diff.diff

```
diff --git a/FEM/Output.cpp b/FEM/Output.cpp
index 8874dbd..1b5179b 100644
--- a/FEM/Output.cpp
+++ b/FEM/Output.cpp
@@ -13,6 +13,8 @@
#include <iostream>
#include <string>

+#include <cfloat> // DBL_EPSILON
+
#include "Configure.h"

#include "FEMIO/GeoIO.h"
@@ -69,6 +71,11 @@ COutput::COutput() :
vtk = NULL; //NW
tecplot_zone_share = false; // 10.2012. WW
VARIABLESHARING = false; //BG
+#if defined(USE_PETSC) || defined(USE_MPI) //|| defined(other parallel libs)//01.3014. WW
+    int_disp = 0;
+    offset = 0;
+    domain_output_counter = 0;
+#endif
}

COutput::COutput(size_t id) :
@@ -80,6 +87,10 @@ COutput::COutput(size_t id) :
vtk = NULL; //NW
tecplot_zone_share = false; // 10.2012. WW
VARIABLESHARING = false; //BG
+#if defined(USE_PETSC) || defined(USE_MPI) //|| defined(other parallel libs)//01.3014. WW
+    int_disp = 0;
+ domain_output_counter = 0;
+#endif
}
#if defined(USE_PETSC) || defined(USE_MPI) //|| defined(other parallel libs)//03.3012. WW
void COutput::setMPI_Info(const int rank, const int size, std::string rank_str)
@@ -132,6 +143,15 @@ void COutput::init()

setInternalVarialbeNames(m_msh); //NW

+    // For binary output of the domain data
+#if defined(USE_PETSC) // || defined(other solver libs)//01.3014. WW
+    if(getGeoType() == GEOLIB::GEODOMAIN)
+    {
```

```
+      //dat_type_name = "BINARY";
+       setDataArrayDisp();
+    }
+#endif
+
}

COutput::~COutput()
@@ -843,6 +863,235 @@ void COutput::NODWriteDOMDataTEC()
}
}

+/*
+   Set data array displacement for parallel output
+
+    WW  12.2013
+*/
+#if defined(USE_PETSC) // || defined(other solver libs)//01.2014. WW
+void COutput::setDataArrayDisp()
+{
+
+   if(getGeoType() != GEOLIB::GEODOMAIN)
+      return;
+
+   //   MPI_Barrier (MPI_COMM_WORLD);
+
+   //
+   int *i_cnt;
+   int *i_disp;
+   int *i_recv;
+
+   i_cnt =  new int[msize];
+   i_disp = new int[msize];
+   i_recv = new int[msize];
+
+   for(int i=0; i<msize; i++)
+   {
+     i_cnt[i] = 1;
+     i_disp[i] = i;
+   }
+
+   int size_local =  fem_msh_vector[0]->getNumNodesLocal();
+
+   MPI_Allgatherv(&size_local, 1, MPI_INT, i_recv, i_cnt, i_disp,
+                  MPI_INT, MPI_COMM_WORLD);
+
```

```
+   int_disp = 0;
+   for(int i=0; i<mrank; i++)
+   {
+       int_disp += i_recv[i];
+   }
+
+   delete [] i_cnt;
+   delete [] i_disp;
+   delete [] i_recv;
+   //   MPI_Barrier (MPI_COMM_WORLD);
+}
+#endif
+
+/*
+    Write variable informations of the domain
+
+    WW  12.2013
+*/
+void COutput::DomainWrite_Header()
+{
+#if defined(USE_PETSC) // || defined(other solver libs)//01.2014. WW
+
+   if(mrank != 0)
+      return;
+
+#endif
+
+   string file_name;
+
+   file_name = file_base_name +  "_" + convertProcessTypeToString(getProcessType()) + "_d
+   std::cout << "Name of the binary file for node and element data: " << file_name <<
+
+
+   if(!_new_file_opened)
+   {
+   remove(file_name.c_str());
+   }
+
+   ofstream os (file_name.data(), ios::trunc | ios::out);
+   if (!os.good())
+   {
+   return;
+   }
+
+#if defined(USE_PETSC) // || defined(other solver libs)//01.2014. WW
+   os << msize << "\n";
```

```
+#endif
+
+   m_pcs =  GetPCS();
+
+   os  << domain_output_counter  <<  "\n";
+
+   const size_t num_prim_unknowns = m_pcs->GetPrimaryVNumber();
+   const size_t num_2nd_unknowns = m_pcs->GetSecondaryVNumber();
+
+   os <<  convertProcessTypeToString(getProcessType()) << "\n";
+   os << num_prim_unknowns + num_2nd_unknowns <<  "\n";
+
+   for(size_t i=0; i < num_prim_unknowns; i++)
+   {
+      os <<  m_pcs->GetPrimaryVName(i) << " ";
+   }
+   for(size_t i=0; i < num_2nd_unknowns; i++)
+   {
+       os << m_pcs->GetSecondaryVName(i) << " ";
+   }
+   os << "\n";
+   // Write number of unknowns
+   size_t n_unknowns = 0;
+#if defined(USE_PETSC) // || defined(other solver libs)//01.2014. WW
+   n_unknowns = static_cast<size_t>(m_pcs->m_msh->getNumNodesGlobal() );
+#else
+   n_unknowns = m_pcs->m_msh->GetNodesNumber(false);
+#endif
+   os << n_unknowns << "\n";
+
+   os.close();
+
+}
+
+/*
+  WW 08.2013
+*/
+void COutput:: BinaryDomainWrite()
+{
+   string file_name;
+
+   file_name = file_base_name + "_" + convertProcessTypeToString(getProcessType()) + "_do
+   std::cout << "Name of the binary file for node and element data: " << file_name <<
+
+   domain_output_counter++;
+
```

```
+   if(!_new_file_opened)
+   {
+      remove(file_name.c_str());
+   }
+
+   m_pcs =  GetPCS();
+
+#if defined(USE_PETSC) // || defined(other solver libs)//01.2014. WW
+   MPI_Barrier (MPI_COMM_WORLD);
+
+   MPI_Offset offset_new;
+   MPI_File fh;
+   int rc = 0;
+
+   if(!_new_file_opened)
+   {
+      rc = MPI_File_open(MPI_COMM_WORLD, &file_name[0], MPI_MODE_WRONLY | MPI_MODE_CREATE,
+      offset = 0;
+   }
+   else
+   {
+       rc = MPI_File_open(MPI_COMM_WORLD, &file_name[0], MPI_MODE_WRONLY | MPI_MODE_APPEND
+   }
+   if (rc )
+   {
+
+       MPI_Finalize();
+       cout<<"Cannot open "<<file_name<<"does not exist." <<"\n";
+       exit(0);
+   }
+
+
+   //MPI_File_get_position( fh, &offset );
+   // Write time and remember the number of processes#
+   string ftype = "native";
+
+   offset_new = offset + mrank*sizeof(double);
+   MPI_File_set_view(fh, offset_new, MPI_DOUBLE, MPI_DOUBLE,  &ftype[0], MPI_INFO_NULL);
+   MPI_File_write(fh, &_time, 1, MPI_DOUBLE, MPI_STATUS_IGNORE); //_all
+   offset += msize*sizeof(double);
+
+   const size_t num_prim_unknowns = m_pcs->GetPrimaryVNumber();
+   const size_t num_2nd_unknowns = m_pcs->GetSecondaryVNumber();
+   // Write unknowns
+   size_t n_unknowns = 0;
+   n_unknowns = m_pcs->m_msh->getNumNodesLocal();
```

```
+   const int nn = m_pcs->m_msh->getNumNodesGlobal();
+
+   // Write primary unknowns
+   for(size_t i=0; i < num_prim_unknowns; i++)
+   {
+      double *node_values = m_pcs->getNodeValue_per_Variable(2*i + 1);
+      offset_new = offset + int_disp*sizeof(double);
+      MPI_File_set_view(fh, offset_new, MPI_DOUBLE, MPI_DOUBLE,  &ftype[0], MPI_INFO_NULL)
+      MPI_File_write(fh, node_values, n_unknowns, MPI_DOUBLE, MPI_STATUS_IGNORE); //_all
+      offset += nn * sizeof(double);
+   }
+
+   // Write primary unknowns
+   for(size_t i=0; i < num_2nd_unknowns; i++)
+   {
+       double *node_values = m_pcs->getNodeValue_per_Variable(2*num_prim_unknowns + i);
+       offset_new = offset + int_disp*sizeof(double);
+       MPI_File_set_view(fh, offset_new, MPI_DOUBLE, MPI_DOUBLE,  &ftype[0], MPI_INFO_NULL
+       MPI_File_write(fh, node_values, n_unknowns, MPI_DOUBLE, MPI_STATUS_IGNORE); //_all
+       offset += nn * sizeof(double);
+   }
+
+
+   MPI_File_sync( fh ) ;
+   MPI_Barrier( MPI_COMM_WORLD ) ;
+   MPI_File_sync( fh ) ;
+   MPI_File_close(&fh);
+#else
+   ofstream bin_file (file_name.data(), ios::app | ios::out | ios::binary);
+   if (!bin_file.good())
+   {
+   return;
+   }
+
+   const size_t num_prim_unknowns = m_pcs->GetPrimaryVNumber();
+   const size_t num_2nd_unknowns = m_pcs->GetSecondaryVNumber();
+   // Write unknowns
+   size_t n_unknowns = 0;
+   n_unknowns = m_pcs->m_msh->GetNodesNumber(false);
+   // Write primary unknowns
+   for(size_t i=0; i < num_prim_unknowns; i++)
+   {
+      const double *node_values = m_pcs->getNodeValue_per_Variable(2*i + 1);
+       bin_file.write((char*)(node_values), n_unknowns * sizeof(double));
+   }
+
```

```
+   // Write primary unknowns
+   for(size_t i=0; i < num_2nd_unknowns; i++)
+   {
+      const double *node_values = m_pcs->getNodeValue_per_Variable(2*num_prim_unknowns +
