# IMPLEMENTAÇÃO DE UM MODELO DE PARTÍCULAS LAGRANGIANAS UTILIZANDO TABELA *HASH*, DENTRO DO CÓDIGO COMPUTACIONAL AMR3D

**Renato Pacheco Silva, UFU, rpacheco_br@yahoo.com.br**
**Aristeu da Silveira Neto, UFU, aristeus@mecanica.ufu.br**
**Millena Martins Villar Vale, UFU, millena.villar@gmail.com**

***Resumo:*** *O desenvolvimento de códigos computacionais para a solução de problemas de engenharia é sempre uma tarefa difícil, que envolve conceitos multidisciplinares. Almejando sempre soluções mais precisas, existem as formas mais comuns de se atingir este objetivo: discretizações de alta ordem, refino de malha, malhas móveis, solvers diferentes, assim como modelos numéricos diferentes. Quando se trata de problemas de engenharia, não só estas questões devem ser levadas em conta, mas também o custo computacional envolvido. Sendo assim, um ponto importante a ser observado é o tipo de estrutura de dados a ser utilizada. É neste contexto que este trabalho está sendo desenvolvido. Quando comparadas diferentes estruturas de dados para um modelo Lagrangiano (Discrete Phase Model - DPM), algumas mostram melhores resultados em termos de custo computacional. Esta avaliação é sempre feita com base nas operações que serão necessárias para os problemas envolvidos. Neste trabalho, duas estruturas de dados diferentes foram comparadas, e outras duas foram estudadas. Dos resultados obtidos, uma estrutura foi escolhida e utilizada para implementação dentro do código computacional AMR3D.*

***Palavras chave****: Lagrangiana, DPM, estrutura de dados, AMR3D*

## 1. INTRODUÇÃO

Na utilização de um modelo de fase discreta (DPM), é também necessário uma boa proposta para a estrutura de dados. A forma como os dados utilizados vão ser alocados e utilizaados, são de extrema importância para uma melhor performance do código. Algumas propostas são então escolhidas para serem estudadas, sendo que duas delas foram testadas para comparação.

Outro aspecto importante é no que diz respeito aos mecanismos de busca utilizados para obtenção das informações Eulerianas referentes às partículas Lagrangianas. Esse mecanismo de busca é chamado de *Search Engine* (SE) e utiliza de um mapa Euleriano, chamado de *Multi-Level Euler Map* (MLEM) para o correto funcionamento.

A terceira etapa é a integração de todas estas ferramentas, dentro do código AMR3D, que é um com refinamento adaptativo de malha (*Adaptive Mesh Refinement*), para escoamentos bifásicos (Villar, 2007). O AMR3D está escrito em linguagem FORTRAN, e todo o modelo de partículas está escrito em linguagem C, o que torna o trabalho mais desafiador.

## 2. MODELO DE FASE DISCRETA

A integração da equação 01 nos dá a velocidade da partícula:

$$\frac{du_p}{dt} = F_D\left(u - u_p\right) + \frac{g(\rho_p - \rho)}{\rho_p} \tag{1}$$

Onde o índice $p$ indica informações Lagrangianas e sem índice indica informações Eulerianas, $u$ é a velocidade, $t$ é o tempo, $F_D$ é a força de arrasto, $g$ é a força da gravidade e $\rho$ é a densidade.

A força de arrasto é dada por:

$$F_D = \frac{18\mu}{\rho_p d_p^2}\frac{C_D Re}{24} \tag{2}$$

Onde $d$ é o diâmetro, $\mu$ é a viscosidade, $C_D$ é o coeficiente de arrasto e $Re$ é o número de Reynolds da partícula, dado por:

$$Re \equiv \frac{\rho d_p \left|u_p - u\right|}{\mu} \tag{3}$$

Nenhuma outra força é considerada.

