REVISTA INTERDISCIPLINAR DE PESQUISA EM ENGENHARIA

CILAMCE 2016
XXXVII IBERIAN LATIN AMERICAN CONGRESS ON COMPUTATIONAL METHODS IN ENGINEERING
BRASÍLIA - DF - BRAZIL

DESENVOLVIMENTO DA INTTEGRAÇÃO MED-MEF (LIGGGHTS®-PUBLIC E CODE_ASTER) PARA A ANÁLISE DA DEPOSIÇÃO DE PARTÍCULAS COM COESÃO EM UMA CAÇAMBA

Daniel Borges de Oliveira

Luiza Fabrino Favato

danielbo17@gmail.com

lfavato@gmail.com

Jánes Landre Junior

Pedro Américo Almeida Magalhães Júnior

Claysson Bruno Santos Vimieiro

janes@pucminas.br

pamerico@pucminas.br

claysson@pucminas.br

Pontifícia Universidade Católica de Minas Gerais

Av. Dom José Gaspar, 500 Coração Eucarístico, CEP 30535-901, Minas Gerais, Belo Horizonte, Brasila

Resumo. *Este artigo aborda o desenvolvimento da integração de dois métodos numéricos diferentes: o Método de Elementos Discretos (MED) e o Método de Elementos Finitos (MEF), de forma a apresentar alguns métodos desenvolvidos previamente por diferentes autores e as operações realizadas pelos autores deste artigo para iniciar o acoplamento de dois softwares open-source, LIGGGHTS®-PUBLIC 3.4.1 e code_aster 12.6, que aplicam o MED e MEF respectivamente. Para tal, foi utilizado como objeto de estudo o impacto da interação de partículas com coesão, depositadas sobre um modelo de caçamba de caminhão. A integração foi executada em apenas um sentido: de MED para FEM. O objetivo do trabalho foi o acoplamento dos métodos através do rastreamento do número de identificação do elemento (ID element number) das duas malhas: parede granular e malha FEM, utilizadas no LIGGGHTS e no*

code_aster respectivamente, através de programação desenvolvida pelos autores. O processo de integração desses dois métodos utilizando softwares open-source viabiliza futuros estudos e simulações, tanto para o meio acadêmico como industrial, uma vez que possibilita uma análise mais completa do comportamento estrutural de uma geometria, utilizando o MEF, exposta a um carregamento dinâmico e quase aleatório, mas bem definido, gerado pela simulação MED, sem custos de licenciamento de software.

Palavras-chave: *LIGGGHTS®-PUBLIC, code_aster, Método de Elementos Discretos, Método de Elementos Finitos, integraçãoo MEDMEF*

1 INTRODUÇÃO

A deposição e movimentação dinâmica de material particulado em componentes estruturais é um fenômeno comum no cotidiano da indústria. O Método de Elementos Discretos (MED) possibilita a simulação virtual da interação de partículas, sendo possível analisar as interações entre as partículas e a interação das partículas com uma geometria previamente definida. O minério, por exemplo, é um material granular porque apesar de ser sólido, o seu escoamento tem características de fluido. Sendo assim é possível executar uma simulação MED para prever o seu comportamento. As máquinas envolvidas no processo de extração, transporte e beneficiamento do minério, como por exemplo: escavadeiras, tratores, caminhões, transportadores de correia, britadeiras, moinhos e chutes; são expostas às cargas e ao desgaste abrasivo que ocorre devido a interação das partículas de minério nas superfícies dessas máquinas. Essas implicações têm profundas consequências econômicas que envolvem não só os custos de substituição, mas também os custos envolvidos no tempo de máquina parada e perda de produção. O método apropriado para analisar as tensões, deslocamento e deformações da geometria, que recebe o impacto dessas partículas, é o Método de Elementos Finitos (MEF), sendo possível prever o desgaste, a falha por fadiga e melhor dimensionamento essas estruturas. Com isso, o acoplamento dos dois métodos (MED e MEF) possibilita um melhor entendimento do comportamento dessas partículas e as implicações desse comportamento sobre a estrutura de estudo, de forma a possibilitar um melhor estudo dos revestimentos dessas máquinas, além de possibilitar uma otimização visando a integridade e eficiência estrutural.

1.1 Problema

O objetivo dos autores foi executar uma integração do método dos elementos discretos e método dos elementos finitos através do monitoramento dos números de identificação de cada elemento (ID element's number) das duas malhas utilizadas na análise, a parede granular DEM e a malha FEM de uma caçamba de caminhão, de forma a obter os resultados das tensões generalizadas, deslocamentos e deformações nos eixos X, Y e Z, dessa geometria de estudo. O acoplamento dos métodos MED-MEF foi executado em apenas um sentido, ou seja, foram desenvolvidos métodos para que os resultados obtidos pela simulação executada no software LIGGGHTS fossem exportados, tratados e inseridos no software code_aster, softwares MED e MEF respectivamente. No LIGGGHTS foi executada a deposição de material granular composto de 314.522 partículas, com a propriedade de coesão, sobre a parede granular gerada pela discretização de uma caçamba de caminhão. Para criar uma compatibilidade entre os solvers foi desenvolvido um módulo complementar no código fonte do LIGGGHTS. Esse módulo teve como objetivo gerar a malha no formato de trabalho do code_aster e também extrair os dados de tensão superficial normal média por elemento, resultantes dos impactos das partículas sobre a caçamba.

2 REFERENCIAL TEÓRICO

Existem várias metodologias e operações de integração do Método de Elementos Discretos (MED) com o Método de Elementos Finitos (MEF) também conhecido como *Combined Continuum and Discrete Method* (CCDM). Alguns autores e métodos são abordados a seguir:

No trabalho de Michael, Vogel e Peters (2015) é apresentado e validado a combinação eficiente dos métodos MED e MEF para o estudo da performance da tração na interação entre

a borracha de um pneu, meio contínuo; e um terreno de cascalho seco, material particulado. O acoplamento proposto foi baseado no compartilhamento de uma interface entre os domínios de cada método, a superfície limite do MED e a malha do MEF de forma que possibilitou a extração de ambas respostas simultaneamente. Cada grão do material granular em contato com a superfície do pneu gerou uma força de contato que reagiu com uma força de repulsão e a soma das cargas gerou uma deformação sobre o pneu. Neste estudo, a malha do pneu foi gerada utilizando elementos triangulares e o cascalho foi representado por esferas. A carga gerada pelo contato da partícula foi interpretada por uma força pontual, pelo MEF, sobre a superfície do elemento triangular e os deslocamentos foram computados nos nós do elemento. A interpolação da força foi baseada nos trabalhos virtuais para assegurar a força nodal constante para o MEF, de forma que os estados equivalentes do trabalho da partícula emparelhada com a interpolação dos deslocamentos é igual ao trabalho realizado pelas forças e deslocamentos nodais. O algoritmo utilizado foi criado de forma a atualizar os dois domínios na ocorrência de deformação gerada pelas interações das partículas na malha MEF do pneu. A simulação virtual foi validada por experimento prático executado por outra autora e foi possível a tração do pneu sobre o cascalho.

