Uso de GPU computing en Matemática Experimental

Ing. Mg. Carlos A. Correa

Departamento de Informática / Facultad de Ciencias Exactas, Físicas y Naturales / UNSJ
Av. Ignacio de la Roza 590 (O)
Teléfonos: 4234129 - 4260355
carcorrea@hotmail.com.ar

Resumen

Este trabajo supone el uso de herramientas computacionales, capaces de analizar grandes cantidades de datos, para dar credibilidad, mediante la búsqueda de contraejemplos, a una nueva conjetura sobre números coprimos. Dicha conjetura postula que: *"Entre dos pares de múltiplos consecutivos de todo número natural $k>1$ siempre hay al menos un número que es coprimo de todos los números naturales $\leq k$"*.

Cuanto más extensiva sea la búsqueda de contraejemplos más fuerte se hará la conjetura planteada. Por ello, a fin de hacer que dicha búsqueda resulte además eficiente, en este trabajo se exploran y aplican técnicas de computación paralela SIMD de alta performance. Se trata, en concreto, del uso de GPU Computing y programación CUDA como plataforma para la implementación y ensayo de dos algoritmos de cribado especialmente diseñados para la búsqueda de coprimos. Uno de estos algoritmos fue diseñado respetando estrictamente lo que establece la conjetura, pero es a su vez el que demanda más recursos computacionales (tiempo + memoria).

El segundo algoritmo fue concebido aplicando más restricciones que las establecidas por la conjetura original. Esta idea permitió, a riesgo de toparse con "pseudo-contraejemplos", reducir significativamente la demanda de dichos recursos computacionales. Existiendo de todos modos la posibilidad de volver a aplicar la conjetura original, y verificarla sobre algún punto de la búsqueda, si apareciera un posible "pseudo-contraejemplo".

Palabras Claves: Teoría de Números, Procesamiento Paralelo, GPU Computing, CUDA, C++

Contexto

Este artículo es una sinopsis del trabajo de tesis denominado "Uso de GPU Computing en Matemática Experimental: Búsqueda de evidencias factuales que justifiquen la postulación de una nueva conjetura sobre distribución de números coprimos". Defendida en la UTFSM para optar al grado de Magíster en Ciencias de la Ingeniería Informática.

Introducción

La Matemática está encarnada, como soporte oculto, en todas las otras ciencias y actividades del hombre. Pero particularmente su aplicación a la informática es, en los tiempos actuales, responsable de muchos de los avances técnicos que deslumbran al mundo. Por ejemplo, en áreas tales como la de las Tecnologías de la Información y la Comunicación (TIC's), la Criptografía, la Compresión de Datos y la Teoría Computacional de Números, resulta fundamental la aplicación de conceptos y algoritmos de la Teoría de Números, importante rama de la Matemática.

En contrapartida las tecnologías de computación avanzada y especialmente la computación de alta performance (HPC), contribuyen a la investigación matemática convirtiéndose en una herramienta básica de lo que actualmente se denomina Matemática Experimental.

Computación Avanzada aplicada a la Investigación Matemática

La utilización de tecnología de computación avanzada aplicada a la investigación matemática es a menudo llamada Matemática Experimental. Un ejemplo reciente en el campo de la teoría probabilística de números lo constituyen los test de primalidad probabilística que permiten juzgar, con alto grado de certeza, si un número es primo o no, probabilidad que aumenta con el tiempo y esfuerzo de computación que se esté dispuesto a invertir.

Las computadoras facilitan la sinergia vital entre teoría, experimentación y aplicación, evidenciada en el amplio impacto de la computación moderna en ANTC. En contrapartida, la investigación en matemáticas empuja al desarrollo tecnológico; los avances en Matemáticas producen mejores algoritmos, computadoras más rápidas y una comunicación más fiable. La ciencia hoy es digital y computacional y se debe en buena medida a las Matemáticas Discretas. La influencia del Algebra, la Teoría de Números, la Combinatoria, etc., se manifiesta en áreas de Computación tales como: Compresión de datos (Algebra, T. de Números, T. de la Información, T. de Wavelets). Robótica (Algebra, Combinatoria, Análisis Numérico, Lógica Difusa). Criptografía y Seguridad (Teoría de Números, Teoría de la Información). Optimización Combinatoria y técnicas de búsqueda (Algebra, Combinatoria, Teoría de Grafos).