+      bin_file.write((char*)(node_values), n_unknowns * sizeof(double));
+
+   }
+#endif
+
+}
+
/*************************************************************************
FEMLib-Method:
Programing:
diff --git a/FEM/Output.h b/FEM/Output.h
index 8cb3f52..4cda925 100644
--- a/FEM/Output.h
+++ b/FEM/Output.h
@@ -146,6 +146,12 @@ public:
void NODWriteWaterBalancePNT(double); // 6/2012 JOD
void NODWritePointsCombined(double); // 6/2012 JOD
void CalculateThroughflow(MeshLib::CFEMesh*, std::vector<long>&, std::vector<double>&)
// 6/2012 JOD
+
+    //------------------------------------------------------------
+    /// Binary output for parallel computing. 01.2014. WW
+ void DomainWrite_Header();
+ void BinaryDomainWrite();
+    //------------------------------------------------------------

void setTime (double time) { _time = time; }
/**
@@ -238,6 +244,12 @@ private:
int mrank;
int msize;
std::string mrank_str;
+
+    int int_disp;
+    MPI_Offset offset;
+
+    void setDataArrayDisp();
#endif
+    unsigned domain_output_counter; // WW 04.2014
};
#endif // OUTPUT_H
diff --git a/FEM/fem_ele.cpp b/FEM/fem_ele.cpp
```

```
index 391f504..687f9b2 100755
--- a/FEM/fem_ele.cpp
+++ b/FEM/fem_ele.cpp
@@ -1293,13 +1293,13 @@ void ElementMatrix::AllocateMemory(CElem* ele, int type)
switch(type)
{
case 0:                                 // H || T Process
- Mass = new SymMatrix(nnodes);
+ Mass = new Matrix(nnodes, nnodes);
//        Laplace = new SymMatrix(nnodes);
Laplace = new Matrix(nnodes, nnodes);
RHS = new Vec(nnodes);
break;
case 1:                                 // HM Partioned scheme, Flow
- Mass = new SymMatrix(nnodes);
+ Mass = new Matrix(nnodes, nnodes);
//        Laplace = new SymMatrix(nnodes);
Laplace = new Matrix(nnodes, nnodes);
RHS = new Vec(nnodes);
@@ -1317,7 +1317,7 @@ void ElementMatrix::AllocateMemory(CElem* ele, int type)
CouplingA = new Matrix(dim * nnodesHQ, nnodes);
break;
case 4:                                 // HM monothlic scheme
- Mass = new SymMatrix(nnodes);
+ Mass = new Matrix(nnodes, nnodes);
//        Laplace = new SymMatrix(nnodes);
Laplace = new Matrix(nnodes, nnodes);
size = dim * nnodesHQ;
@@ -1327,7 +1327,7 @@ void ElementMatrix::AllocateMemory(CElem* ele, int type)
CouplingB = new Matrix(nnodes, dim * nnodesHQ);
break;
case 5:                                 // Mass Transport process
- Mass = new SymMatrix(nnodes);
+ Mass = new Matrix(nnodes, nnodes);
Laplace = new Matrix(nnodes, nnodes);
Advection = new Matrix(nnodes, nnodes);
Storage = new Matrix(nnodes, nnodes);
diff --git a/FEM/fem_ele.h b/FEM/fem_ele.h
index ecac040..28024fb 100644
--- a/FEM/fem_ele.h
+++ b/FEM/fem_ele.h
@@ -255,7 +255,7 @@ public:
// Allocate memory for strain coupling matrix
void AllocateMemory(CElem* ele, int type = 0);
// Set members
- void SetMass(SymMatrix* mass) { Mass = mass; }
```

```
+ void SetMass(Matrix* mass) { Mass = mass; }
void SetMass_notsym(Matrix* mass) { Mass_notsym = mass; }
void SetLaplace(Matrix* laplace) { Laplace = laplace; }
void SetStiffness(Matrix* x) { Stiffness = x; }
@@ -266,7 +266,7 @@ public:
void SetCouplingMatrixB(Matrix* cplM) {CouplingB = cplM; }
void SetRHS(Vec* rhs) {RHS = rhs; }
// Get members
- SymMatrix* GetMass() {return Mass; }
+ Matrix* GetMass() {return Mass; }
Matrix* GetMass_notsym() {return Mass_notsym; }
Matrix* GetLaplace() {return Laplace; }
Matrix* GetStiffness() {return Stiffness; }
@@ -288,7 +288,7 @@ public:
private:
//TODO in more gernal way for the case of sym and unsym. WW      SymMatrix *Mass;
//      SymMatrix *Laplace;
- SymMatrix* Mass;
+ Matrix* Mass;
Matrix* Mass_notsym;
Matrix* Laplace;
Matrix* Advection;
diff --git a/FEM/fem_ele_std.cpp b/FEM/fem_ele_std.cpp
index fb84cb4..4c4f992 100755
--- a/FEM/fem_ele_std.cpp
+++ b/FEM/fem_ele_std.cpp
@@ -1969,7 +1969,7 @@ void CFiniteElementStd::CalCoefMassMCF()
for(in = 0; in<nDF; in++)
{
arg_PV[in] = interpolate(NodalValue[in]);
- std::cout << "NB-debug: " << arg_PV[in] << "\n";
+//WW std::cout << "NB-debug: " << arg_PV[in] << "\n";
}

rho = FluidProp->Density(arg_PV);
@@ -2496,7 +2496,7 @@ void CFiniteElementStd::CalCoefLaplace(bool Gravity, int ip)
mat[i * dim + i] = SolidProp->Heat_Conductivity(TG);
}
// DECOVALEX THM1 or Curce 12.09. WW
- else if(SolidProp->GetConductModel() == 3 || SolidProp->GetConductModel() == 4)
+ else if(SolidProp->GetConductModel()%3 == 0 || SolidProp->GetConductModel() == 4)
{
// WW
PG = interpolate(NodalValC1);
@@ -4832,8 +4832,8 @@ void CFiniteElementStd::CalcMassPSGLOBAL()
fkt = GetGaussData(gp, gp_r, gp_s, gp_t);
```

```
// Compute geometry
ComputeShapefct(1);       // Linear interpolation function
-      for(in = 0; in < dof_n; in++)
- for(jn = 0; jn < dof_n; jn++)
+     for(in = 0; in < dof_n; in++)
+      for(jn = 0; jn < dof_n; jn++)
{
// Material
mat_fac = CalCoefMassPSGLOBAL(in * dof_n + jn);
@@ -4841,20 +4841,19 @@ void CFiniteElementStd::CalcMassPSGLOBAL()
// Calculate mass matrix
#if defined(USE_PETSC) // || defined(other parallel libs)//03~04.3012. WW
for (i = 0; i < act_nodes; i++)
-       {
- const int  ia = local_idx[i];
- for (j = 0; j < nnodes; j++)
-   {
-     (*Mass2)(ia + in * nnodes,j + jn *
-      nnodes) += mat_fac * shapefct[ia] *  shapefct[j];
-   }
-       }
-
+     {
+           const int  ia = local_idx[i];
+           for (j = 0; j < nnodes; j++)
+           {
+              (*Mass2)(ia + in * nnodes,j + jn *
+                        nnodes) += mat_fac * shapefct[ia] *  shapefct[j];
+           }
+        }
#else
for (i = 0; i < nnodes; i++)
for (j = 0; j < nnodes; j++)
- (*Mass2)(i + in * nnodes,j + jn *
-  nnodes) += mat_fac * shapefct[i] *  shapefct[j];
+            (*Mass2)(i + in * nnodes,j + jn *
+                       nnodes) += mat_fac * shapefct[i] *  shapefct[j];
#endif
}
}
@@ -5144,7 +5143,7 @@ void CFiniteElementStd::CalcStorage()
const int ia = local_idx[i];
for (j = 0; j < nnodes; j++)
{
- (*Storage)(ia, j) += fkt * shapefct[ia] * shapefct[j];
+                  (*Storage)(ia, j) += fkt * shapefct[ia] * shapefct[j];
```

```
}
}
#else
@@ -5198,7 +5197,7 @@ void CFiniteElementStd::CalcContent()
const int ia = local_idx[i];
for (j = 0; j < nnodes; j++)
{
- (*Content)(ia,j) += fkt * shapefct[ia] * shapefct[j];
+                  (*Content)(ia,j) += fkt * shapefct[ia] * shapefct[j];
}
}

@@ -5281,12 +5280,12 @@ void CFiniteElementStd::CalcLaplace()
else if (PcsType == P)
CalCoefLaplacePSGLOBAL(false,ishd + jn);
}
-   else if (PcsType == N)
+        else if (PcsType == N)
CalCoefLaplaceTNEQ(ishd + jn);
const int jsh = jn*nnodes;
#if defined(USE_PETSC) // || defined(other parallel libs)//03~04.3012. WW
-       //---------------------------------------------------------
-       for (i = 0; i < act_nodes; i++)
+     //---------------------------------------------------------
+     for (i = 0; i < act_nodes; i++)
{
const int ia = local_idx[i];
const int iish = ia + ish;
@@ -5295,44 +5294,42 @@ void CFiniteElementStd::CalcLaplace()
const int jjsh = j + jsh;
//  if(j>i) continue;
for (k = 0; k < dim; k++)
- {
-   const int ksh = k*nnodes + ia;
-   const int km = dim *k ;
-   for(l=0; l< dim; l++)
-     {
-       (*Laplace)(iish, jjsh) += fkt * dshapefct[ksh] \
- * mat[km + l] * dshapefct[l*nnodes+j];
-
-     }
- }
-     } // j: nodes
- } // i: nodes
+              {
+                 const int ksh = k*nnodes + ia;
```

```
+                 const int km = dim *k ;
+                 for(l=0; l< dim; l++)