Já no trabalho de Guo e Zhao (2014) foi proposto uma estrutura hierárquica multiescalar do acoplamento dos dois métodos para que a simulação pudesse ser executada através do processamento paralelo. No MED é gerado um arquivo com o histórico dos seus carregamentos em cada ponto de Gauss integrado à malha FEM para servir como a representação de um elemento de volume ou partícula. O MED recebe as deformações globais nos pontos de Gauss da malha MEF como entrada das condições de contorno e é resolvido para obter a relação constitutiva no ponto específico do material. O cálculo DEM emprega as leis de contato baseados fisicamente em conjunto com o atrito de Coulomb para os contatos interpartículas para capturar a dependência de carga-histórica e resposta dissipadora altamente não-linear de um material granular. A simulação foi executada de forma a prever o comportamento da areia submetida a compressão monotônica biaxial, assim como a liquefação e a mobilidade cíclica da areia nos ensaios de cisalhamento cíclicos simples. Tanto o MED como o MEF compartilham informações de tensão, deformação e módulo tangente. O processo de solução da abordagem multiescalar hierárquica envolve o esquema de interação de Newton-Raphson, em que o operador tangente necessário para a solução MEF é estimado ou pelo módulo elástico do conjunto MED ou usando o método da perturbação.

O trabalho de Forsström e Jonsén (2016) utiliza o acoplamento dos métodos para simular o comportamento de corpo basculante de uma caçamba de caminhão discretizado pelo MEF em relação a um material granular representado pelo MED de forma a analisar a resposta do desgaste na estrutura perante o comportamento do fluxo granular utilizando o modelo de contato.

O trabalho de Chung e Ooi (2012) aborda a metodologia de conexão entre os métodos para avaliar o carregamento particulado em contato com uma estrutura. Na formulação matemática de acoplamento entre os métodos, as forças de contato da condição de contorno do MED foram convertidas para as forças e momentos nodais e aplicados na malha triangular do MEF usando as coordenadas globais do sistema. A metodologia utilizada foi derivada com base no conceito das funções de forma do elemento. Para provar a viabilidade desta metodologia proposta foram analisados dois modelos numéricos: compressão confinada e expansão confinada de esferas de aço dentro de um recipiente cilíndrico. Segundo os autores os resultados mostraram que a metodologia proposta pode proporcionar uma maneira eficaz para determinar a distribuições

de tensãoes nas estruturas submetidas às cargas provocadas pelas interações com as partículas, aplicados para solucionar problemas de interações sólido-estruturais.

Han, Peric, Crook et al. (2000) e Han, Peric, Owen et al. (2000) apresentou dois trabalhos que descrevem o processo do acoplamento dos métodos e da aplicação para a simulação do processo de jateamento (shot peening). Esse é um processo de tratamento, de trabalho a frio, usado para aumentar a vida útil do componente. As equações não lineares dos dois sistemas acoplados foram resolvidas explicitamente pelo algoritmo da diferença central em intervalos de tempo variados. No primeiro trabalho foi apresentado a interação dos métodos em uma malha MEF 2D e uma partícula esférica MED. Já no segundo a interação foi realizada utilizando uma malha MEF 3D.

3 METODOLOGIA

A metodologia foi baseada no fluxo de trabalho necessário para efetuar a integração dos dois métodos. Para tal, foi necessário primeiramente criar um módulo no código fonte do LIGGGHTS para viabilizar a extração dos resultados no formato desejado. Essa extração se baseou em gerar a malha compatível com o solver code_aster e também extrair as cargas em cada elemento para cada timestep. Em seguida foi desenvolvido um script de integração no code_aster de forma a criar as funções de carga utilizando os parâmetros extraídos anteriormente e aplicar estas cargas em cada elemento. Por fim foi executado o estudo FEM.

Durante o processo de integração dos métodos, foi observado que a numeração dos elementos da malha, gerada no software LIGGGHTS, é definida pelo número de processos utilizados durante o estudo. A malha é construída de forma paralela e os ID's dos elementos depende da quantidade de processos que foram definidos no momento da execução do estudo. Para o presente estudo foi utilizado um cluster MPI com 8 computadores e 16 processos no total. O software então divide o estudo pelo número dos processo e posteriormente gera a malha utilizando as informações destes processos. A Figura 1 exibe as malhas geradas no estudo para a quantidade de processos 1,2,8,16. Com isso, a criação do módulo de exportação da malha pelo LIGGGHTS foi uma etapa impressindível para o acoplamento dos dois métodos.

3.1 Código LIGGGHTS

Neste capitulo será descrito a programação e o raciocínio lógico utilizado para a geração do código fonte do software LIGGGHTS utilizado para a geração do arquivo de texto com o campo de tensões e a malha no formato .med. Foi necessário criar dois arquivos fontes: dump_mesh_med.cpp e dump_mesh_med.h. O arquivo dump_mesh_med.h é o arquivo de cabeçalho. Este arquivo tem como função apenas citar as funções, variáveis e demais declarações do módulo dump_mesh_med.cpp para que o compilador possa importar o modulo criado em outras partes do programa de maneira que as funções descritas em dump_mesh_med.cpp possam ser utilizadas por outros módulos.

No arquivo dump_mesh_med.cpp são descritas as funções que realizam os objetivos propostos. Este arquivo foi baseado no arquivo dump_mesh_vtk.cpp, que é o modulo responsável por gerar a malha de resultados em formato .vtk. Dentro deste arquivo foram criadas duas funções: write_data_MED e write_mesh_MED.

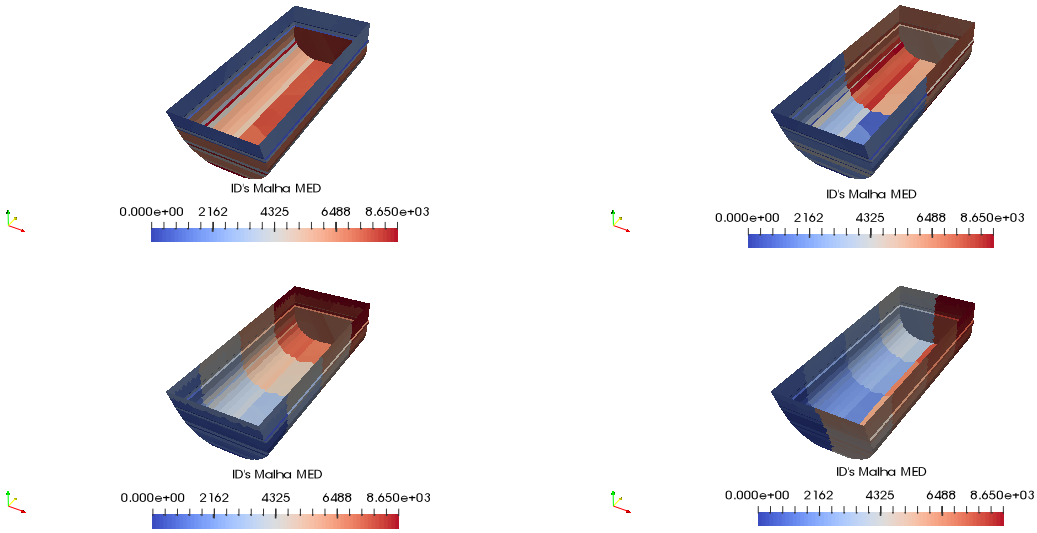

Figura 1: Comparação dos ID's nas malhas criadas com 1, 2, 4 e 8 processos

A função write_data_MED tem como objetivo gerar o arquivo dump.csv em que a primeira linha é gerada por uma estrutura de repetição que salva o intervalo [0,8650] (ID's dos elementos). As linhas subsequentes são geradas utilizando as informações de tensão normal média. A função só começa a registrar valores de tensão se houver algum valor maior que 0 em um dado timestep, ou seja, momentos em que a geometria estiver totalmente sem carga não são registrados no arquivo dump.csv. Cada linha, após a primeira, registra os valores das tensões normais médias para cada elemento em cada timestep. Caso este valor de tensão em um dado elemento e em um dado timestep seja 0 ele é transformado em 1×10^{-6}. Este processo é necessário para que o processo de convergência para o resultado no solver code_aster seja mais rápido. O código fonte desta função está disponível no capítulo 4.1.

