Multi-thread and Mpi usage in GRID

Roberto Alfieri - Parma University & INFN, Gr.Coll. di Parma

Ferrara , Thursday, March 11, 2010

Outline

MPI and multi-thread support in EGEE

- *Past , Present and Future*

A case study: *porting Cellular Automata algorithms on the Grid*

- *Implemented in MPI, OpenMP and hybrid MPI/OpenMP*

MPI support in EGEE: the past

EGEE-I (2004-2006)

- Typical WN: SMP with 2 CPUs
- No support for multi-core architectures.
- Support for a single flavour of MPI : **MPICH**
- No possibility to run **pre/post execution scripts**
- Files had to be distributed **manually** on the WNs if the home wasn't shared.
- The user had to compile the MPI program on the UI using MPICH.

Example:

```
UI> mpicc  mympi-prog.c    -o mympi-prog
UI> edg-job-submit      mympi-prog.jdl
```

mympi-prog.jdl

```
JobType="MPICH";
NodeNumber=4;
Executable="mympi-prog";
InputSandbox ={"mympi-prog"};
```

In this phase MPI has been mainly used for functionality and performance tests.

MPI support in EGEE: the present

EGEE-II (2006-2008)

Typical WN: 2 CPUs, 2 cores per CPU

Start-up of a specific **MPI Working Group**.

- Purpose of the MPI-WG: To recommend a method for deploying MPI support that will work for both users and site administrators.
- Recommendation document released in **late 2006**.
- Most of the recommendations have been integrated in the middleware
- This new **MPI support** is currently in use
- **Multithread programming** is not supported yet.
- Main user related recommendations:
  1) Introduction of **a more flexible JDL** syntax
  2) Integration of **MPI-start** tool.

MPI support in EGEE: new JDL syntax

NodeNumber is misleading → **deprecated**

CPUNumber fits better the meaning of the attribute

JobType="MPICH" is a wrong level of abstraction → **deprecated**

JobType="normal" should be used, so user can submit MPI jobs using wrapper scripts that set up their environment

- select a specific MPI flavour
- distribute files if the home is not shared
- execute pre/post execution scripts (e.g compile the source on site)
- start mpirun

JDL Example:

```
JobType="Normal";
CPUNumber=4;
Executable="mympi_wrapper.sh";
Arguments="mpi-prog OPENMPI";
InputSandbox={"mympi-wrapper.sh","mympi-prg.c"};

Requirements = other.GlueHostArchitectureSMPSize >= 4;
```

- Depend on the batch scheduler policy

MPI support in EGEE: mpi-start architecture

Integration of **Mpi-Start** in gLite since 02/2008

MPI-Start, developed by HLRS, is a set of scripts which main advantage is the possibility to detect and use site-specific configuration features, like:

- MPI implementations (openMPI, MPICH, MPICH2, PACX-MPI, LAM)

- **Batch scheduler** (SGE, PBS, LSF)
- **File distribution** (if the home isn't shared)
- **Workflow control** (user's pre/post execution scripts)

MPI support in EGEE: mpi-start scripts

mpistart-wrapper.jdl (example)

```
JobType="Normal";
CPUNumber=4;
Executable="mpistart-wrapper.sh";
Arguments="mympi-prog OPENMPI";
StdOutput="std.out";
StdError="std.err";
InputSandbox={"mympi-prog.c","mpistart-wrapper.sh","mpi-hooks.sh"};
```

mpistart-wrapper.sh (simplified version)

```
export I2G_MPI_APP=`pwd`/$1
export I2G_MPI_TYPE=$2
export I2G_MPI_PRE_RUN_HOOK=mpihooks.sh
export I2G_MPI_POST_RUN_HOOK=mpihooks.sh
/opt/i2g/bin/mpi-start   #Invoke mpi-start
```

mpi-hooks.sh (example)

```
pre_run_hook()
{ mpicc -o ${MPI_APPLICATION} ${MPI_APPLICATION}.c }

post_run_hook()
{ mail user@domain -s "${MPI_APPLICATION} completed" < std.out }
```

MPI and multi-thread support in EGEE : the future

EGEE-III (2008-2010)

- Typical WN: 2-4 CPUs, 4-6 cores per CPU
- **Multi-thread programming** should be supported in EGEE to exploit the upcoming multi-core architectures.
- Survey for users and administrators in April 2009: **MPI is still scarcely used**
- Set-up of a new **MPI WG.** Recommendation document will be released soon.
- **New attributes will be introduced in the JDL** for multi-thread support.
- **Discussion ongoing** concerning semantics and implementation.