+                 {
+                    (*Laplace)(iish, jjsh) += fkt * dshapefct[ksh] \
+                              * mat[km + l] * dshapefct[l*nnodes+j];

+                 }
+              }
+           } // j: nodes
+        } // i: nodes
#else
-       //-------------------------------------------------------
-       for (i = 0; i < nnodes; i++)
+        //-------------------------------------------------------
+        for (i = 0; i < nnodes; i++)
{
const int iish = i + ish;
for (j = 0; j < nnodes; j++)
-     {
+    {
const int jjsh = j + jsh;
//  if(j>i) continue;
for (k = 0; k < dim; k++)
- {
-   const int ksh = k*nnodes + i;
-   const int km = dim *k ;
-   for(l=0; l< (int)dim; l++)
-     {
-       (*Laplace)(iish, jjsh) += fkt * dshapefct[ksh] \
- * mat[km + l] * dshapefct[l*nnodes+j];
-
-     }
- }
-     } // j: nodes
+               {
+                  const int ksh = k*nnodes + i;
+                  const int km = dim *k ;
+                  for(l=0; l< (int)dim; l++)
+                  {
+                     (*Laplace)(iish, jjsh) += fkt * dshapefct[ksh] \
+                           * mat[km + l] * dshapefct[l*nnodes+j];
+                  }
+               }
+           } // j: nodes
} // i: nodes
#endif
```

```
-     }
+   }
}
-    } // //TEST OUTPUT
+   } // //TEST OUTPUT
// Laplace->Write();
}
/**************************************************************************
@@ -5967,11 +5964,11 @@ void CFiniteElementStd::CalcRHS_by_ThermalDiffusion()
{
for (j = 0; j < nnodes; j++)
{
- (*RHS)(i) -= (*Laplace)(i,j) * (NodalValC[j] + T_KILVIN_ZERO);
- (*RHS)(i) += (*Mass)(i,j) * (NodalValC1[j] - NodalValC[j]) / dt;
+ (*RHS)[i] -= (*Laplace)(i,j) * (NodalValC[j] + T_KILVIN_ZERO);
+ (*RHS)[i] += (*Mass)(i,j) * (NodalValC1[j] - NodalValC[j]) / dt;
}
eqs_rhs[cshift + eqs_number[i]]
-         += (*RHS)(i);
+         += (*RHS)[i];
}

//TEST OUTPUT
@@ -6247,7 +6244,7 @@ void CFiniteElementStd::Assemble_Gravity()
//          eqs_rhs[cshift + eqs_number[i]]
//                  += k_rel_iteration* geo_fac*NodalVal[i+ii_sh];
#endif
- (*RHS)(i + LocalShift + ii_sh) += NodalVal[i + ii_sh];
+ (*RHS)[i + LocalShift + ii_sh] += NodalVal[i + ii_sh];
}
}
//TEST OUTPUT
@@ -6316,7 +6313,7 @@ void CFiniteElementStd::Assemble_GravityMCF()
for (i = 0; i < nnodes; i++)
{
eqs_rhs[cshift + eqs_number[i]] += NodalVal[i];
- (*RHS)(i + LocalShift) += NodalVal[i];
+ (*RHS)[i + LocalShift] += NodalVal[i];
}
//RHS->Write();
}
@@ -6472,7 +6469,7 @@ void CFiniteElementStd::Assemble_Gravity_Multiphase()
eqs_rhs[cshift + eqs_number[i]]
+= k_rel_iteration * geo_fac * NodalVal[i + ii_sh];
#endif
- (*RHS)(i + LocalShift + ii_sh) += NodalVal[i + ii_sh];
```

```
+ (*RHS)[i + LocalShift + ii_sh] += NodalVal[i + ii_sh];
}
}
//TEST OUTPUT
@@ -8114,7 +8111,7 @@ void CFiniteElementStd::AssembleParabolicEquation()
#if !defined(USE_PETSC) // && !defined(other parallel libs)//03~04.3012. WW
eqs_rhs[i_sh + eqs_number[i]] += NodalVal[i + ii_sh];
#endif
- (*RHS)(i + LocalShift + ii_sh) +=  NodalVal[i + ii_sh];
+ (*RHS)[i + LocalShift + ii_sh] +=  NodalVal[i + ii_sh];
}
}
}
@@ -8126,7 +8123,7 @@ void CFiniteElementStd::AssembleParabolicEquation()
#if !defined(USE_PETSC) // && !defined(other parallel libs)//03~04.3012. WW
eqs_rhs[cshift + eqs_number[i]] += NodalVal[i];
#endif
- (*RHS)(i + LocalShift) +=  NodalVal[i];
+ (*RHS)[i + LocalShift] +=  NodalVal[i];
}
}
//
@@ -8478,7 +8475,7 @@ void CFiniteElementStd::CalcFEM_FCT()
}
AuxMatrix1->multi(NodalVal1, NodalVal);
for (int i = 0; i < nnodes; i++)
-        (*RHS)(i + LocalShift) +=  NodalVal[i];  // RHS is later added into a global RHS
+        (*RHS)[i + LocalShift] +=  NodalVal[i];  // RHS is later added into a global RHS

#else
// assemble part of RHS: b_i += 1/dt * ml_i * u_i^n
@@ -8488,7 +8485,7 @@ void CFiniteElementStd::CalcFEM_FCT()
for (int i = 0; i < nnodes; i++)
{
eqs_rhs[NodeShift[problem_dimension_dm] + eqs_number[i]] += NodalVal[i];
-        (*RHS)(i + LocalShift) +=  NodalVal[i];
+        (*RHS)[i + LocalShift] +=  NodalVal[i];
}
#endif
}
@@ -8683,7 +8680,7 @@ void CFiniteElementStd::AssembleMixedHyperbolicParabolicEquation()
#if !defined(USE_PETSC) // && !defined(other parallel libs)//03~04.3012. WW
eqs_rhs[NodeShift[problem_dimension_dm] + eqs_number[i]] += NodalVal[i];
#endif
- (*RHS)(i + LocalShift) +=  NodalVal[i];
+ (*RHS)[i + LocalShift] +=  NodalVal[i];
```

```
}
}                                        //end: femFCTmode
//------------------------------------------------------------------------
@@ -9005,7 +9002,7 @@ void CFiniteElementStd::Assemble_strainCPL(const int phase)
eqs_rhs[NodeShift[shift_index] + eqs_number[i]]
+= NodalVal[i];
#endif
- (*RHS)(i + LocalShift) +=  NodalVal[i];
+ (*RHS)[i + LocalShift] +=  NodalVal[i];
}
}
// Monolithic scheme.
@@ -9122,16 +9119,14 @@ void CFiniteElementStd::AssembleMassMatrix(int option)
// Add local matrix to global matrix
if(PcsType == V || PcsType == P)      // For DOF>1: 03.03.2009 PCH
{
- int ii_sh, jj_sh;
- long i_sh, j_sh = 0;
for(int ii = 0; ii < pcs->dof; ii++)
{
- i_sh = NodeShift[ii];
- ii_sh = ii * nnodes;
+ long i_sh = NodeShift[ii];
+ long ii_sh = ii * nnodes;
for(int jj = 0; jj < pcs->dof; jj++)
{
- j_sh = NodeShift[jj];
- jj_sh = jj * nnodes;
+ long j_sh = NodeShift[jj];
+ long jj_sh = jj * nnodes;
for(int i = 0; i < nnodes; i++)
{
for(int j = 0; j < nnodes; j++)
@@ -9265,7 +9260,7 @@ void CFiniteElementStd::Config()
// Initialize RHS
if(pcs->Memory_Type > 0)
for(i = LocalShift; (size_t)i < RHS->Size(); i++)
- (*RHS)(i) = 0.0;
+ (*RHS)[i] = 0.0;
else
(*RHS) = 0.0;
//------------------------------------------------------------------------
@@ -9631,7 +9626,7 @@ void CFiniteElementStd::Assembly(int option, int dimension)
(*Laplace) = 0.0;
if(pcs->Memory_Type > 0)
for(i = LocalShift; (size_t)i < RHS->Size(); i++)
```

```
- (*RHS)(i) = 0.0;
+ (*RHS)[i] = 0.0;
else
(*RHS) = 0.0;

@@ -9642,7 +9637,7 @@ void CFiniteElementStd::Assembly(int option, int dimension)
if(pcs->Write_Matrix)
{
for (i = 0; i < nnodes; i++)
- (*RHS)(i) = NodalVal[i];
+ (*RHS)[i] = NodalVal[i];
(*pcs->matrix_file) << "### Element: " << Index << "\n";
(*pcs->matrix_file) << "---Mass matrix: " << "\n";
Mass->Write(*pcs->matrix_file);
@@ -10813,8 +10808,16 @@ void CFiniteElementStd::Assemble_RHS_T_MPhaseFlow()
// Material
fac = fkt * CalCoef_RHS_T_MPhase(ii) / dt;
// Calculate THS
+#if defined(USE_PETSC) //|| defined (other parallel solver) //WW 04.2014
+ for (int ia = 0; ia < act_nodes; ia++)
+         {
+     const int i = local_idx[ia];
+#else
for (i = 0; i < nnodes; i++)
+ {
+#endif
NodalVal[i + ii * nnodes] += fac * shapefct[i];
+ }
}
// grad T
for(ii = 0; ii < dof_n; ii++)
@@ -10822,7 +10825,14 @@ void CFiniteElementStd::Assemble_RHS_T_MPhaseFlow()
// Material
fac = fkt * CalCoef_RHS_T_MPhase(ii + dof_n);
// Calculate THS
+#if defined(USE_PETSC) //|| defined (other parallel solver) //WW 04.2014
+ for (int ia = 0; ia < act_nodes; ia++)
+         {
+     const int i = local_idx[ia];
+#else
for (i = 0; i < nnodes; i++)
+ {
+#endif
for (j = 0; j < nnodes; j++)
for (size_t k = 0; k < dim; k++)
NodalVal[i + ii * nnodes] +=
```

```
@@ -10830,6 +10840,7 @@ void CFiniteElementStd::Assemble_RHS_T_MPhaseFlow()
dshapefct[k * nnodes +
i] * dshapefct[k * nnodes + j]
* (NodalValC1[j] + T_KILVIN_ZERO);
+ }
}
}
int ii_sh;
@@ -10843,7 +10854,7 @@ void CFiniteElementStd::Assemble_RHS_T_MPhaseFlow()
#if !defined(USE_PETSC) // && !defined(other parallel libs)//03~04.3012. WW
