CPU + GPU: Un entorno de cómputo de alto rendimiento. Su aplicación a métodos de mallado de elementos finitos.

Santiago Montiel (1) , Adriana A. Gaudiani (1),
Gabriel Acosta Rodríguez (2)

Departamento de Matemática, Facultad de Ciencias Exactas,
Universidad de Buenos Aires.
Area de Computación, Instituto de Ciencias,
Universidad Nacional de General Sarmiento.
(1) {agaudi,smontiel}@ungs.edu.ar, (2) gacosta@dm.uba.ar.

Resumen

NVIDIA abrió la arquitectura de los procesadores gráficos de sus placas de video (GPUs) para ser utilizados en aplicaciones de propósito general y desarrolló el lenguaje de programación CUDA permitiendo a los programadores crear funciones con explícito paralelismo de datos, brindando una plataforma de cómputo de alto rendimiento y paralelismo a gran escala.

Este trabajo forma parte del proyecto *"Métodos numéricos para ecuaciones diferenciales y aplicaciones"* que reúne investigadores de la Universidad de Buenos Aires y de la Universidad Nacional de General Sarmiento, y tiene como objetivo utilizar dicha arquitectura de cómputo paralelo para acelerar el tiempo de ejecución de un método de mallado de elementos finitos creado por P. Persson y G. Strang para MATLAB.
Dado que los GPUs poseen decenas de unidades de procesamiento, son adecuadas para el procesamiento de datos en paralelo. Esto es una ventaja para el algoritmo seleccionado, el cual presenta un evidente paralelismo de datos.

Palabras claves Algoritmos Paralelos, Procesadores Gráficos, GPU, Cómputo de Alto Rendimiento, Paralelismo de datos.

Contexto de la Investigación

Este trabajo es parte del proyecto PICT2007-910, "Métodos numéricos para ecuaciones diferenciales y aplicaciones"que reúne investigadores de la Facultad de Ciencias Exactas y Naturales de la Universidad de Buenos Aires y del Instituto de Ciencias de la Universidad Nacional de General Sarmiento.
La generación de mallas es un insumo indispensable en la implementación del

método de elementos finitos. En particular, para mallas en 3D, el costo computacional es muy elevado. Sería de enorme utilidad acelerar el tiempo de ejecución de estos métodos aprovechando el poder de cómputo que ofrecen actualmente los procesadores gráficos (GPUs) presentes en las tarjetas gráficas. [2] [7]

- Las causas de esta elección

La elección de esta tecnología de cómputo paralelo se debió a diversos factores. Ellos son:

- La facilidad de acceso a los GPUs como los que se encuentran en las tarjetas gráficas NVIDIA, algunas de bajo costo.
- El modelo de programacion de CUDA permite programar aplicaciones de propósito general, fácilmente escalables a GPUs con más cantidad de cores. Esta arquitectura se comporta como un co-procesador paralelo de la CPU ofreciendo un gran rendimiento a aplicaciones que puedan correr sobre una gran cantidad de hilos de ejecución independientes.[6] [3]
- El paralelismo de datos es una característica propia del algoritmo de mallado. Esta propiedad permite realizar muchas operaciones a-ritméticas sobre las estructuras de datos y *de manera simultánea*. Esta es una situación ideal para el modelo de programación de CUDA, *obteniendo el máximo provecho de la arquitectura SIMT*-Single Instruction-Multiple Thread que la caracteriza. [1] [5]

- El problema de Aplicación

Per-Olof Persson y Gilbert Strang, desarrollaron un código generador de mallas para MATLAB, llamado *DistMesh*, el cual está a disposición del público[1]. Este código fue escrito con el objetivo de brindar un método simple y de muchas menos líneas que otras técnicas de mallado. Persson-Strang toman el mallado que provee el algoritmo de Delaunay[12],[11] y, según se explica a continuación, utilizan una técnica que le permite mejorarlo y obtener un mallado de más calidad.[8]

La idea de este algoritmo es utilizar una simple analogía mecánica entre una malla de triángulos o tetraedros, y una estructura de resortes. Cualquier conjunto de puntos en el plano puede ser triangulado por el algoritmo de Delaunay.[10] Los puntos de la malla son nodos de una estructura armada con "barras" o "resortes" que se encuentran comprimidos dentro del dominio. Se trata de dejar evolucionar el sistema mediante un proceso iterativo hasta llegar algún equilibrio establecido para el método, por ejemplo cuando la sumatoria de fuerzas en cada nodo sea cero, a menos de una tolerancia. Si en cada iteración algún nodo es reubicado fuera del dominio, entonces debe ser reproyectado hacia la superficie. En el caso que los puntos se separen en más de alguna tolerancia preestablecida para el método se vuelve a utilizar Delaunay para generar un nuevo mallado, y se

[1] http://persson.berkeley.edu/distmesh/

sigue adelante con el método general.

