Frontal and multi-frontal solvers: hypermatrix generalization to higher orders and multi-physics

Maciej Paszynski

Department of Computer Science

AGH University of Science and Technology, Krakow, Poland

maciej.paszynski@agh.edu.pl

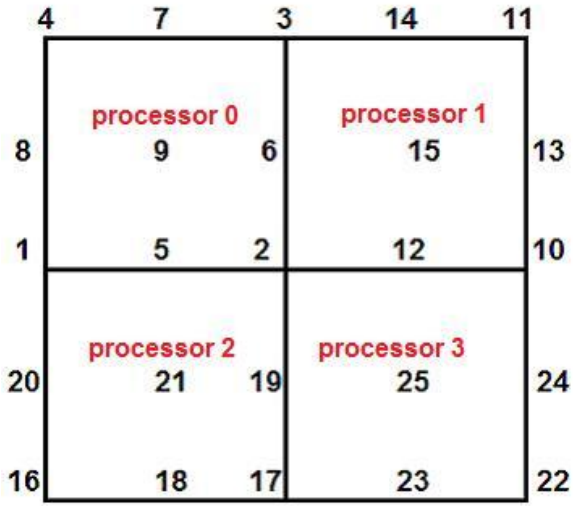
http://www.ki.agh.edu.pl/en/staff/paszynski-maciej

http://www.ki.agh.edu.pl/en/research-groups/a2s

Main collaborators

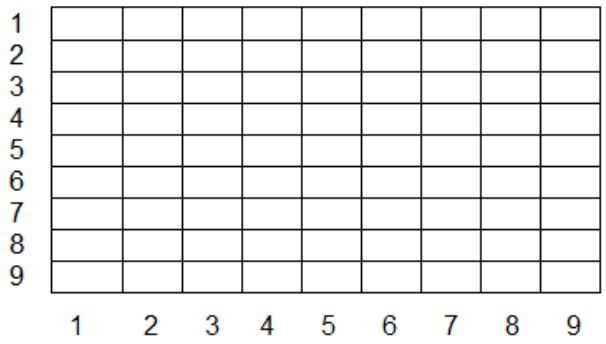
David Pardo (IKERBASQUE)
Victor Calo (KAUST)
Leszek Demkowicz (ICES, UT)

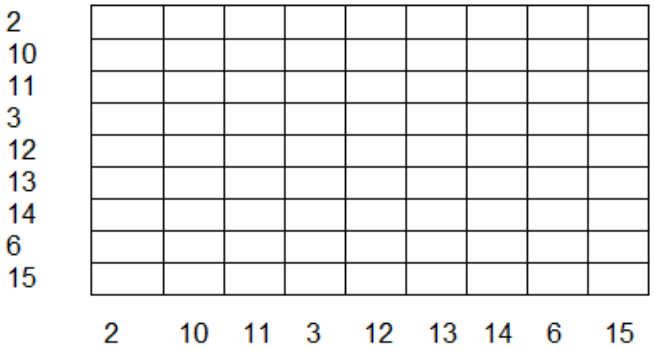
INPUT DATA

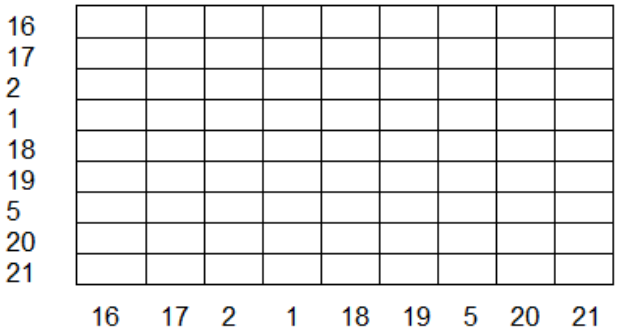


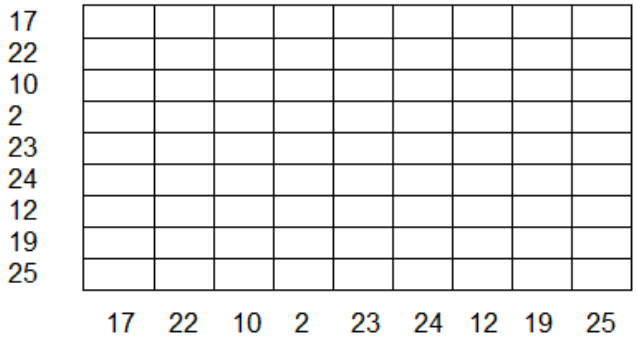
1. Supernodes matrices
(one per processor)

Assumption: work on the level of nodes

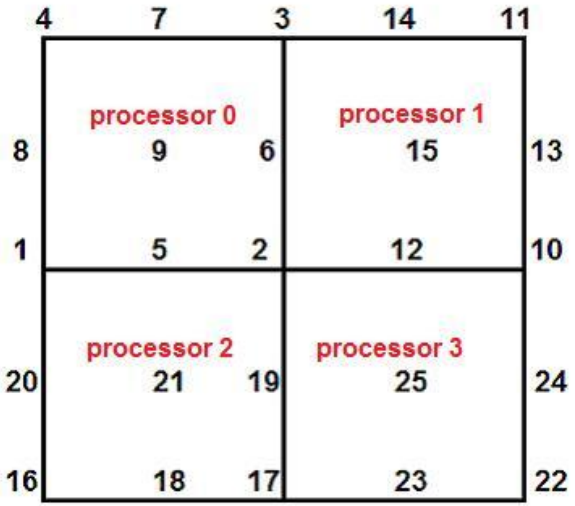






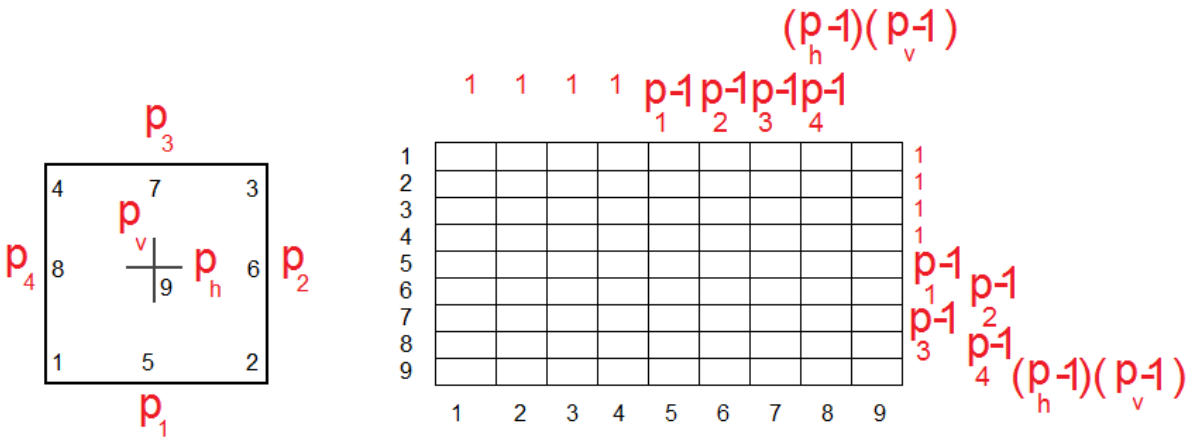


INPUT DATA



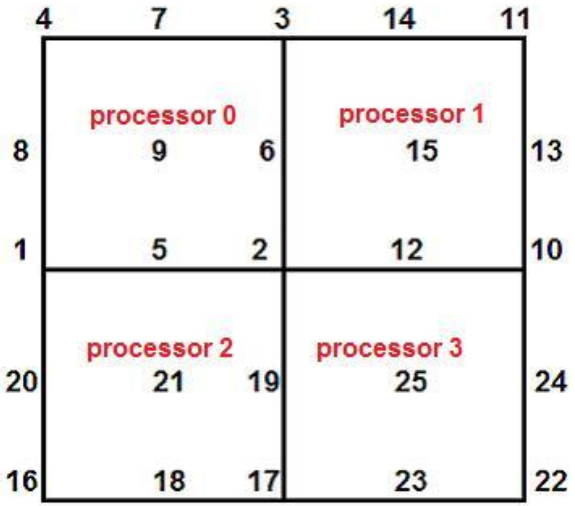
1. Supernodes matrices
(one per processor)

