Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXXII, págs. 947-960 (artículo completo)
Carlos G. García Garino, Aníbal E. Mirasso, Mario A. Storti, Miguel E. Tornello (Eds.)
Mendoza, Argentina, 19-22 Noviembre 2013

OPTIMIZACIÓN DE LA CALIDAD DE MALLAS DE TETRAEDROS EN PARALELO

Juan P. D'Amato[b,c], **Marcelo Vénere**[a,c] **y Ricardo Rossi**[d]

[a]*CNEA – Comisión Nacional de Energía Atómica, Argentina*

[b]*CONICET - Consejo Nacional de Investigaciones Científicas y Técnicas , Argentina*

[c]*UNICEN – Universidad Nacional del Centro de la Prov. De Buenos, Argentina*

[d]*CIMNE – Centro internacional de Métodos Numéricos en Ingeniería, España*

Palabras claves: Optimización, Mallas, Paralelismo, GPU, Calidad.

Resumen. Uno de los métodos más efectivos para la generación de mallas de calidad a partir de descripciones geométricas se basa en modificaciones locales que mejoran una métrica dada. El principal problema de este algoritmo es que requiere de muchas operaciones hasta converger a una malla satisfactoria, por lo que en general se debe interrumpir el proceso luego de un determinado número de mejoras.

En este trabajo se presenta un esquema para la paralelización masiva del remallado utilizando múltiples procesadores. Se presentan los resultados de "speed-up" y de calidad obtenidos en mallas de tetraedros con cientos de miles de elemento, comparando con otras bibliotecas de uso libre. También se analiza su aplicación cuando la malla se utiliza en simulaciones numéricas con desplazamiento de nodos.

1 INTRODUCCIÓN

Las simulaciones basadas en elementos finitos permiten modelar fenómenos físicos, muchos de los cuales utilizan mallas que evolucionan en el tiempo. Durante esta evolución, los nodos se mueven de una manera arbitraria, el cual produce que los elementos se deformen. Esta deformación significa que la calidad de los elementos empeora, y muchas veces los resultados de las simulaciones son inválidos (por ejemplo, cuando los elementos tienen volumen negativo).

Para mejorar la calidad de los elementos, se suelen aplicar distintas estrategias. En muchos trabajos se propone remallar todo el dominio (Wojtan and Turk 2008},(Dehne et al 2000) utilizando estrategias como Delaunnay restricto. En otros trabajos, se proponen estrategias marginales, aplicando reconexiones de elementos en forma localizada (Coupez et al 2000), (Klingner 2009). Como otra alternativa, el trabajo de (Zhang 2005) plantea el problema como un problema de optimización global, en el cual únicamente se mueven los nodos a una posición optima. Este tipo de estrategias no asegura que se corrijan todos los problemas, como los elementos de volumen negativo, por lo que no se utilizan por si solos.

En las simulaciones dinámicas, una de las variables más importante es el paso del tiempo. El paso del tiempo determina la deformación de la malla y en consecuencia la degradación de la misma. Cuando el paso de tiempo es pequeño, se aplican las estrategias basadas en mejoras locales. Cuando este paso es grande, es más conveniente utilizar una estrategia de remallado de todo el dominio.

En este trabajo presentamos una estrategia basada en mejoras locales, que optimiza un criterio de calidad de elementos. Cuando el dominio se deforma considerablemente, esta estrategia se combina con un algoritmo de remallado de Delaunnay, logrando obtener mallas de excelente calidad. La estrategia de remallado ya ha sido presentado en (D'Amato Venere 2013), mostrando excelentes resultados para mallas estáticas y con bajos tiempos de procesamiento al correr en una arquitectura CPU + GPU. Esta estrategia ahora se extiende, para resolver el problema de mallas móviles.

2 ANTECEDENTES

La precisión de las simulaciones depende en gran parte de la calidad y tamaño de los elementos. Si los elementos se degradan al sufrir una deformación, se requiere aplicar algún método de remallado que permite recuperar un cierto grado de calidad. En general, para que este proceso sea más efectivo, el contorno del dominio debe preservarse.

Existen muchas propuestas que abordan esta problemática en lo que se conoce como contextos adaptativos. En (Budd et al 2009) se realiza un resumen de métodos de optimización de la calidad para mallas dinámicas. En general, aún se aplica un remallado total, en dominios con movimiento libre (Wojtan 2009).

Otro método muy aceptado para el tratamiento de mallas aplicables al cálculo científico es el basado en cambios locales, propuesto por (Coupez 1994), y aplicado en (Frey & Borouchaki 1998),(Wang et al 2006), entre otros. Esta estrategia permite lograr elementos de buena calidad incluso con diferencias de discretización significativas, como en la Figura 1. Algunos trabajos trabajan sobre parametrizaciones del dominio para realizar dichos cambios, tal como (Wicke et al 2010). pero se encuentra limitado a cierto tipo de deformaciones. En (Loseille & Löhner 2010), se propone un algoritmo adaptativo aplicado en simulaciones aerodinámicas que aplica una serie de operaciones de eliminar, añadir e intercambiar elementos de la malla, tal como aristas, puntos o caras.