eqs_rhs[i_sh + eqs_number[i]] -= NodalVal[i + ii_sh];
#endif
- (*RHS)(i + LocalShift + ii_sh) -=  NodalVal[i + ii_sh];
+ (*RHS)[i + LocalShift + ii_sh] -=  NodalVal[i + ii_sh];
}
}
//
@@ -10887,11 +10898,16 @@ void CFiniteElementStd::Assemble_RHS_T_PSGlobal()
// Material
fac = fkt * CalCoef_RHS_T_PSGlobal(ii) / dt;
// Calculate THS
+#if defined(USE_PETSC) //|| defined (other parallel solver) //WW 04.2014
+ for (int ia = 0; ia < act_nodes; ia++)
+         {
+     const int i = local_idx[ia];
+#else
for (i = 0; i < nnodes; i++)
- {
+            {
+#endif
NodalVal[i + ii * nnodes] += fac * shapefct[i];
- //remove
- temp[i + ii * nnodes] += fac * shapefct[i];
+ //remove temp[i + ii * nnodes] += fac * shapefct[i];
}
}
}
@@ -10906,7 +10922,7 @@ void CFiniteElementStd::Assemble_RHS_T_PSGlobal()
#if !defined(USE_PETSC) // && !defined(other parallel libs)//03~04.3012. WW
eqs_rhs[i_sh + eqs_number[i]] -= NodalVal[i + ii_sh];
#endif
- (*RHS)(i + LocalShift + ii_sh) -=  NodalVal[i + ii_sh];
+ (*RHS)[i + LocalShift + ii_sh] -=  NodalVal[i + ii_sh];
}
}
//
```

```
@@ -10959,7 +10975,14 @@ void CFiniteElementStd::Assemble_RHS_Pc()
// Material
CalCoef_RHS_Pc(ii + dof_n);
// Calculate Pc
+#if defined(USE_PETSC) //|| defined (other parallel solver) //WW 04.2014
+ for (int ia = 0; ia < act_nodes; ia++)
+         {
+     const int i = local_idx[ia];
+#else
for (i = 0; i < nnodes; i++)
+ {
+#endif
for (j = 0; j < nnodes; j++)
for (size_t k = 0; k < dim; k++)
for (size_t l = 0; l < dim; l++)
@@ -10972,6 +10995,7 @@ void CFiniteElementStd::Assemble_RHS_Pc()
dshapefct[l * nnodes + j]
* NodalVal1[j +
dof_n];
+             }
}
}

@@ -10989,7 +11013,7 @@ void CFiniteElementStd::Assemble_RHS_Pc()
#if !defined(USE_PETSC) // && !defined(other parallel libs)//03~04.3012. WW
eqs_rhs[i_sh + eqs_number[i]] += NodalVal[i + ii_sh];
#endif
- (*RHS)(i + LocalShift + ii_sh) +=  NodalVal[i + ii_sh];
+ (*RHS)[i + LocalShift + ii_sh] +=  NodalVal[i + ii_sh];
}
}
//
@@ -11063,14 +11087,24 @@ void CFiniteElementStd::Assemble_RHS_LIQUIDFLOW()
//  Compute RHS+=int{N^T alpha_T dT/dt}
//------------------------------------------------------------
const double fac = eff_thermal_expansion * dT / dt / time_unit_factor;//WX:bug fixed
+#if defined(USE_PETSC) //|| defined (other parallel solver) //WW 04.2014
+        for (int ia = 0; ia < act_nodes; ia++)
+     {
+    const int i = local_idx[ia];
+#else
for (int i = 0; i < nnodes; i++)
+        {
+#endif
NodalVal[i] += gp_fkt * fac * shapefct[i];
+        }
```

```
}
int i_sh = NodeShift[dm_shift];
for (int i = 0; i < nnodes; i++)
{
+#if !defined(USE_PETSC) // && !defined(other parallel libs)//03~04.3012. WW
eqs_rhs[i_sh + eqs_number[i]] += NodalVal[i];
-        (*RHS)(i + LocalShift) +=  NodalVal[i];
+#endif
+        (*RHS)[i + LocalShift] +=  NodalVal[i];
}
}

@@ -11151,8 +11185,16 @@ void CFiniteElementStd::Assemble_RHS_M()
fac *= SolidProp->biot_const;

// Calculate MHS
- for (i = 0; i < nnodes; i++)
+#if defined(USE_PETSC) //|| defined (other parallel solver) //WW 04.2014
+            for (int ia = 0; ia < act_nodes; ia++)
+            {
+              const int i = local_idx[ia];
+#else
+            for (i = 0; i < nnodes; i++)
+            {
+#endif
NodalVal[i + ii * nnodes] += fac * shapefct[i];
+     }
}
}
//
@@ -11167,7 +11209,7 @@ void CFiniteElementStd::Assemble_RHS_M()
#if !defined(USE_PETSC) // && !defined(other parallel libs)//03~04.3012. WW
eqs_rhs[i_sh + eqs_number[i]] -= NodalVal[i + ii_sh];
#endif
- (*RHS)(i + LocalShift + ii_sh) -=  NodalVal[i + ii_sh];
+ (*RHS)[i + LocalShift + ii_sh] -=  NodalVal[i + ii_sh];
}
}
setOrder(1);
@@ -11272,7 +11314,7 @@ void CFiniteElementStd::Assemble_RHS_AIR_FLOW()
#if !defined(USE_PETSC) // && !defined(other parallel libs)//03~04.3012. WW
eqs_rhs[i_sh + eqs_number[i]] -= NodalVal[i + ii_sh];
#endif
- (*RHS)(i + LocalShift + ii_sh) -=  NodalVal[i + ii_sh];
+ (*RHS)[i + LocalShift + ii_sh] -=  NodalVal[i + ii_sh];
}
```

```
}
}
@@ -11350,7 +11392,7 @@ void CFiniteElementStd::Assemble_RHS_HEAT_TRANSPORT()
#if !defined(USE_PETSC) // && !defined(other parallel libs)//03~04.3012. WW
eqs_rhs[i_sh + eqs_number[i]] -= NodalVal[i + ii_sh];
#endif
- (*RHS)(i + LocalShift + ii_sh) -=  NodalVal[i + ii_sh];
+ (*RHS)[i + LocalShift + ii_sh] -=  NodalVal[i + ii_sh];
}
}
}
@@ -11536,7 +11578,7 @@ void CFiniteElementStd::Assemble_RHS_HEAT_TRANSPORT2()
#if !defined(USE_PETSC) // && !defined(other parallel libs)//03~04.3012. WW
eqs_rhs[i_sh + eqs_number[i]] -= NodalVal[i + ii_sh];
#endif
- (*RHS)(i + LocalShift + ii_sh) -=  NodalVal[i + ii_sh];
+ (*RHS)[i + LocalShift + ii_sh] -=  NodalVal[i + ii_sh];
}
}
}
@@ -11580,7 +11622,7 @@ void CFiniteElementStd::Assemble_RHS_HEAT_TRANSPORT2()
for (i=0;i<nnodes;i++)
{
eqs_rhs[i_sh + eqs_number[i]] += NodalVal[i+ii_sh];
- (*RHS)(i+LocalShift+ii_sh) +=  NodalVal[i+ii_sh];
+ (*RHS)[i+LocalShift+ii_sh] +=  NodalVal[i+ii_sh];
//std::cout << eqs_rhs[i_sh + eqs_number[i]] << " " << (*RHS)(i+LocalShift+ii_sh) << "
}
}
@@ -11763,7 +11805,7 @@ void CFiniteElementStd::AssembleRHSVector()
#else
pcs->eqs->b[NodeShift[problem_dimension_dm] + eqs_number[i]] += NodalVal[i];
#endif
- (*RHS)(i + LocalShift) += NodalVal[i];
+ (*RHS)[i + LocalShift] += NodalVal[i];
}
//------------------------------------------------------------------------
//RHS->Write();
@@ -11841,7 +11883,7 @@ void CFiniteElementStd::AssembleCapillaryEffect()
#else
pcs->eqs->b[NodeShift[problem_dimension_dm] + eqs_number[i]] += NodalVal[i];
#endif
- (*RHS)(i + LocalShift) += NodalVal[i];
+ (*RHS)[i + LocalShift] += NodalVal[i];
}
//------------------------------------------------------------------------
```

```
//RHS->Write();
diff --git a/FEM/fem_ele_vec.cpp b/FEM/fem_ele_vec.cpp
index 9e7c863..19a0ca6 100755
--- a/FEM/fem_ele_vec.cpp
+++ b/FEM/fem_ele_vec.cpp
@@ -102,7 +102,7 @@ CFiniteElementVec::CFiniteElementVec(process::CRFProcessDeformation*
Idx_dm1[2] = Idx_dm1[2] + 1;
Idx_Vel[2] = pcs->GetNodeValueIndex("VELOCITY_DM_Z");
}
- Mass = new SymMatrix(20);
+ Mass = new Matrix(20, 20);
dAcceleration = new Vec(60);

beta2 = dm_pcs->m_num->GetDynamicDamping_beta2();
@@ -584,7 +584,7 @@ void CFiniteElementVec::SetMemory()

if(dynamic)
{
- Mass->LimitSize(nnodesHQ);
+ Mass->LimitSize(nnodesHQ, nnodesHQ);
dAcceleration->LimitSize(nnodesHQ * dim);
}
}
@@ -882,7 +882,7 @@ void CFiniteElementVec::ComputeMatrix_RHS(const double fkt,
// Local assembly of A*u=int(B^t*sigma) for Newton-Raphson method
for (j = 0; j < ele_dim; j++)
for (k = 0; k < ns; k++)
- (*RHS)(j * nnodesHQ + ia) +=
+ (*RHS)[j * nnodesHQ + ia] +=
(*tmp_B_matrix_T)(j,k) * (dstress[k] - stress0[k]) * fkt;
//TEST              (*B_matrix_T)(j,k)*dstress[k]*fkt;
if(PreLoad == 11)
@@ -1007,7 +1007,7 @@ void CFiniteElementVec::ComputeMatrix_RHS(const double fkt,
#else
const int ka = k;
#endif
- (*RHS)(i + ka) += coeff * shapefctHQ[ka];
+ (*RHS)[i + ka] += coeff * shapefctHQ[ka];
//        (*RHS)(i+ka) += LoadFactor * rho * smat->grav_const * shapefctHQ[ka] * fkt;
}
}
@@ -2090,13 +2090,15 @@ void CFiniteElementVec::GlobalAssembly_RHS()
}
else
{
- /*
```

```
+
+                if(pcs->Neglect_H_ini==2)
+ {
for (i=0;i<nnodes;i++)
{
-    if(AuxNodal0[i]<0.0)
-       AuxNodal0[i] = 0.;
+           AuxNodal0[i] -= h_pcs->GetNodeValue(nodes[i],idx_p1_ini);
+           AuxNodal2[i] -= h_pcs->GetNodeValue(nodes[i],idx_p2_ini);
}
-  */
+ }

PressureC->multi(AuxNodal2, AuxNodal1, LoadFactor);
PressureC_S->multi(AuxNodal0, AuxNodal1, -1.0 * LoadFactor);