A segunda função gerada recebeu o nome de write_mesh_MED. Para compilar esta função foi necessário instalar as bibliotecas de geração de malha MED-3.2.0. Este é o formato de malha aceito pelo code_aster. As instruções de compilação do pacote MED-3.2.0 estão neste arquivo e por isto não serão comentadas. Após a instalação da biblioteca MED-3.2.0 é necessário reconfigurar o arquivo Makefile.g++ (Localizado na pasta MAKE do código fonte do LIGGGHTS) indicando a localização das pastas "include"e "lib"na pasta de instalação da biblioteca. É necessário também incluir no comando de compilação o parâmetro *-lmed* indicando que a biblioteca MED-3.2.0 será utilizada no processo. As modificações necessárias no arquivo Makefile.g++ estão descritas a seguir:

```
# MED library, OPTIONAL
# Used to export to MED format (code_aster)
MED_INC = -I/opt/aster/public/med-3.2.0/include/ -I/opt/aster/public/
  hdf5-1.8.14/include/
MED_PATH = -L/opt/aster/public/med-3.2.0/lib/
MED_LIB = -lmed

EXTRA_INC = $(LMP_INC) $(PKG_INC) $(MPI_INC) $(FFT_INC) $(JPG_INC) $(
  PKG_SYSINC) $(MED_INC)
EXTRA_PATH = $(PKG_PATH) $(MPI_PATH) $(FFT_PATH) $(JPG_PATH) $(
  PKG_SYSPATH) $(MED_PATH)
```

```
9 EXTRA_LIB = $(PKG_LIB) $(MPI_LIB) $(FFT_LIB) $(JPG_LIB) $(PKG_SYSLIB) $(
  MED_LIB)
```

Algoritmo 1: Modificações em MakeFile.g++

Após estas modificações o binário resultante do LIGGGHTS já pode ser linkado usando a biblioteca MED-3.2.0. Para que a função criada possa trabalhar com as funções disponíveis na biblioteca MED-3.2.0 é necessário indicar no cabeçalho do arquivo dump_mesh_med.cpp os arquivos que serão inseridos:

```
1 #include <med.h>
  #include <med_utils.>
```

Algoritmo 2: Inserções de Código em C++

Dessa forma as funções disponíveis na biblioteca MED-3.2.0 já podem ser utilizadas.

Para a construção da malha .med é necessário indicar uma matriz de coordenadas na forma [X0, Y0, Z0, X1, Y1, Z1, X2, Y2, Z2, ...], onde X, Y e Z são as coordenadas de cada nó, e também é necessário a construção de uma matriz de conectividade na forma [1,2,3, 4,5,6, 7,8,9 ...], em que os números indicam a numeração de cada nó na malha. Toda a lógica de construção destas matrizes foi baseada na lógica existe no módulo dump_mesh_vtk.cpp.

Após a criação desta função e a compilação do código do LIGGGHTS, a opção de exportação da malha em formato .med e a extração do campo de tensões normais médias ficam disponíveis no binário resultante. O arquivo para a realização do estudo no LIGGGHTS deve ter a instrução abaixo:

dump 1 *all mesh/med* 1000 *post/dump*.txt stress cad1*

A instrução *mesh/med* do comando *dump* indica que o programa utilizará as instruções do novo módulo criado e a instrução *stresss* indica que a tensão de contato deve ser extraída. A execução do estudo gerará a malha (*dump.med*) e o .csv (*dump.csv*).

3.2 Parâmetros e resultados da simulação no LIGGGHTS®-PUBLIC

A Figura 2 ilustra alguns parâmetros utilizados na simulação no LIGGGHTS. Como pode ser visto, foram criadas três regiões de inserção nas posições indicadas. As inserções de partículas nas regiões não ocorrem de forma simultânea, a simulação foi seccionada em três fases de inserção nas três regiões de inserção e uma fase de assentamento das partículas. As regiões de inserção possuem o volume de 1m^3 cada e foi estipulado que deveriam ser introduzidos 1000kg de partículas a cada 20.000 timesteps. A cada carregamento foram inseridos três tipos de partículas, sendo 35% de partículas de 3cm, 35% de 5cm e 30% de 7cm. A malha da caçamba foi gerada no software NX Nastran, foi convertida para o formato .stl (ASCII) e importada no LIGGGHTS. No LIGGGHTS essa malha foi definida como uma parede granular, de forma que as propriedades do modelo de análise para o sistema (modelo de Hertz com coesão SJKR, JKR simplificado) foram válidos também para a geometria importada. O domínio da análise é a região que envolve todo o ambiente de estudo, no caso envolve as regiões de inserção das partículas e a malha da caçamba.

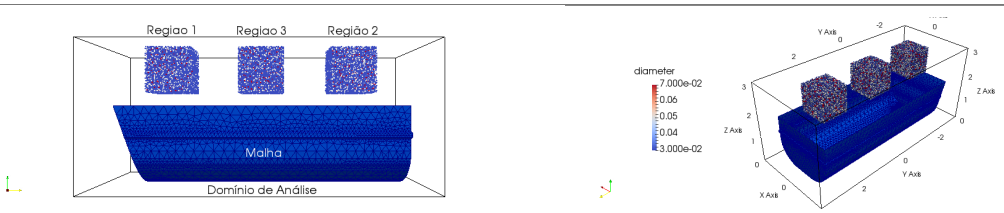

Figura 2: Domínio da simulação: Malha e Região de inserção

3.3 Código code_aster

O solver code_aster utiliza um arquivo de parâmetros que deve ser indicado no momento da execução do estudo (arquivo cacamba.dktg.py). Neste arquivo são definidas todas as premissas do estudo. Este arquivo segue uma sintaxe própria do solver, mas também aceita entradas de comandos para o interpretador Python. Por este motivo o arquivo de parâmetros utilizado neste estudo utilizou as duas sintaxes.

O presente estudo necessitou ser segmentado em onze intervalos de análise, conforme Figura 3. Nesta figura as linhas tracejadas verticais indicam cada intervalo de análise e a evolução em vermelho indica a tensão aplicada em cada momento da análise.