Attribute	Meaning
CPUNumber=P	Total number of required CPUs
SMPGranularity=C	Minimum number of cores per node
NodeNumber=N	Total number of required nodes
WholeNode=true	Reserve the whole node (all cores)

MPI and multi-thread support in EGEE : examples

PURE MULTI_THREAD

```
# e.g. single whole node with a minimum of 4 cores:
SMPGranularity = 4 ;
WholeNode = True ;
```

PURE MPI

```
#e.g. 16 MPI processes:
CPUnumber = 16 ;

# e.g. 16 MPI processes, whole nodes, a minimum of 4 cores each:
CPUnumber = 16 ;
SMPGranularity = 4 ;
WholeNode = True ;
```

HYBRID MULTI-THREAD/MPI

```
# e.g. 4 MPI processes, 1 per node, a minimum of 4 cores each:
NodeNumber = 4 ;
SMPGranularity = 4 ;
WholeNode = True ;
```

A case study: 2D Cellular Automata (CA)

CA with periodic boundary conditions is a simple example suitable for our purpose
- fine grained parallelism, domain decomposition, high rate of local communications

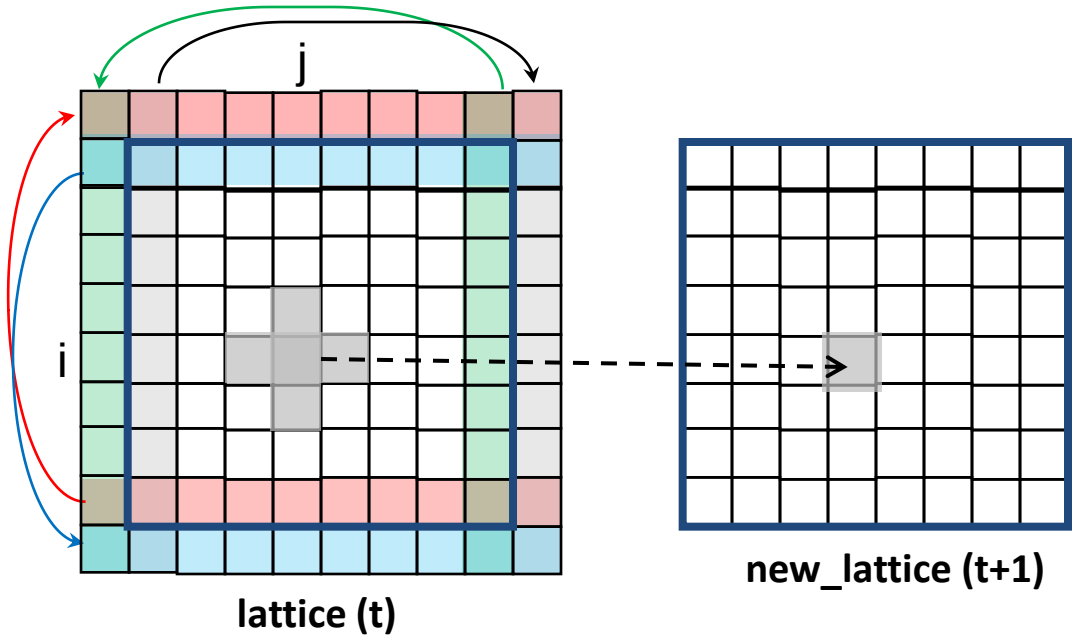
Algorithm and parallel approach are similar to other problems in computational physics
- e.g. Finite Difference Method, Computational Fluid Dynamics, Lattice QCD, etc.

```
init(lattice)
```

```
for(i=0;i<Nrows;i++)
 for(j=0;j<Ncols;j++)
  update_function(i,j)

lattice=new_lattice
```

```
copy_border()
```

```
save(lattice)
```