Assumption: work on the level of nodes



In 3D the internal node block may have 10^3 unknowns

INPUT DATA

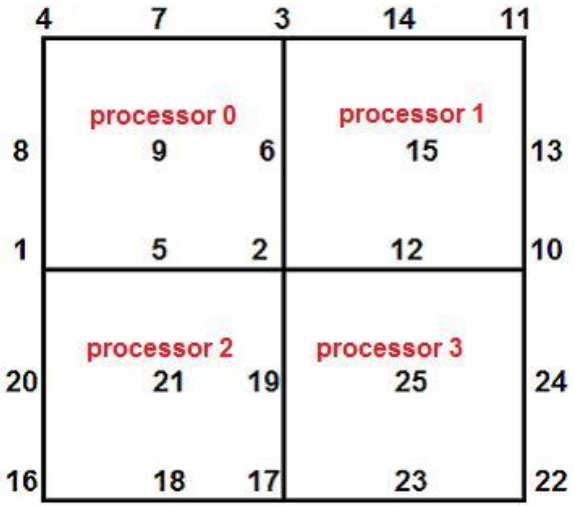


2. List of processors assigned to nodes

Node	1	2	3	4	5	6	7	8	9	10	11	12
Processors	0,2	0,1,2,3	0,1	0	0,1	0,1	0	0	0	1,3	1	1,3

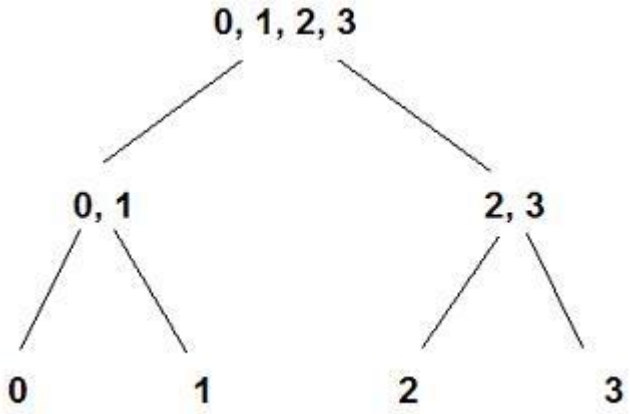
Node	13	14	15	16	17	18	19	20	21	22	23	24	25
Processors	1	1	1	2	2,3	2	2,3	2	2	3	3	3	3

INPUT DATA



3. Elimination tree

Can be constructed automatically by the solver or provided by the user



DATA STRUCTURES : HYPERMATRIX

```
 type hyper_matrix
c
c     type: -2 = rectangular matrix of hyper-matrices
c           -1 = lower-triangular matrix of hyper-matrices
c            0 = zero (null) matrix
c            1 = dense lower-triangular matrix
c            2 = dense rectangular matrix
c     mrow: number of rows
c     ncol: number of columns
c     D:    pointer to memory for dense block
c     S:    recursive pointer to hyper_matrix blocks
      integer :: type
      integer :: mrow,ncol
#if C_MODE
      complex*16,   pointer :: D(:,:)
#else
      double precision,   pointer :: D(:,:)
#endif
      type(hyper_matrix), pointer :: H(:,:)
c
      endtype hyper_matrix
```

DATA STRUCTURES : SUPERNODE

```
type supernode_in_matrix
      integer :: global_id      !global numbering of supernodes
      integer :: matrix_id      !either row or column of supernode in the matrix

c.......subdomains where the supernode belongs
c      to know when can we eliminate the supernode
      integer :: nr_subdomains
      integer, pointer, dimension(:) :: subdomains
end type supernode_in_matrix
```

DATA STRUCTURES :
SUPERNODES SYSTEM

```
type supernodes_system
c.......number of supernodes in rows / columns
      integer :: nr_rows
      integer :: nr_columns
      type(hyper_matrix) :: A
      type(hyper_matrix) :: b

c.......global ids of supernodes,
c      with pointers to matrix supernodes
      type(supernode_in_matrix), dimension(:),
   .        pointer ::row_supernodes
      type(supernode_in_matrix), dimension(:),
   .        pointer ::column_supernodes

c.......number of eliminated nodes (=0 if full system)
      integer :: ncount
end type supernodes_system
```

DATA STRUCTURES : SUPERNODES SOLUTION

```
 type supernodes_solution
c.......number of components
       integer :: nr_components

c.......solution values for several supernodes
       type(hyper_matrix) :: x

c.......global id's of these supernodes
       type(supernode_in_matrix), dimension(:),
    .          pointer :: supernodes
end type supernodes_solution
```

DATA STRUCTURES : STORING PROCESSOR OWNERS

```
c  ...initial mesh element...
      type element
c  .....
      integer        :: proc_owner
      endtype element
```

```
c  ...vertex node
     type vertex
c  ......
       integer        :: global_id
       integer        :: nr_proc_owners
       integer        :: proc_owners(MAX_PROC_OWNERS)
     endtype vertex
```

```
c  ...non-vertex (higher order) node
     type node
c  ......
       integer        :: global_id
       integer        :: nr_proc_owners
       integer        :: proc_owners(MAX_PROC_OWNERS)
     endtype node
```

INTERFACE

```
subroutine distribute_mesh
     use data_structure2D
c  .....
end subroutine distribute_mesh
```

```
subroutine get_elimination_tree
use data_structure2D
c   ......
end subroutine get_elimination_tree
```

```
subroutine get_subdomain_system(s)
     use data_structure2D
     use hyper_matrix_mod
     use supernodes_system_mod
c.....arguments
     type(supernodes_system) :: s
c   .....
end subroutine get_subdomain_system
```

```
subroutine return_solution_to_subdomain(sol)
      use supernodes_system_mod
c.....arguments
     type(supernodes_solution) :: sol
c  .....
end subroutine return_solution_to_subdomain
```

INTERFACE TO LINEAR ALGEBRA SOLVER

```
subroutine get_schur_complement(s)
      use supernodes_system_mod
      use communicators
c.....arguments
     type(supernodes_system),pointer :: s
c  .....
end subroutine get_schur_complement
```

```
subroutine execute_backward_substitution(s, sol)
      use supernodes_system_mod
      use communicators
c.....arguments
     type(supernodes_solution) :: sol
     type(tree_node) :: s
c  .....
end subroutine execute_backward_substitution
```

```
module communicators
c.....communicator involving processors of the node
     integer :: COMM_MYNODE
```

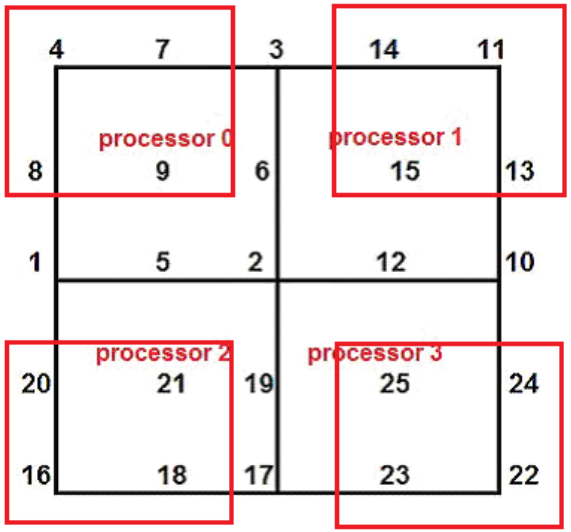
HOW DOES IT WORK?

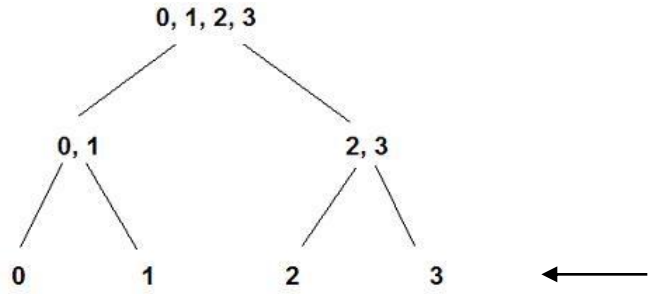
```
system function recursive_solver(tree_node)
  system = 0
c...leaf node computes Schur complement of subdomain internal supernodes
c   with respect to subdomain interface supernodes
  if only 1 proc is assigned to tree_node then
    system = schur complement of subdomain internal supernodes
      with respect to subdomain interface supernodes
  else
c...other nodes:
c   1. send/recv contributions between son nodes
    system = 0
    do ison for each son_node of tree_node
      if MYRANK is assigned to node then
        system_contributions(1) = recursive_solver1(son_node)
        if MYRANK is n/2+1 on the list of n processors assigned to node then
          send system_contributions(1) to 1st processor from the list
        else if MYRANK is 1st on the list of n processors assigned to node then
          receive system_contributions(2) from n/2+1 processor from the list
        endif
      endif
    enddo
```

HOW DOES IT WORK?

```
c   2. eliminate fully assembled supernodes
    create NODE_COMMUNICATOR with processors assigned to tree_node
    barrier(NODE_COMMUNICATOR)
    if MYRANK is 1st processor on the list of n processors assigned to node then
      decide which supernodes from system_contributions can be eliminated
      create system
      submit the system
    endif
      system = resulting Schur complement
        (using all processors from NODE_COMMUNICATOR)
    if MYRANK is 1st processor on the list of processor assinged to node
c   3. store Schur complement at the node
      store system at tree_node
    endif
    delete NODE_COMMUNICATOR
  endif
  return system
end
```

HOW DOES IT WORK?