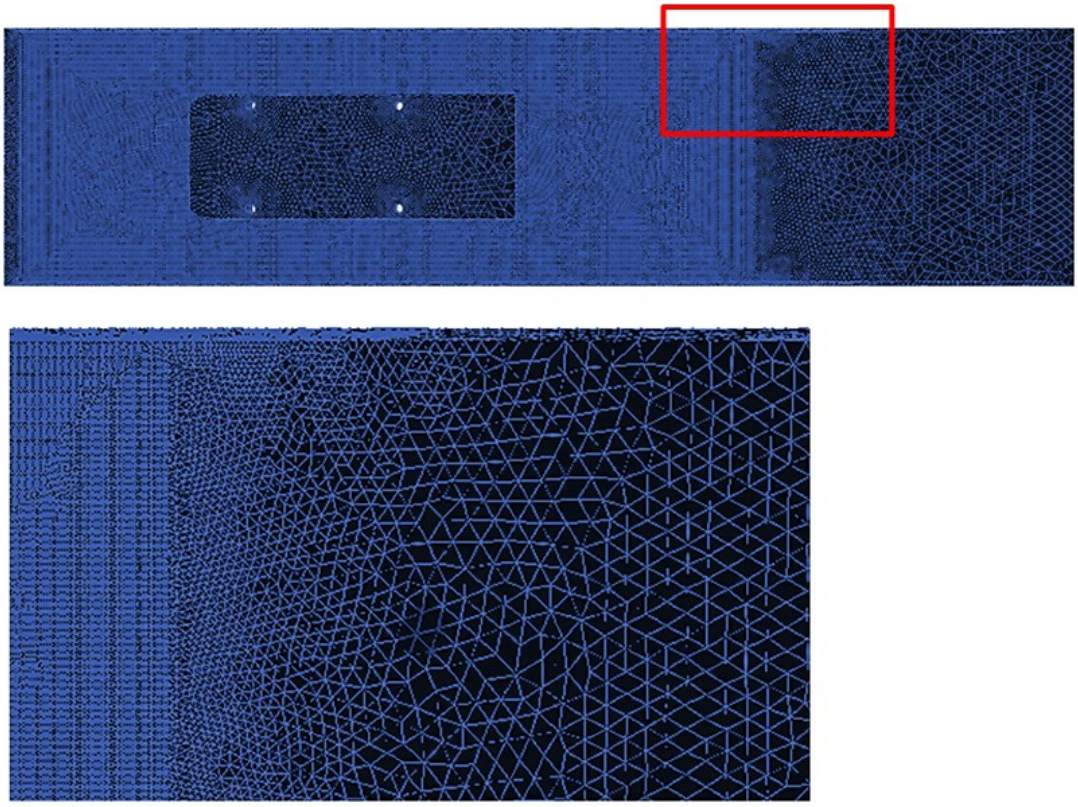

Figura 1: Ejemplo de malla con gran diferencia de densidad

Estas estrategias generalmente son iterativas. Cuando el número de elementos es muy elevado, los tiempos de remallado aumentan rápidamente; consumiendo más tiempo que la propia simulación. En estos casos, utilizar estrategias de aceleración a partir de arquitecturas en paralelo resulta muy adecuado. En general, la estrategia más convencional para el trabajo en paralelo, ya ideada en (Schloegel et al 1997) consiste en partir el dominio en partes o *parches;* cada parte de la malla se distribuye a diferentes procesadores en una red, en la cual se procesan los elementos internos y en una etapa posterior los elementos del contorno compartidos entre las secciones de la malla. Esta estrategia resulta interesante para mallas muy densas, pero añade un tiempo considerable de sincronización, y no suelen aprovechar las capacidades multi-núcleo de los procesadores modernos o incluso de las GPUs que cuentan con cientos de *threads* para la ejecución.

En este trabajo, presentamos una variante del método de remallado que corre en arquitecturas en paralelo con un esquema de memoria compartida. Este método, presenta ciertas similitudes con los anteriores, pero la idea es procesar pequeñas partes de la malla, con a lo sumo 100 elementos, que denominamos *clusters*. Estos *clusters* pueden trabajarse de forma independiente, sin necesidad de utilizar estrategias de bloqueo de datos, tal como se utiliza generalmente en sistemas de tiempo real. Tal como mostraremos la estrategia reduce considerablemente los tiempos de proceso; por lo que ahora extendemos la idea para trabajar con mallas que evolucionan en el tiempo. Algunos resultado de este grupo ya se han presentado en (D'Amato et al 2010) para mallas de superficie y (D'Amato & Vénere 201) para mallas de tetraedros,

3 ESTRATEGIA DE OPTIMIZACION DE LA CALIDAD

Los resultados de las simulaciones dependen en gran medida de la calidad de los elementos. Si las calidades son negativas o cercanas a 0, generalmente los resultados no tendrán sentido. Si a su vez, los nodos de la malla se encuentran en movimiento, es probable que la calidad de los elementos se degrade; es por esto que tiene importancia mantener una alta calidad de los tetraedros, aplicando un proceso de remallado en cada paso de la simulación.

Los métodos tradicionales (Hudson et al 2007), (Ito et al 2007) a veces no resultan ser los más adecuados, debido a que deben regenerar todo el dominio, con un costo computacional elevado y obteniéndose mallas con una calidad baja. Por otro lado, los métodos basados en cambios locales, generan mallas de mejor calidad pero también requieren de gran cantidad de procesamiento.

Lo que proponemos en este trabajo, es extender las estrategias tradicionales con un método basado en mejoras locales. Este método, propone tomar sub-conjuntos de tetraedros para conformar un *clúster* y evaluar posibles mejoras. Para medir la calidad de los tetraedros, utilizamos la métrica descripta en (Parthasarathy et al 1994) midiendo el peor ángulo diedro de los elementos. Como criterio global, utilizamos la calidad mínima de toda la malla. Este ángulo define la precisión numérica que puede lograrse en cada paso.