Búsqueda de contraejemplos con HPC

La validez experimental de la búsqueda de contraejemplos, a fin de fortalecer la credibilidad de una conjetura, aumenta con el volumen de datos explorado. No obstante factores como la complejidad algorítmica, la duración del tiempo de computación y las limitaciones de memoria, hacen que una computación factible en lo teórico no siempre resulte viable en la práctica. Por este motivo los algoritmos de búsqueda serán implementados usando técnicas y recursos de computación que posibiliten el manejo eficiente de grandes volúmenes de datos. Esto es factible mediante el uso de sistemas de computación de alto rendimiento y de computación paralela.

En este caso se decidió utilizar equipamiento que puede calificarse como "Home Supercomputing", es decir emplear técnicas de HPC sobre computadoras personales. Concretamente GPGPU computing sobre tarjetas gráficas de Nvidia..

GPGPU Computing

Hace unas décadas, cuando aparecen los sistemas multiprocesador y los primeros procesadores vectoriales, el acceso a la computación paralela estaba restringido a unas pocas instituciones y empresas. El desarrollo que ha tenido lugar desde entonces en materia de hardware, y el constante decremento de su costo, han hecho que la potencia de un supercomputador esté hoy disponible a un precio relativamente bajo.

En la actualidad la Computación de Altas Prestaciones se aplica en numerosos campos de la ciencia y la ingeniería para el desarrollo de aplicaciones y el estudio de problemas que requieren, bien por su complejidad o bien por la necesidad de abordarlos en tiempo real, una gran capacidad de cómputo. El uso cada vez más extendido de procesadores gráficos (GPUs) en aplicaciones de propósito general (GPGPUs), representan una oportunidad y un desafío a la vez. La potencia de cálculo de estas nuevas arquitecturas permite abordar problemas complejos en computadores personales, pero a la vez supone para los desarrolladores un cambio drástico en el modo de concebir el software. El programador es quien deberá esforzarse para que su aplicación se ejecute eficientemente y haga uso de todos los recursos disponibles

Computación sobre GPU

Arquitectónicamente, una GPU podría ser incluida en el grupo de los procesadores vectoriales. Este tipo de procesadores se caracteriza por ser capaz de ejecutar una misma operación sobre múltiples datos de manera simultánea. La computación sobre GPU se basa en el modelo GPGPU (General Purpose Computing on Graphics Processing Units) que consiste en el uso combinado de CPU y GPU en un sistema de co-procesamiento heterogéneo. En este sistema las partes de la aplicación que tienen mayor carga computacional aprovechan el alto grado de paralelismo de la GPU (paralelismo de datos), mientras que el resto del código se ejecuta en la CPU. Esta combinación de CPU y GPU, en una plataforma multinúcleo heterogénea, da soporte a la computación de alto desempeño (High performance Computing, o HPC en inglés) sobre computadoras de costo relativamente bajo.

GPUs de las tarjetas gráficas NVIDIA

Existen diferentes alternativas para procesamiento sobre GPUs. Actualmente la opción más utilizada es la que brindan las tarjetas gráficas Nvidia, para las cuales se dispone del conjunto de herramientas de desarrollo CUDA6 creado por nVidia Corporation. Cuda permite aprovechar el gran poder computacional de estas tarjetas gráficas mediante la paralelización de algoritmos secuenciales, tarea que demanda un esfuerzo moderado si se está familiarizado con el lenguaje C. Dicho poder computacional ha sido utilizado para resolver problemas en diversas áreas de la ciencia, como: criptografía, procesamiento de imágenes, predicción climática, búsqueda en bases de datos, dinámica molecular, etc. No obstante, no todos los problemas pueden ser resueltos utilizando GPU Computing, los más adecuados son aquellos que pueden abordarse mediante la aplicación del paradigma paralelo de datos. Los algoritmos con un alto grado de paralelismo, con estructuras de datos poco complejas, y con un alto costo aritmético, son los que mayor beneficio obtienen de las GPUs, valiéndose de la multiplicidad de núcleos que estas poseen.