Processor 0:

call get_subdomain_system
1,2,3,4,5,6,7,8,9

call find_nodes_to_eliminate
4,7,8,9 || 1,2,3,5,6

call get_schur_complement

Processor 1:

call get_subdomain_system
2,10,11,3,12,13,14,6,15

call find_nodes_to_eliminate
11,13,14,15 || 2,10,3,12,6

call get_schur_complement

Processor 2:

call get_subdomain_system
16,17,2,1,18,19,5,20,21

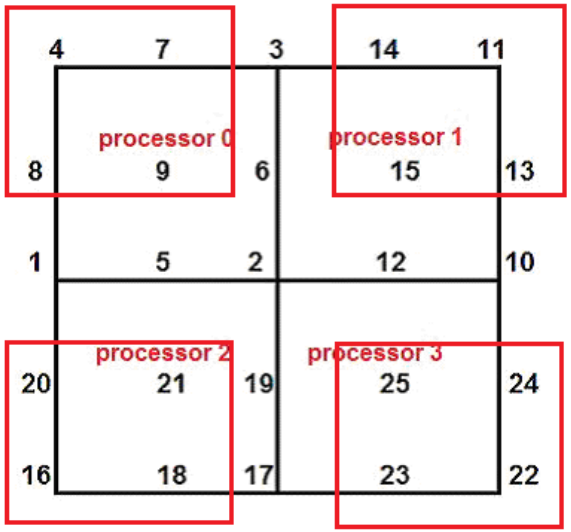
call find_nodes_to_eliminate
16,18,20,21 || 17,2,1,19,5
call get_schur_complement

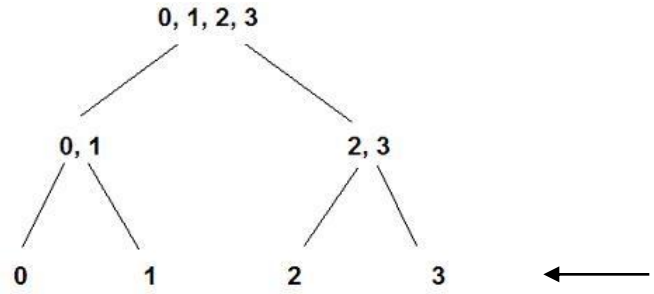
Processor 3:

call get_subdomain_system
17,22,10,2,23,24,12,19,25

call find_nodes_to_eliminate
22,23,24,25 || 17,10,2,12,19
call get_schur_complement

HOW DOES IT WORK?





Processor 0: ~~2,10,3,12,6~~

call get_subdomain_system
1,2,3,4,5,6,7,8,9

call find_nodes_to_eliminate
4,7,8,9 || 1,2,3,5,6

call get_schur_complement

Processor 1:

call get_subdomain_system
2,10,11,3,12,13,14,6,15

call find_nodes_to_eliminate
11,13,14,15 || 2,10,3,12,6

call get_schur_complement

Processor 2: ~~17,10,2,12,19~~

call get_subdomain_system
16,17,2,1,18,19,5,20,21

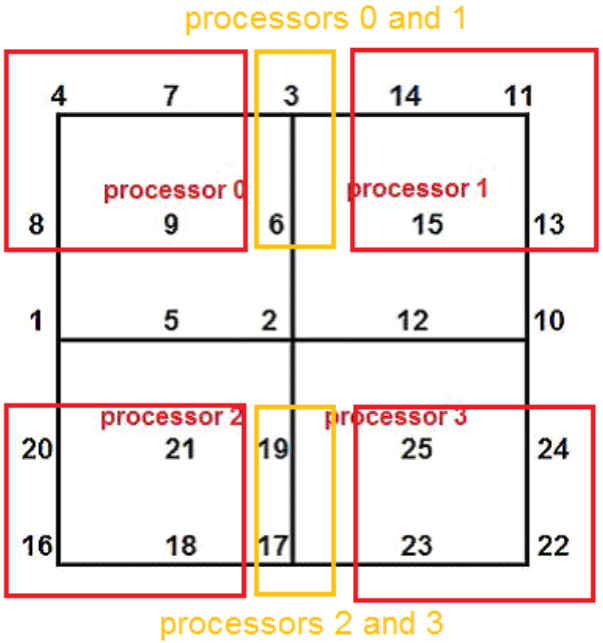
call find_nodes_to_eliminate
16,18,20,21 || 17,2,1,19,5
call get_schur_complement

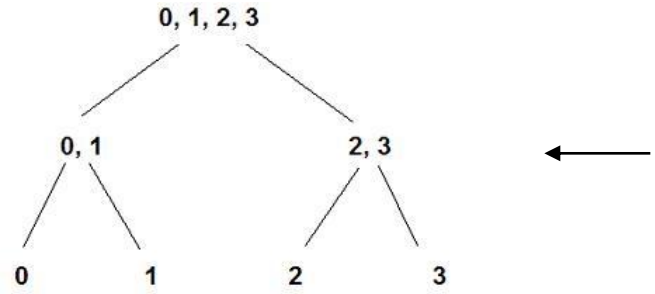
Processor 3:

call get_subdomain_system
17,22,10,2,23,24,12,19,25

call find_nodes_to_eliminate
22,23,24,25 || 17,10,2,12,19
call get_schur_complement

HOW DOES IT WORK?





Processor 0, 1

call create_system
1,2,3,5,6 + 2,10,3,12,6 = 1,2,3,5,6,10,12

call find_nodes_to_eliminate
3,6 || 1,2,5,10,12
call get_schur_complement

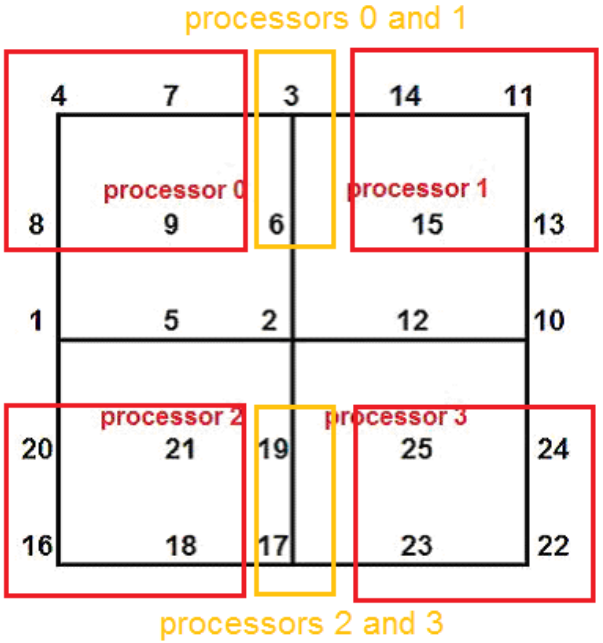
Processor 2, 3

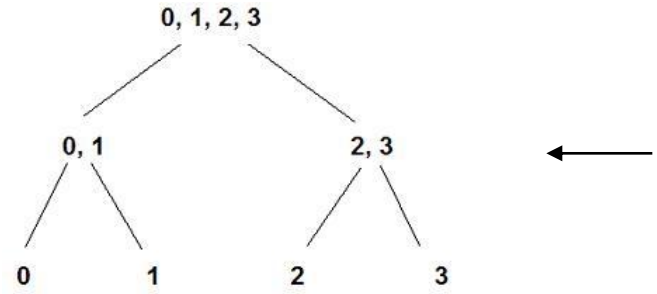
call create_system
17,2,1,19,5 + 17,10,2,12,19 = 1,2,5,10,12,17,19

call find_nodes_to_eliminate
19,17 || 1,2,5,10,12

call get_schur_complement

HOW DOES IT WORK?