Proponemos dos formas de mejoras. La primera que realiza cambios en la topología de la malla, denominada "basada en reconexiones". La siguiente forma busca la posición del nodo central dentro del volumen delimitado por el *clúster* que mejore el criterio propuesto, denominada como "basada en óptima ubicación". Aunque estos métodos son muy costosos, asegura una mejora continua de la malla y tal como se describirá más adelante, pueden ser paralelizables bajo ciertas condiciones.

3.1 Optimización basada en reconexiones

El método propuesto es un método iterativo. Inicia cuando se toma un conjunto de tetraedros, que conforman un *clúster*, y se realizan un conjunto de reconexiones para generar nuevos elementos. Si esta nueva configuración mejora la métrica, se aceptan los cambios; si no se prosigue hasta analizar todas las configuraciones posibles. Una vez que se ha evaluado este *clúster*, se genera otro y se repite el proceso hasta que se han visitado todos los elementos de mala calidad.

En general, para formar el *clúster* se toma un nodo y todos los tetraedros que inciden en él, tal como se muestra en la Figura 2. Para que este proceso sea eficiente, es necesario mantener en alguna estructura la relación bidireccional entre todos los nodos y los tetraedros. En cada análisis local, el volumen y la superficie externa del *clúster* se mantienen; propiedad que luego utilizaremos para procesar varios conjuntos en simultáneo.

Dicho proceso es de mejora continua, y nunca se toman configuraciones que empeoren la calidad mínima global. Por otro lado, el análisis de cada *clúster* puede ser costoso, ya que se evalúan todas las conexiones posibles entre elementos, con un costo computacional cuadrático con respecto a la cantidad de elementos probados.

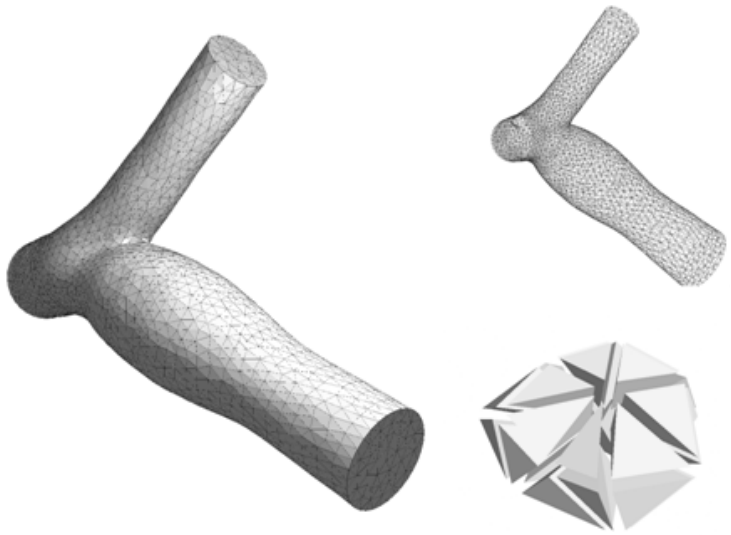

Figura 2: Elementos para formar un *cluster* incidentes a un vértice.

En el análisis de posibles conexiones, se utiliza un esquema general, descripto en (D'Amato & Vénere 2013), con ciertas similitudes al originalmente propuesto por (Coupez 1994). Este esquema analiza gran cantidad de variantes, mientras que otros trabajos proponen un conjunto acotado de cambios (Wicke 2010). El método se inicia a partir de la formación de un clúster de tetraedros, del cual se extrae su superficie exterior; formada por triángulos. A continuación, por cada uno de esos triángulos, se eligen un nodo del *cluster* que también esté sobre la superficie, y se crea un nuevo tetraedro. Este tetraedro, debe mejorar la métrica de calidad inicial del clúster; sino es descartado. A continuación, se elige otro triángulo del tetraedro y otro nodo. Esto se repite hasta que se han visitado todas las caras y se han utilizado todos los nodos de la superficie del clúster.

En caso que se ha llegado a una solución posible donde se han utilizado todas las caras, se evalúa la calidad mínima del conjunto. Si esta calidad, es mejor que la anterior; se almacena la solución de forma parcial. En general, existen varias combinaciones que mejoren la configuración inicial. Al terminar de analizar todas las combinaciones posibles y haber encontrado una mejor solución; se actualiza la mala, generando los nuevos tetraedros y eliminando los anteriores.

Dado que existen varias estructuras que describen a los tetraedros (nodos, tetraedros, tetraedros incidentes a cada nodo, entre otras), la actualización debe realizarse con cuidado asegurando la coherencia estructural. El proceso de actualización suele ser costoso, por lo que se propone una actualización masiva; todos los cambios se realizan en una sola pasada, asegurando que no se han cambiado dos tetraedros en distintos *clusters*. Además, detectamos una relación, que la cantidad de elementos de la nuevo configuración es +/- 1 la cantidad de elementos del *cluster* original; por lo que la cantidad de memoria puede acotarse.

A pesar que no podemos asegurar que se hayan analizado todas las conexiones posibles, si podemos afirmar que el número de casos evaluados en mucho mayor que los previamente publicados.

3.2 Optimización basada en ubicación óptima de los nodos

Se reconoce en la literatura del tema, que el suavizado es un buen método para mejorar la calidad de la malla con un bajo costo computacional, manteniendo la conectividad de los

elementos de la malla. En general, como se observa en los trabajo de (Zhang et al 2005), el suavizado tiene a mejorar la calidad promedio, pero muchas veces perjudica considerablemente la calidad mínima; e incluso llegan a generar elementos invertidos. Este fenómenos se produce en mallas con muchas concavidades.

