical mappings

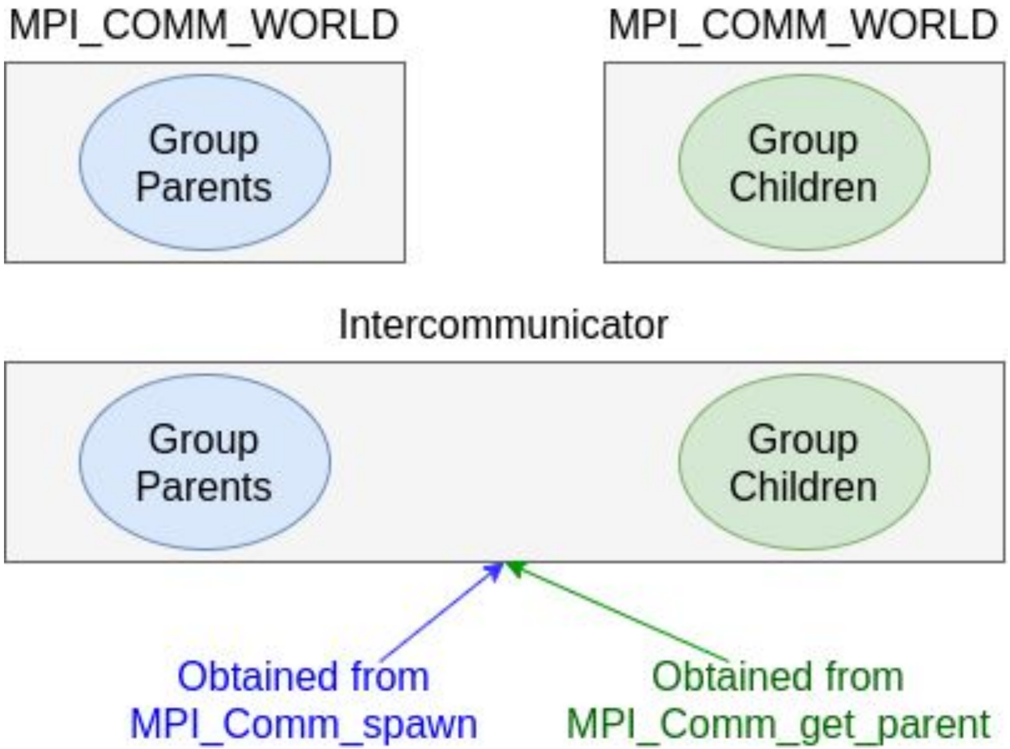


Balanced-mapping

Node 0	Node 1
Core 0 Process 0 / Group 0	Core 0 Process 0 / Group 0
Core 1 Process 1 / Group 0	Core 1 Process 1 / Group 0
Core 2 Free	Core 2 Free
Core 3 Free	Core 3 Free

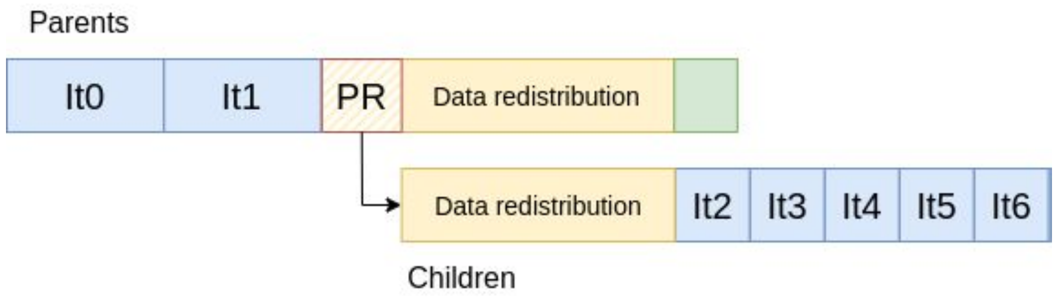
Compact-mapping

Node 0	Node 1
Core 0 Process 0 / Group 0	Core 0 Free
Core 1 Process 1 / Group 0	Core 1 Free
Core 2 Process 2 / Group 0	Core 2 Free
Core 3 Proceso 3 / Group 0	Core 3 Free

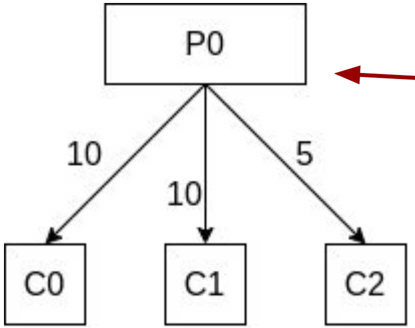


Reconfiguration module - Process spawn

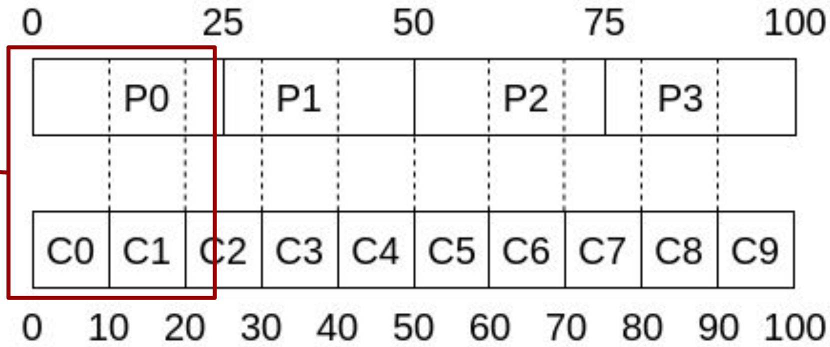
Data redistribution - Basics



Based on MPI_Alltoallv



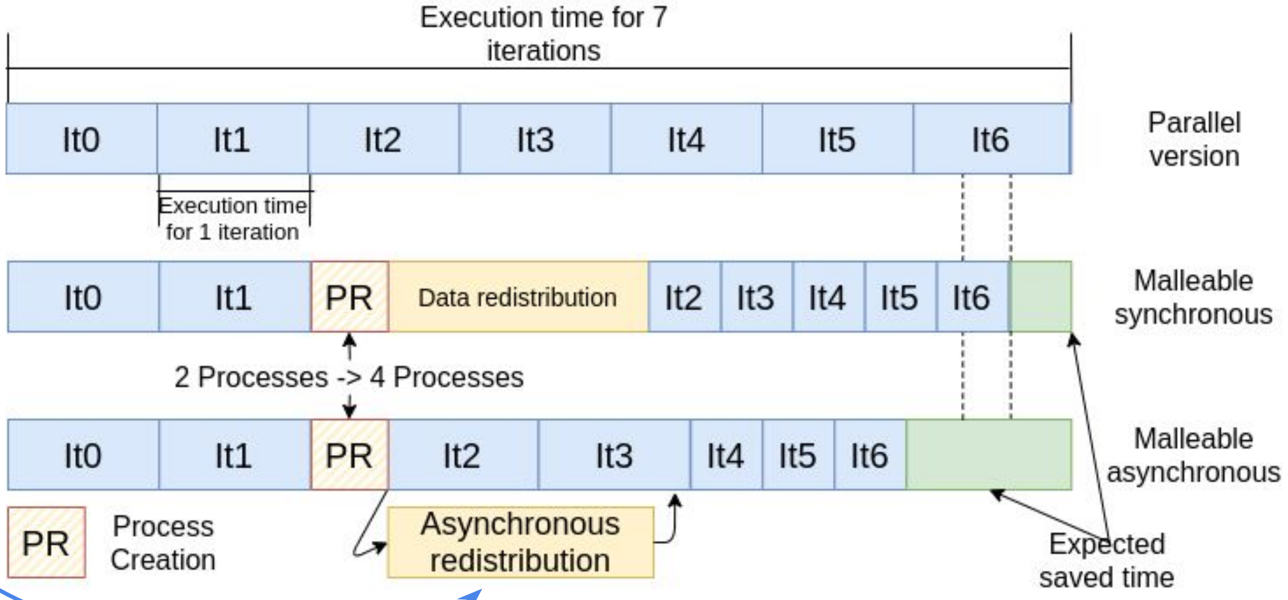
Distribution of elements from parent P0 to its children C0, C1 and C2



Distribution from 4 to 10 processes of 100 elements

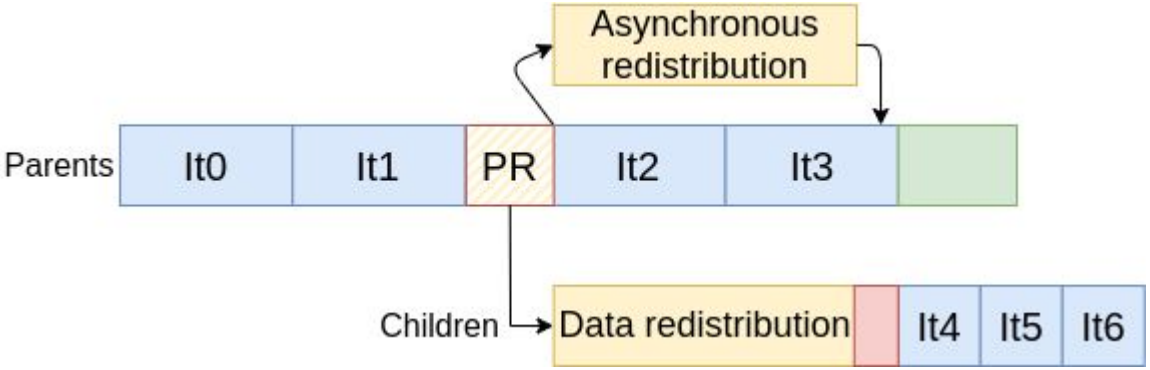
Data redistribution - Asynchronous (I)