Processor 0, 1

1,2,5,10,12

call create_system
1,2,3,5,6 + 2,10,3,12,6 = 1,2,3,5,6,10,12

call find_nodes_to_eliminate
3,6 || 1,2,5,10,12
call get_schur_complement

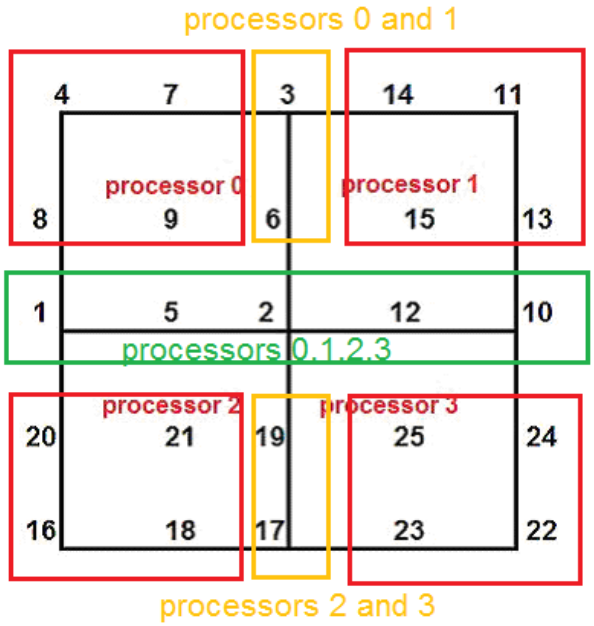
Processor 2, 3

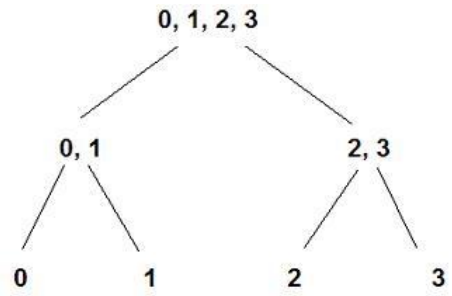
call create_system
17,2,1,19,5 + 17,10,2,12,19 = 1,2,5,10,12,17,19

call find_nodes_to_eliminate
19,17 || 1,2,5,10,12

call get_schur_complement

HOW DOES IT WORK?



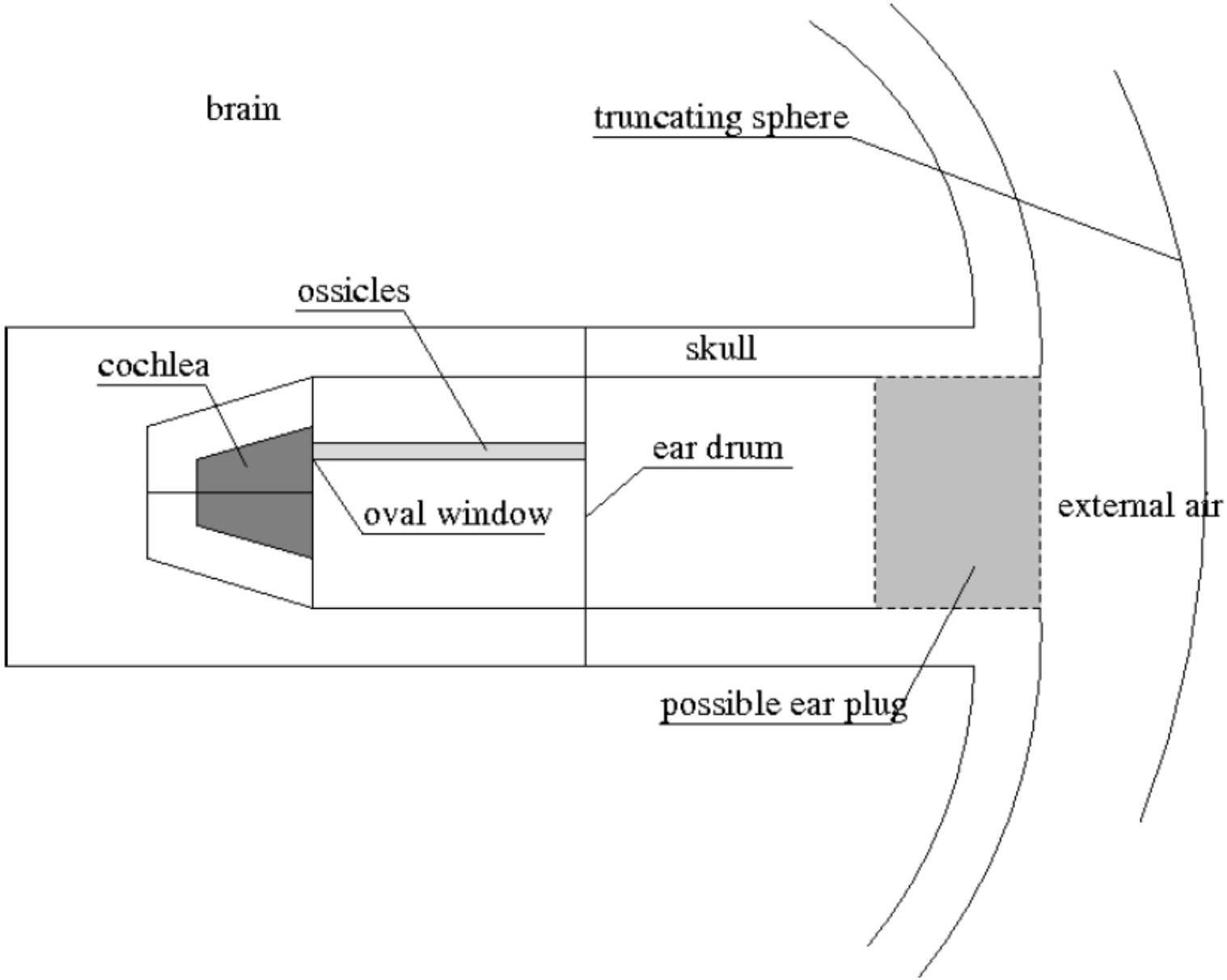


Processors 0, 1, 2, 3

call create_system
1,2,5,10,12 + 1,2,5,10,12 = 1,2,5,10,12

call get_schur_complement
(system%ncount=0 results in
full forward elimination
followed by backward substitution)

NUMERICAL RESULTS



LINEAR ELASTICITY COUPLED WITH LINEAR ACOUSTICS

$$\begin{cases} \boldsymbol{u} \in \tilde{\boldsymbol{u}}_D + \boldsymbol{V},\ p \in \tilde{p}_D + V, \\ b_{ee}(\boldsymbol{u}, \boldsymbol{v}) + b_{ae}(p, \boldsymbol{v}) = l_e(\boldsymbol{v}), \quad \forall \boldsymbol{v} \in \boldsymbol{V} \\ b_{ea}(\boldsymbol{u}, q) + b_{aa}(p, q) = l_a(q), \quad \forall q \in V \end{cases}$$

$$b_{ee}(\boldsymbol{u}, \boldsymbol{v}) = \int_{\Omega_e} \left(E_{ijkl} u_{k,l} v_{i,j} - \rho_s \omega^2 u_i v_i \right)\, d\boldsymbol{x} + i\omega \int_{\Gamma_{Ce}} \beta_{ij} u_j v_i \, dS$$