Para asegurar que siempre se mejora la calidad de los elementos, se propone un algoritmo de mejora incremental. Básicamente, se modifica la posición del nodo central del clúster. Para eso, se proponen un conjunto de puntos candidatos, ubicados a una distancia inicial r de la posición del nodo y se escoge aquel que mejore el criterio de calidad propuesto. A continuación, se actualiza el nodo con la posición escogida, y se inicia la búsqueda nuevamente. Si no se ha encontrado ninguna posición mejor, se actualiza la distancia búsqueda como $r= r/2$ y se evalúa nuevamente. El algoritmo continua hasta que r es menor que una tolerancia dada.

Si el nodo en algún momento sale del clúster, los elementos se invierten, por lo que nunca se considera como posición válida. En la Figura 3 esquematizamos cómo funciona el método.

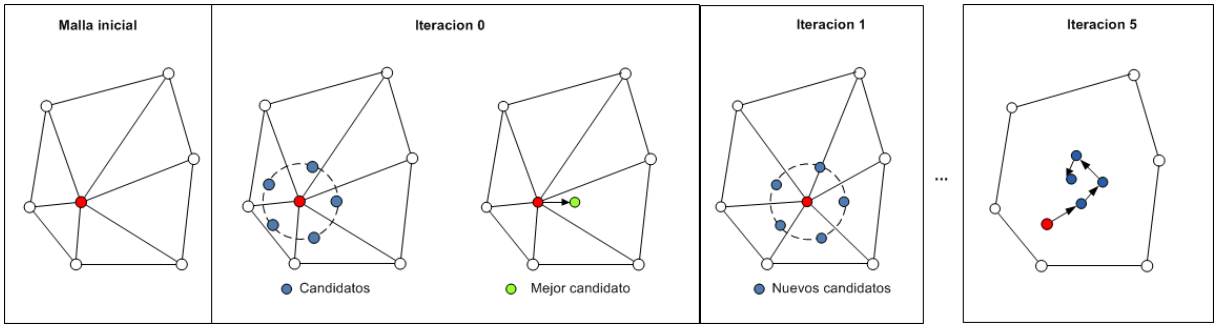

Figura 3: Ubicación óptima del nodo.

Como r inicial se escoge la mitad de la arista mínima de los tetraedros del clúster y 25 puntos candidatos ubicados al azar a esta distancia. Con estos parámetros, se requieren cerca de 20 iteraciones para converger. Se puede reducir la cantidad de puntos candidatos, para que el método converja antes.

3.3 Secuencia de operaciones

En el proceso de remallado, se debe minimizar el movimiento de nodos; porque esto produce un fenómeno de difusión artificial; producto de la interpolación de los datos de la simulación para los nuevos elementos generados. Para reducir este fenómeno, es conveniente realizar primero las operaciones conservativas (como las operaciones topológicas) y luego las operaciones más agresivas (como las de movimiento de nodos).

Para generar los clústeres de manera eficiente, se requiere cierta información de conectividad, tal como los tetraedros incidentes al nodo y los nodos vecinos a cada nodo, estructuras que deben ser accesibles en todo momento. Otras estructuras útiles, tal como la superficie externa a un clúster (compuesta por triángulos) se calculan dinámicamente solo cuando se analiza dicho clúster. De esta manera, se reduce considerablemente la cantidad de memoria utilizada y la cantidad de estructuras a actualizar.

Los cambios en la topología se registran en dos listas, una para los elementos creados; otra para los elementos eliminados. Una vez que se han procesado todos los posibles clústeres, se

actualiza la estructura de la malla en un único paso, llamado "REESTRUCTURAR". De esta forma, se reduce la cantidad de solicitudes y actualizaciones de memoria, acelerando considerablemente la etapa de actualización de estructuras.

En el siguiente seudocódigo, resumimos el proceso de remallado propuesto.

Function *RESTRUCTURAR (Mesh M , list newElements, list oldElements)*

For each *e* ***in*** *newElements*

If (e is node) M.addNode(e)
If (e is tetrahedra)
M.addtetra(e)
End for
For each *e* ***in*** *oldElements*
If *(e* ***is*** *node) M.removeNode(e)*
If *(e* ***is*** *tetrahedra) M.removetetra(e)*
End for
updateNeighborhood(M , newElements) { Actualiza las relaciones de los elementos creados }

Function *OPTIMIZACIÓN DE ELEMENTOS (Mesh M, ElementType ET)*

elements = conjunto de todos los elementos del tipo de *M elegido (nodo, arista, cara)*
For each *e* ***in*** *elements*
Cluster C = generateCluster(n) {conjunto de tetraedros incidentes a e}
Q = Optimizar(C, newElements, oldElements) {almacena los cambios en una lista y retorna la calidad obtenida }
If (Q > minQuality)
acceptChanges(C)
end for
Restructurar (M , *newElements, oldElements);*

Function Suavizar(Mesh *M*)
nodes = conjunto de todos los nodos de *M*
For each *n* ***in*** *node*
Cluster C = generateCluster(n) {conjunto de tetraedros incidentes a e}
Q = BuscarMejorPosicion(C , newPos) {almacena la mejor posición obtenida }
If *(Q > minQuality)*
n.position = newPos;
end for