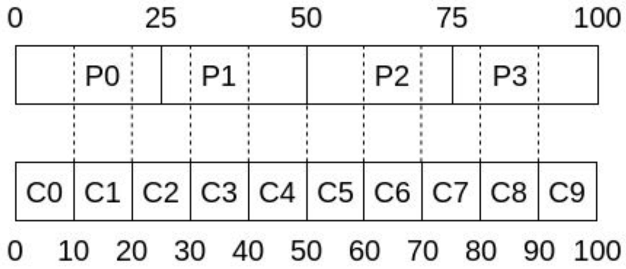
Additional contribution: Allow asynchronous data redistribution



SR = Synchronous redistribution

AR = Asynchronous redistribution



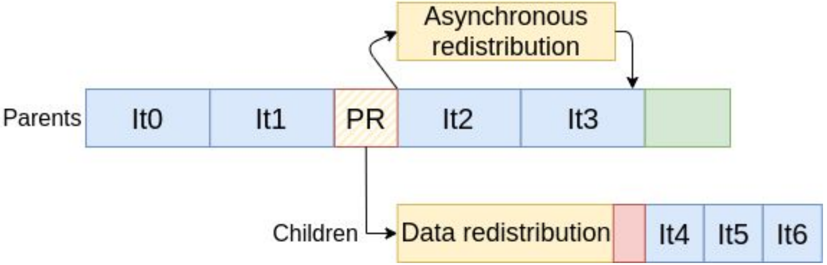


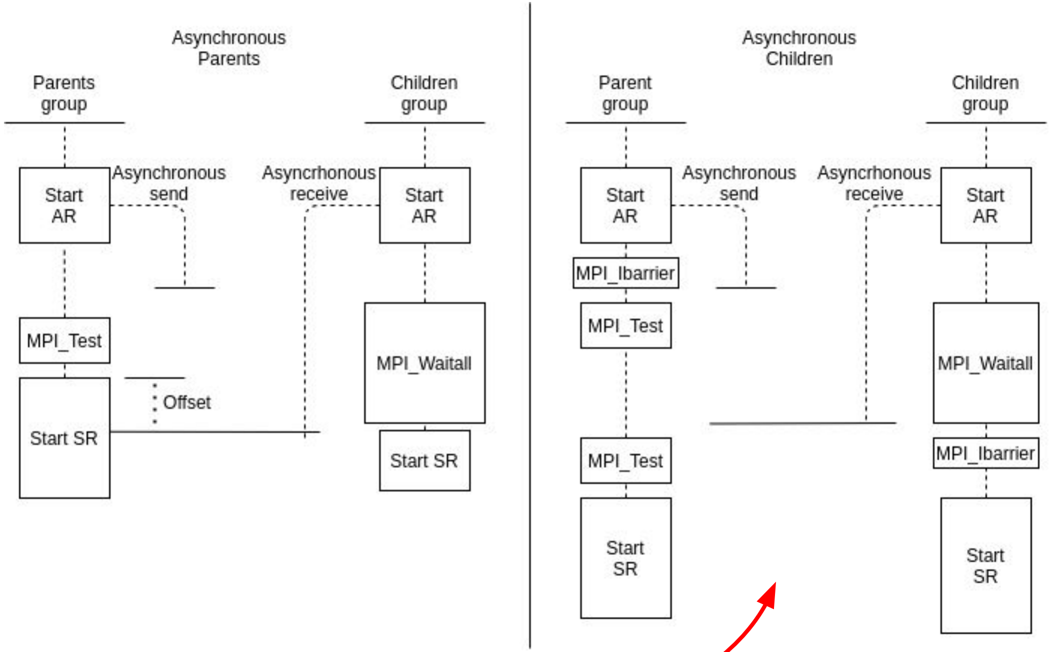
Asynchronous procedures:

- Point to point (MPI_Isend/Irecv)
- Collective all-to-all Parents (MPI_Ialltoallv)
- Collective all-to-all Children (MPI_Ialltoallv)
- Pthreads (MPI_Alltoallv)

MPI Non-blocking Primitives

Data redistribution - Asynchronous (II)

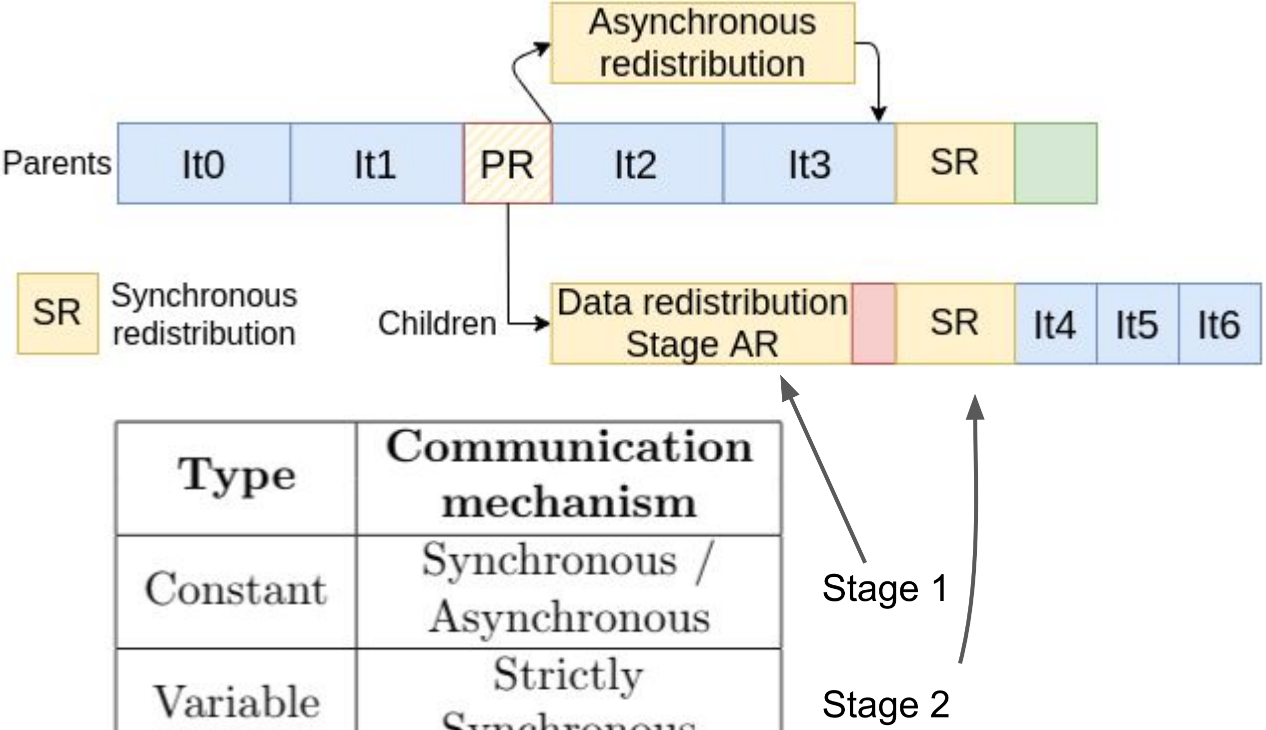




Asynchronous procedures:

- Point to point (MPI_Isend/Irecv)
- Collective all-to-all Parents (MPI_Ialltoallv)
- Collective all-to-all Children (MPI_Ialltoallv)
- Pthreads (MPI_Alltoallv)

Data redistribution - Asynchronous (II.1)

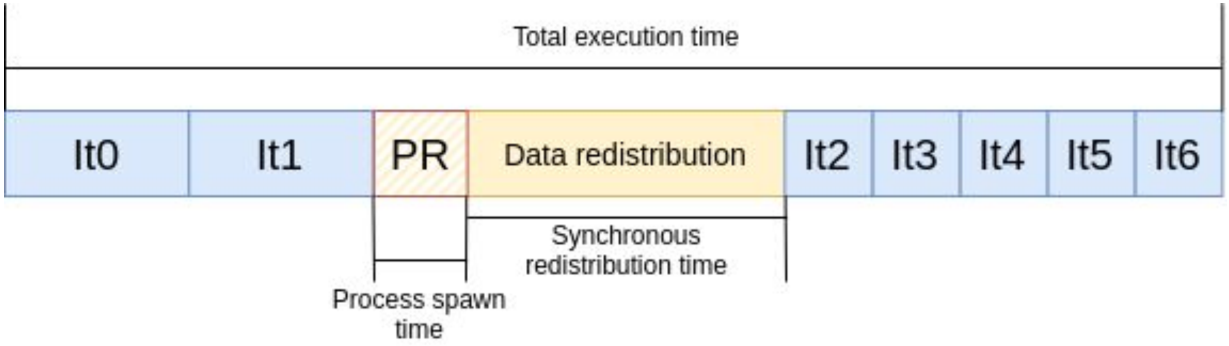

Asynchronous redistribution
Parents
It0
It1
PR
It2
It3
SR
SR
Synchronous redistribution
Children
Data redistribution Stage AR
SR
It4
It5
It6
Stage 1
Stage 2