El código MATLAB de DistMesh fue vectorizado para evitar los ciclos-for y lograr mayor eficiencia aunque aun podría ser mejorado según explican los autores en [9].

Para este trabajo hemos seleccionado, como herramienta para el desarrollo, una implementación del algoritmo de Delaunay, del tipo Divide y Vencerás escrito por Geo Leach en lenguaje C[2]. Este algoritmo es de complejidad O(n logn), la mejor complejidad lograda, siendo n la cantidad de puntos del dominio. El mismo fue optimizado como explica su autor en [4].

Objetivos teóricos y experimentales

Se propone disminuir el tiempo de ejecución del método propuesto por Per-Olof Persson y Gilbert Strang, para generar un mallado de elementos finitos en 2D y 3D, migrando a una arquitectura de cómputo paralelo de alto rendimiento como la ofrecida por los procesadores gráficos GPUs.

En una primera etapa se implementa la versión paralela y 2D de DistMesh usando CUDA. En esta etapa es de suma importancia enfocar los esfuerzos en maximizar el paralelismo de datos del algoritmo. Una manera de lograrlo será concentrándose en la naturaleza de los datos y en su organización durante el cómputo.

Este desarrollo en 2D contiene los ingredientes conceptuales necesarios para permitir la extensión al caso 3D. En esta segunda etapa del trabajo se implementará el caso 3D del método, resultando en consecuencia un incremento de la necesidad de memoria. Esto requiere mejorar aun más el acceso a los diferentes niveles de memoria presentes en la arquitectura de los GPUs mediante el uso del modelo de programación CUDA.

En una etapa final se escribirá la nueva función de mallado para Matlab optimizada, o sea, se elaborará una nueva función DistMesh modificada para que ejecute haciendo uso de los GPUs. Esto implica la escritura de una función de tipo MEX para ser incluida dentro de las funciones de Matlab.

Formación de recursos Humanos

Este segmento del proyecto pretende fortalecer el área de matemática aplicada de la UNGS promoviendo el contacto con investigadores del departamento de matemática de la Facultad de Ciencias Exactas y Naturales de la UBA. Además, uno de los coautores del trabajo está ingresando al Doctorado en Ciencia y Tecnología de la UNGS, sirviendo los temas desarrollados en este trabajo de guía para su inserción en los tópicos que abordará en su tesis.

Agradecimientos

Los integrantes de este trabajo quieren agradecer a los investigadores del Instituto de Investigación en Informática-LIDI de la Universidad Nacional de La Plata, dirigido por el Ing. Armando De Giusti, cuya ayuda inestimable está presente siempre en el mo-

mento de requerirla.

Referencias

[1] Michael Garland. Sparse matrix computation on manycore gpu´s. *Annual ACM IEEE Design Automation Conference. DAC '08: Proceedings of the 45th annual Design Automation Conference.*, pages 2–6, 2008.

[2] Tom R. Halfhill. Parallel processing with cuda. nvidia´s high-performance computing platform uses massive multithreading. *Microprocessor Report*, January 2008.

[3] Tianyl Han and Tarek. Abdelrahman. hicuda: High-level gpgpu programming. *IEEE Transaction on Parallel and Distributed Systems.*, 21, 2010.

[4] Geoff Leach. Improving worst-case optimal delaunay triangulation algorithms. In *In 4th Canadian Conference on Computational Geometry*, page 15, 1992.

[5] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. *Scalable Parallel Programming*. ACM QUEUE, April 2008.

[6] NVIDIA Corporation, Cuda Toolkit. *Optimization. NVIDIA CUDA C Programming. Best Practice Guides.*, July 2009.

[7] John Owens, Mike Houston, David Luebke, Simon Green, John Stone, and James Phillips. Gpu computing. *Proceedings of the IEEE*, 96(5):879–899, Mayo 2008.

[8] Per-Olof Persson. *Mesh Generation for Implicit Geometries*. PhD thesis, Massachusetts Institute of Technology, February 2005.

[9] Per-Olof Persson and Gilbert Strang. A simple mesh generator in matlab. *SIAM Review*, 46:329–345, 2004.

[10] Guodong Rong, Tiow-Seng Tan, Thanh-Tung Cao, and Stephanus. Computing two-dimensional delaunay triangulation using graphics hardware. In *I3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games*, pages 89–97, New York, NY, USA, 2008. ACM.

[11] Jonathan R. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications*, 22:21–74, 2002.

[12] Jonathan Richard Shewchuk. Mesh generation for domains with small angles. In *SCG '00: Proceedings of the sixteenth annual symposium on Computational geometry*, pages 1–10, New York, NY, USA, 2000. ACM.