Function OPTIMIZACION DE LA MALLA (Mesh *M, doublé minQuality*)

Tetras = conjunto de todos los tetrahedros de *M*
While (true)
OPTIMIZACIÓN DE ELEMENTOS(Mesh *M, Node*)
OPTIMIZACIÓN DE ELEMENTOS(Mesh *M, Edge*)
OPTIMIZACIÓN DE ELEMENTOS(Mesh *M, Face*)
SMOOTH MESH(M)

```
    badElements = conjunto de tetraedris con calidad inferior a minQuality
    if (badElements.count == 0) break;
  end while
```

4 REMALLADO EN PARALELO

El método de remallado propuesto tiene le ventaja que los elementos son analizados en grupos acotados. Como se ha descripto en la sección anterior, cada cambio en la topología, requiere que se actualicen ciertas estructuras de vecinos. En otros trabajos, estas actualizaciones simultáneas, suelen resolverse con mecanismos de sincronización tal como barreras o semáforos. Cuando la cantidad de actualizaciones es muy elevada, como en los casos que proponemos, el mecanismo de sincronización frenan todo el tiempo de proceso. Por otra parte, se puede prever que los cambios entre dos o más clústeres que no comparten elementos (ni nodos, ni aristas, ni tetraedros) pueden procesarse en paralelo, condición que denominamos de "independencia mutua" entre pares de clústeres.

Proponemos entonces, que es posible procesar múltiples conjuntos en paralelo, si cumplen esta condición al mismo tiempo. Para esto, es necesario definir un algoritmo de planificación del procesamiento, el cual tiene la función de escoger de forma eficiente aquellos clústeres mutuamente independientes. Aunque esto impacta en el orden que se realiza el procesamiento, se sigue cumpliendo la condición de optimización de la calidad.

Este esquema es muy versátil, dado que se puede aplicar tanto para las mejoras basadas en re conexiones, como para las basadas en movimiento de vértices. En esta sección, se describe el algoritmo.

4.1 Planificación de la ejecución

Dado que se desea procesar la mayor cantidad posible de elementos en simultáneo, debe existir un método de distribución eficiente del procesamiento, que asegure las condiciones antes planteadas. Para el caso particular de conjuntos de tetraedros, la asignación puede ser bastante sencilla. En primer lugar, se debe escoger los elementos que tengan peor calidad, a fin de procesarlos primero. Para esto, se ordenan en forma creciente de acuerdo al criterio de ángulos diedros nombrados.

A continuación, se toma el primer elemento, se genera un clúster con este elemento y sus vecinos y se marcan todos como visitados y se continúa. Si ni el elemento siguiente en la lista ni ninguno de sus vecinos ha sido visitado, se crea un nuevo clúster, sino, se omite y se toma el siguiente. Una vez que se han asignado todos los posibles elementos, se inicia el procesamiento en paralelo.

```
Function SuavizarParalelo(Mesh M)

  Nodes = conjunto de todos los nodos de M
  nt = numero de threads/procesadores disponibles
  while not empty(nodes)
      Np = distribuirNodosaProcesar(M )
      Parallel for each node in Np
          Cluster C = generateCluster(n) {conjunto de tetraedros incidentes a e}
          Q = BuscarMejorPosicion( C , newPos ) {almacena la mejor posición obtenida }
          If (Q > minQuality)
            n.position = newPos;
```

end parallel for
remove(Nodes, Np);//quito los nodos ya procesados
end for

Cuando se compara la ejecución de la versión secuencial, con respecto a su versión en paralelo, surgen ciertas cuestiones a resolver. Primero, los elementos que se descartaron pero aún tienen mala calidad, deben ser procesados. Segundo, si se crearon nuevos elementos, los mismos pueden seguir teniendo una calidad inferior a la esperada. Es por esto, que luego de cada iteración; se debe repetir el proceso hasta que no existan elementos inválidos. También sucede, que la cantidad de elementos a procesar decrece, y en la última iteración quedan procesadores ociosos. Cuando la cantidad de elementos a procesar es muy alta (Numero elementos >> numero de procesadores), el tiempo ocioso es despreciable.

4.2 Procesamiento en la GPU

La GPU es un recurso muy interesante, por la gran cantidad de unidades de cálculo disponibles. Por otro lado; tiene una menor cantidad set de operaciones que de una CPU por lo que se requiere rever como se implementan ciertas estrategias. El principal problema surge cuando se requiere realizar muchos accesos a memoria del dispositivo o alocaciones de memoria en forma dinámica. Aunque a partir de la versión 4.0 de CUDA incluye ciertas funciones tal como *new* o *free*, estas aun no están disponibles en OpenCL. En estas condiciones, la actualización de estructuras en GPU resulta siendo igual o incluso más lento que en su versión equivalente a CPU.

Por otro lado, el algoritmo de ubicación optima de nodos resulta más atractivo para implementar en estas arquitecturas, ya que la topología de la malla no se modifica y la cantidad de cálculos realizadas en muy elevada. En estas condiciones, se realiza una implementación del algoritmo de suavizado para GPU. Tal como se conoce, las placas graficas tienen su propio espacio de memoria, por lo que deben añadirse dos pasos de copia, desde y hacia la placa grafica, a fin de compartir los datos entre ambas arquitecturas. Previo a la ejecución en GPU, se aplicar el esquema de planificación ya propuesto.