$$b_{ae}(p, \boldsymbol{v}) = \int_{\Gamma_I} p v_n \, dS$$

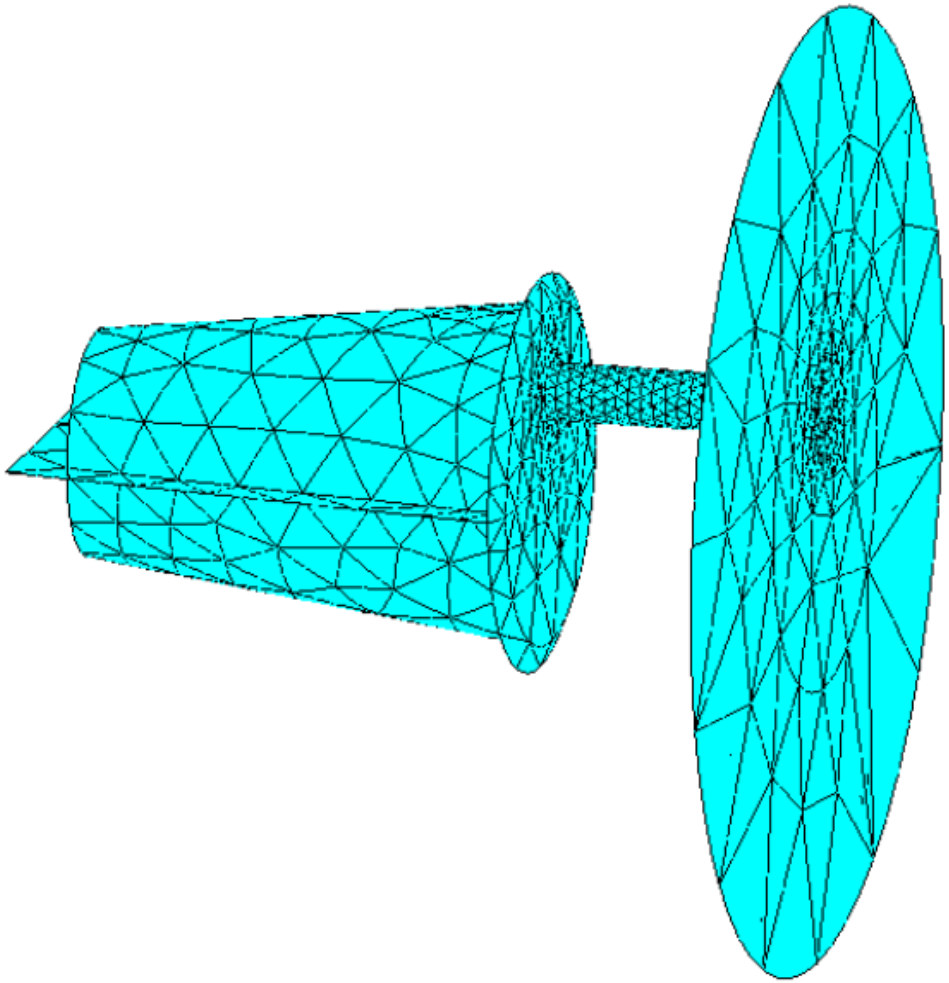
$$b_{ea}(\boldsymbol{u}, q) = -\omega^2 \rho_f \int_{\Gamma_I} u_n q \, dS$$

$$b_{aa}(p, q) = \int_{\Omega_a} \left(\boldsymbol{\nabla} p \boldsymbol{\nabla} q - k^2 pq \right)\, d\boldsymbol{x} + i\omega \int_{\Gamma_{Ca}} \rho_f d\, pq \, dS$$

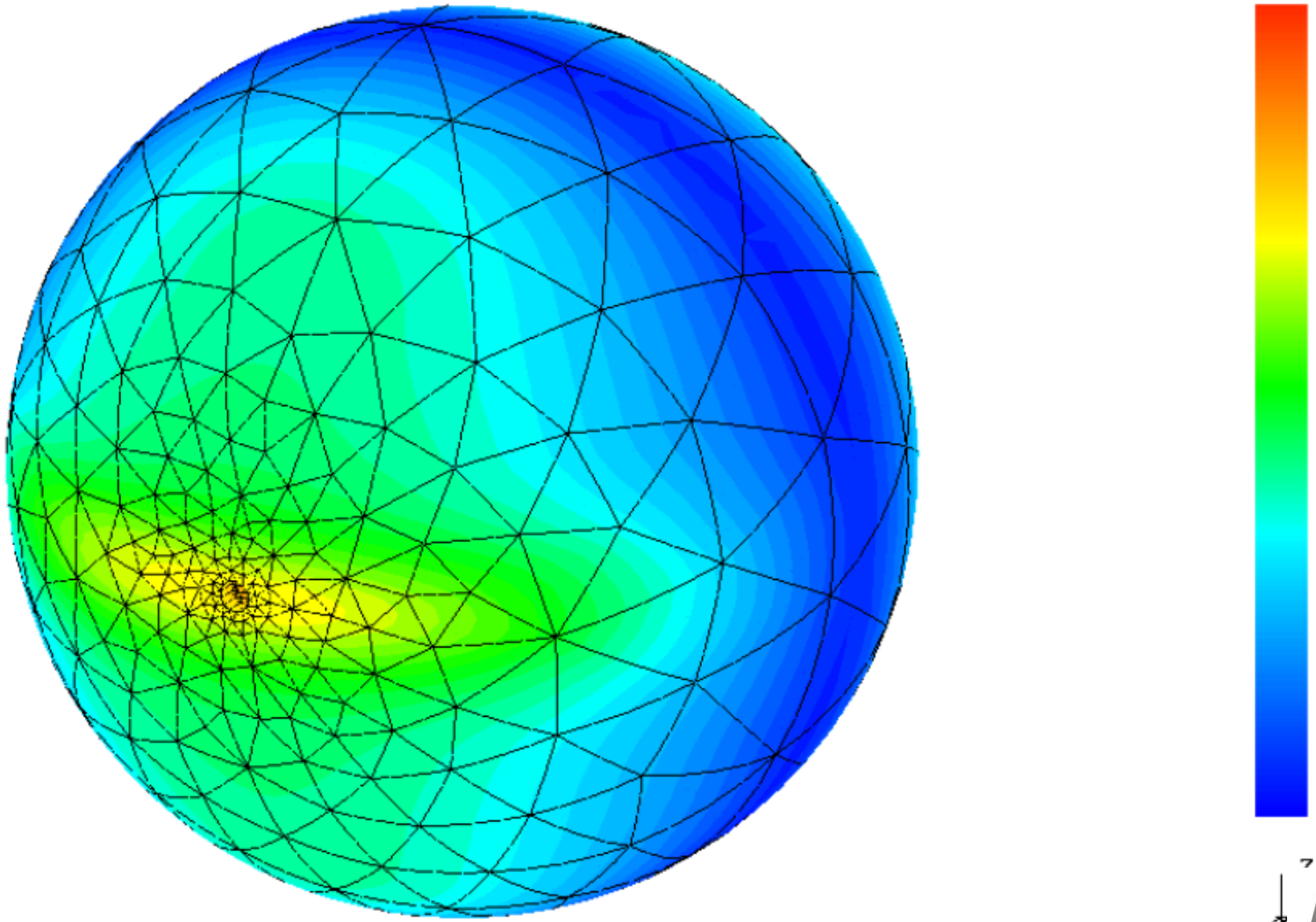
$$l_e(\boldsymbol{v}) = \int_{\Omega_e} f_i v_i \, d\boldsymbol{x} + \int_{\Gamma_{Ne} \cup \Gamma_{Ce}} g_i v_i \, dS$$