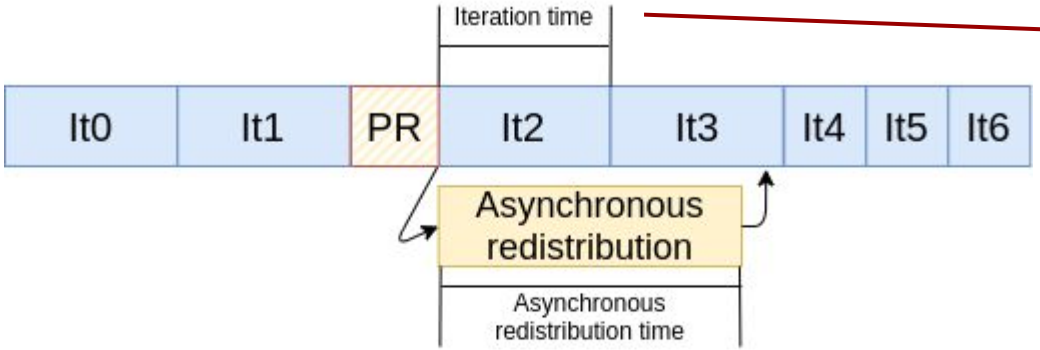
Type	Communication mechanism
Constant	Synchronous / Asynchronous
Variable	Strictly Synchronous

Data redistribution - Asynchronous (III)

Metric gatherer module

Based on MPI_Wtime





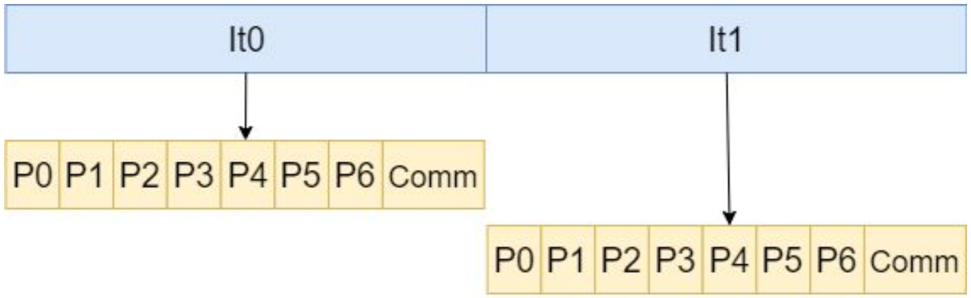
Also stored with each iteration:

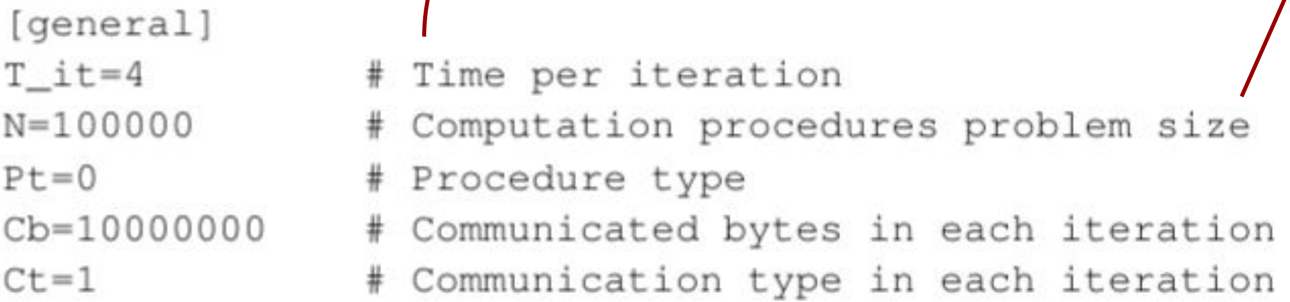
- Number of *Op* executed
- If AR is being performed or not

Metric gatherer module

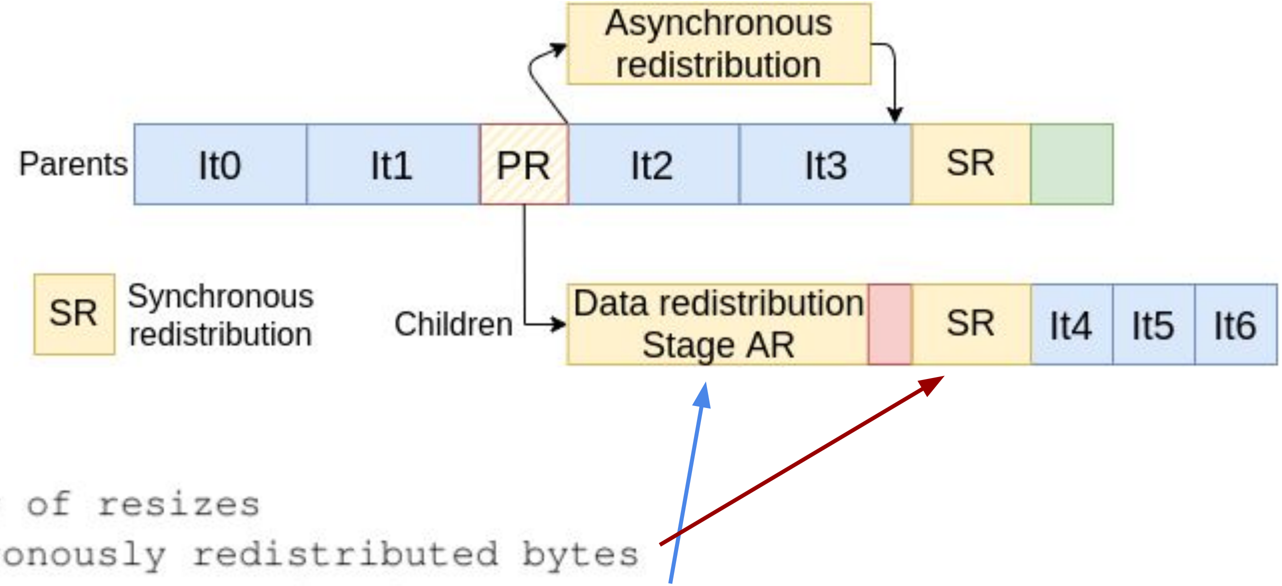
Configuration files

Configuration file - General



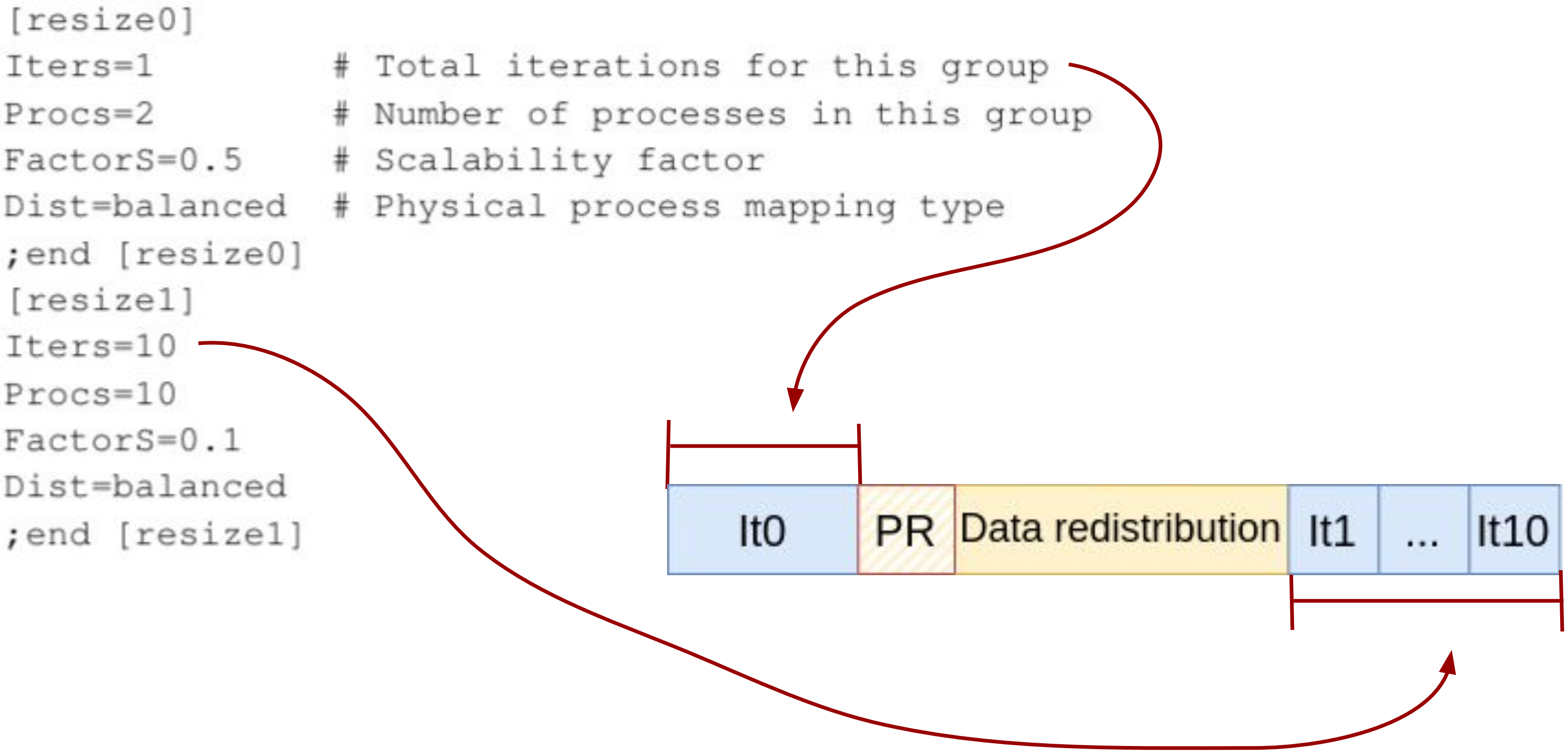


```
[general]
T_it=4              # Time per iteration
N=100000            # Computation procedures problem size
Pt=0                # Procedure type
Cb=10000000         # Communicated bytes in each iteration
Ct=1                # Communication type in each iteration
```