A posição da partícula é dada por:

$$posx^t = posx^{t+1} + ux_p^{t-1} * dt \tag{4}$$

## 2. ESTRUTURAS DE DADOS

Para o modelo Lagrangiano proposto (DPM), é necessária uma estrutura de dados eficiente. A forma como os dados serão armazenados e acessados são muito importantes para uma boa performance do código. Então, estruturas de dados como lista ligada, vetor dinâmico, tabela hash ou árvore binária, devem ser analizadas para se encontrar a melhor opção. Foram analisadas computacionalmente, duas estruturas de dados: lista ligada e tabela *hash*.

### 2.1. Lista ligada

É uma estrutura onde uma sequência de células conectadas. Cada célula tem toda informação necessária. No caso do modelo Lagrangiano, cada célula carrega toda a informação da partícula necessária; A referência é o head da tabela, que é o primeiro elemento. O head aponta para o próximo elemento, até que o próximo seja nulo, e ai se encontra o final da lista. É uma estrutura fácil de ser construída, mas limitada em termos de performance em algumas operações como busca, por exemplo.

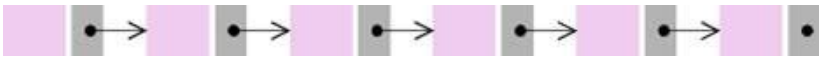

Figura 1: Exemplo de lista ligada (Feofiloff, 2013)

### 2.2. Tabela *Hash*

Outra opção é a tabela *hash*. Este tipo de estrutura de dados é basicamente uma lista ligada de um tipo de dados bem barato, normalmente um inteiro. Existe uma tabela com as informações de cada ponto da lista, e um link entre o ponto e a tabela. Esse link é chamado de função *hash*. É então garantido o a acesso à tabela, de ordem constante desde que o acesso tenha sido feito pela chave da tabela. A chave é o número inteiro que aponta para as informações da tabela, através da função *hash*.

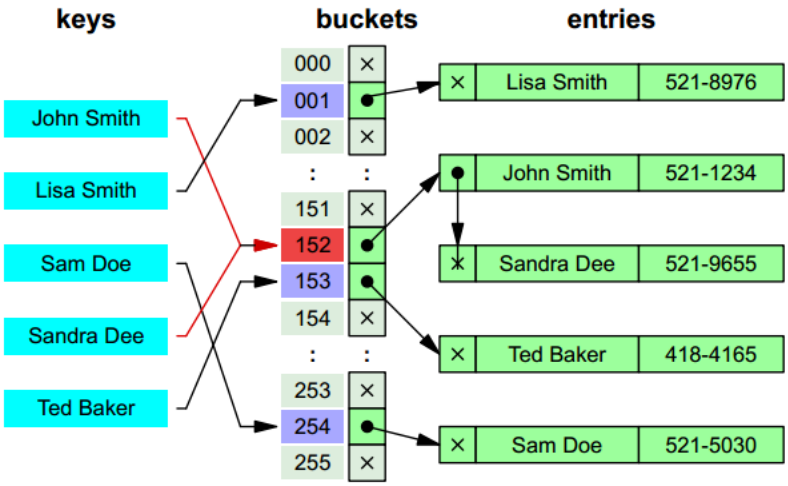


Figura 2: Exemplo de tabela *hash* com a chave sendo um tipo char (MIT OpenCourseWare, 2013)

## 3. RESULTADOS

Após a análise das estruturas de dados, foi implementada a tabela *hash*, e testada no comparativo com a lista ligada, com relação à sua performance. Foi também testada a implementação com relação ao transporte Lagrangiano no AMR3D. Estes testes mostraram bons resultados como podem ser vistos nas próximas sub-seções.