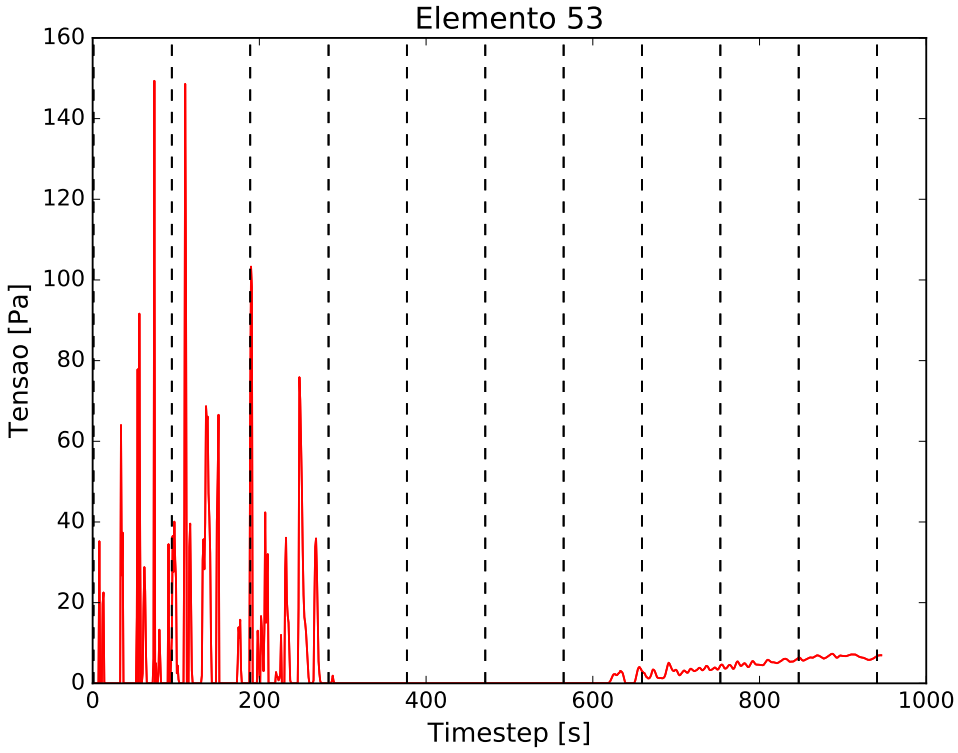

Figura 3: Função carregamento elemento 53

Os intervalos de análise foram construídos utilizando estruturas de repetição do Python. O código para realizar esta função está descrito a seguir:

```
for j in range(11):
```

```
    file = open("cacamba"+str(j)+".comm",'w')
```

Algoritmo 3: Função para gerar os intervalos de análise

Esta estrutura gera onze arquivos de parâmetros em que os nomes são assimilados na forma: cacamba0.comm, cacamba1.comm, ..., cacamba10.comm. Os comandos subsequentes geram textos dentro dos arquivos de parâmetros que foram interpretados posteriormente pelo solver.

Os onze arquivos de parâmetros gerados têm os mesmos parâmetros definidos com exceção do carregamento aplicado que foi definido conforme a função de carga que é construída para cada intervalo de análise de cada arquivo de parâmetro. As funções de carga têm a estrutura a seguir.

Funcao53 = [0, 1×10^{-9}, 1, 1×10^{-9}, 2, 1×10^{-9}, 3, 1×10^{-9}, 4, 1×10^{-9}, 5, 1×10^{-9}, 6, 1×10^{-9}, 7, 1×10^{-9}, 8, 35.224,...]

Esta função descreve o carregamento por timestep para o elemento 53. Cada elemento tem uma função de carga definida para cada timestep. Os valores de cada timestep e os valores de tensão por elemento são descritos no arquivo dump.csv gerado no estudo MED. A construção destas funções é realizada pelo módulo CSV Python. São geradas 8651 funções de carregamento para os 8651 elementos da malha em cada intervalo de análise.

Estas funções foram então declaradas no arquivo de parâmetros e aplicadas em cada elemento da malha utilizando os ID's. Esse processo foi possível utilizando os comandos de repetição da linguagem utilizada e o comando para aplicação de carga por elemento do code_aster.

Após a geração destes arquivos de parâmetro iniciou-se a execução dos estudos.

Cada timestep é divido em 10 partes de forma que a iteração básica é 0.1. O solver realiza interpolações lineares para definir os valores de carregamento entre as iterações. Cada intervalo de análise gera um arquivo de resultado com as informações de tensões generalizadas, deformações generalizadas, deslocamento, força nodal e reação nodal.

4 RESULTADOS

A subseção 4.1 exibe os códigos fonte das funções desenvolvidas. A subseção 4.2 exibe os resultados extraídos no módulo de pós processamento do SALOME V-7.8.0.

4.1 Código Fonte

Função 1

```
void DumpMeshMED::write_data_MED(int n, double *mybuf)
{
  fp=fopen("./post/dump.csv","a");
  int m=0, buf_pos = 9;

  // Dump Element Number
  if( update->ntimestep == 0 )
  {
    for (int i = 0; i < n; i++)
    {
```

```
        if ( i == n-1 )
        {
          fprintf(fp,"%i\n",i);
        }
        else
        {
          fprintf(fp,"%i,",i);
        }
      }
    }

    // Dump Stress
    if(dump_what_ & DUMP_STRESS)
    {
        // ---------- Normal Stress Average ---------- //
        // Verifica se existe valores nao nulos no array de tensoes
        m = buf_pos; // Variavel para posicionar o ponteiro dentro do array
        int g = 0; // Variavel para verificar a existencia de valores nao
    nulos no array
        for (int i = 0; i < n; i++)
        {
          if ( mybuf[m] != 0 ) g++;
          m += size_one;
        }

        // Caso exista valores nao nulos, inicia a gravacao dos valores no
    arquivo
        m = buf_pos; // Variavel para posicionar o ponteiro dentro do array
        if ( g != 0 )
        {
          for (int i = 0; i < n; i++)
          {
            if ( i == n-1 ) // Condicional necessario para adicionar uma
    quebra de linha no ultimo elemento
            {
              if ( mybuf[m] == 0 ) fprintf(fp,"1e-6\n"); // Se o valor for
    zero ele se torna 0.000001 ( Ajuda na convergencia do solver)
              else fprintf(fp,"%f\n",mybuf[m]);
              m += size_one;
            }
            else
            {
              if ( mybuf[m] == 0 ) fprintf(fp,"1e-6,");
              else fprintf(fp,"%f,",mybuf[m]);
              m += size_one;
            }
          }
        }
        buf_pos++; // Incrementa a posicao do ponteiro para que a proxima
    extracao de resultados possa ocorrer normalmente
    }
  }
```