@@ -2138,7 +2140,7 @@ void CFiniteElementVec::GlobalAssembly_RHS()
PressureC->multi(AuxNodal, AuxNodal1);
}
for (i = 0; i < dim_times_nnodesHQ; i++)
- (*RHS)(i) -= fabs(biot) * AuxNodal1[i];
+ (*RHS)[i] -= fabs(biot) * AuxNodal1[i];
}                                       // End if partioned

// If dymanic
@@ -2146,7 +2148,7 @@ void CFiniteElementVec::GlobalAssembly_RHS()
for (size_t i = 0; i < dim; i++)
for (j = 0; j < nnodesHQ; j++)
for (k = 0; k < nnodesHQ; k++)
- (*RHS)(i * nnodesHQ + j) += (*Mass)(j,k) * (
+ (*RHS)[i * nnodesHQ + j] += (*Mass)(j,k) * (
(*dAcceleration)(i * nnodesHQ + k)
+ a_n[nodes[k] + NodeShift[i]]);

@@ -2154,7 +2156,7 @@ void CFiniteElementVec::GlobalAssembly_RHS()
#if !defined(USE_PETSC) // && !defined(other parallel libs)//06.2013. WW
for (size_t i = 0; i < dim; i++)
for (j = 0; j < nnodesHQ; j++)
- b_rhs[eqs_number[j] + NodeShift[i]] -= (*RHS)(i * nnodesHQ + j);
+ b_rhs[eqs_number[j] + NodeShift[i]] -= (*RHS)[i * nnodesHQ + j];
#endif

//WX:07.2011 if not on excav boundary, RHS=0
diff --git a/FEM/fem_ele_vec.h b/FEM/fem_ele_vec.h
index 91ec5d3..7dbca74 100755
--- a/FEM/fem_ele_vec.h
+++ b/FEM/fem_ele_vec.h
```

```
@@ -159,7 +159,7 @@ private:
Matrix* PressureC;
Matrix* PressureC_S;                    // Function of S
Matrix* PressureC_S_dp;                    // Function of S and ds_dp
- SymMatrix* Mass;                         // For dynamic analysis
+ Matrix* Mass;                          // For dynamic analysis
Vec* RHS;
// Global RHS. 08.2010. WW
double* b_rhs;
diff --git a/FEM/matrix_class.cpp b/FEM/matrix_class.cpp
index b9e071d..6810b9d 100644
--- a/FEM/matrix_class.cpp
+++ b/FEM/matrix_class.cpp
@@ -28,7 +28,7 @@ namespace Math_Group
// Constructors
Matrix::Matrix(size_t rows, size_t cols) :
nrows (rows), nrows0 (rows), ncols (cols), ncols0 (cols),
- size (nrows * ncols), data (new double[size]), Sym (false)
+ size (nrows * ncols), data (new double[size])
{
for(size_t i = 0; i < size; i++)
data[i] = 0.0;
@@ -36,12 +36,12 @@ Matrix::Matrix(size_t rows, size_t cols) :

Matrix::Matrix() :
nrows (0), nrows0 (0), ncols (0), ncols0 (0),
- size (nrows * ncols), data (NULL), Sym (false)
+ size (nrows * ncols), data (NULL)
{}

Matrix::Matrix(const Matrix& m) :
nrows (m.nrows), nrows0 (m.nrows), ncols (m.ncols), ncols0 (m.ncols),
- size (nrows * ncols), data (new double[size]), Sym (m.Sym)
+ size (nrows * ncols), data (new double[size])
{
for(size_t i = 0; i < size; i++)
data[i] = m.data[i];
@@ -55,7 +55,6 @@ void Matrix::resize(size_t rows, size_t cols)
data = NULL;
}

- Sym = false;
nrows = rows;
ncols = cols;
nrows0 = rows;
@@ -71,6 +70,7 @@ Matrix::~Matrix()
```

```
delete [] data;
data = NULL;
}
+
// 06.2010. WW
void Matrix::ReleaseMemory()
{
@@ -78,70 +78,28 @@ void Matrix::ReleaseMemory()
data = NULL;
}

-void Matrix::operator= (double a)
-{
- for(size_t i = 0; i < size; i++)
- data[i] = a;
-}
-void Matrix::operator *= (double a)
-{
- for(size_t i = 0; i < size; i++)
- data[i] *= a;
-}
-void Matrix::operator /= (double a)
-{
- for(size_t i = 0; i < size; i++)
- data[i] /= a;
-}
-void Matrix::operator += (double a)
-{
- for(size_t i = 0; i < size; i++)
- data[i] += a;
-}
-//
-void Matrix::operator = (const Matrix& m)
+/*
+void Matrix::operator= (const SymMatrix& m)
{
-#ifdef gDEBUG
- if(nrows != m.Rows() || ncols != m.Cols())
- {
- std::cout << "\n The sizes of the two matrices are not matched" << "\n";
- abort();
- }
-#endif
- for(size_t i = 0; i < nrows; i++)
- for(size_t j = 0; j < ncols; j++)
- data[i * ncols + j] = m(i,j);
```

```
-}
+    const double *m_data = m.getEntryArray_const();

-//
-void Matrix::operator += (const Matrix& m)
-{
-#ifdef gDEBUG
- if(nrows != m.Rows() || ncols != m.Cols())
- {
- std::cout << "\n The sizes of the two matrices are not matched" << "\n";
- abort();
- }
-#endif
- for(size_t i = 0; i < nrows; i++)
- for(size_t j = 0; j < ncols; j++)
- data[i * ncols + j] += m(i,j);
-}
+    for(size_t i = 0; i < nrows; i++)
+    {
+        double *row_data = &data[i * ncols] ;
+        const double *row_data_m = &data[(i * (i + 1) / 2)] ;
+
+        // diagonal
+        row_data[i] = row_data_m[i];

-//
-void Matrix::operator -= (const Matrix& m)
-{
-#ifdef gDEBUG
- if(nrows != m.Rows() || ncols != m.Cols()) //Assertion, will be removed
- {
- std::cout << "\n The sizes of the two matrices are not matched" << "\n";
- abort();
- }
-#endif
- for(size_t i = 0; i < nrows; i++)
- for(size_t j = 0; j < ncols; j++)
- data[i * ncols + j] -= m(i,j);
+        for(size_t j = 0; j<ncols; j++)
+        {
+           row_data[j] = row_data_m[j];
+           data[j*ncols + i] = row_data_m[j];
+        }
+    }
}
+*/
```

```
+
//
void Matrix::GetTranspose(Matrix& m)
{
@@ -153,12 +111,19 @@ void Matrix::GetTranspose(Matrix& m)
}
#endif

- for(size_t i = 0; i < m.Rows(); i++)
- for(size_t j = 0; j < m.Cols(); j++)
- //          m(i,j) = data[j*ncols+i];
- m(i,j) = (*this)(j,i);
+    double *m_data = m.getEntryArray();
+    const size_t mrows = m.Rows();
+    const size_t mcols = m.Cols();
+ for(size_t i = 0; i < mrows; i++)
+ {
+        double *row_m_data = &m_data[i * mcols] ;
+ for(size_t j = 0; j < mcols; j++)
+ {
+            row_m_data[j] = data[j*ncols+i];
+ }
+ }
}
-//
+
// m_results = this*m. m_results must be initialized
void Matrix::multi(const Matrix& m, Matrix& m_result, double fac)
{
@@ -169,16 +134,28 @@ void Matrix::multi(const Matrix& m, Matrix& m_result, double fac)
abort();
}
#endif
- for(size_t i = 0; i < m_result.Rows(); i++)
- for(size_t j = 0; j < m_result.Cols(); j++)
- {
- if(Sym && (j > i))
- continue;
- // m_result(i,j) = 0.0;
- for(size_t k = 0; k < ncols; k++)
- //            m_result(i,j) += fac*data[i*ncols+k]*m(k,j);
- m_result(i,j) += fac * (*this)(i,k) * m(k,j);
- }
+   const double *m_data = m.getEntryArray_const();
+    const size_t mrows = m.Rows();
+    const size_t mcols = m.Cols();
```

```
+    double *r_data = m_result.getEntryArray();
+    const size_t r_rows = m_result.Rows();
+    const size_t r_cols = m_result.Cols();
+
+    for(size_t i = 0; i < r_rows; i++)
+    {
+        const double *row_data = &data[i * ncols] ;
+        double *r_row_data = &r_data[i * r_cols] ;
+        for(size_t j = 0; j < r_cols; j++)
+        {
+            // r_row_data[j] = 0.0;
+            double val = 0.;
+            for(size_t k = 0; k < ncols; k++)
+            {
+                val += row_data[k] * m_data[k*mcols + j];
+            }
+            r_row_data[j] += val * fac;
+        }
+    }
}

//
@@ -196,8 +173,6 @@ void Matrix::multi(const Matrix& m1, const Matrix& m2, Matrix& m_resu
for(size_t i = 0; i < m_result.Rows(); i++)
for(size_t j = 0; j < m_result.Cols(); j++)
{
- if(Sym && (j > i))
- continue;
//m_result(i,j) = 0.0;
for(size_t k = 0; k < ncols; k++)
for(size_t l = 0; l < m2.Rows(); l++)
@@ -208,9 +183,16 @@ void Matrix::multi(const Matrix& m1, const Matrix& m2, Matrix& m_res
// vec_result = This*vec. vec_result must be  initialized
void Matrix::multi(const double* vec, double* vec_result, double fac)
{
- for(int i = 0; (size_t)i < nrows; i++)
- for(int j = 0; (size_t)j < ncols; j++)
- vec_result[i] += fac * (*this)(i,j) * vec[j];
+ for(size_t  i = 0; i < nrows; i++)
+ {
+        double val = 0.;
+        const double *row_data = &data[i * ncols] ;
+ for(size_t j = 0; j < ncols; j++)
+ {
+ val += row_data[j] * vec[j];
+ }
```

```
+        vec_result[i] += fac * val;
+ }
}