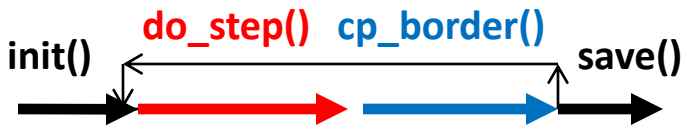


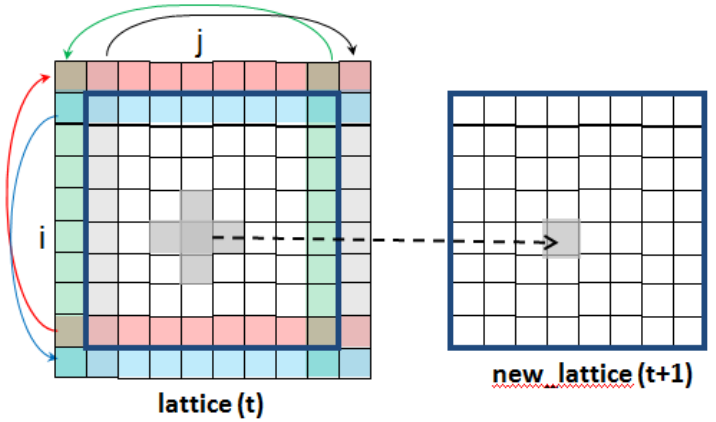
EXAMPLE OF UPDATE FUNCTION: 5 POINT STENCIL

$$x_{i,j}(t+1)=f(x_{i,j}(t),x_{i-1,j}(t),x_{i+1,j}(t),x_{i,j-1}(t),x_{i,j+1}(t))$$

Serial implementation: the code

Single processor
- Borders are exchanged via memory transfers





CA.c

```
init(lattice);
for(step=0; step<max; step++)
   {
   do_step();
   cp_border();
   }
save("lattice.dat");
```

```
do_step(){
for(i=1;i<nrows;i++)
 for (j=1;j<ncols;j++)
  new_lattice[i][j]=update(lattice,i,j);
 cp_lattice(lattice,new_lattice);
}
```

```
cp_border(){
for (i=0;i<=nrows+1;i++) {
    lattice[i][0]=lattice[i][ncols];
    lattice[i][ncols+1]=lattice[i][1]; }
for (j=0;j<=ncols+1;j++) {
    lattice[0][j]=lattice[nrows][j];
    lattice[nrows+1][j]=lattice[1][j]; }
}
```

Serial implementation: execution in the Grid

CA.jdl

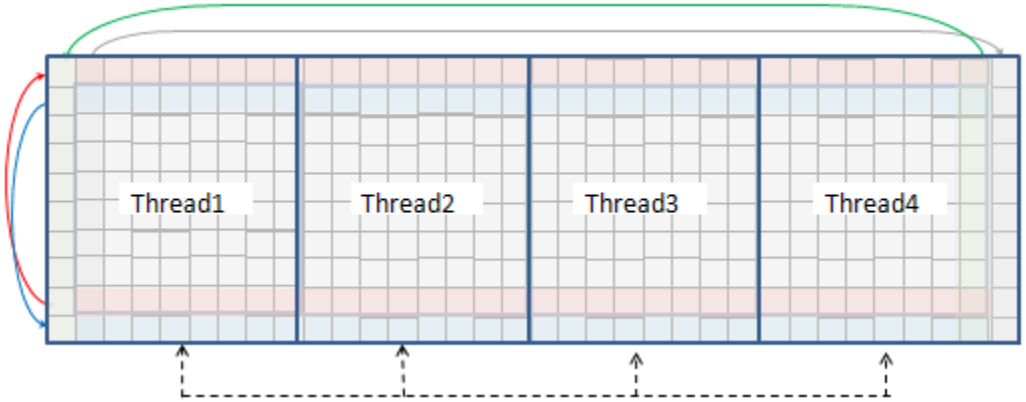
```
JobType="Normal";
Executable="/bin/bash";
Arguments="CA.sh";
StdOutput = "std.out";
StdError = "std.err";
InputSandbox= {"CA.sh","CA.c"};
OutputSandbox= {"std.out","std.err"};

Requirements= other.GlueHostMainMemoryRAMSize >= 8000;
```

CA.sh

```
#compile
gcc CA.c -o CA
#execute
./CA
#save the results
lcg-cr  lattice.dat  -l lfn:/grid/theophys/lattice.dat
```

OpenMP implementation: program flow and code

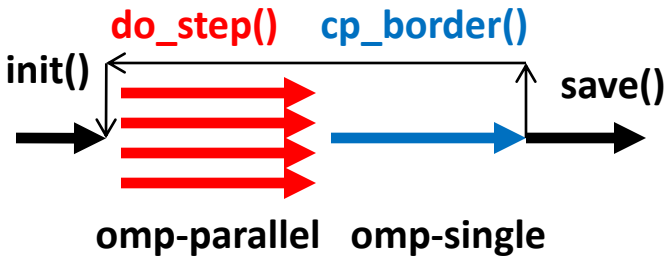
Example: a Whole Node 4 cores
- 1D domain decomposition
- Rows and columns are exchanged
  via memory transfers by a single thread