$$l_a(q) = i\omega \rho_f \int_{\Gamma_{Na} \cup \Gamma_{Ca}} v_0 q \, dS$$

NUMERICAL RESULTS

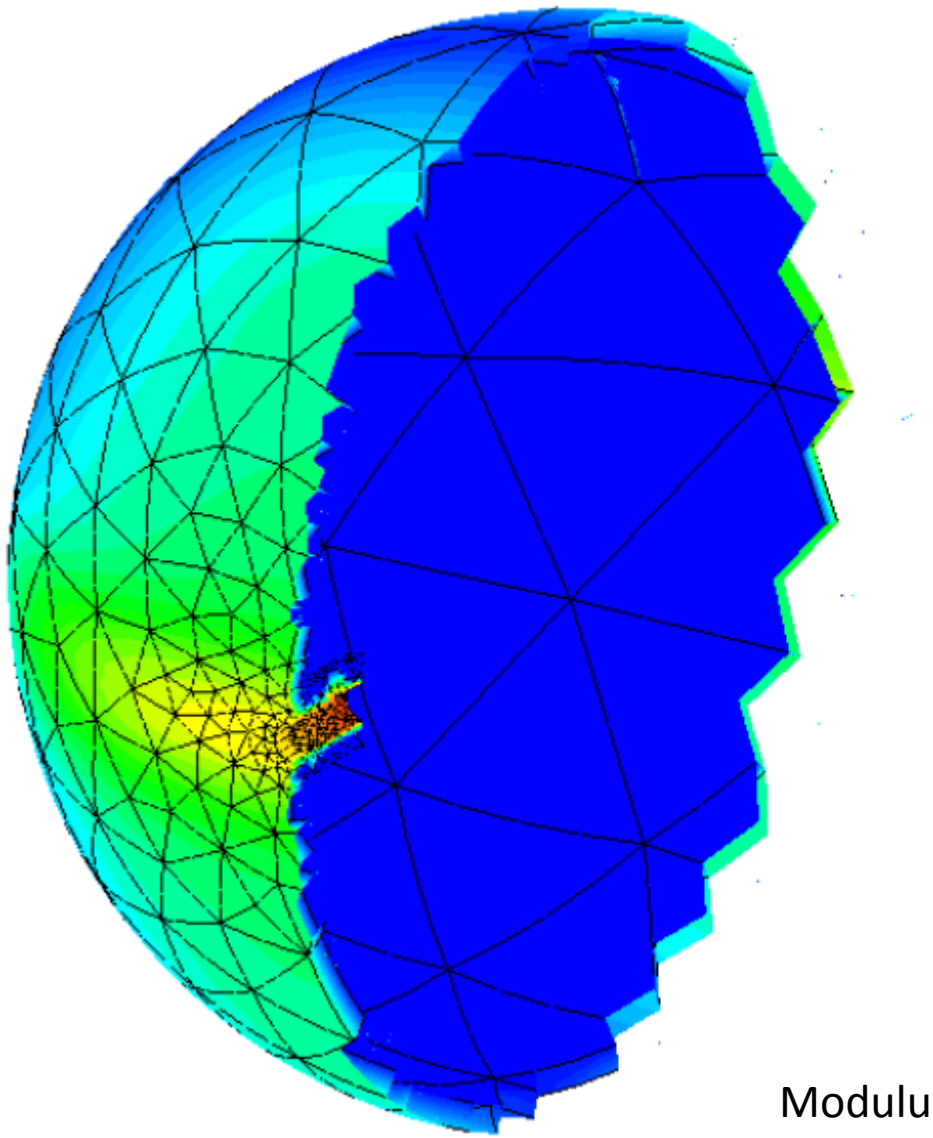


NUMERICAL RESULTS



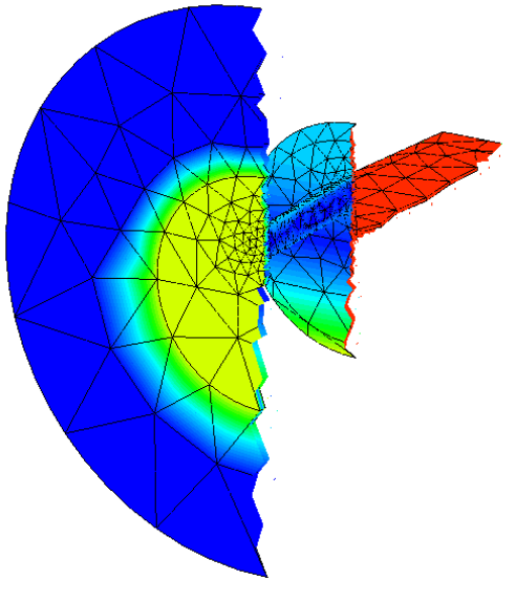
Modulus of acoustic pressure

NUMERICAL RESULTS

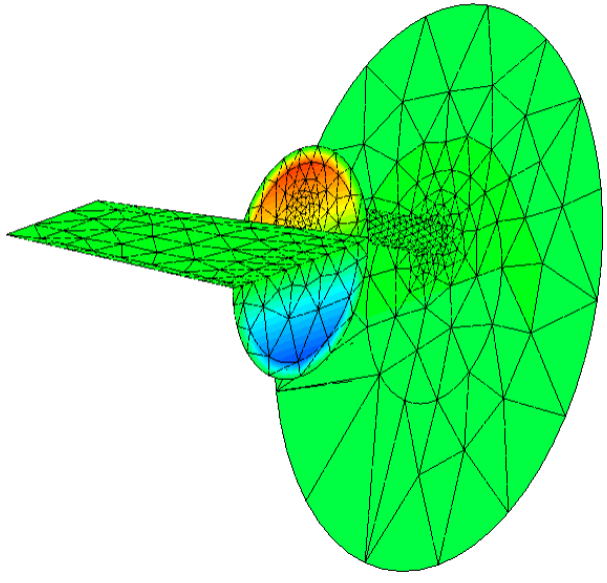


Modulus of acoustic pressure

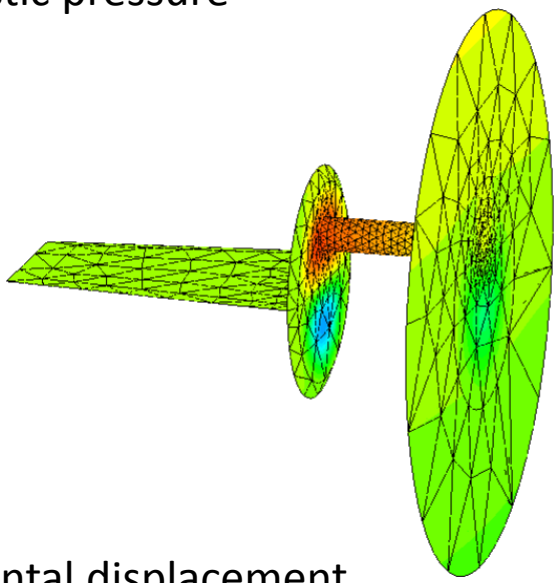
NUMERICAL RESULTS



Modulus of acoustic pressure

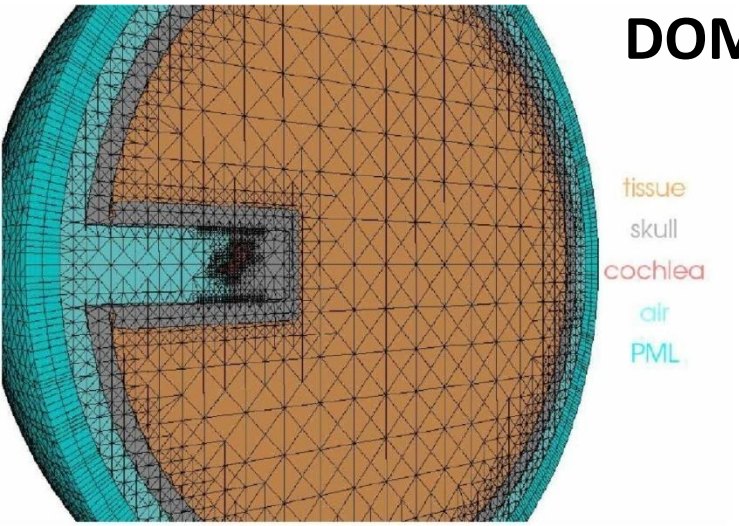


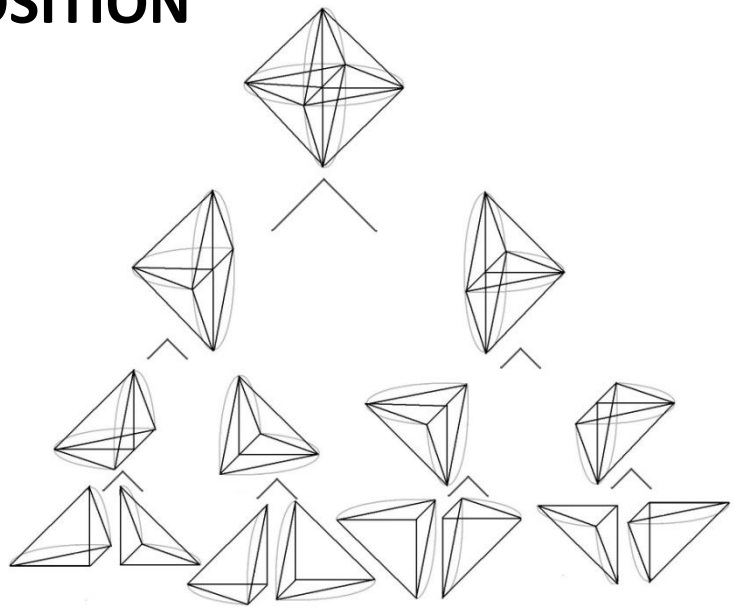
Real part of horizontal displacement



Real part of horizontal displacement

DOMAIN DECOMPOSITION

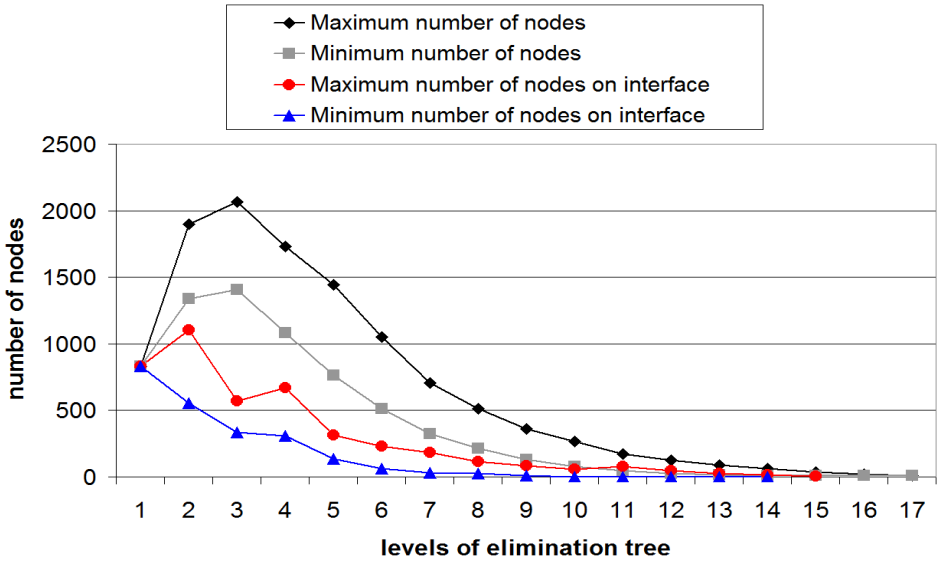




```
direction = (1,0,0)
nbeg = 1; nend = number_of_elements;
call bisection(direction,nbeg,nend,nlevel)
```

```
recursive subroutine bisection(direction,ibeg,iend,ilevel)
call sort_elements_along_direction(direction,ibeg,iend)
call rotate_direction(direction)
ihalf = (iend+ibeg)/2
if(ihalf>ibeg)then
   nbeg = ibeg; nhalf = ihalf; nlevel = ilevel+1;
   call bisection(direction,nbeg,nhalf,nlevel)
endif
if(ihalf<iend)then
   nend = iend; nhalf = ihalf+1; nlevel = ilevel+1;
   call bisection(direction,nhalf,nend,nlevel)
endif
```