Las implementaciones del método de suavizado para CPU y GPU son prácticamente iguales, ambas fueron implementadas con C Ansi90. Se utilizan directivas de compilación para invocar funciones especificas de cada arquitectura, tal como obtener el ID del Thread que se está accediendo en cada momento (función get_threadID). Además, se definen tipos de punteros a datos. La desventaja de estas definiciones, es que siempre se utilizan estructuras alojadas en memoria global del dispositivo, la cual es considerablemente más lenta. Tambien difieren el modo de invocar al procesamiento en paralelo.

A continuación, se muestran algunas secciones del código propuesto:

```
#if defined GPU
    typedef global int* global_int_ptr ;
    typedef global float4* global_float4_ptr ;
#else
    typedef int* global_int_ptr;
    typedef float4* global_float4_ptr
#endif

int get_ThreadID(int dim)
```

```
{
#if defined GPU
      return get_global_id(dim);
#if defined OPENMP
                return  omp_get_thread_num();
#if defined GPU
}

#if defined GPU
kernel
#END
Void SmoothKernel(global_int_ptr  assignment,
         global_int_ptr neighbours
        ,global_int_ptr* tetra,
        global_float4_ptr* vertexPositions)
{

    size_t x = get_ThreadID(0);
    int vertexID = assignment[x];
   ....
}
```

A pesar que no se aprovechan al máximo las prestaciones de las GPUs, el código resultante es portable (corre tanto en CPU como GPU) y de mejor mantenibilidad.

4.3 Tratamiento de mallas evolutivas

El procesamiento de mallas que evolucionan en el tiempo considera que en cada paso de tiempo se procesa una malla estática. Se comienza con una malla en un estado inicial. A esta malla se le aplica un primer proceso de mejora, buscando una mejor configuración de calidad. A continuación, se le aplica una función de movimiento determinada con un paso de tiempo *dt*. Luego, se analiza el estado de la malla, evaluando cuantos elementos se han deformado, eligiendo aquellos cuya calidad se encuentra por debajo de una tolerancia dada y discriminando entre elementos de baja calidad y elementos invertidos.

De acuerdo a la cantidad de elementos de baja calidad detectados, se elige que estrategia adoptar. Si la cantidad de elementos invertidos es significativa, es más conveniente aplicar un remallado global. En otro caso, siempre es más conveniente aplicar nuestra estrategia de remallado localizado, dado que requiere de pocas iteraciones para converger.

Empíricamente, hemos determinado que cuando la cantidad de elementos invertidos es mayor al 5%, es conveniente aplicar un remallado global. En otro caso, cuando la mayoría son elementos de baja calidad pero de valor positivo, siempre conviene aplicar nuestra estrategia, obteniendo una calidad mínima de 15º.

Ciertamente, cuanto mayor es el paso del tiempo, se generan mayor cantidad de elementos negativos. Por otro lado, como las mallas son adaptativas, ya sea para modelar una interfaz entre materiales o una sección particular de volumen, el paso de tiempo es proporcional al tamaño de los elementos más pequeños. Esto significa que por cada iteración, generalmente son muy pocos los elementos que se modifican; y es donde nuestra estrategia se obtiene una relación de aceleración mucho mayor.

5 RESULTADOS

Para evaluar cómo se comporta nuestro algoritmo corriendo en paralelo, utilizamos mallas con gran cantidad de elementos. En la Figura 4 presentamos los casos de estudio. En (D'Amato & Vénere 2013), mostramos que la aceleración se comporta cuasi-linealmente con respecto a la cantidad de procesadores. Esta relación, se debe a que la etapa de generación de nuevos elementos y la etapa de asignación del procesamiento se resuelve de forma lineal.

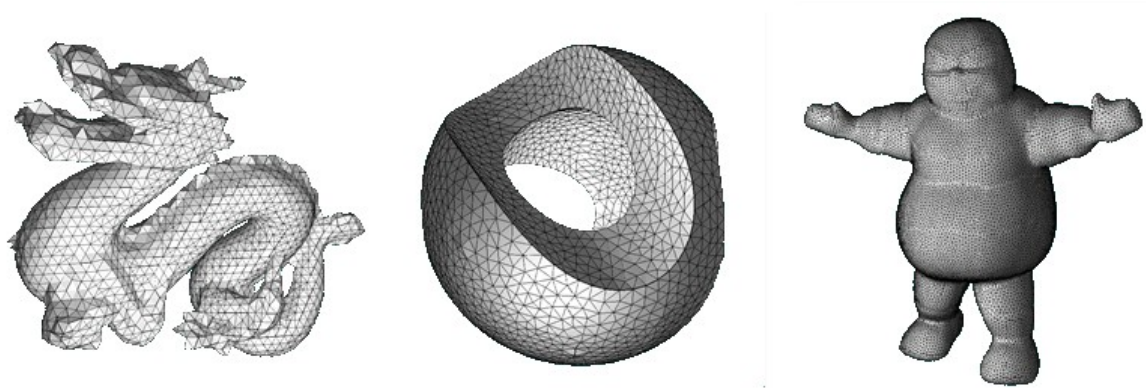

Figura 4: Mallas utilizadas en las pruebas.

En las pruebas se utilizó un AMD 6X de 6 núcleos a 3.3 GHZ, con 4GB de RAM y una Nvidia GTX 550i, corriendo con OpenMP. En la Tabla 1, mostramos los resultados de aceleración obtenidos comparando los tiempos y las calidades de nuestro algoritmo con respecto a un optimizador de calidad de mallas de uso abierto como Stellar (Wicke et al 2010). Como las mallas originales se generaron con un método de Delaunnay, en estos casos no lo consideramos para la evaluación. También se probó la biblioteca de Intel® Thread Building Blocks(TBB) para la implementación, obteniendo aceleraciones similares con respecto a la versión OpenMP.