Asynchronous redistribution
Parents
It0
It1
PR
It2
It3
SR
SR
Synchronous redistribution
Children
Data redistribution Stage AR
SR
It4
It5
It6

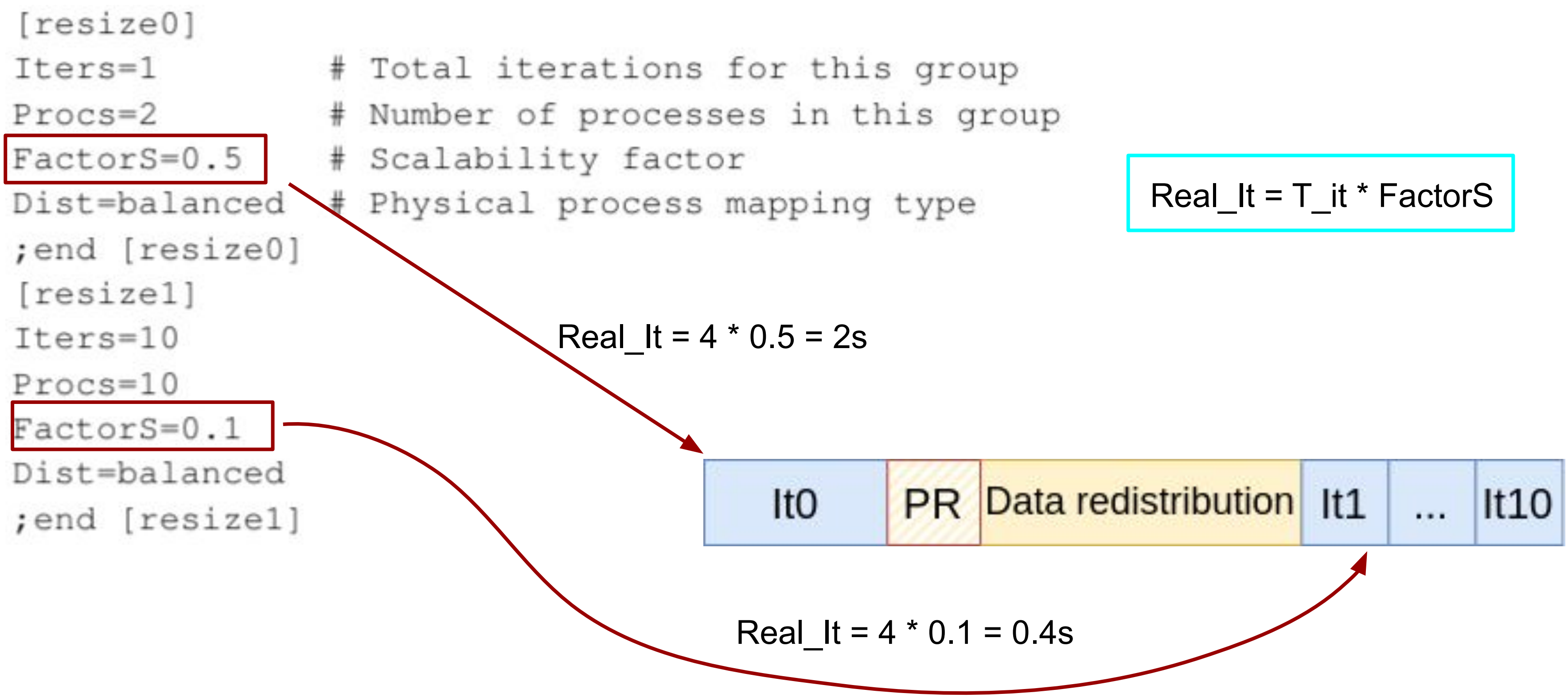
```
R=1            # Number of resizes
SDR=1000000000 # Synchronously redistributed bytes
ADR=1000000000 # Asynchronously redistributed bytes
AT=3           # Type of asynchronous redistribution
;end [general]
```

Configuration file - Malleability



```
[resize0]
Iters=1          # Total iterations for this group
Procs=2          # Number of processes in this group
FactorS=0.5      # Scalability factor
Dist=balanced    # Physical process mapping type
;end [resize0]
[resize1]
Iters=10
Procs=10
FactorS=0.1
Dist=balanced
;end [resize1]
```

Configuration file - Groups (I)



```
[resize0]
Iters=1       # Total iterations for this group
Procs=2       # Number of processes in this group
FactorS=0.5   # Scalability factor
Dist=balanced # Physical process mapping type
;end [resize0]
[resize1]
Iters=10
Procs=10
FactorS=0.1
Dist=balanced
;end [resize1]
```

Real_It = T_it * FactorS

Real_It = 4 * 0.5 = 2s

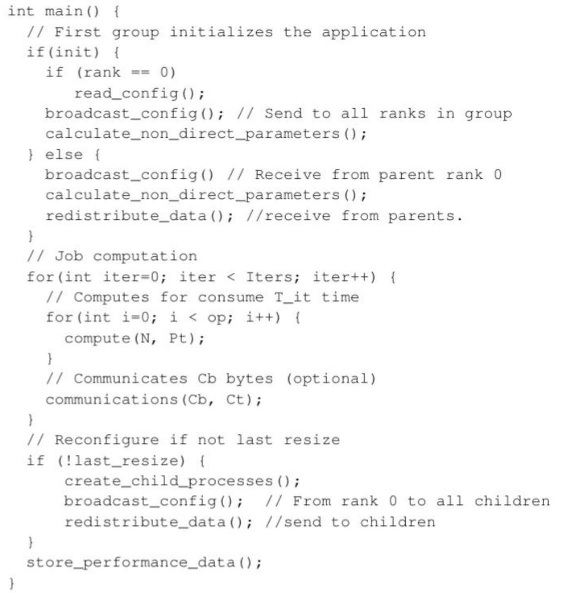
Real_It = 4 * 0.1 = 0.4s

Configuration file - Groups (II)

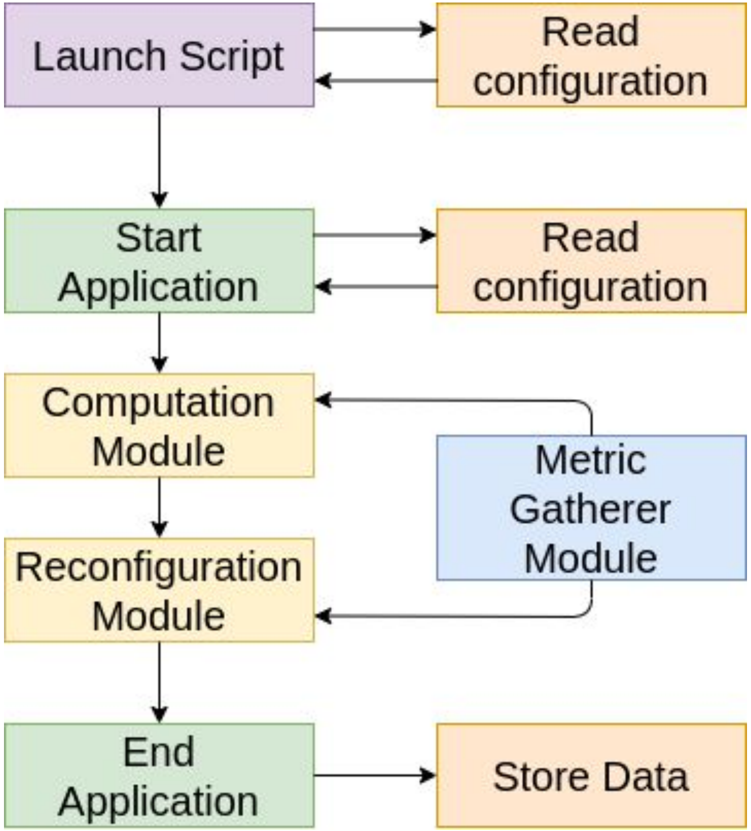
Application composed of three modules:

- Computation module
- Reconfiguration module
- Metric gatherer module

and a configuration file



```
int main() {
  // First group initializes the application
  if(init) {
    if (rank == 0)
       read_config();
    broadcast_config(); // Send to all ranks in group
    calculate_non_direct_parameters();
  } else {
    broadcast_config() // Receive from parent rank 0
    calculate_non_direct_parameters();
    redistribute_data(); //receive from parents.
  }
  // Job computation
  for(int iter=0; iter < Iters; iter++) {
    // Computes for consume T_it time
    for(int i=0; i < op; i++) {
      compute(N, Pt);
    }
    // Communicates Cb bytes (optional)
    communications(Cb, Ct);
  }
  // Reconfigure if not last resize
  if (!last_resize) {
      create_child_processes();
      broadcast_config();  // From rank 0 to all children
      redistribute_data(); //send to children
  }
  store_performance_data();
}
```