double& Matrix::operator() (size_t i, size_t j) const
@@ -286,9 +268,8 @@ void Matrix::Read_BIN(std::fstream& is)
//-----------------------------------------------------
// Symmetrical matrix
SymMatrix::SymMatrix(size_t dim) :
- Matrix(0)
+ Matrix()
{
- Sym = true;
nrows = ncols = dim;
size = (int)nrows * (nrows + 1) / 2;
data = new double[size];
@@ -297,19 +278,8 @@ SymMatrix::SymMatrix(size_t dim) :
data[i] = 0.0;
}

-SymMatrix::SymMatrix() : Matrix(0)
-{
- Sym = true;
- nrows = 0;
- ncols = 0;
- nrows0 = 0;
- ncols0 = 0;
- size = 0;
- data = 0;
-}
-SymMatrix::SymMatrix(const SymMatrix& m) : Matrix(0)
+SymMatrix::SymMatrix(const SymMatrix& m) : Matrix()
{
- Sym = m.Sym;
nrows = m.nrows;
ncols = m.ncols;
nrows0 = m.nrows0;
@@ -317,7 +287,7 @@ SymMatrix::SymMatrix(const SymMatrix& m) : Matrix(0)
size = m.size;
data = new double[size];
for(size_t i = 0; i < size; i++)
- data[i] = 0.0;
+ data[i] = m.data[i];
}

void SymMatrix::resize(size_t dim)
```

```
@@ -328,7 +298,6 @@ void SymMatrix::resize(size_t dim)
data = NULL;
}

- Sym = true;
nrows = ncols = dim;
size = (int) nrows * (nrows + 1) / 2;
data = new double[size];
@@ -337,119 +306,117 @@ void SymMatrix::resize(size_t dim)
data[i] = 0.0;
}

-void SymMatrix::operator = (double a)
-{
- for(size_t i = 0; i < size; i++)
- data[i] = a;
-}
-void SymMatrix::operator *= (double a)
-{
- for(size_t i = 0; i < size; i++)
- data[i] *= a;
-}
-void SymMatrix::operator += (double a)
-{
- for(size_t i = 0; i < size; i++)
- data[i] += a;
-}
-
//
-void SymMatrix::operator = (const SymMatrix& m)
+void SymMatrix::LimitSize(size_t dim)
{
#ifdef gDEBUG
- if(nrows != m.Rows() || ncols != m.Cols())
+ if(dim > nrows0)
{
- std::cout << "\n The sizes of the two matrices are not matched" << "\n";
+ std::cout << "\n Given size exceeds the original size of the matrix" << "\n";
abort();
}
#endif
- size_t id = 0;
- for (size_t i = 0; i < nrows; i++)
- for (size_t j = 0; j < ncols; j++)
- {
- if (j > i)
```

```
- continue;
- id = i * (i + 1) / 2 + j; // temporary
- data[id] = m(i, j);
- }
+ nrows = ncols = dim;
+ size = nrows * (nrows + 1) / 2;
}

-//
-void SymMatrix::operator += (const SymMatrix& m)
+// m_results = this*m. m_results must be initialized
+void SymMatrix::multi(const Matrix& m, Matrix& m_result, double fac)
{
#ifdef gDEBUG
- if(nrows != m.Rows())
+ if(ncols != m.Rows() && nrows != m_result.Rows() && m.Cols() != m_result.Cols())
{
std::cout << "\n The sizes of the two matrices are not matched" << "\n";
abort();
}
#endif
- size_t id = 0;
- for (size_t i = 0; i < nrows; i++)
- for (size_t j = 0; j < ncols; j++)
- {
- if (j > i)
- continue;
- id = i * (i + 1) / 2 + j; // temporary
- data[id] += m(i, j);
- }
+    const size_t mrows = m.Rows();
+    const size_t mcols = m.Cols();
+    double *r_data = m_result.getEntryArray();
+    const size_t r_rows = m_result.Rows();
+    const size_t r_cols = m_result.Cols();
+
+    for(size_t i = 0; i < r_rows; i++)
+    {
+        const double *row_data = &data[(i * (i + 1) / 2)] ;
+        double *row_data_r = &r_data[i * r_cols] ;
+
+        for(size_t j = 0; j<r_cols; j++)
+        {
+            // row_data_r[j] = 0.0;
+            double val = 0.;
+            for(size_t k = 0; k <= i; k++)
```

```
+            {
+                val += row_data[k] * m(k,j);
+            }
+            for(size_t k = i+1; k < ncols; k++)
+            {
+                val += data[(k * (k + 1) / 2) + i] * m(k,j);
+            }
+            row_data_r[j] += val * fac;
+        }
+    }
}

//
-void SymMatrix::operator -= (const SymMatrix& m)
+// m_results = this*m1*m2. m_results must be  initialized
+void SymMatrix::multi(const Matrix& m1, const Matrix& m2, Matrix& m_result)
{
#ifdef gDEBUG
- if(nrows != m.Rows())                  //Assertion, will be removed
+ if(ncols != m1.Rows() && m1.Cols() != m2.Rows()
+    && m2.Cols() != m_result.Cols() && nrows != m_result.Rows())
{
std::cout << "\n The sizes of the two matrices are not matched" << "\n";
abort();
}
#endif
- size_t id = 0;
- for (size_t i = 0; i < nrows; i++)
- for (size_t j = 0; j < ncols; j++)
+    double *r_data = m_result.getEntryArray();
+    const size_t r_rows = m_result.Rows();
+ const size_t r_cols = m_result.Cols();
+
+ for(size_t i = 0; i < r_rows; i++)
+ {
+        double *row_data_r = &r_data[i * r_cols] ;
+ for(size_t j = 0; j < r_cols; j++)
{
- if (j > i)
- continue;
- id = i * (i + 1) / 2 + j; // temporary
- data[id] -= m(i, j);
+ //m_result(i,j) = 0.0;
+            double val = 0.;
+ for(size_t k = 0; k < ncols; k++)
+ {
```

```
+                const double entry_of_this = data[getArrayIndex(i,k)];
+ for(size_t l = 0; l < m2.Rows(); l++)
+ val += entry_of_this * m1(k,l) * m2(l,j);
+ }
+            row_data_r[j] += val;
}
+ }
}
-//
-double& SymMatrix::operator() (size_t i, size_t j) const
+// vec_result = This*vec. vec_result must be  initialized
+void SymMatrix::multi(const double* vec, double* vec_result, double fac)
{
-#ifdef gDEBUG
- if(i >= nrows || j >= nrows)
- {
- std::cout << "\n Index exceeds the size of the matrix" << "\n";
- abort();
- }
-#endif
+    for(size_t  i = 0; i < nrows; i++)
+    {
+        double val = 0.;

- if(i >= j)
- return data[i * (i + 1) / 2 + j];
- else
- return data[j * (j + 1) / 2 + i];
-}
+        const double *row_data = &data[static_cast<size_t>(i * (i + 1) / 2)] ;
+        for(size_t j = 0; j <= i; j++)
+        {
+            val += row_data[j] * vec[j];
+        }

-void SymMatrix::LimitSize(size_t dim)
-{
-#ifdef gDEBUG
- if(dim > nrows0)
- {
- std::cout << "\n Given size exceeds the original size of the matrix" << "\n";
- abort();
- }
-#endif
- nrows = ncols = dim;
- size = nrows * (nrows + 1) / 2;
```

```
+        for(size_t j = i+1; j < ncols; j++)
+        {
+            val += data[static_cast<size_t>(j * (j + 1) / 2) + i] * vec[j];
+        }
+
+        vec_result[i] += fac * val;
+    }
}

//-----------------------------------------------------
// Diagonal matrix
-DiagonalMatrix::DiagonalMatrix(size_t dim) : Matrix(0)
+DiagonalMatrix::DiagonalMatrix(size_t dim) : Matrix()
{
- Sym = true;
nrows = ncols = dim;
size = dim;
data = new double[dim];
@@ -459,9 +426,8 @@ DiagonalMatrix::DiagonalMatrix(size_t dim) : Matrix(0)
dummy_zero = 0.0;
}

-DiagonalMatrix::DiagonalMatrix() : Matrix(0)
+DiagonalMatrix::DiagonalMatrix() : Matrix()
{
- Sym = true;
nrows = 0;
ncols = 0;
nrows0 = 0;
@@ -470,9 +436,8 @@ DiagonalMatrix::DiagonalMatrix() : Matrix(0)
data = 0;
dummy_zero = 0.0;
}
-DiagonalMatrix::DiagonalMatrix(const DiagonalMatrix& m) : Matrix(0)
+DiagonalMatrix::DiagonalMatrix(const DiagonalMatrix& m) : Matrix()
{
- Sym = m.Sym;
nrows = m.nrows;
ncols = m.ncols;
nrows0 = m.nrows0;
@@ -492,7 +457,6 @@ void DiagonalMatrix::resize(size_t dim)
data = NULL;
}

- Sym = true;
nrows = ncols = dim;
```

```
size = dim;
data = new double[size];
@@ -501,63 +465,6 @@ void DiagonalMatrix::resize(size_t dim)
data[i] = 0.0;
}

-void DiagonalMatrix::operator = (double a)
-{
- for(size_t i = 0; i < size; i++)
- data[i] = a;
-}
-void DiagonalMatrix::operator *= (double a)
-{
- for(size_t i = 0; i < size; i++)
- data[i] *= a;
-}
-void DiagonalMatrix::operator += (double a)
-{
- for(size_t i = 0; i < size; i++)
- data[i] += a;
-}
-
-//
-void DiagonalMatrix::operator = (const DiagonalMatrix& m)
-{
-#ifdef gDEBUG
- if(nrows != m.Rows() || ncols != m.Cols())
- {
- cout << "\n The sizes of the two matrices are not matched" << "\n";
- abort();
- }
-#endif
- for(size_t i = 0; i < size; i++)