	#elementos	Calidad inicial	Calidad minima (Stellar)		Tiempo de proceso		
			Stellar	Nuestro metodo	Stellar	Nuestro metodo	speedup
Dragon	32959	15.45º	27.11º	28.11º	28.0s	2.17s	12.9x
Sculp	50391	11.45º	30.05º	26.4º	110.0s	3.58s	30.7x
staypuft	102392	1.14º	18.10º	11.07º	198,0s	5.72s	34.6x

Table 1: Comparativa de tiempo y mejoras de calidad.

Las mejores aceleraciones se obtuvieron para el caso de la malla de *staypuft*, con una aceleración de 34.6x con respecto a Stellar. En general, las calidades obtenidas con ambos métodos son similares pero las aceleraciones de nuestra propuesta son mucho mayores.

5.1 Evaluación de mallas que evolucionan en el tiempo

Para la evaluación de nuestra estrategia, tanto en un contexto dinámico, utilizamos una malla de una cavidad cúbica definida en el rango (0,0,0) al (1,1,1) con una resolución de $6x10^5$ elementos. Esta malla tiene una calidad inicial de 1º, tal cual como se muestra en la sección anterior. A continuación, aplicamos un movimiento circular en torno al eje Z del centro de la cavidad. Los nodos de la superficie de la cavidad se mantienen fijos. A partir de la primera deformación, la calidad mínima fue de -0.05º con 48 elementos invertidos. Aplicando nuestro algoritmo de optimización, logramos una calidad de 6º. A continuación repetimos el proceso reiteradamente.

En la comparación, repetimos el mismo proceso de deformación de la malla, pero en cada paso aplicamos un remallado global para eliminar los elementos invertidos. Utilizamos un algoritmo basado en Delaunnay 3D con restricciones con la biblioteca gratuita *tetGe (*SI 2011). En esta evaluación utilizamos la misma configuración de hardware descripta en la sección anterior.

En la Figura 5, mostramos cómo se comportan las calidades en los distintos casos. En la misma gráfica, podemos observar que aparecen elementos invertidos luego de la 1ra iteración. También podemos observar, que a partir de la 8va iteración, aparecen gran cantidad de elementos invertidos que no pueden ser resueltos, por lo que conviene aplicar un remallado global.

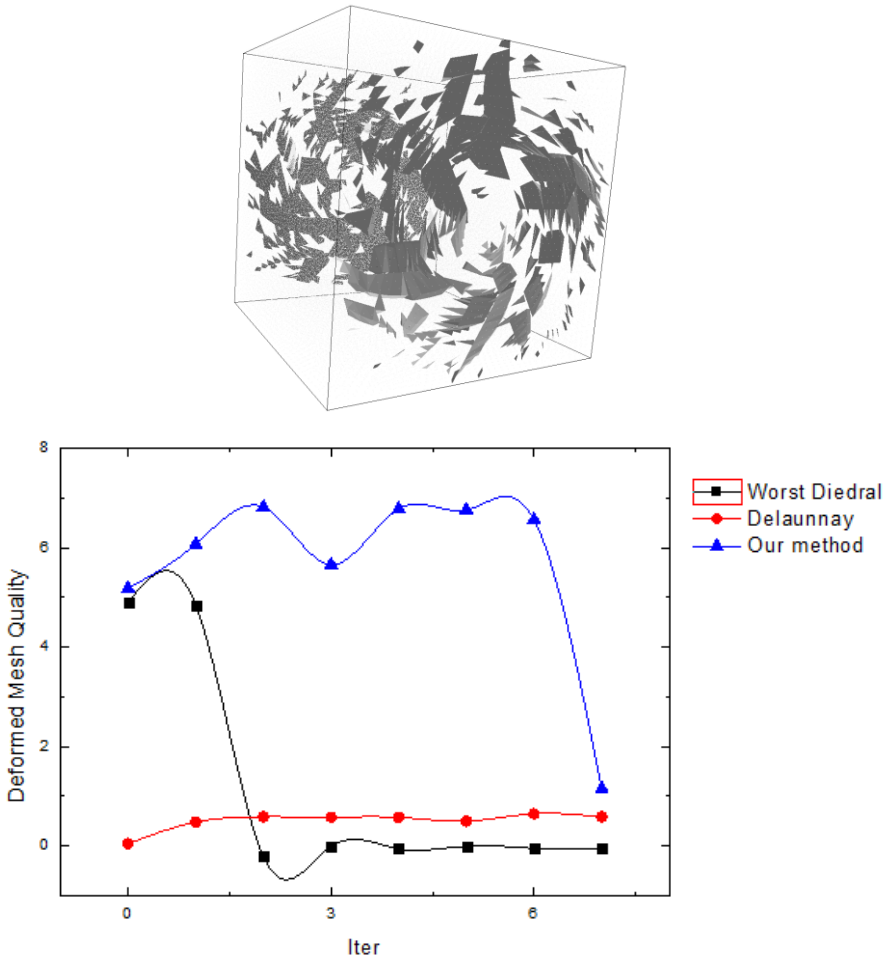

Figure 5: (arriba) Cantidad de elementos invertidos luego de 2 iteraciones (abajo) Comparativa de la evolución de la peor calidad, sin remallado y con los 2 métodos propuestos.