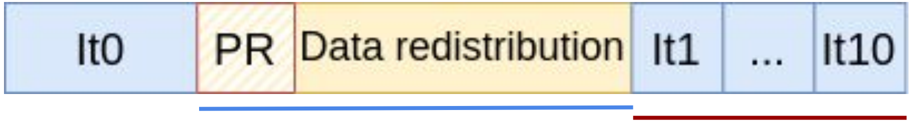


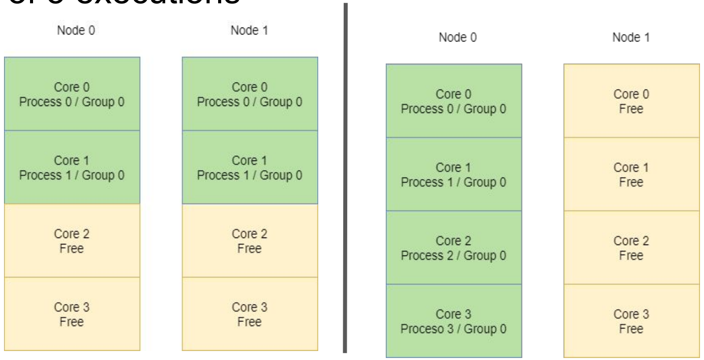
Synthetic application

Results

Results - Experiments

- Two 10-core processor Intel Xeon 4210
- Two servers (40 cores)
- 11 Iterations of 4s or 0.4s per execution
- 10MB iteration communication
- One reconfiguration per execution
- *FactorS* is applied to have a perfect SpeedUp
- Both physical mappings
- Fully SR or AR of 1GB
- Pthreads option
- Mean of 5 executions




Node 0

Core 0
Process 0 / Group 0

Core 1
Process 1 / Group 0

Core 2
Free

Core 3
Free

Node 1

Core 0
Process 0 / Group 0

Core 1
Process 1 / Group 0

Core 2
Free

Core 3
Free

Node 0

Core 0
Process 0 / Group 0

Core 1
Process 1 / Group 0

Core 2
Process 2 / Group 0

Core 3
Proceso 3 / Group 0

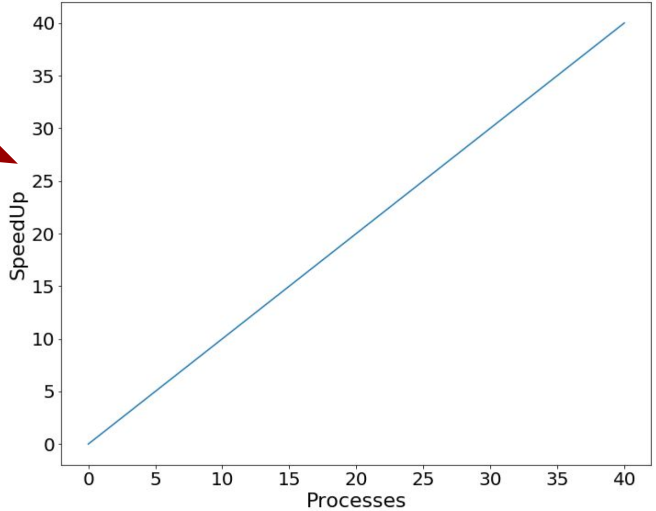
Node 1

Core 0
Free

Core 1
Free

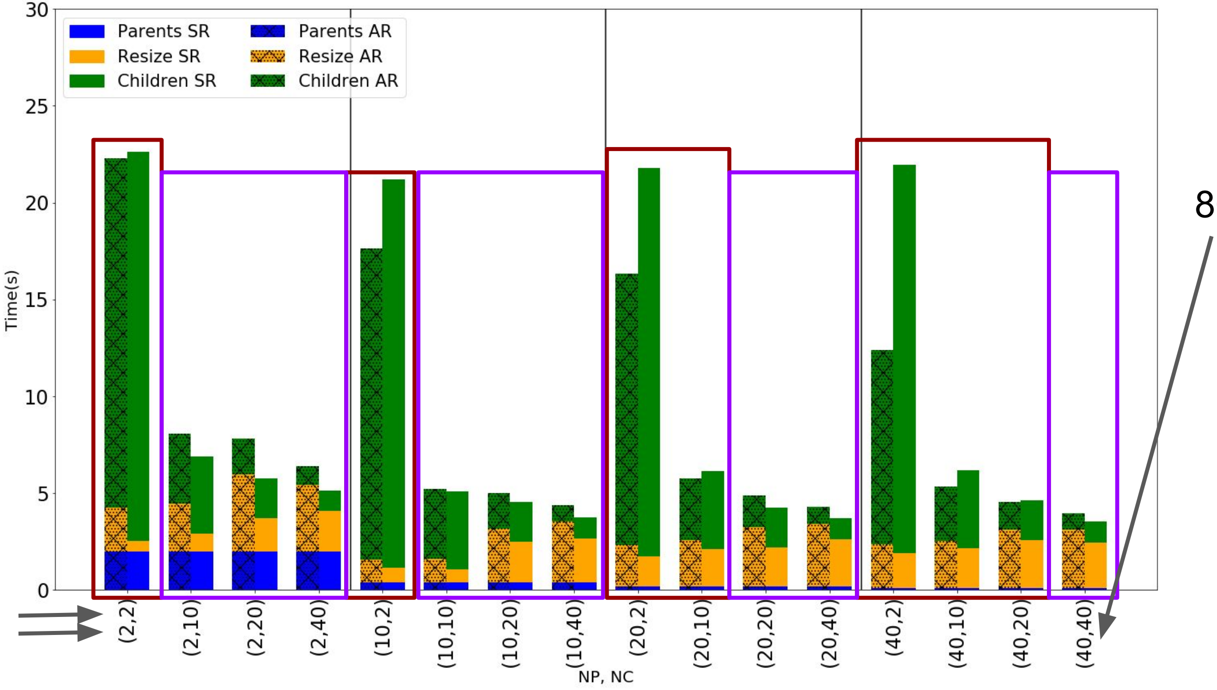
Core 2
Free

Core 3
Free



Compact mapping

T_it = 4 seconds

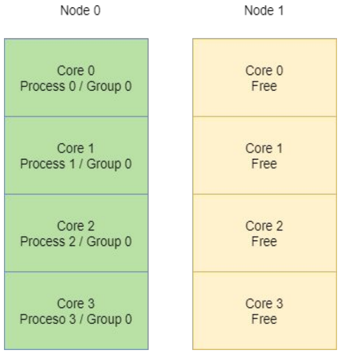


Shrinking with AR

Expanding with SR

80 processes, 120 threads in AR

Oversubscription when more than 20 threads in one node.

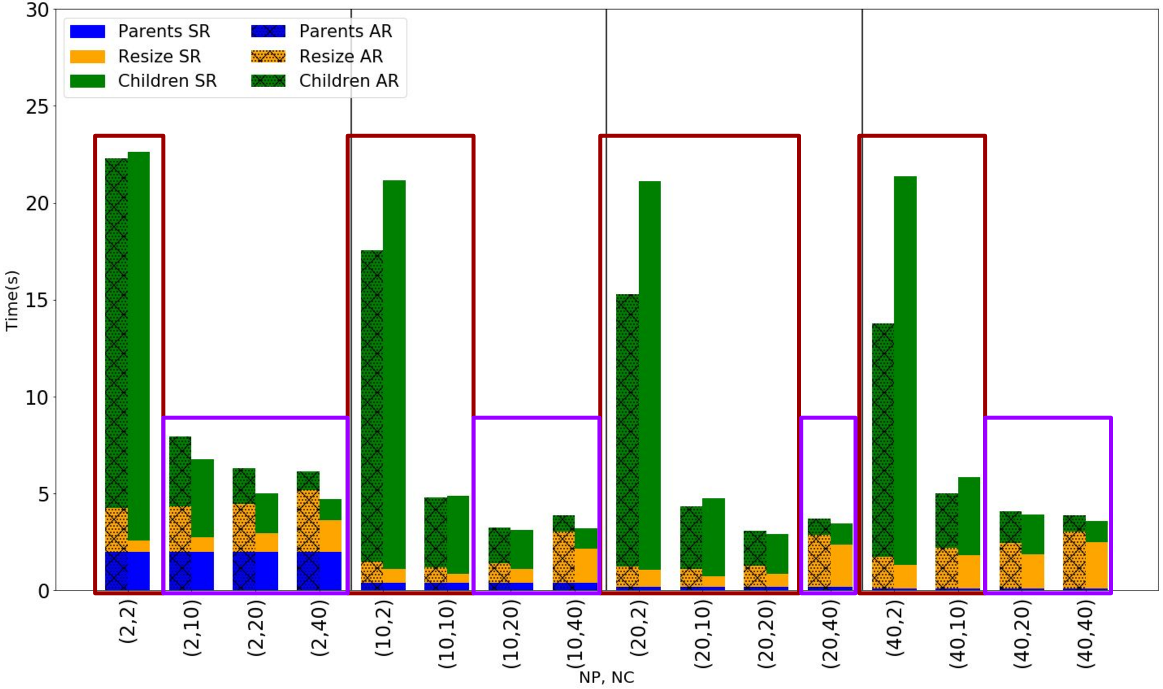


Node 0	Node 1
Core 0 Process 0 / Group 0	Core 0 Free
Core 1 Process 1 / Group 0	Core 1 Free
Core 2 Process 2 / Group 0	Core 2 Free
Core 3 Proceso 3 / Group 0	Core 3 Free

Results - Total execution times (I)

Balanced mapping

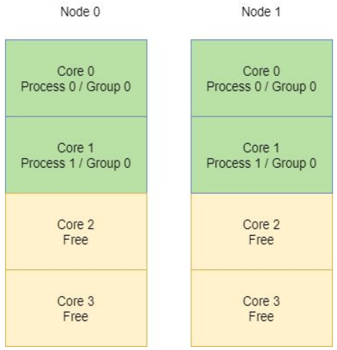
T_it = 4 seconds



Shrinking with AR

Expanding with SR

Oversubscription when more than 20 threads.