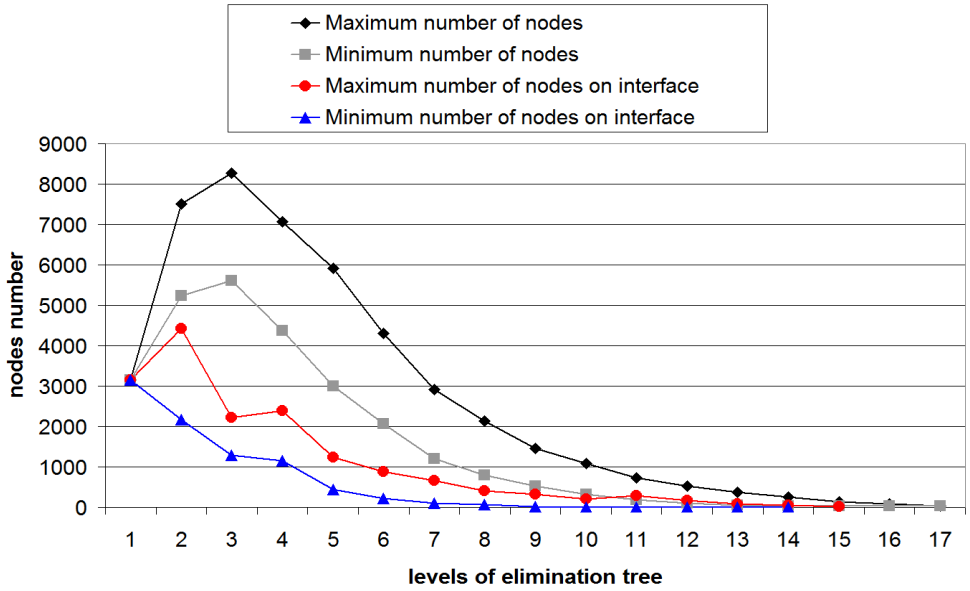
NUMBER OF NODES



Coarse mesh:
19,288 finite elements
125,754 degrees of freedom
15,321,155 non zero entries

Largest problem:

2000 (nodes) * (2000) nodes *
* 4 (equations) * 3*3 (p=3)
* 8 (complex double precision) =
1 GB

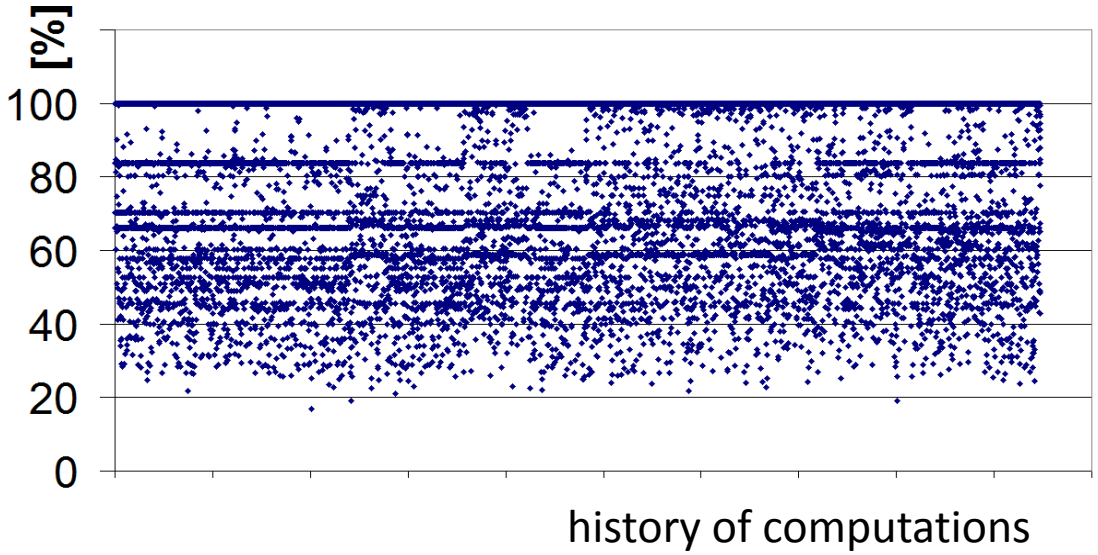


Fine mesh:
154,304 finite elements
1,058,622 degrees of freedom
119,942,591 non zero entries

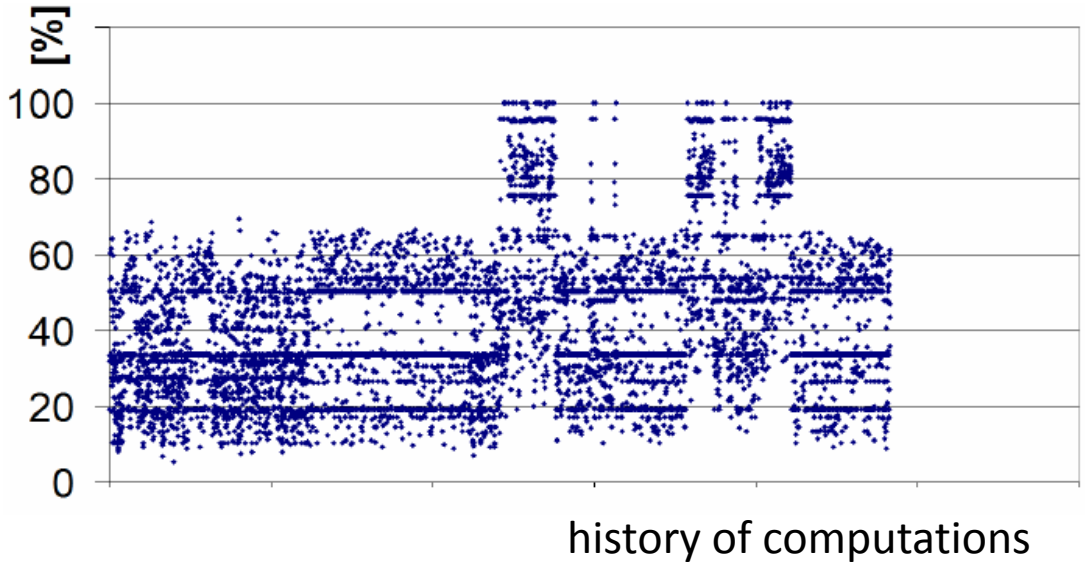
Largest problem:

8000 (nodes) * 8000 (nodes) *
* 4 (equations) * 4*4 (p=4)
* 8 (complex double precision) =
32 GB

SPARSITY OF LINEAR SYSTEMS



Coarse mesh:
19,288 finite elements
125,754 degrees of freedom
15,321,155 non zero entries
polynomial order p=3