Tomando los tiempos de remallado de cada paso, obtuvimos que el tiempo promedio de un paso de Delaunnay es de 25,14s, dando un total de 201s para las 8 iteraciones. Por otro lado, nuestro método requiere de 8.57s por iteración, con un total de 72s. De la misma gráfica se observa como siempre la calidad obtenida por nuestro método es mucho mejor que con el otro método.

6 CONCLUSIONES

En este trabajo se presenta un algoritmo de remallado, diseñado especialmente para trabajar en arquitecturas altamente paralelizables. La implementación de la solución es relativamente sencilla, ya que utiliza bucles en paralelo, provistos por bibliotecas tales como OpenMP, TBB e incluso OpenCL para las GPUs.

En este desarrollo, hemos presentado solo un conjunto de posibles configuraciones. La estrategia propuesta es muy flexible ya que tanto las operaciones, como la métrica o la forma de generar los clústeres pueden ser modificadas o extendidas. A su vez, asegurando las condiciones de independencia mutua, es posible paralelizar fácilmente su procesamiento. La misma idea ha sido aplicada también para mallas de superficie, tomando agrupaciones de triángulos y evaluando el perímetro de los elementos.

Las aceleraciones obtenidas creemos son importante, aunque pueden ser mejoradas. La idea es realizar todo el proceso en la GPU, especialmente las modificaciones de la topología. Para lograr esto, estamos trabajando en un esquema de manejo de memoria seudo-dinámico que permite que tanto la creación como destrucción de los elementos se realice enteramente en la placa gráfica.

REFERENCES

Budd, C. J., Huang, W., Russell, R. D. Adaptivity with moving grids. In Acta Numerica 2009, vol. 18. pp. 1–131 (2009).

Cohen, Varshney, Manocha, Turk, Weber, Agarwal, Brooks, and Wright. Simplification envelopes. In: ACM SIGGRAPH ’93 Conference Proceedings. New Orleans, Louisiana; pp. 119–28 (1996).

Coupez T. , Digonnet H. , Ducloux R. Parallel meshing and remeshing, Applied Mathematical Modelling vol. 25, 2 pp.153-175(2000).

Coupez T. A mesh improvement method for 3D auto-matic remeshing. 4th International conference on numerical grid generation in computational fluid dynamics and related fields, pp. 615-626, (1994).

Dehne, Langis, and Roth. Mesh simplification in parallel. In: Proceedings 4th International Conference on Algorithms and Architectures for Parallel Processing. Hong Kong; pp. 281–90 (2000).

D'Amato J.P; Vénere, M.. A CPU–GPU framework for optimizing the quality of large meshes”. Journal Of Parallel And Distributed Computing. Academic Press Inc Elsevier Science. (2013) .

D’Amato, J.P., Lotito P., Vénere M., Remallado en Paralelo Utilizando un Esquema de Partición Implícita de Datos, Mecánica Computacional, Volume XXIX. Number 62, pp5999-6012,ISSN 1666-6070 (2010) .

Frey P. and Borouchaki H.. “Geometric surface mesh optimization”. Computing and Visualization in Science, pp 113–121, (1998).

Hudson, B., Miller, G., Phillips, T.Sparse Parallel Delaunay Mesh Refinement. In:ACM

Symposium on Parallelism in Algorithms and Architectures, California, USA (2007).
Ito Y., Shih A., Erukala A., Soni B.,Chernikov A., Chrisochoides N., Nakahashi N., Parallel unstructured mesh generation by an advancing front method. In : Mathematics and Computers in Simulation vol. 75 pp. 200–209 (2007).
Klingner, B. M. Tetrahedral Mesh Improvement. PhD thesis, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, California.(2009).
Loseille A., Löhner R. Anisotropic Adaptive Simulations in Aerodynamics; AIAA-10-0169 (2010)..
Parthasarathy, V. N., Graiche, C. M., Athaway, A. F. A comparison of tetrahedron quality measures.Finite Elements in Analysis and Design vol. 15-3, pp. 255-261 (1994).
Schloegel K., Karypis G., Kumar V. Multilevel diffusion schemes for repartitioning of adaptive meshes. Journal of Parallel and Distributed Computing, vol.47-2, pp.109-124 (1997).
Si, H. A Quality Tetrahedral Mesh Generator and a 3D Delaunay Triangulator. Site : http://tetgen.org (2011).
Southern, J., Gorman G.J. , Piggott M.D. , Farrell P.E. Parallel anisotropic mesh adaptivity with dynamic load balancing for cardiac electrophysiology. Journal of Computational Science vol.3 pp. 8-16 (2012).
Wang D., Hassan O., Morgan K. , Weatherill N. “EQSM: An efficient high quality surface grid generation method based on remeshing” Comput. Methods Appl. Mech. Engrg. Vol.195 pp. 5621–5633(2006).
Wicke M. Bryan D., Klingner M., Burke S., Klingner M., Shewchuk J. O’Brien J.Dynamic Local Remeshing for Elastoplastic Simulation. Computer Graphics Proceedings, Annual Conference Series, vol. 49 pp.1-12 (2010).
Wojtan, C., Turk G. Fast viscoelastic behavior with thin features. ACM Transactions on Graphics vol. 27-3 pp.47 (2008).
Wojtan, C., Thurey, N., Gross, M., and Turk G. . Deforming meshes that split and merge. ACM Transactions on Graphics vol. 28-3 (2009).
Zhang Y., Bajaj C., and Xu G.. Surface smoothing and quality improvement of quadrilateral/hexahedral meshes with geometric flow. In Proc. 13th Int. Meshing Roundtable, (2005)

.