Algoritmo 4: Função para extrair o campo de tensões

Função 2

```
void DumpMeshMED::write_mesh_MED(int n, double *mybuf, const char *
    meshfile_base)
{
  med_idt fid;
  med_err ret=-1;
  const char meshname[MED_NAME_SIZE+1] = "Unstructured mesh created by
   LIGGGHTS";
  const char meshfile_result[MED_NAME_SIZE+1] = "post/dump.rmed";
  const med_int spacedim = 3;
  const med_int meshdim = 3;
  const char axisname[3*MED_SNAME_SIZE+1] = "x               y
     z"; // Nomes dos eixos coordenados
  const char unitname[3*MED_SNAME_SIZE+1] = "m               m
   m"; // Escala da geometria
  const med_int ntria3 = n; //Numero de elementos na malha
  int  buf_pos = 0; // Variavel para posicionar o array de resultados de
   forma correta
  int sub=0; // Variavel de controle para definir a quantidade a ser
   subtraido na inicializacao do array de coordenadas.

  double coordinates0[n*9]; // Matriz de coordenadas base
  double coordinates1[n*9]; // Matriz de coordenadas para conversao
  int triaconnectivity[n*3];

  // ********** Construcao da matriz de coordenadas base. ********** //
  int j = 0;
  int m = 0;
  for (int i = 0; i < n; i++)
  {
    coordinates0[j+0] = mybuf[m+0];
    coordinates0[j+1] = mybuf[m+1];
    coordinates0[j+2] = mybuf[m+2];
    coordinates0[j+3] = mybuf[m+3];
    coordinates0[j+4] = mybuf[m+4];
    coordinates0[j+5] = mybuf[m+5];
    coordinates0[j+6] = mybuf[m+6];
    coordinates0[j+7] = mybuf[m+7];
    coordinates0[j+8] = mybuf[m+8];
    j += 9;
    m += size_one;
  }
  buf_pos += 9;

  // ********** Construcao da matriz de coordenadas para conversao e da
   matriz de conectividade. ********** //
  int p = 0; // Variaveis para posicionar o array de coordendas
  int t = 0; // Variavel para controlar o numero dos nos

  for (int i = 0; i < n*3 ; i++) //Loop nos nos.
  {
    int i3=i*3; // Cada no tem 3 coordenadas

    triaconnectivity[i] = t+1; // Matriz de conectividade

```

```
    if ( i >= 1 )
49    for (int k = p-1; k > 0; k-=3) //  Loop regressivo na matriz de
    coordenadas que esta sendo montada
        if ( coordinates0[i3+0] == coordinates1[k-2] && coordinates0[i3+1]
    == coordinates1[k-1] && coordinates0[i3+2] == coordinates1[k-0] ) // Se
    a coordenada a ser adicionada ja existe no array, salva o numero do no
    na matriz de conectividade
51        triaconnectivity[i] = ((k-2)/3)+1;

53    if ( triaconnectivity[i] == t+1 ) // Se o valor da matriz de
    conectividade nao tiver sido alterado, salva a coordenada do no na
    matriz de coordenadas
    {
55    coordinates1[p+0] = coordinates0[i3+0];
      coordinates1[p+1] = coordinates0[i3+1];
57    coordinates1[p+2] = coordinates0[i3+2];
      p+=3;
59    t+=1;
    }
61  }

63  // Este bloco e necessario para remover os zeros no final do array
    coordinates1
    double c[p];
65  for(int i=0 ; i < p ; i++ )
      c[i]=coordinates1[i];
67
    const med_int nnodes=(p/3); // Variavel com todos os nos da malha a ser
    criada
69
    ret = 0;
71  ERROR :

73  // open MED file
    fid = MEDfileOpen(meshfile_base,MED_ACC_CREAT);
75  if (fid < 0)
    {
77    MESSAGE("ERROR : file creation ...");
      goto ERROR;
79  }
    // write a comment in the file
81  if (MEDfileCommentWr(fid,"A 3D unstructured mesh") < 0)
    {
83    MESSAGE("ERROR : write file description ...");
      goto ERROR;
85  }

87  // mesh creation : a 3D unstructured mesh
    if (MEDmeshCr(fid, meshname, spacedim, meshdim, MED_UNSTRUCTURED_MESH,
89        "A 3D unstructured mesh","",MED_SORT_DTIT,MED_CARTESIAN, axisname,
    unitname) < 0)
    {
91    MESSAGE("ERROR : mesh creation ...");
      goto ERROR;
93  }
```

```
  // nodes coordinates in a cartesian axis in full interlace mode
  //    (X1,Y1, X2,Y2, X3,Y3, ...) with no iteration and computation step
  if (MEDmeshNodeCoordinateWr(fid, meshname, MED_NO_DT, MED_NO_IT, 0.0,
        MED_FULL_INTERLACE, nnodes,c) < 0)
  {
    MESSAGE("ERROR : nodes coordinates ...");
    goto ERROR;
  }

  // cells connectiviy is defined in nodal mode with no iteration and
   computation step
  if (MEDmeshElementConnectivityWr(fid, meshname, MED_NO_DT, MED_NO_IT,
   0.0, MED_CELL, MED_TRIA3,
        MED_NODAL, MED_FULL_INTERLACE, ntria3, triaconnectivity) < 0)
  {
    MESSAGE("ERROR : triangular cells connectivity ...");
    goto ERROR;
  }

  // create family 0 : by default, all mesh entities family number is 0
  if (MEDfamilyCr(fid, meshname,MED_NO_NAME, 0, 0, MED_NO_GROUP) < 0)
  {
    MESSAGE("ERROR : family 0 creation ...");
    goto ERROR;
  }

  // close MED file
  if (MEDfileClose(fid)  < 0)
  {
    MESSAGE("ERROR : close file ...");
  }
}
```

Algoritmo 5: Função para salvar malha em formato MED

Função 3

```
import csv
teste=[]
with open("/home/daniel/Desktop/FEA-DEA-Analise_Cacamba/code_aster/dump.
 csv") as csvfile:
    reader = csv.DictReader(csvfile, delimiter=',')
    for row in reader:
        teste.append(row)

elementos = len(teste[0]) # Quantidade de elementos na malha
subtimestep = 1     # Divisor do tamanho do timestep

decimo=(len(teste)-len(teste)%10)/10 # Calcula o modulo para remover o
 resto da divisao por 10.
resto = len(teste)%10   # Salva o resto na variavel resto. O ultimo
 timestep utiliza este valor para montar o range.

timestep0  = range(0,decimo)
timestep1  = range(timestep0[-1]+1, timestep0[-1]+1 + decimo)
```

```
  timestep2  = range(timestep1[-1]+1, timestep1[-1]+1 + decimo)
  timestep3  = range(timestep2[-1]+1, timestep2[-1]+1 + decimo)
  timestep4  = range(timestep3[-1]+1, timestep3[-1]+1 + decimo)
  timestep5  = range(timestep4[-1]+1, timestep4[-1]+1 + decimo)
  timestep6  = range(timestep5[-1]+1, timestep5[-1]+1 + decimo)
  timestep7  = range(timestep6[-1]+1, timestep6[-1]+1 + decimo)
  timestep8  = range(timestep7[-1]+1, timestep7[-1]+1 + decimo)
  timestep9  = range(timestep8[-1]+1, timestep8[-1]+1 + decimo)
  timestep10 = range(timestep9[-1]+1, timestep9[-1]+1 + resto)

  # Cria as funcoes com as informacoes de tensao em cada momento
  for n in range(elementos):  # Laco para varrer as colunas
      globals()['funcao'+str(n)] = [] # Cria variaveis de forma dinamica
    para cada elemento
      for i in range(timestep"""+str(j)+"""[0],timestep"""+str(j)+"""[-1]):
     # Laco para varrer as linhas
          # Monta o array com a funcao tensao x timestep
          globals()['funcao'+str(n)].append(int(i)) # Timestep
          globals()['funcao'+str(n)].append(float(teste[i][str(n)])*1e-3) #
     Tensao
```