### 3.1. Resultados do Teste de Performance

Para comparação entre lista ligada e tabela *hash*, foi escolhida a advecção sem acoplamento com Navier-Stokes de um determinado número de partículas por 150 passos de tempo e verificado o tempo computacional a partir do comando *./time*. Foram feitas então três rodadas para cada quantidade de partículas e comparado utilizando a média dos resultados obtidos. As quantidades de partículas foram 1, 2, 3, 4, 5, 6, 7, 8, 9 e 10 milhões de partículas, como pode ser visto na Figura 3. A tabela *hash* utilizada foi a UTHASH (Hanson, 2013):

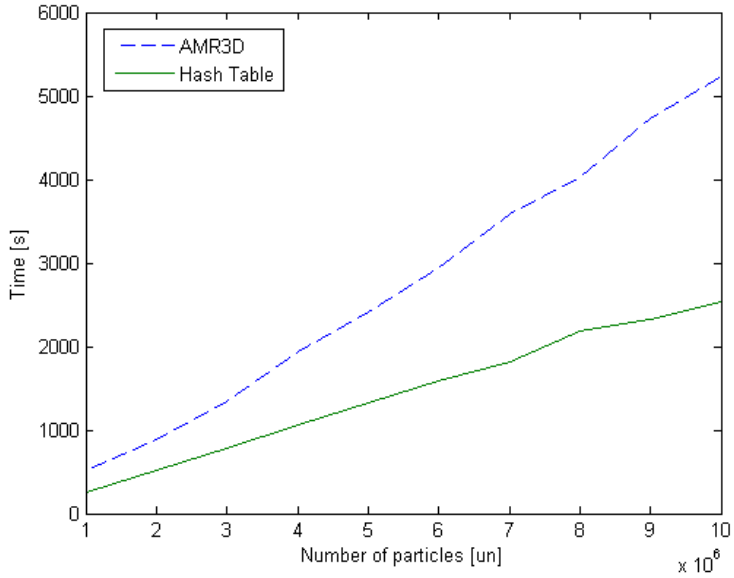


Figura 3: Comparativo entre lista ligada já existente no AMR3D e tabela *hash*

### 3.2. Resultados do Acoplamento com AMR3D

A partir de uma malha com três níveis de refinamento, foi imposto um campo de velocidade $u=0.0$, $v=0.0$ e $w=z$. Desta forma, a velocidade $w$ da partícula deverá sempre ser igual à sua posição em $z$, o que facilita a verificação se a advecção está sendo feita de maneira correta. Foram geradas $10000$ partículas com posição randômica entre $0.0 < x < 1.0$, posição randômica entre $0.0 < y < 1.0$ e posição randômica entre $0.0 < z < 3.0$, sendo o domínio global $0.0 < x < 1.0$, $0.0 < y < 1.0$ e $0.0 < z < 8.0$.

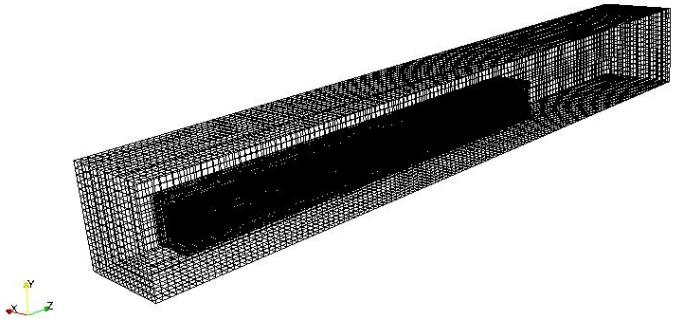

Figura 4: Malha com três níveis de refinamento

a) b)

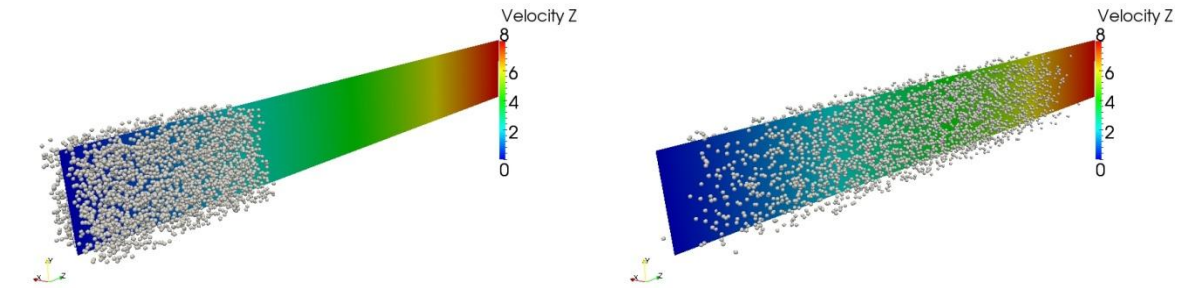


Figura 5: Advecção de partículas após a) 0 passos de tempo e b) 150 passos de tempo