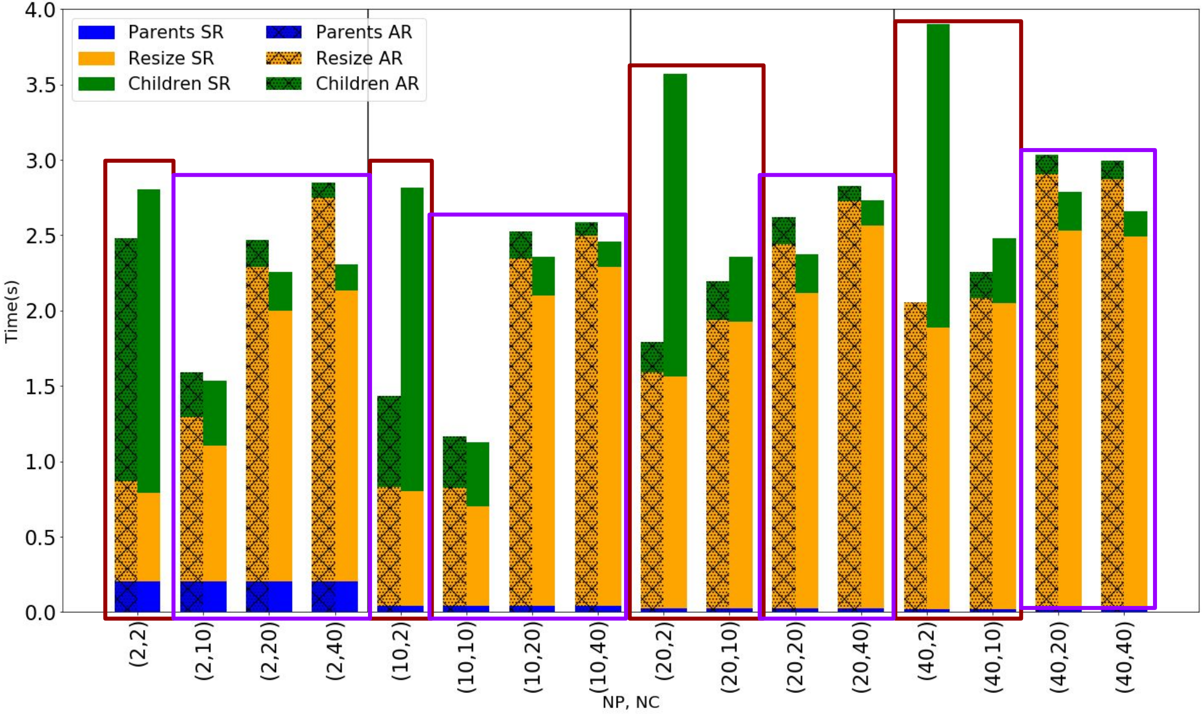


Node 0	Node 1
Core 0 Process 0 / Group 0	Core 0 Process 0 / Group 0
Core 1 Process 1 / Group 0	Core 1 Process 1 / Group 0
Core 2 Free	Core 2 Free
Core 3 Free	Core 3 Free

Results - Total execution times (II)

Compact mapping

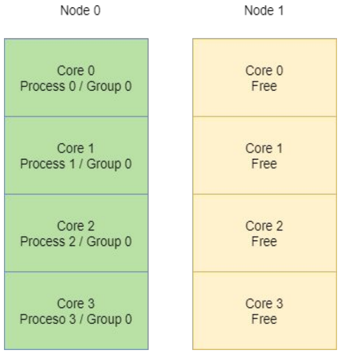
T_it = 0.4 seconds



Shrinking with AR

Expanding with SR

Oversubscription when more than 20 threads.

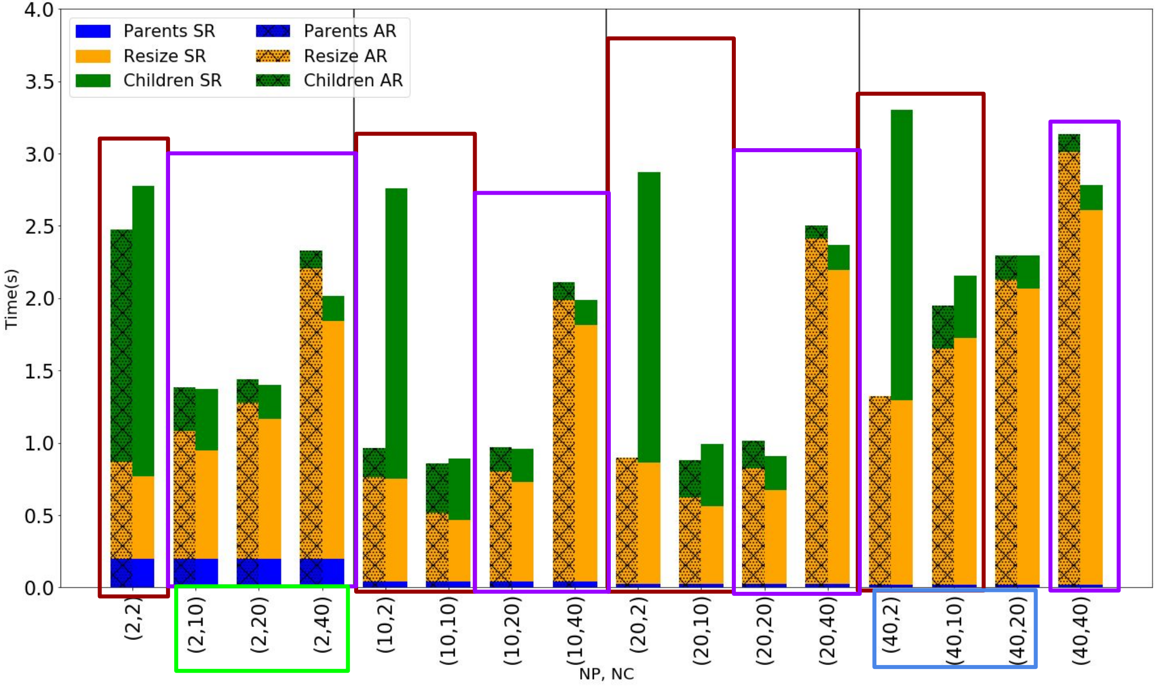


Node 0	Node 1
Core 0 Process 0 / Group 0	Core 0 Free
Core 1 Process 1 / Group 0	Core 1 Free
Core 2 Process 2 / Group 0	Core 2 Free
Core 3 Proceso 3 / Group 0	Core 3 Free

Results - Final execution times (III)

Balanced mapping

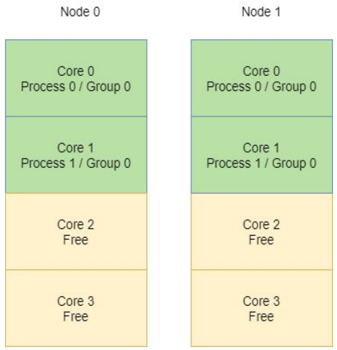
T_it = 0.4 seconds



Shrinking with AR

Expanding with SR

Oversubscription when more than 20 threads.

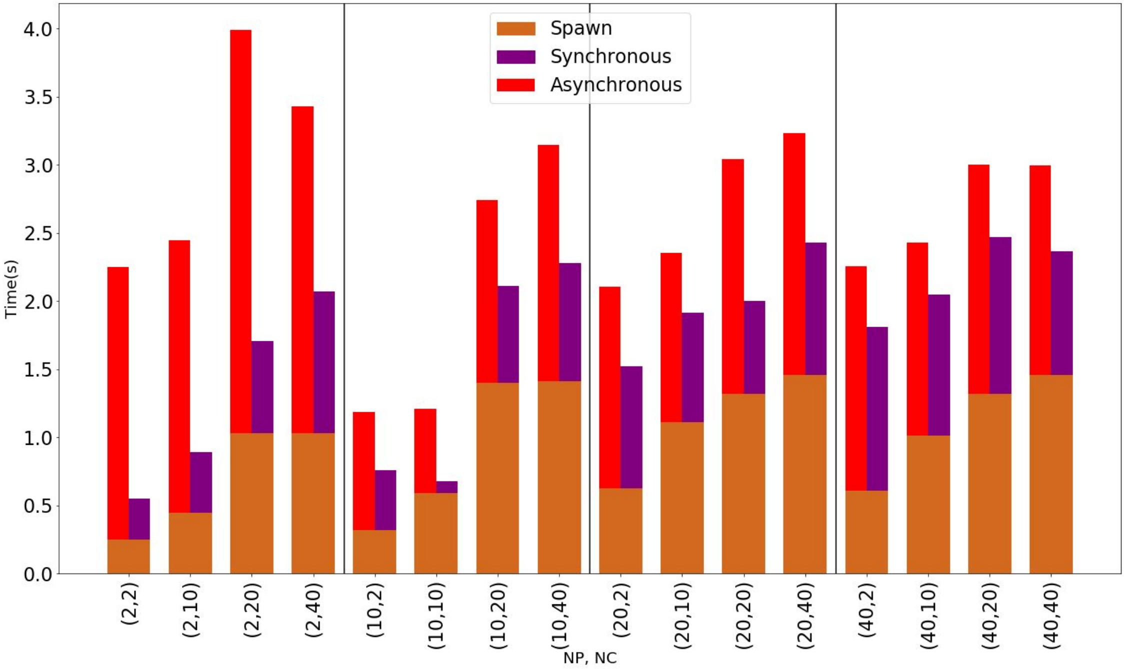


Node 0	Node 1
Core 0 Process 0 / Group 0	Core 0 Process 0 / Group 0
Core 1 Process 1 / Group 0	Core 1 Process 1 / Group 0
Core 2 Free	Core 2 Free
Core 3 Free	Core 3 Free

Results - Final execution times (IV)