- data[i] = m(i);
-}
-
-//
-void DiagonalMatrix::operator += (const DiagonalMatrix& m)
-{
-#ifdef gDEBUG
- if(nrows != m.Rows())
- {
- cout << "\n The sizes of the two matrices are not matched" << "\n";
- abort();
- }
-#endif
```

```
- for(size_t i = 0; i < size; i++)
- data[i] += m(i);
-}
-
-//
-void DiagonalMatrix::operator -= (const DiagonalMatrix& m)
-{
-#ifdef gDEBUG
- if(nrows != m.Rows())                 //Assertion, will be removed
- {
- cout << "\n The sizes of the two matrices are not matched" << "\n";
- abort();
- }
-#endif
- for(size_t i = 0; i < size; i++)
- data[i] -= m(i);
-}
//
double& DiagonalMatrix::operator() (size_t i, size_t j) const
{
diff --git a/FEM/matrix_class.h b/FEM/matrix_class.h
index 0f70b55..34255bf 100644
--- a/FEM/matrix_class.h
+++ b/FEM/matrix_class.h
@@ -5,6 +5,7 @@
Function:   See the declaration below
Design and programm WW
03/2010 some improvements TF
+   03/2014 Rewritten A*B and A*x functions WW
==========================================================================*/
#ifndef matrix_class_INC
#define matrix_class_INC
@@ -27,7 +28,7 @@ class CPARDomain;

namespace Math_Group
{
-
+class SymMatrix;
class Matrix
{
public:
@@ -41,30 +42,80 @@ public:
void ReleaseMemory();                 //06.2010. WW

// Operators
- virtual void operator= (double a);
```

```
- virtual void operator*= (double a);
- virtual void operator/= (double a);
- virtual void operator+= (double a);
- void operator= (const Matrix& m);
- void operator+= (const Matrix& m);
- void operator-= (const Matrix& m);
+ virtual inline void operator= (double a)
+ {
+    for(size_t i = 0; i < size; i++)
+   data[i] = a;
+ }
+ virtual inline void operator*= (double a)
+ {
+    for(size_t i = 0; i < size; i++)
+   data[i] *= a;
+ }
+
+ virtual inline void operator/= (double a)
+ {
+    for(size_t i = 0; i < size; i++)
+   data[i] /= a;
+ }
+
+ virtual inline void operator+= (double a)
+ {
+     for(size_t i = 0; i < size; i++)
+   data[i] += a;
+ }
+
+ //void operator= (const SymMatrix& m);
+
+ virtual inline void operator= (const Matrix& m)
+ {
+        const double *m_data = m.getEntryArray_const() ;
+     for(size_t i = 0; i < size; i++)
+   data[i] = m_data[i];
+ }
+
+ virtual inline void operator+= (const Matrix& m)
+ {
+        const double *m_data = m.getEntryArray_const() ;
+     for(size_t i = 0; i < size; i++)
+   data[i] += m_data[i];
+ }
+
+ virtual inline void operator-= (const Matrix& m)
```

```
+ {
+        const double *m_data = m.getEntryArray_const() ;
+     for(size_t i = 0; i < size; i++)
+   data[i] -= m_data[i];
+ }

void GetTranspose(Matrix& m);

-        double *getEntryArray()
-        {
-          return data;
-        }
+ double *getEntryArray()
+ {
+   return data;
+ }
+
+  double *getEntryArray_const() const
+  {
+    return data;
+  }

- // vec_result = This*vec. vec_result must be initialized
- void multi(const double* vec, double* vec_result, double fac = 1.0);
- // m_result = this*m. m_result must be initialized
- void multi(const Matrix& m, Matrix& m_result, double fac = 1.0);
- // m_result = this*m1*m2. m_result must be initialized
- void multi(const Matrix& m1, const Matrix& m2, Matrix& m_result);
+ // vec_result = This*vec. vec_result must be initialized.
+ virtual void multi(const double* vec, double* vec_result, double fac = 1.0);
+ // m_result = this*m. m_result must be initialized.
+ virtual void multi(const Matrix& m, Matrix& m_result, double fac = 1.0);
+ // m_result = this*m1*m2. m_result must be initialized.To be removed
+ virtual void multi(const Matrix& m1, const Matrix& m2, Matrix& m_result);

// Access to members
- virtual double& operator() (size_t i, size_t j = 0) const;
+ virtual double& operator() (const size_t i, const size_t j = 0) const;
+
+ // Operator: set or get an entry of the raw data array. 03.2014. WW
+ double& operator[] (const size_t i) const
+ {
+   return data[i];
+ }
+
void LimitSize(size_t nRows, size_t nCols = 1);
```

```
size_t Rows() const {return nrows; }
@@ -75,12 +126,12 @@ public:
void Write(std::ostream& os = std::cout);
void Write_BIN(std::fstream& os);
void Read_BIN(std::fstream& is);
-protected:
+
+ protected:
size_t nrows, nrows0;
size_t ncols, ncols0;
size_t size;
double* data;
- bool Sym;
};

// Symmetrical matrix. 12-01-2005. WW
@@ -88,23 +139,70 @@ class SymMatrix : public Matrix
{
public:
SymMatrix(size_t dim);
- SymMatrix();
+ SymMatrix() : Matrix() { }
explicit SymMatrix(const SymMatrix& m);

void resize(size_t dim);
~SymMatrix() {}

- // Operators
- void operator= (double a);
- void operator*= (double a);
- void operator+= (double a);
- void operator= (const SymMatrix& m);
- void operator+= (const SymMatrix& m);
- void operator-= (const SymMatrix& m);
- void LimitSize(size_t dim);
+ // Forwarded operators
+ virtual inline void operator= (double a)
+ {
+    return Matrix::operator=(a);
+ }
+ virtual inline void operator*= (double a)
+ {
+    return Matrix::operator*=(a);
+ }
+
```

```
+ virtual inline void operator/= (double a)
+ {
+    return Matrix::operator/=(a);
+ }
+
+ virtual inline void operator+= (double a)
+ {
+    return Matrix::operator+=(a);
+ }
+
+ virtual inline void operator= (const Matrix& m)
+ {
+    return Matrix::operator=(m);
+ }
+
+ virtual inline void operator+= (const Matrix& m)
+ {
+    return Matrix::operator+=(m);
+ }
+
+ virtual inline void operator-= (const Matrix& m)
+ {
+    return Matrix::operator-=(m);
+ }
+
+ // Access the element
+ virtual inline double& operator() (const size_t i, const size_t j = 1) const
+ {
+   return data[getArrayIndex(i, j)];
+ }

// Access to members
- double& operator() (size_t i, size_t j) const;
+ void LimitSize(size_t dim);
+
+ // vec_result = This*vec. vec_result must be initialized
+ virtual void multi(const double* vec, double* vec_result, double fac = 1.0);
+ // m_result = this*m. m_result must be initialized. m_result must be a full stored matr
+ virtual void multi(const Matrix& m, Matrix& m_result, double fac = 1.0);
+ // m_result = this*m1*m2. m_result must be initialized.  m_result must be a full stored
+ virtual void multi(const Matrix& m1, const Matrix& m2, Matrix& m_result);
+
+ inline size_t getArrayIndex(const size_t i, const size_t j)  const
+ {
+   if(i >= j)
+     return static_cast<size_t>(i * (i + 1) / 2) + j;
```

```
+    else
+      return static_cast<size_t>(j * (j + 1) / 2) + i;
+ }
};

class DiagonalMatrix : public Matrix
@@ -120,18 +218,64 @@ public:

~DiagonalMatrix() {}

- // Operators
- void operator = (double a);
- void operator *= (double a);
- void operator += (double a);
- void operator = (const DiagonalMatrix& m);
- void operator += (const DiagonalMatrix& m);
- void operator -= (const DiagonalMatrix& m);
+ // Forwarded operators
+ virtual inline void operator= (double a)
+ {
+    return Matrix::operator=(a);
+ }
+ virtual inline void operator*= (double a)
+ {
+    return Matrix::operator*=(a);
+ }
+
+ virtual inline void operator+= (double a)
+ {
+    return Matrix::operator+=(a);
+ }
+
+ virtual inline void operator= (const Matrix& m)
+ {
+    return Matrix::operator=(m);
+ }
+
+ virtual inline void operator+= (const Matrix& m)
+ {
+    return Matrix::operator+=(m);
+ }
+
+ virtual inline void operator-= (const Matrix& m)
+ {
+    return Matrix::operator-=(m);
+ }
```

```
+
void LimitSize(size_t dim);

// Access to members
- double& operator() (size_t i, size_t j) const;
- double& operator() (size_t i) const;