Algoritmo 6: Função para gerar as funções de carregamento no Code_Aster

Função 4

```
  for i in range(elementos):
      file.write("Ramp"+str(i)+"=DEFI_FONCTION(NOM_PARA='INST',VALE=funcao"
    +str(i)+",INTERPOL='LIN',PROL_DROITE='LINEAIRE',PROL_GAUCHE='CONSTANT',)
    ;\n")
  file.write("""
  Load=AFFE_CHAR_MECA_F(
  MODELE=cocModel,
  VERI_NORM='NON',
  FORCE_COQUE=(
  """)
  for i in range(elementos):
      file.write("_F(MAILLE=('M"+str(i+1)+"',),PRES=Ramp"+str(i)+",),\n""")
  file.write("""
    ),
  );
```

Algoritmo 7: Função para gerar dinamicamente os carregamentos em cada elemento

Função 5

```
for j in range(11):
  exportfile = os.path.join(WORKING_DIR+'dktg/'+JOBNAME+'.export')
  e = open(exportfile,'w')
  e.write('P actions make_etude\n')
  e.write('P nomjob '+JOBNAME+'\n') # Name of the job
  e.write('P consbtc oui\n') # to build (without soumetttre) the file btc.
    corefilesize
  e.write('P soumbtc oui\n')  # submission or not of the file .btc tpsjob
    maximum Time of the job
```

```
e.write('P follow_output yes\n') # To ask (or not) the interactive follow
  -up of the execution.
e.write('P tpsjob 99999999999999999999\n') # Time limit of the
  conclusion of the job in minutes
e.write('P mem_aster 100\n') # Percentage of memjob alocate to study
e.write('P memjob 7781\n') # Memory used in job in MB
e.write('P memory_limit 7781\n') # Memory for Aster computation in MB
e.write('P mode interactif\n')
e.write('P mpi_nbcpu 1\n') # Total number of processors for parallelism
  MPI
e.write('P mpi_nbnoeud 1\n') # Number of nodes used by MPI
e.write('P ncpus 4\n') # Number of process used by OpemMP
e.write('P platform LINUX64\n') # Operational system of the host
e.write('P proxy_dir /tmp\n') # Directory to store temporary files
e.write('P rep_trav /tmp/'+JOBNAME+'\n') # Necessary to destroy working
  directory
e.write('P version '+ASTER_VERSION+'\n') # Aster version used in study
e.write('A args\n')
e.write('A memjeveux 972\n') # Size of the memory taken by the execution
  (in MB). memjeveux = Total memory / 8
e.write('A tpmax 35996400\n') # Limit of the time of the execution (in
  seconds)
e.write('P time_limit\n') # time in seconds of subjected work
e.write('P aster_root '+ASTER_ROOT+'\n') # Root directory of aster binary
e.write('F mmed '+WORKING_DIR+'meshes/'+JOBNAME+'.med  D 20\n') # MED
  file path and name
e.write('F comm '+WORKING_DIR+'dktg/'+JOBNAME+str(j)+'.comm D  1\n') #
  COMM file path and name
e.write('F mmed '+WORKING_DIR+'dktg/'+JOBNAME+str(j)+'.rmed R 80\n') #
  RMED file path and name
e.write('F mess '+WORKING_DIR+'dktg/'+JOBNAME+str(j)+'.mess R  6\n') #
  MESS file path and name
e.write('F resu '+WORKING_DIR+'dktg/'+JOBNAME+str(j)+'.resu R  8\n') #
  RESU file path and name
e.write('R base '+WORKING_DIR+'dktg/'+JOBNAME+str(j)+'.base RC 0\n') #
  BASE file path and name
e.close()
aster_run = Popen(ASTER_ROOT+'/bin/as_run --vers='+ASTER_VERSION+' '+
  WORKING_DIR+JOBNAME+'dktg/'+'.export', shell='TRUE')
aster_run.wait()
```

Algoritmo 8: Função para executar cada intervalo de análise

4.2 Resultados da integração no pós-processador ParaView

A função foi criada para a extração das tensões do solver LIGGGHTS para o arquivo .csv somente se o valor fosse diferente de 0 para cada 1.000 timesteps. O primeiro contato do material de deposição no estudo MED aconteceu no timestep 46.000, sendo, portanto, o primeiro registro no arquivo .csv. Isto fez com que a correlação entre os timesteps dos solvers fosse modificada. Tabela 1 descreve essa correlação entre os timesteps entre os dois solvers. A Tabela 1 descreve a correlação dos timesteps dos dois solvers.

As Figuras 4, 5 e 6 ilustram os resultados das simulações MED e MEF em três momentos em que os timesteps foram correlacionados. Por exemplo, na Figura 4, o timestep no

Tabela 1: Correlação entre timesteps no solver LIGGGHTS e no solver code_aster

Arquivo de resultados	Intervalo de estudo code_aster (intervalo de timesteps)	Intervalo de estudo LIGGGHTS (intervalo de timesteps)
cacamba0	0 - 93	46.000 - 139.000
cacamba1	94 - 187	140.000 - 233.000
cacamba2	188 - 281	234.000 - 327.000
cacamba3	282 - 375	328.000 - 421.000
cacamba4	376 - 469	422.000 - 515.000
cacamba5	470 - 563	516.000 - 609.000
cacamba6	564 - 657	610.000 - 703.000
cacamba7	658 - 751	704.000 - 797.000
cacamba8	752 - 845	798.000 - 891.000
cacamba9	846 - 939	892.000 - 985.000
cacamba10	940 - 947	986.000 - 1.040.000