Parallel do_step() with domain decomposition

CAomp.c

```
init(lattice);
omp_set_num_threads(NUM_THREADS);
#pragma omp parallel private(step)
for(step=0; step<max; step++)
 {
  do_step();
  #pragma omp barrier
  #pragma omp single
   {
    cp_border();
   }
 }
save("lattice.dat");
```



OpenMP execution in the Grid

CAomp.jdl

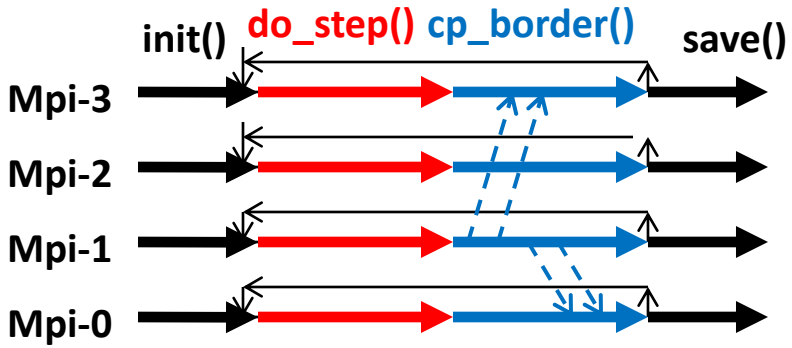
```
JobType="Normal";
SMPGranularity = 4;      # Number of cores on a single node (minimum)
WholeNode = true;        # whole node required
Executable="/bin/bash";
Arguments="CAomp.sh";
StdOutput="std.out";
StdError="std.err";
InputSandbox={"CAomp.sh","CAomp.c"};
OutputSandbox={"std.out","std.err"};

Requirements= other.GlueHostMainMemoryRAMSize >= 8000;
```

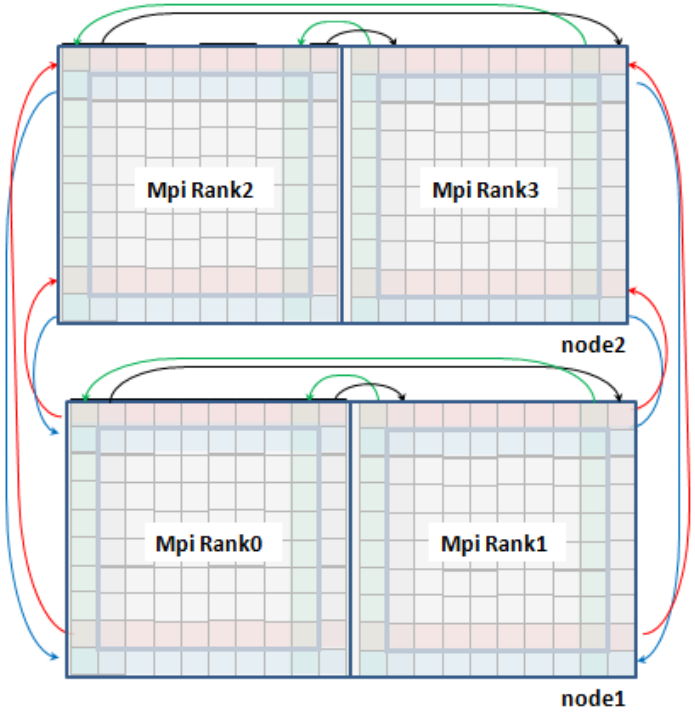
CAomp.sh

```
#compile
gcc -fopenmp CAopm.c -o CAomp
#execute
./CAomp
#save the results
lcg-cr lattice.dat -l lfn:/grid/theophys/lattice.dat
```

MPI implementation : program flow and code

Example: 4 MPI processes (2 on node1, 2 on node2)
- Rows and columns are exchanged via Mpi messages





CAmpi.c

```
MPI_Init(&argc,&argv);
init(lattice);
for(step=0; step<max; step++){
 do_step();
 cp_border();---------->
 }
MPI_Finalize();
save("lattice.dat");
```

```
cp_border() {
// copy rows via MPI messages
MPI_Irecv(FirstRrow-1 from up);
MPI_Irecv(LastRow+1 from down);
MPI_Send (LastRow  to down);
MPI_Send (FirstRow to up);
MPI_Waitall(2);

// copy columns via MPI messages
..
}
```