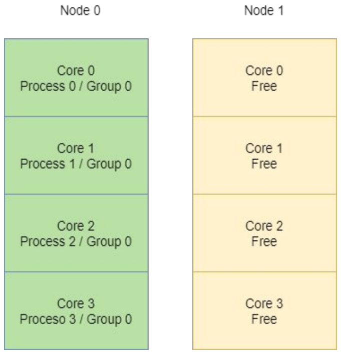
Compact mapping

T_it = 4 seconds



Spawning processes is the most expensive operation

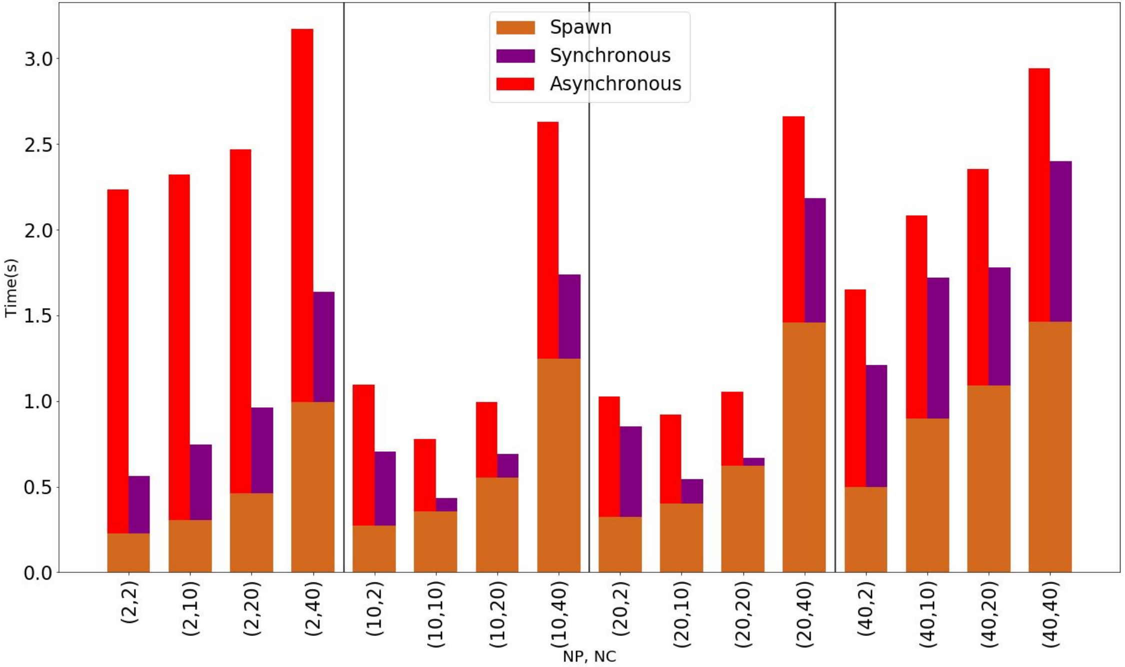
AR is always more expensive than SR



Node 0	Node 1
Core 0 Process 0 / Group 0	Core 0 Free
Core 1 Process 1 / Group 0	Core 1 Free
Core 2 Process 2 / Group 0	Core 2 Free
Core 3 Proceso 3 / Group 0	Core 3 Free

Results - Reconfiguration times (I)

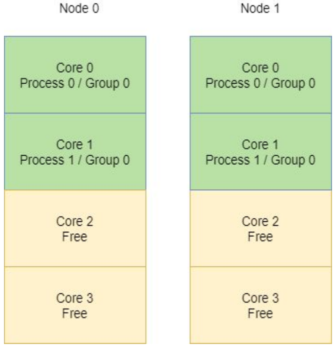
Balanced mapping T_it = 4 seconds



Spawning processes is the most expensive operation

AR is always more expensive than SR

Better performance than Cm: Oversubscription appears in less configurations



Node 0	Node 1
Core 0 Process 0 / Group 0	Core 0 Process 0 / Group 0
Core 1 Process 1 / Group 0	Core 1 Process 1 / Group 0
Core 2 Free	Core 2 Free
Core 3 Free	Core 3 Free

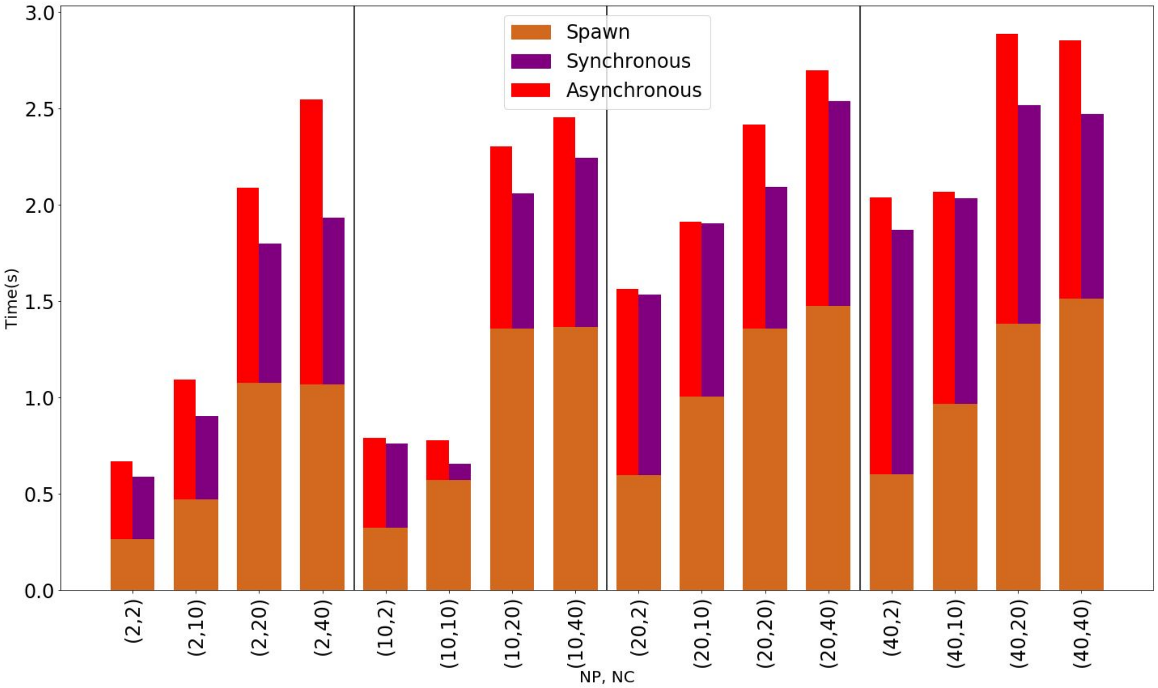
UNIVERSITAT JAUME I

Results - Reconfiguration times (II)

Results - Reconfiguration times (III)

Compact mapping

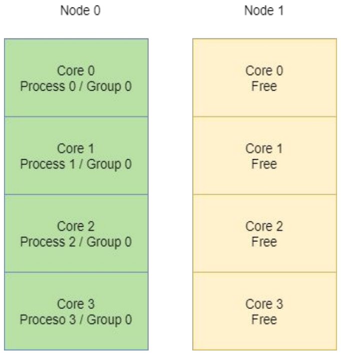
T_it = 0.4 seconds



Spawning processes is the most expensive operation

AR is always more expensive than SR

AR is better for lower T_it values

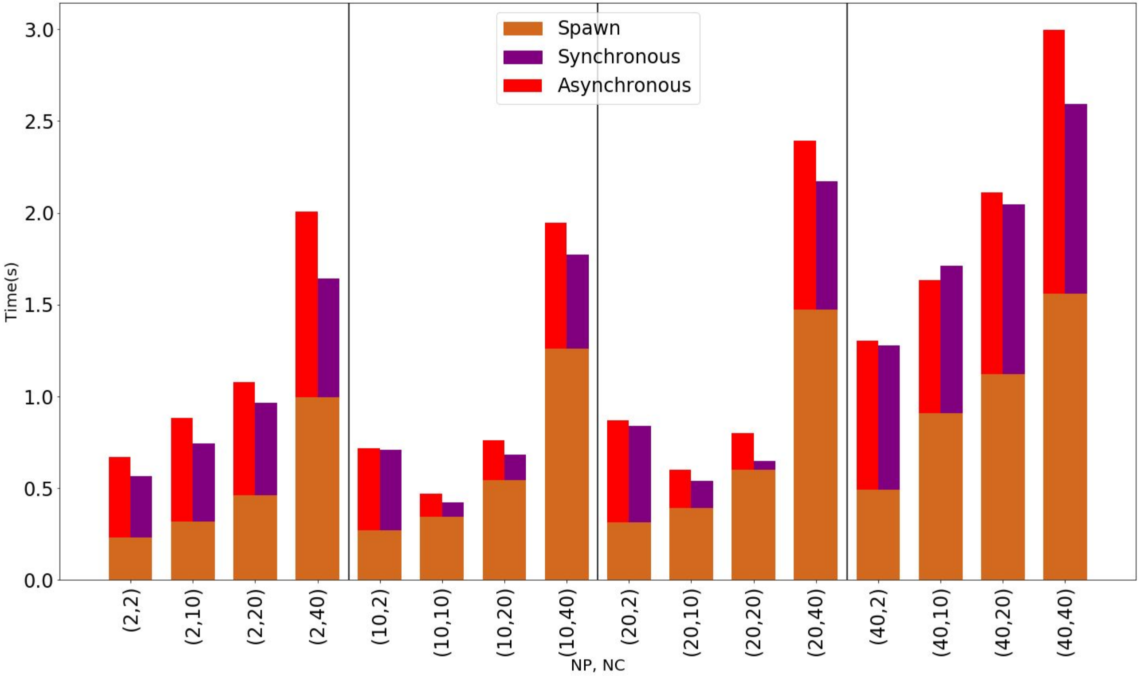


Node 0	Node 1
Core 0 Process 0 / Group 0	Core 0 Free
Core 1 Process 1 / Group 0	Core 1 Free
Core 2 Process 2 / Group 0	Core 2 Free
Core 3 Proceso 3 / Group 0	Core 3 Free