Modelo de programación CUDA

El modelo de programación asociado a la arquitectura CUDA es SIMD (Single Instruction Multiple Data) y el modelo de ejecución de programas es SIMT (Single Instruction, Multiple Threads). En CUDA se diferencian dos ámbitos denominados "Host" y "Device". Host es el procesador o CPU al que está conectada la tarjeta gráfica, y será quien la controle, y Device es la GPU en sí. Un programa Cuda tiene una parte secuencial (Host code) que se ejecuta en la CPU, y código paralelo, especificado mediante Kernels, que es ejecutado sobre la GPU.

CUDA C es una extensión del lenguaje C/C++, que permite gestionar el trabajo paralelo a través de threads.

Algoritmos para la Búsqueda de Contraejemplos

La conjetura a ser explorada postula que:

"Entre dos pares de múltiplos consecutivos de todo número natural k>1 siempre hay al menos un número que es coprimo de todos los números naturales ≤ k".

Antes de diseñar los algoritmos para la búsqueda de contraejemplos, se analizaron los algoritmos de cribado más conocidos como son el de la criba de Eratóstenes y el de la de criba de Atkin. Se estudiaron algunas ideas y se adaptaron para la búsqueda de coprimos y luego para su implementación en CUDA.

El trabajo de computación que demanda la obtención de los primos ≤n (aplicando la criba de Eratóstenes) presenta complejidad lineal con respecto a n; esto es O(n) operaciones en términos de complejidad asintótica. Adaptar uno de los algoritmos de cribado a la búsqueda de coprimos no supone reducir su complejidad; por lo tanto, para poder mantener los tiempos de ejecución en valores razonables y aumentar los valores de n a los que se puede dar un tratamiento eficiente, se diseñaron algoritmos optimizados y luego paralelizados para correr sobre GPUs. Esto teniendo en cuenta que el principal inconveniente de los algoritmos de cribado es que demandan grandes cantidades de memoria.

Primer Algoritmo

Inicialmente se pensó en usar los *n* números a ser explorados como direcciones para apuntar a la memoria a nivel de bit, de modo que controlando el estado del bit correspondiente a cada número natural ≤n se puede indicar, tal como se hace en otros algoritmos de cribado, si dicho número ha sido o no marcado como múltiplo de algún primo interviniente en la criba. Esta alternativa favorece, en cierta medida, el aprovechamiento de la memoria, pero no resultó la más adecuada para plantear la búsqueda de contraejemplos. En lugar de asignar un bit o "flag" a cada número, se optó por el usar un acumulador de $\lceil \log2\ x \rceil + 1$ bits por cada número a ser explorado. Mediante este acumulador cada número apuntará, durante el proceso de cribado, al número aún no marcado más cercano, por lo que será actualizado cuando dicho número no marcado resulte "cribado" o marcado al avanzar el proceso.

Segundo Algoritmo

El primer algoritmo descripto fue diseñado respetando estrictamente lo que establece la conjetura. Sin embargo, para poder ampliar el espacio de búsqueda de contra-ejemplos es necesario reducir la cantidad de recursos computacionales (memoria sobre todo) que dicho algoritmo demanda. Esto se debe a que este primer algoritmo debe almacenar por cada nº a ser explorado un acumulador de $\lceil \log2\ x \rceil + 1$ bits (siendo x el mayor número natural entre cuyos múltiplos se buscarán contraejemplos). Este segundo algoritmo fue concebido

aplicando más restricciones que las establecidas por la conjetura original, de este modo se puede reducir significativamente la demanda de recursos computacionales. Sin embargo existe la posibilidad de tener que volver en algún punto de la búsqueda a la conjetura original para verificarla si apareciera un posible "pseudo-contraejemplo".