MPI execution in the Grid with mpi-start

mpistart-wrapper.jdl

```
JobType = "Normal";
CPUNumber = 4;                    # Total number of processors
SMPGranularity=2;                 # Number of cores per node (minimum)
WholeNode=true;
Executable = "mpistart-wrapper.sh";
Arguments = "CAmpi OPENMPI";
StdOutput = "std.out";
StdError = "std.err";
InputSandbox = {"mpistart-wrapper.sh","mpi-hooks.sh","CAmpi.c"};
OutputSandbox = {"std.err","std.out"};
```

mpi-hooks.sh

Remote compilation

Transfer of the results

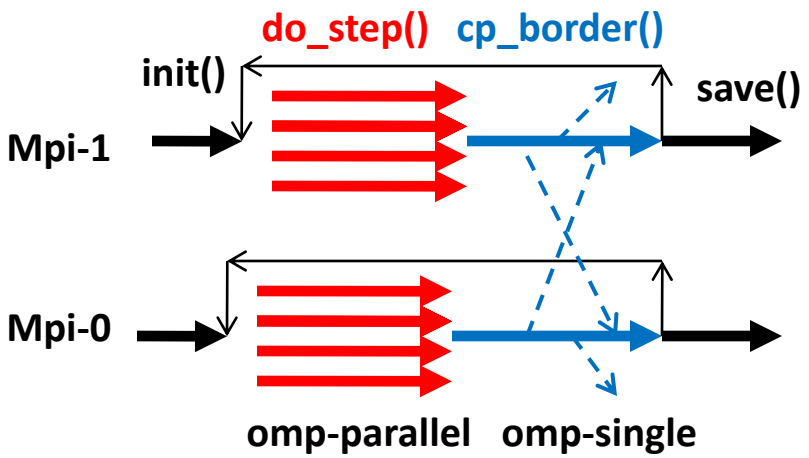
```
pre_run_hook(){
mpicc   -o ${MPI_APPLICATION} ${MPI_APPLICATION}.c }

post_run_hook()
{ lcg-cr  lattice.dat -l lfn:/grid/theophys/lattice.dat }
```

Hybrid OpenMP and MPI: program flow and code

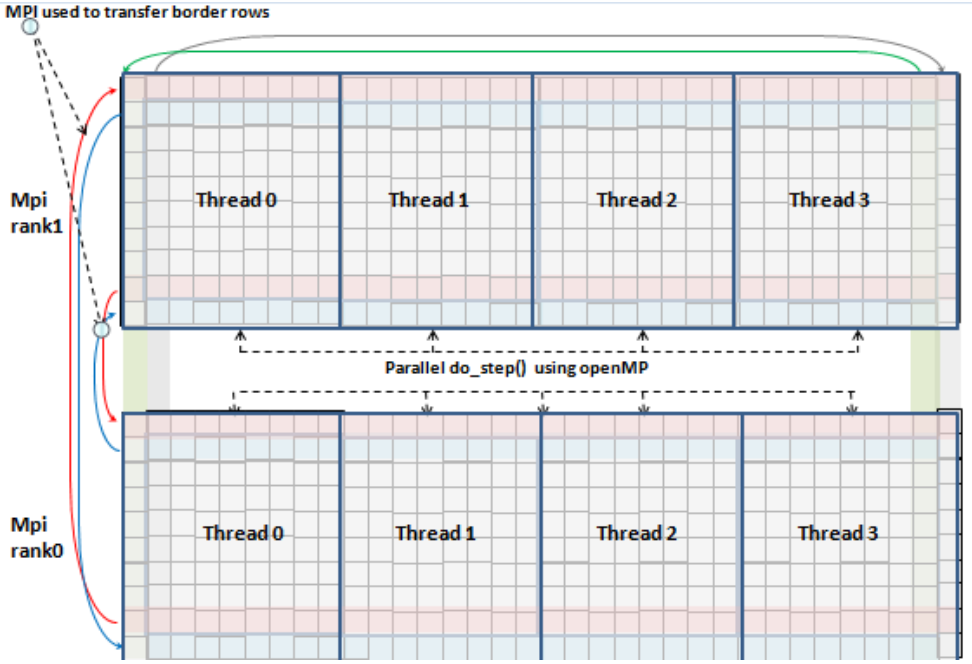
Example: 2 MPI processes with 4 threads each

openMP 1D horizontal domain decomposition
- Columns are exchanged via memory transfers