### 3.3. Solução da Cavidade com Presença de Partículas

Foram selecionados dois casos de cavidade com tampa deslizante, com presença de *1000* partículas, *Reynolds = 3000*, sendo no primeiro caso partículas de fluido (mesmas propriedades do campo euleriano) e no segundo caso, partículas de areia.

a) b)

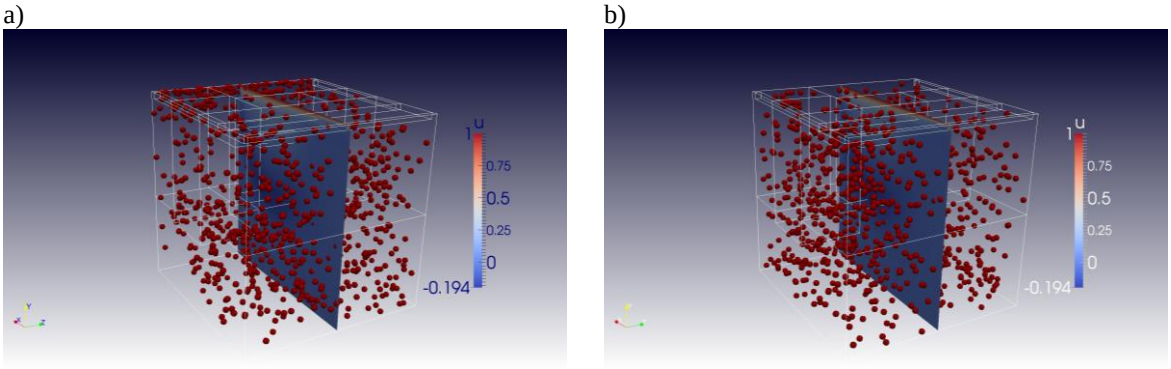


Figura 6: Cavidade com tampa deslizante com a) partículas de fluido e b) partículas de areia

## 4. CONCLUSÕES

Ficaram evidentes grandes benefícios a respeito da utilização de estrutura de dados do tipo tabela *hash* ao invés de lista ligada. Pode-se questionar sobre a perda de ferformance utilizando interfaces C-FORTRAN, mas muito provavelmente não existem, pois o compilador trata as rotinas linkadas conjuntamente. O código AMR3D está agora apto a rodar casos com DPM, mais realísticos.

## 5. REFERÊNCIAS

Feofiloff, P., Listas Encadeadas, Em: <http://www.ime.usp.br/~pf/algoritmos/aulas/lista.html>. Acesso em: 09 de Agosto, 2013.

Hanson, T. D., A Hash Table for C Structures, Em: <http://troydhanson.github.io/uthash/>. Acesso em: 09 de Agosto, 2013.

Krenzel, S., Dynamic Arrays, Em: <http://stevekrenzel.com/articles/dynamic-arrays-no-copy>. Acesso em: 09 de Agosto, 2013.

MIT OpenCourseWare, Void and function pointers. Hash tables, Em: <http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-087-practical-programming-in-c-january-iap-2010/lecture-notes/>. Acesso em: 16 de Setembro, 2013.

MIT OpenCourseWare, Algorithm Analysis: Reasonable vs. Unreasonable Algorithms, Complexity, Em: <http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-01-unified-engineering-i-ii-iii-iv-fall-2005-spring-2006/comps-programming/>. Access in: 16 de Setembro, 2013.

Villar, M. M. V, 2007, Análise Numérica Detalhada de Escoamentos Multifásicos Bidimensionais, Tese de Doutorado, Universidade Federal de Uberlândia, Brasil.