```
/* Example of problem solving in parallel */

// Usage:
// ff-mpirun -np 12 LaplacianParallel.edp (here 12 is the number of threads (command nproc to know that)
// Need FreeFem++ with PETSc

// Parallel stuff
load "PETSc"
macro partitioner()metis//
macro dimension()2//
include "./macro_ddm.idp"

macro def(i)[i]//
macro init(i)[i]//
//macro meshN()mesh// //these macro are defined in macro_ddm.idp
//macro intN()int2d//

// Parameters
int nn = 500;
real L = 1.;
real H = 1.;

func f = 1.;

func Pk = P1;

// Mesh
border b1(t=0, L){x=t; y=0; label=1;}
border b2(t=0, H){x=L; y=t; label=2;}
border b3(t=L, 0){x=t; y=H; label=3;}
border b4(t=H, 0){x=0; y=t; label=4;}

meshN Th = buildmesh(b1(1) + b2(1) + b3(1) + b4(1)); //build a really coarse mesh (just to build the fespace later)
//meshN Th = square(1, 1, [L*x, H*y]);

int[int] Wall = [1, 2, 3, 4];

// Fespace
fespace Uh(Th, Pk);

// Mesh partition
int[int] ArrayIntersection;
int[int][int] RestrictionIntersection(0);
real[int] D;

meshN ThBorder;
meshN ThGlobal = buildmesh(b1(nn*L) + b2(nn*H) + b3(nn*L) + b4(nn*H)); //build the mesh to partition
//meshN ThGlobal = square(nn*L, nn*H, [L*x, H*y]);
int InterfaceLabel = 10;
int Split = 1;
int Overlap = 1;
build(Th, ThBorder, ThGlobal, InterfaceLabel, Split, Overlap, D, ArrayIntersection, RestrictionIntersection, Uh, Pk,
    ↪ mpiCommWorld, false); //see macro_ddm.idp for detailed parameters

// Macro
macro grad(u) [dx(u), dy(u)] //

// Problem
varf vLaplacian (u, uh) //Problem in varf formulation mandatory
        = intN(Th)(
```

```
                grad(u)' * grad(uh)
        )
        - intN(Th)(
                f * uh
        )
        + on(Wall, u=0)
        ;

matrix<real> Laplacian = vLaplacian(Uh, Uh); //build the sequential matrix
real[int] LaplacianBoundary = vLaplacian(0, Uh);// and right hand side

//// In sequential, you normally do that:
//// Solve
//Uh def(u)=init(0);
//u[] = Laplacian^-1 * LaplacianBoundary;

//// Plot
//plot(u);

// In parallel:
// Matrix construction
dmatrix PLaplacian(Laplacian, ArrayIntersection, RestrictionIntersection, D, bs=1); //build the parallel matrix
set(PLaplacian, sparams="-pc_type lu -pc_factor_mat_solver_package mumps"); //preconditioner LU and MUMPS
    ↪ solver (see PETSc doc for detailed parameters)

// Solve
Uh def(u)=init(0); //define the unknown (must be defined after mesh partitioning)
u[] = PLaplacian^-1 * LaplacianBoundary;

// Export results to vtk (there is not plot in parallel)
{
        fespace PV(Th, P1);
        PV uu=u;
        int[int] Order = [1];
        export("Result", Th, uu, Order, mpiCommWorld);
}
```