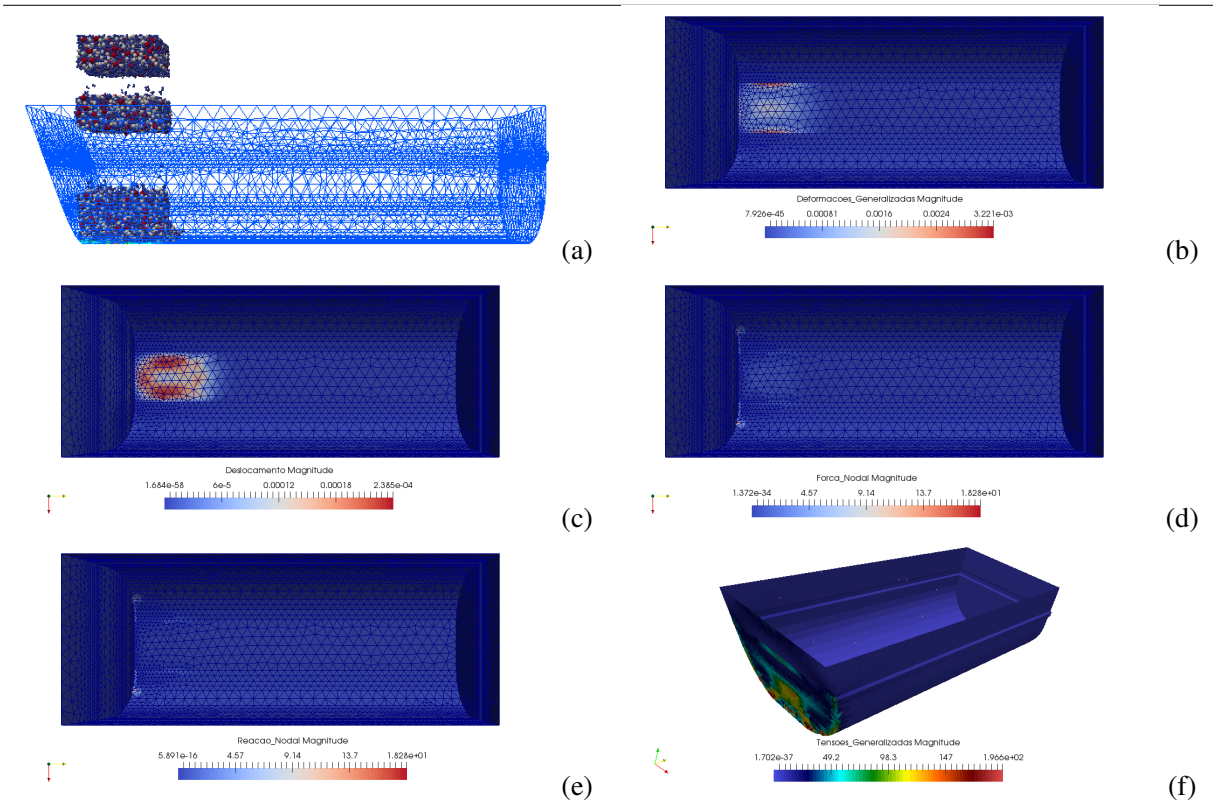

Figura 4: Resultados da Deposição das Partículas - Timestep 59.000 LIGGGHTS e 13 code_aster: (a) deposição das partículas realizada no estudo MED. Resultados da simulação MEF: (b) deformação generalizada, (c) deslocamento, (d) força nas regiões de engaste, (e) reações nas regiões de engaste e (f) tensão generalizada.

LIGGGHTS é o 59.000 e o timestep correspondente no code_aster é o timestep 13. Nessas tabelas é ilustrado na primeira imagem o resultado da simulação MED e nas demais os resultados da simulação MEF.

Foi observado que a transferência de informação entre os softwares foi obtida, conforme pode ser visto nas figuras, as regiões que deveriam apresentar alteração no MEF foram as mesmas regiões de inserção da carga no MED, nos timesteps equivalentes. Como o objetivo foi a

integração entre os softwares os resultados foram satisfatórios.

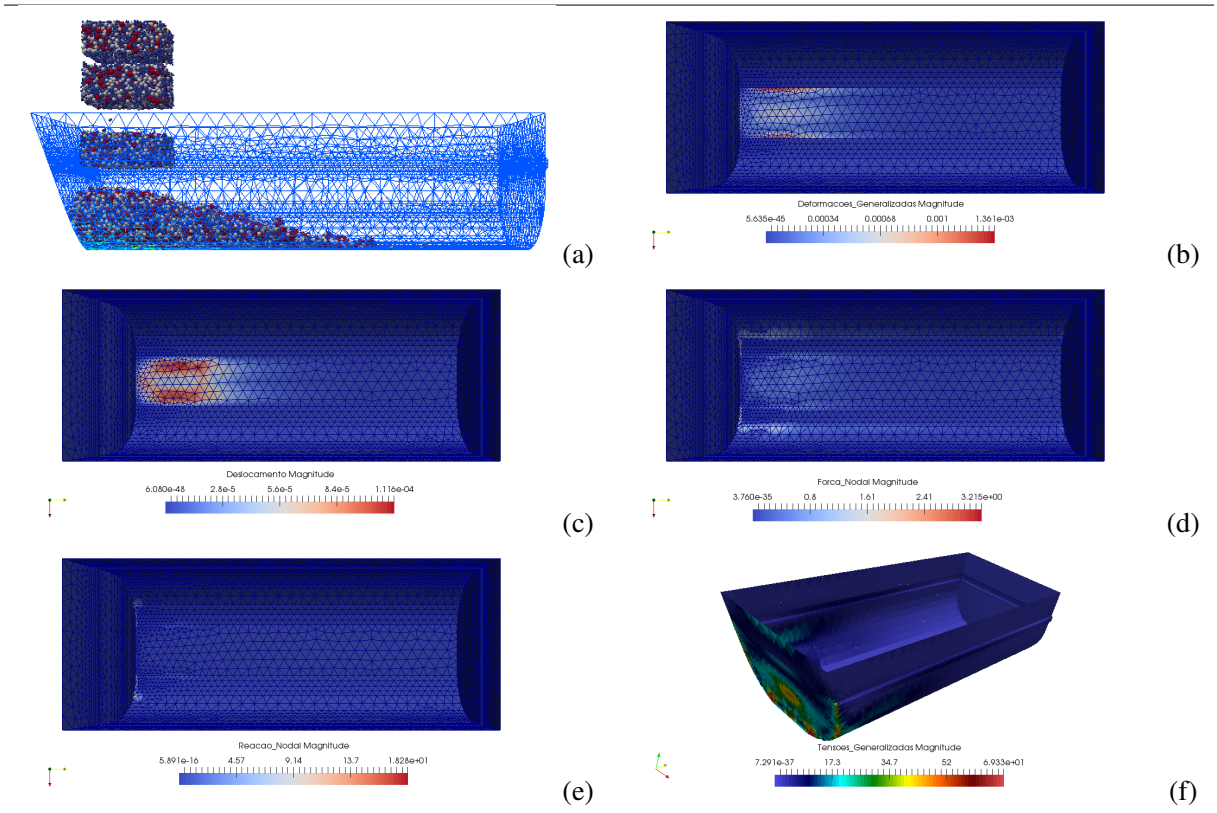

Figura 5: Resultados da Deposição das Partículas - Timestep 147.000 **LIGGGHTS e** 101 **code_aster: (a) deposição das partículas realizada no estudo MED. Resultados da simulação MEF: (b) deformação generalizada, (c) deslocamento, (d) força nas regiões de engaste, (e) reações nas regiões de engaste e (f) tensão generalizada.**