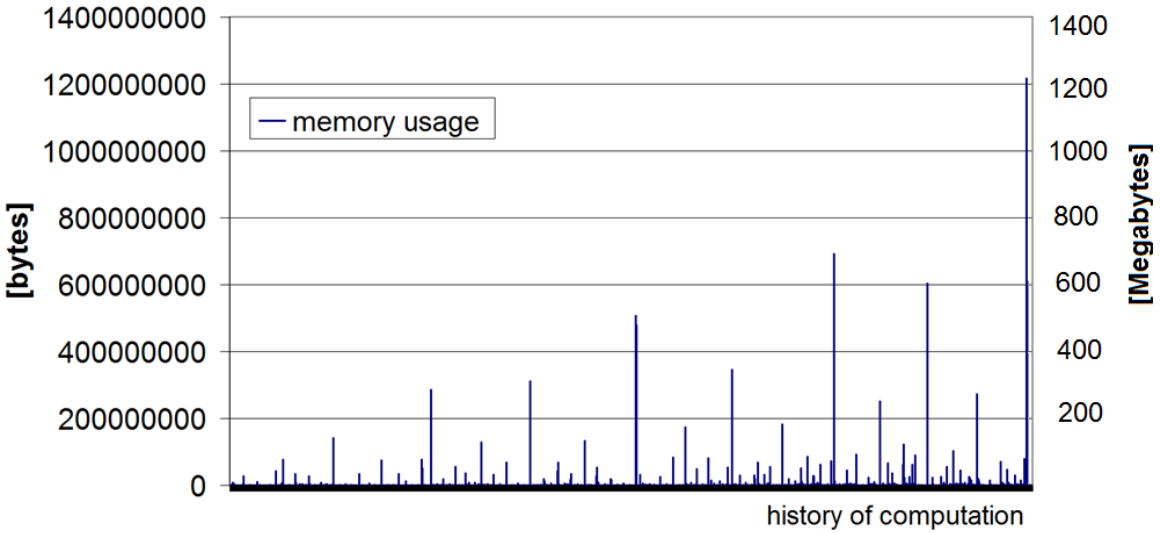


Fine mesh:
154,304 finite elements
1,058,622 degrees of freedom
119,942,591 non zero entries
polynomial order p=4

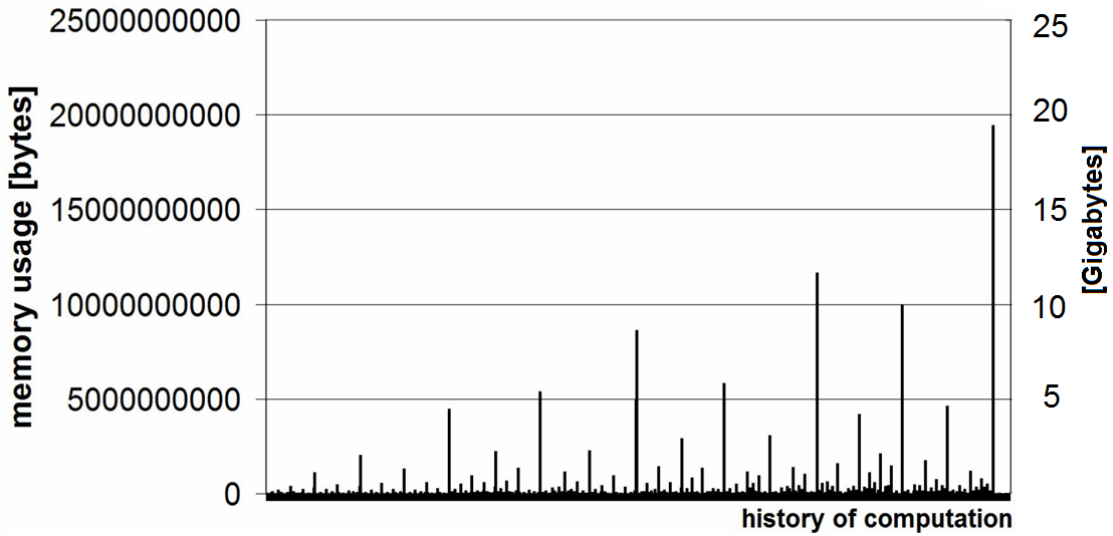
PARALLEL OUT OF CORE SOLVER

```
function out_of_core(node, proc, iret)
if node is a leaf then
    generate local system assigned to node; eliminate internal nodes
else
    loop through son_node's
    if proc is assigned to son_node then
            iret =0; call out_of_core(son_node,proc,iret)
                    if iret==1 then return
                    compute schur1 complement at son_node
                    dumpout and deallocate the system from son_node
                    if proc is 1st proc assigned to 2nd son of node then
                    BUFFER = schur1; deallocate schur1; iret =1; return
                    else if proc is 1st proc assigned to node then
                            dumpout schur1; deallocate(schur1)
                            dest_proc = 1st proc assigned to 2nd son of node
                            mpi_send("dumped out",dest_proc)
    end loop
    if proc is 1st proc assigned to node then
            proc_source = 1st proc assigned to 2nd son of node
            mpi_recv("dumped out",proc_source)
            dumpin previously dumped out schur1
            merge schur1,2 into new system at node
            deallocate(schur1,schur2)
            find nodes to eliminate at node
            compute schur complement at node; dumpout the system
endif
```

NUMBER OF NODES

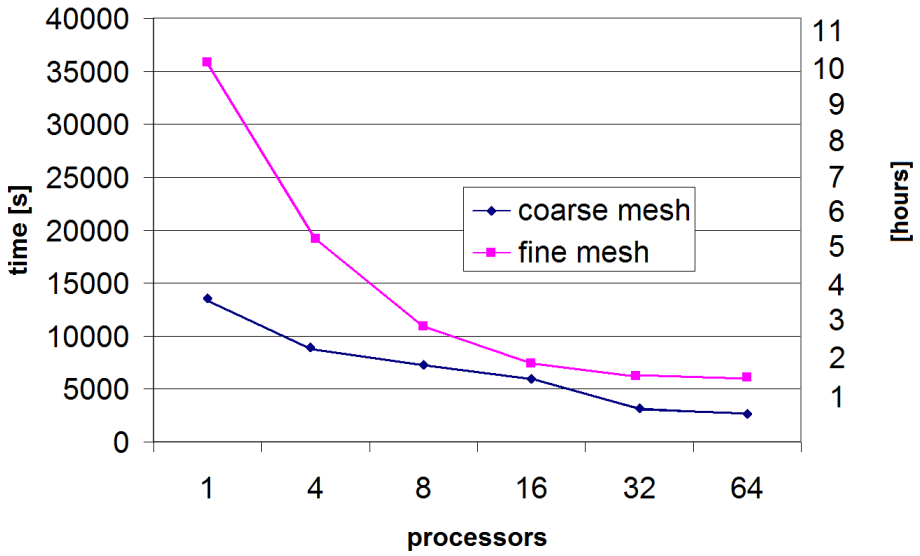


Coarse mesh:
19,288 finite elements
125,754 degrees of freedom
15,321,155 non zero entries
MUMPS memory: 922 MB

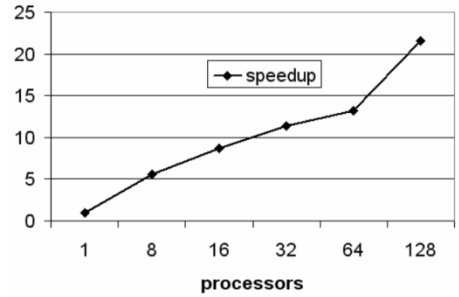


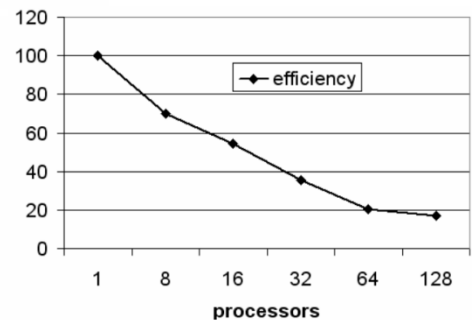
Fine mesh:
154,304 finite elements
1,058,622 degrees of freedom
119,942,591 non zero entries
MUMPS memory: 36 GB

SCALABILITY OF THE PARALLEL OUT OF CORE SOLVER

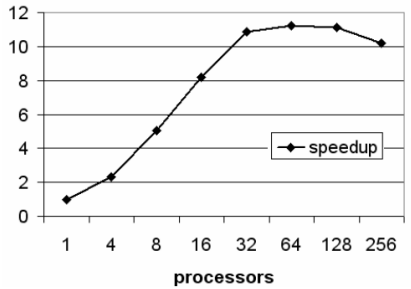


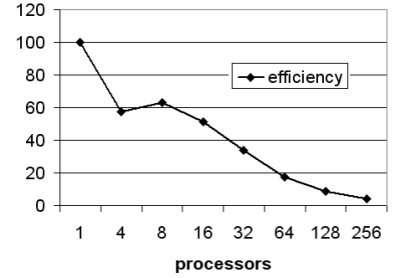
Coarse mesh





Fine mesh





Scalability is limited by the performance of the parallel file system

PAPERS

Maciej Paszyński,

MINIMIZING THE MEMORY USAGE WITH PARALLEL OUT-OF-CORE MULTI-FRONTAL DIRECT SOLVER

Computer Assisted Mechanics and Enginering Science, 20, 1 (2013) 15-41