MPI 1D vertical domain decomposition
- Rows are exchanged via MPI messages




```
MPI_Init();
init(lattice);
omp_set_num_threads(NUM_THREADS);
#pragma omp parallel private(step)
for(step=0; step<max; step++) {
  do_step();
  #pragma omp barrier
  #pragma omp single
   {
    cp_border();
   }
 }
MPI_Finalize();
save("lattice.dat");
```



Hybrid OpenMP and MPI execution in the Grid

mpistart-wrapper.jdl

```
JobType = "Normal";
NodeNumber= 2;                  # Number of MPI processes
SMPGranularity= 4;              # Number of cores per node (minimum)
WholeNode = true;               # Whole Node required
Executable = "mpistart-wrapper.sh";
Arguments = "CAhybrid OPENMPI";
StdOutput = "std.out";
StdError = "std.err";
InputSandbox = {"mpistart-wrapper.sh","mpi-hooks.sh","CAhybrid.c"};
OutputSandbox = {"std.err","std.out"};
```

mpi-hooks.sh

```
pre_run_hook()
{ mpicc -fopenmp -o ${MPI_APPLICATION} ${MPI_APPLICATION}.c }

post_run_hook()
{ lcg-cr lattice.dat -l lfn:/grid/theophys/lattice.dat }
```

Performances

	TOTAL CORES	SPEEDUP	AVAIL. MEM FOR DATA
Sequential	1	1	$\approx M/C$
Pure openMP (WholeNode)	C	C	M
Pure MPI (WholeNodes)	$C * N$	$\approx \frac{Tstep * C * N}{Tstep + 4 * Tcomm}$	$M * N$
Hybrid MPI/openMP (WholeNodes)	$C * N$	$\approx \frac{Tstep * C * N}{Tstep + 2 * Tcomm}$	$M * N$

C = Number of cores per node
N = Number of available nodes
M = Node Memory
Tstep = Computational time needed for a do_step()
Tcomm= Communication time needed to transfer a sub-domain face via MPI

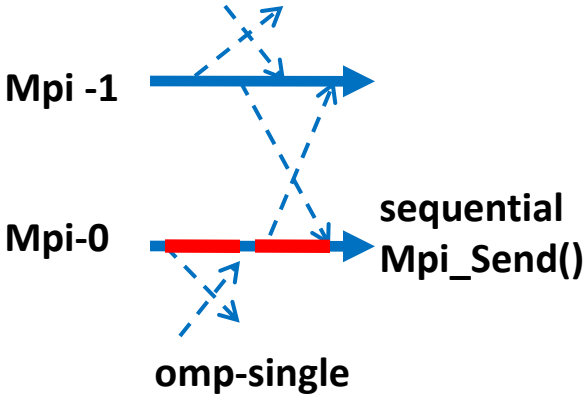
Multi-threaded MPI communications

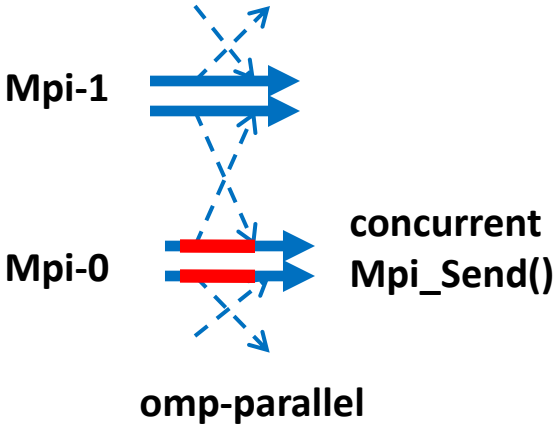
In this Cellular Automata parallel implementation each MPI process executes MPI calls sequentially.

The exploitation of multi-thread parallelism in the MPI communications would allow a further speed-up improvement.

The thread support in MPI should be claimed explicitly (if provided by the MPI implementation):

```
int claimed = MPI_THREAD_MULTIPLE;
int provided;
MPI_Init_thread(&argc,&argc,claimed,&provided);
if(provided < claimed)
 printf("MPI_THREAD_MULTIPLE not supported\n");
```





Thank you
for your attention!