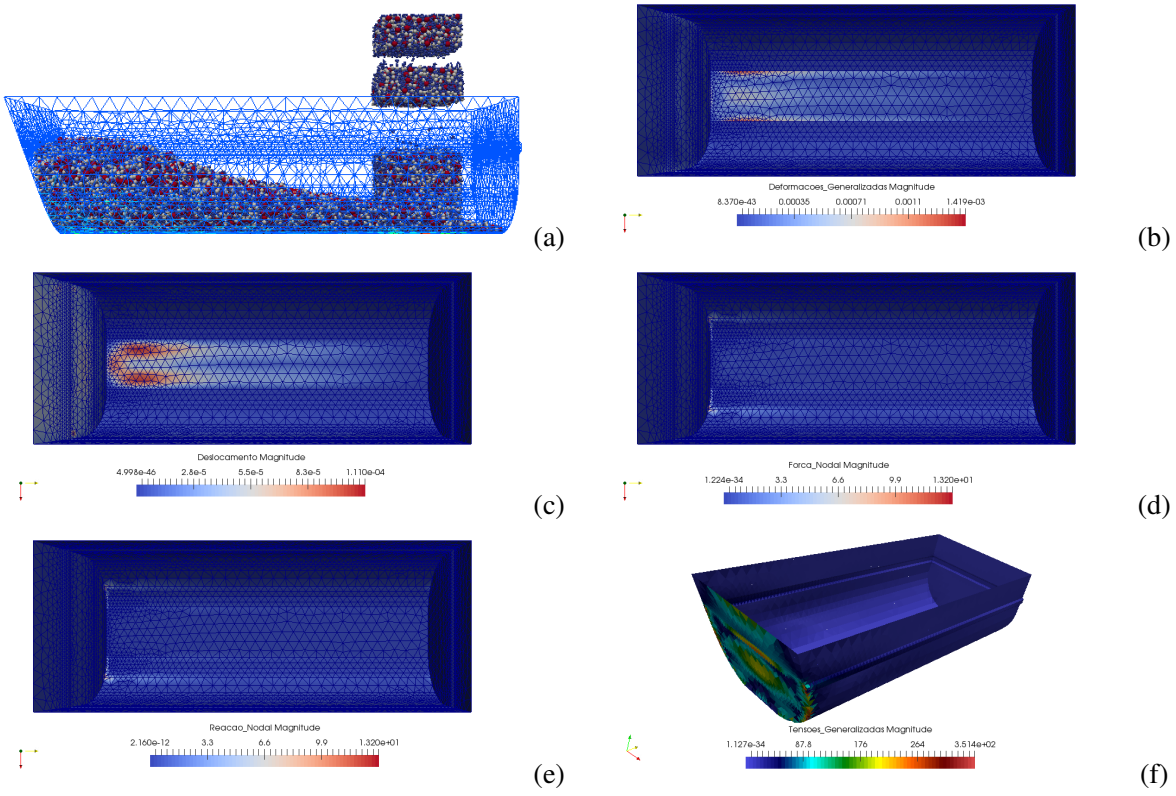

Figura 6: Resultados da Deposição das Partículas - Timestep 333.000 **LIGGGHTS e** 287 **code_aster: (a) deposição das partículas realizada no estudo MED. Resultados da simulação MEF: (b) deformação generalizada, (c) deslocamento, (d) força nas regiões de engaste, (e) reações nas regiões de engaste e (f) tensão generalizada.**

5 CONCLUSÃO

O acoplamento entre os dois softwares foi possível pois o código fonte do LIGGGHTS é aberto (open source). Isto permitiu que este fosse adaptado às necessidades do estudo. As bibliotecas de geração de malha do software code_aster são escritas em linguagem c++ e foram facilmente importadas no processo de criação do novo módulo. O processo de compilação e execução foi todo realizado em um sistema operacional Debian 8 x64. Todos estes softwares dispensam gastos com licenciamento e são facilmente encontrados na internet.

Tanto o software LIGGGHTS quanto o software code_aster tem uma extensa documentação e uma comunidade ativa. Todas as informações necessárias para a realização deste estudo foram encontradas nos manuais (*LIGGGHTS(R)-PUBLIC Documentation* (2016), *LAMMPS Documentation* (2016) e *code_aster Documentation* (2016)) e fóruns.

As bibliotecas Python utilizadas no estudo estão disponibilizadas nos repositórios padrão do compilador (pip).

O estudo precisou ser seccionado em 11 subestudos, pois o uso intenso de memória RAM estava causando a finalização pelo sistema operacional. Para resolver este problema seria necessário um computador com limites de memória superiores a 20GB, pois simulações preliminares indicaram que o limite de 20GB (8GB memória RAM e 12 GB memoria SWAP) de memória disponíveis no computador utilizado estavam sendo atingidos. Este processo não chegou a prejudicar o estudo visto que os carregamentos foram definidos por funções que evoluem com o tempo, conforme Figura 3.

Foi observado que a transferência de informação entre os softwares foi obtida, conforme pode ser visto nas Figuras 4, 5 e 6. As regiões que deveriam apresentar alteração no MEF foram as mesmas regiões de inserção da carga no MED, nos timesteps equivalentes. Como o objetivo foi a integração entre os softwares os resultados foram satisfatórios.

A metodologia desenvolvida poderá auxiliar futuras pesquisas, pois apresenta uma abordagem de integração em que os ID's dos elementos são mantidos nos dois solvers. Isso permite que seja desenvolvido uma metodologia de acoplamento de duas vias, em que os dois solver possam trabalhar de forma totalmente integrada, onde a saída de um solver pode ser a entrada do outro em tempo real.

AGRADECIMENTOS

A Pontifícia Universidade Católica de Minas Gerais – PUCMINAS e aos seus funcionários e professores, a Fundação de Amparo à Pesquisa do Estado de Minas Gerais – FAPEMIG e ao Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq pelo apoio e as contribuições relacionadas à realização da pesquisa e do artigo.

REFERÊNCIAS

Chung, Y.C. e J.Y. Ooi (2012). "Linking of discrete element modelling with finite element analysis for analysing structures in contact with particulate solid". Em: *Powder Technology* 217, pp. 107–120. URL: http://dx.doi.org/10.1016/j.powtec.2011.10.016.

code_aster Documentation (2016). [Acesso em: 10 Maio. 2016]. Électricité de France S.A. Paris, França. URL: http://www.code-aster.org/V2/doc/default/en/index.php?lang=en.

Forsström, D. e P. Jonsén (2016). "Calibration and validation of a large scale abrasive wear model by coupling DEM-FEM: Local failure prediction from abrasive wear of tipper bodies during unloading of granular material". Em: *Engineering Failure Analysis* 66, pp. 274–283. URL: http://dx.doi.org/10.1016/j.engfailanal.2016.04.007.

Guo, Ning e Jidong Zhao (2014). "A coupled FEM/DEM approach for hierarchical multiscale modelling of granular media". Em: *International Journal for Numerical Methods in Engineering* 99, pp. 789–818. URL: http://dx.doi.org/10.1002/nme.4702.

Han, K., D. Peric, A.J.L. Crook et al. (2000). "A combined finite/discrete element simulation of shot peening processes – Part I: Studies on 2D interaction laws". Em: *Engineering Computations* 17, pp. 593–620. URL: http://dx.doi.org/10.1108/02644400010339798.

Han, K., D. Peric, D.R.J. Owen et al. (2000). "A combined finite/discrete element simulation of shot peening processes – Part II: 3D interaction laws". Em: *Engineering Computations* 17, pp. 680–702. URL: http://dx.doi.org.ez93.periodicos.capes.gov.br/10.1108/02644400010340615.

LAMMPS Documentation (2016). [Acesso em: 10 Maio. 2016]. Sandia National Laboratories. Albuquerque, New Mexico; Livermore, California. URL: http://lammps.sandia.gov/doc/Manual.html.

LIGGGHTS(R)-PUBLIC Documentation (2016). 3.X. [Acesso em: 8 Abril. 2016]. DCS Computing Gmbh, Linz, Austria. ndustriezeile 35, 4020 Linz, Austria. URL: http://www.cfdem.com/media/DEM/docu/Manual.html.

Michael, Mark, Frank Vogel e Bernhard Peters (2015). "DEM–FEM coupling simulations of the interactions between a tire tread and granular terrain". Em: *Computer Methods in Applied Mechanics and Engineering* 289, pp. 227–248. URL: http://dx.doi.org/10.1016/j.cma.2015.02.014.