+ double& operator() (const size_t i, const size_t j) const;
+ double& operator() (const size_t i) const;
+
+ // vec_result = This*vec. vec_result must be initialized
+ virtual void multi(const double* vec, double* vec_result, double fac = 1.0)
+ {
+ (void)vec;
+ (void)vec_result;
+ (void)fac;
+ }
+ // m_result = this*m. m_result must be initialized
+ virtual void multi(const Matrix& m, Matrix& m_result, double fac = 1.0)
+ {
+ (void)m;
+ (void)m_result;
+ (void)fac;
+ }
+ // m_result = this*m1*m2. m_result must be initialized
+ virtual void multi(const Matrix& m1, const Matrix& m2, Matrix& m_result)
+ {
+ (void)m1;
+ (void)m2;
+ (void)m_result;
+ }
+
};

typedef Matrix Vec;
diff --git a/FEM/problem.cpp b/FEM/problem.cpp
index e4346c1..b2deb1e 100755
--- a/FEM/problem.cpp
+++ b/FEM/problem.cpp
@@ -1152,7 +1152,15 @@ void Problem::Euler_TimeDiscretize()
}
}
std::cout<<"\n-------------------------------------------------------\n";
-#if defined(USE_PETSC) ||defined(USE_MPI) || defined(USE_MPI_PARPROC) || defined(USE_MPI_REG
+#if defined(USE_PETSC) ||defined(USE_MPI) || defined(USE_MPI_PARPROC) || defined(USE_MPI_REG
+
+#if defined(USE_PETSC) //05.2014. WW
```

```
+ for (size_t i = 0; i < out_vector.size(); i++)
+ {
+       out_vector[i]->DomainWrite_Header();
+    }
+#endif
+
}
#endif
}
diff --git a/FEM/rf_msp_new.cpp b/FEM/rf_msp_new.cpp
index afbd313..040f1b0 100644
--- a/FEM/rf_msp_new.cpp
+++ b/FEM/rf_msp_new.cpp
@@ -263,6 +263,16 @@ std::ios::pos_type CSolidProperties::Read(std::ifstream* msp_file)
capacity_pcs_name_vector.push_back("TEMPERATURE1");
capacity_pcs_name_vector.push_back("SATURATION1");
break;
+ case 30:       // another model for bentonite. WW
+ // 0. maximum conductivity
+ // 1. minimum conductivity
+ // 2. saturation
+ data_Conductivity = new Matrix(3);
+ for(i = 0; i < 3; i++)
+ in_sd >> (*data_Conductivity)(i);
+ in_sd.clear();
+ capacity_pcs_name_vector.push_back("SATURATION1");
+ break;
case 4:       //  = f(S) //21.12.2009 WW
in_sd >> Size;
in_sd.clear();
@@ -1340,24 +1350,34 @@ double CSolidProperties::Heat_Conductivity(double refence)
val = (*data_Conductivity)(0);
break;
case 2:
+ {
+ const double *k_T = data_Conductivity->getEntryArray();
+
// 0. Wet conductivity
// 1. Dry conductivity
// 2. Boiling temperature
// 3. Boiling temperature range
- if(refence < (*data_Conductivity)(2)) // Wet
- val =  (*data_Conductivity)(0);
- else if((refence >= (*data_Conductivity)(2)) &&
-         (refence < ((*data_Conductivity)(2) + (*data_Conductivity)(3))))
- val =  (*data_Conductivity)(0) +
```

```
-       ((*data_Conductivity)(0) - (*data_Conductivity)(1))
-       * (refence - (*data_Conductivity)(2)) / (*data_Conductivity)(3);
+ if(refence < k_T[2]) // Wet
+ val =  k_T[0];
+ else if((refence >= k_T[2]) &&
+         (refence < (k_T[2] + k_T[3])))
+ val =  k_T[0] + (k_T[0] - k_T[1]) * (refence - k_T[2]) / k_T[3];
else
- val =  (*data_Conductivity)(1);
+ val =  k_T[1];
+ }
break;
case 3:                                 // refence: saturation
//val = 1.28-0.71/(1+10.0*exp(refence-0.65));  //MX
val = 1.28 - 0.71 / (1 + exp(10.0 * (refence - 0.65)));
break;
+ case 30:  // Another model for bentonite. 10.2013. WW
+ {
+ //val = k_max-k_min/(1+10.0*exp(refence-S0));
+ const double *k_T = data_Conductivity->getEntryArray();
+// val = k_T[0] - (k_T[0]-k_T[1]) / (1 + exp(10.0 * (refence - k_T[2])));
+ val = k_T[0] + k_T[1]* (refence - k_T[2]);
+ }
+ break;
case 4:                                 //21.12.2009. WW
val = CalulateValue(data_Conductivity, refence);
break;
diff --git a/FEM/rf_out_new.cpp b/FEM/rf_out_new.cpp
index f0b2c39..c57dfe0 100644
--- a/FEM/rf_out_new.cpp
+++ b/FEM/rf_out_new.cpp
@@ -263,7 +263,9 @@ void OUTData(double time_current, int time_step_number, bool force_ou
//========================================================================
// TECPLOT
if (m_out->dat_type_name.compare("TECPLOT") == 0
-     || m_out->dat_type_name.compare("MATLAB") == 0 || m_out->dat_type_name.compare("GNUPL
+     || m_out->dat_type_name.compare("MATLAB") == 0 || m_out->dat_type_name.compare("GNUPL
+     || m_out->dat_type_name.compare("BINARY") == 0 // 08.2012. WW
+          )
{
// m_out->matlab_delim = " ";
// if (m_out->dat_type_name.compare("MATLAB") == 0) // JT, just for commenting header for
@@ -276,6 +278,14 @@ void OUTData(double time_current, int time_step_number, bool force_ou
cout << "Data output: Domain" << "\n";
if (OutputBySteps)
{
```

```
+#if defined (USE_PETSC) // || defined (other parallel solver lib). 12.2012 WW
+    if(m_out->dat_type_name.compare("BINARY") == 0) // 08.2012. WW
+                               {
+                                   m_out->BinaryDomainWrite();
+                               }
+     else
+                               {
+#endif
if (m_out->_pcon_value_vector.size() > 0)
m_out->PCONWriteDOMDataTEC();  //MX
else
@@ -283,6 +293,9 @@ void OUTData(double time_current, int time_step_number, bool force_ou
m_out->NODWriteDOMDataTEC();
m_out->ELEWriteDOMDataTEC();
}
+#if defined (USE_PETSC) // || defined (other parallel solver lib). 12.2012 WW
+     }
+#endif
if (!m_out->_new_file_opened)
//WW
m_out->_new_file_opened = true;
@@ -293,7 +306,15 @@ void OUTData(double time_current, int time_step_number, bool force_ou
if ((time_current > m_out->time_vector[j]) || fabs(
time_current - m_out->time_vector[j])
< MKleinsteZahl) //WW MKleinsteZahl
+                                {
+#if defined (USE_PETSC) // || defined (other parallel solver lib). 01.2014 WW
+        if(m_out->dat_type_name.compare("BINARY") == 0) // 01.2014. WW
+                                {
+                                   m_out->BinaryDomainWrite();
+                                }
+                                else
{
+#endif
if (m_out->_pcon_value_vector.size() > 0)
//MX
m_out->PCONWriteDOMDataTEC();
@@ -302,6 +323,9 @@ void OUTData(double time_current, int time_step_number, bool force_ou
m_out->NODWriteDOMDataTEC();
m_out->ELEWriteDOMDataTEC();
}
+#if defined (USE_PETSC) // || defined (other parallel solver lib). 01.2014 WW
+        }
+#endif
m_out->time_vector.erase(
m_out->time_vector.begin()
```

```
+ j);
diff --git a/FEM/rf_pcs1.cpp b/FEM/rf_pcs1.cpp
index dbafcee..231c2d1 100755
--- a/FEM/rf_pcs1.cpp
+++ b/FEM/rf_pcs1.cpp
@@ -717,13 +717,14 @@ void CreateEQS_LinearSolver()
sparse_info[0] = max_cnct_nodes * dim;
sparse_info[1] = 0; //max_cnct_nodes * dim;
sparse_info[2] = max_cnct_nodes * dim;
-   sparse_info[3] = mesh->getNumNodesLocal_Q() * dim;
+          sparse_info[3] = mesh->getNumNodesLocal_Q() * dim;
eqs = new PETScLinearSolver(eqs_dim);
eqs->Init(sparse_info);
eqs->set_rank_size(rank_p, size_p);
}
else if(  (pcs_type == MULTI_PHASE_FLOW)
-    ||(pcs_type == TWO_PHASE_FLOW) )
+    ||(pcs_type == TWO_PHASE_FLOW)
+    ||(pcs_type == PS_GLOBAL) )
{
sparse_info[0] = max_cnct_nodes * 2;
sparse_info[1] = 0; //max_cnct_nodes * 2;
diff --git a/FEM/rf_tim_new.h b/FEM/rf_tim_new.h
index f2372eb..3c2c638 100644
--- a/FEM/rf_tim_new.h
+++ b/FEM/rf_tim_new.h
@@ -112,6 +112,7 @@ public:
//Begin of function section for PI Time control ------------------------
int GetPITimeStepCrtlType() const {return PI_tsize_ctrl_type; }
double GetTimeStep() const {return this_stepsize; }
+ double GetEndTime() const { return time_end; }
void SetTimeStep( double hnew)  {this_stepsize = hnew; }
double GetRTol() const { return relative_error; }
double GetATol() const { return absolute_error; }
```