Results - Reconfiguration times (IV)

Balanced mapping

T_it = 0.4 seconds

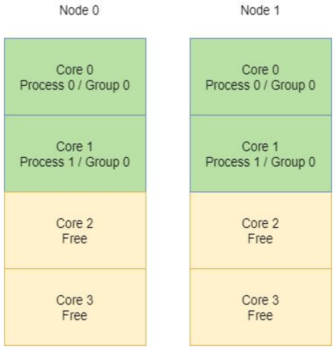


Spawning processes is the most expensive operation

AR is always more expensive than SR

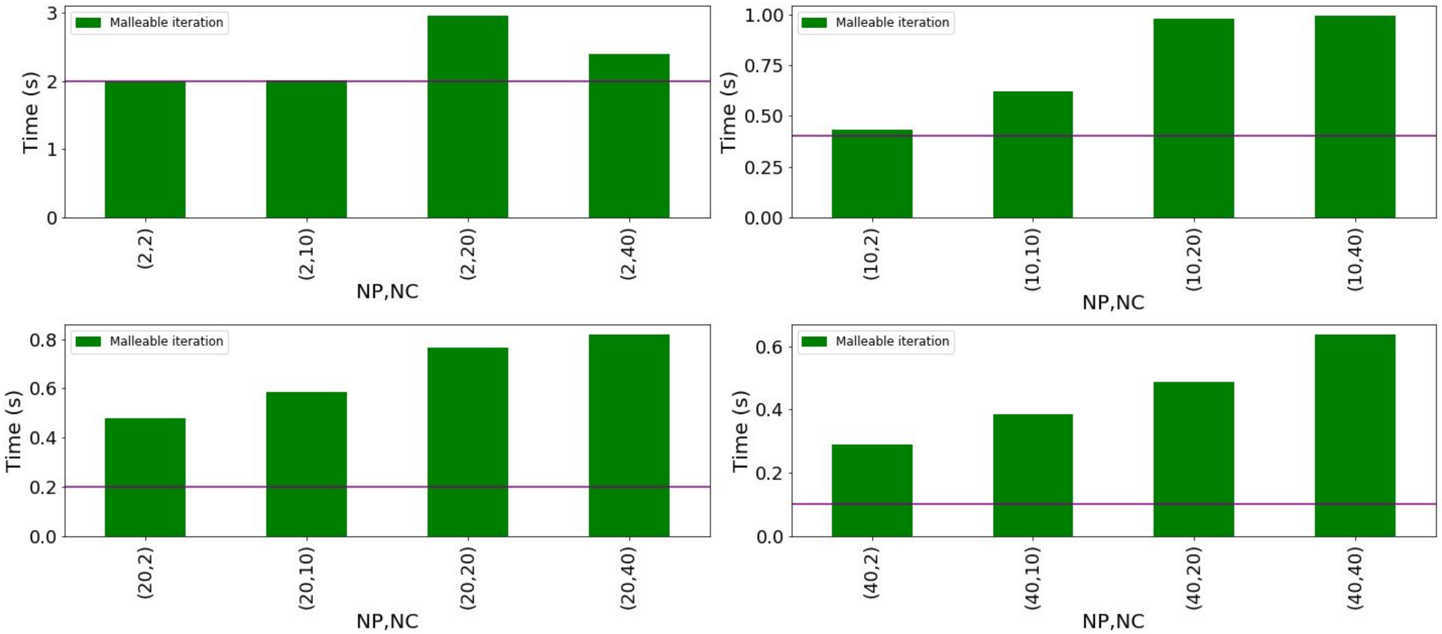
Better performance than Cm: Oversubscription appears in less configurations

AR is better for lower T_it values

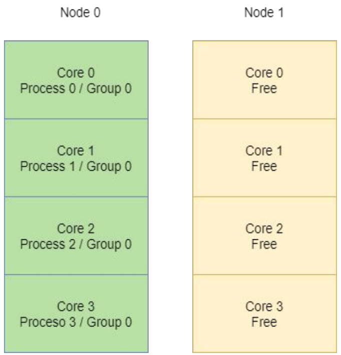


Node 0	Node 1
Core 0 Process 0 / Group 0	Core 0 Process 0 / Group 0
Core 1 Process 1 / Group 0	Core 1 Process 1 / Group 0
Core 2 Free	Core 2 Free
Core 3 Free	Core 3 Free

Compact mapping   T_it = 4 seconds



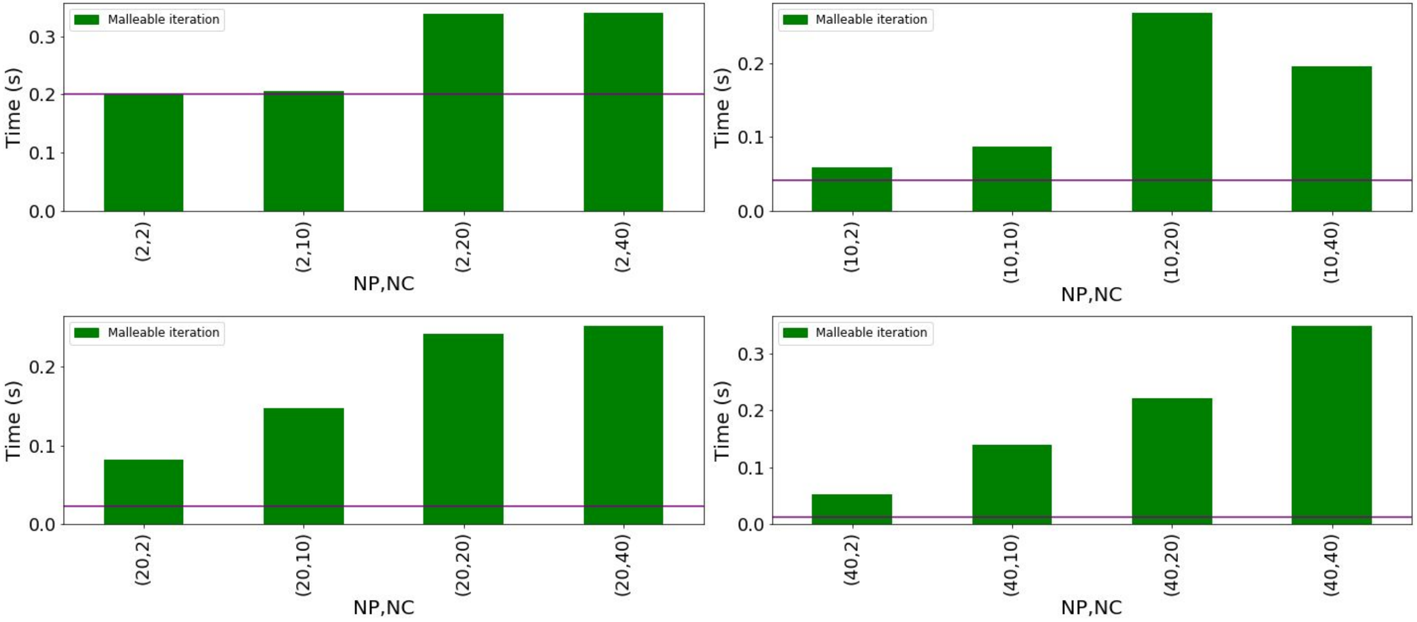
Oversubscription reduces iteration performance drastically



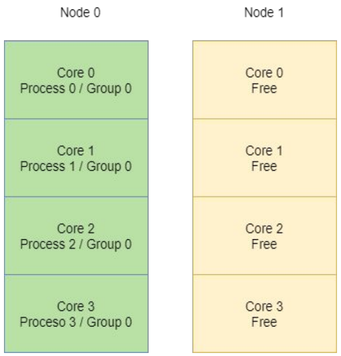
Node 0	Node 1
Core 0 Process 0 / Group 0	Core 0 Free
Core 1 Process 1 / Group 0	Core 1 Free
Core 2 Process 2 / Group 0	Core 2 Free
Core 3 Proceso 3 / Group 0	Core 3 Free

Results - Iteration times under reconfiguration (I)

Compact mapping T_it = 0.4 seconds



Oversubscription reduces iteration performance drastically



Node 0	Node 1
Core 0 Process 0 / Group 0	Core 0 Free
Core 1 Process 1 / Group 0	Core 1 Free
Core 2 Process 2 / Group 0	Core 2 Free
Core 3 Proceso 3 / Group 0	Core 3 Free

Results - Iteration times under reconfiguration (II)

Conclusions
&
Future work

Adaptable synthetic application for expanding, shrinking or migrating

Application allows to study which reconfiguration mechanism is preferred depending on the job state

SR when expanding the job, AR when shrinking

Oversubscription reduces performance for AR and SR

Conclusions

Future Work (I)

Allow users to indicate computational cost for iterations

Allow more complex data redistributions

Allow dynamical SDR & ADR for each reconfiguration in the same execution

Allow emulation of non-iterative applications

Resemble real application from traces

Communicate with RMS

Future Work (II)

A synthetic tool for analysing adaptive workloads

Authors:
Iker Martín-Álvarez (martini@uji.es)
José I. Aliaga
María Isabel Castillo
Sergio Iserte
Rafael Mayo