Diseño del Algoritmo

El 2do algoritmo está inspirado en el uso de contadores en anillo similares, desde el punto de vista funcional, a los usados frecuentemente en electrónica digital. A cada primo impar < se le hace corresponder un contador en anillo regresivo C(*i*) y un contador progresivo G(*i*), ambos contadores serán activados en el momento en que el algoritmo explore el número . La activación del contador C(*i*) consiste en "setearlo" con el valor y habilitarlo para contar regresivamente. La activación del contador G(*i*) consiste en "setearlo" con el valor que tenga el contador G(*i*-1) y habilitarlo para contar progresivamente. Todos los contadores activos, sean progresivos o regresivos, serán actualizados cada vez que el algoritmo explore un nuevo número.

Uso dado a los contadores en anillo regresivos

Al explorar un número cualquiera *b* del espacio de búsqueda el algoritmo puede determinar, analizando el estado de los contadores activos C(*i*), si el número explorado es primo o no y cuáles son sus divisores si no lo es. Serán divisores del número *b* todos los primos cuyos correspondientes contadores en anillo hayan alcanzado el valor cero. Si ninguno de ellos lo hizo, significa que el número *b* es primo. Ahora bien, para establecer la coprimalidad de *b* es necesario determinar cuál es el menor primo divisor de *b*. Para ello, basta con hallar el contador en anillo de menor índice que haya llegado a cero. Hecho esto, el algoritmo establece que *b* es un **Coprimo**$^{pm-1}$, lo que es equivalente a decir que *b* es coprimo de -1.

Uso dado a los contadores en anillo progresivos

Cada contador progresivo G(*i*) registra, a medida que se avanza en la exploración del espacio de búsqueda, la evolución de la brecha o gap entre los coprimos de . Así, analizando el estado de los contadores activos G(*i*) el algoritmo podrá verificar con cada nueva exploración, que se esté cumpliendo la restricción impuesta para dicha brecha.

Capacidades del Algoritmo

- El algoritmo puede verificar la veracidad de la conjetura sobre un espacio de búsqueda ampliado en comparación con el primer algoritmo.
- Generar la tabla de primos, extendiéndola a todo el espacio de búsqueda.
- Obtener los factores primos de los números explorados que no resulten primos.
- Obtener el gap entre cada **Coprimo**^{pi} para valores de < ; siendo *n* el tamaño del espacio de búsqueda.

Implementación de los Algoritmos

Los programas que implementan la algoritmia descripta

han sido desarrollados inicialmente en su versión secuencial. Esto no sólo facilita la comprobación y refinamiento de los algoritmos, sino que permite también identificar, sobre el programa secuencial ya corregido, las partes o tareas más susceptibles de ser paralelizadas. Posteriormente es útil para verificar el correcto funcionamiento y la eficiencia de la correspondiente versión paralela.

Implementación secuencial del 2do algoritmo

Considerando que la API C CUDA fue desarrollada por nVidia como una extensión del lenguaje ANSI C, que permite la programación paralela en entornos CPU/GPU, el lenguaje C es también la opción obvia para la programación secuencial de los algoritmos.

Implementación paralela del 2do algoritmo en CPU/GPU

El desarrollo de software paralelo tiene el insoslayable propósito de mejorar el tiempo de resolución de un problema. Lograr este objetivo, particularmente en GP-GPU Computing, exige esfuerzo y experiencia, sobre todo porque el diseño de programas paralelos para GPU es, en gran medida, un trabajo manual. No existe una única solución paralela, y no siempre un desarrollo sobre GPU es sinónimo de buena performance. Desconocer las características de la arquitectura y algunas de sus limitaciones puede conducir a aplicaciones con bajo rendimiento.

Entorno de desarrollo

El software desarrollado para completar este trabajo ha sido implementado en un equipo ASUS K53S. El entorno de hardware y

software ha sido configurado para admitir aplicaciones de 64 bit con acceso a grandes espacios de memoria. Buscando evitar sobre todo las restricciones de memoria virtual asociadas con las versiones de 32bit de Windows.

La computadora ASUS K53S posee una tarjeta gráfica NVIDIA GeForce 520 Mx, similar a la GeForce GT 540M pero producida con transistores optimizados. Se trata de una GPU de generación Fermi, de gama básica. Cuenta con 48 ALUs, 8 unidades de textura y una interfaz de memoria de 64 bits para VRAM DDR3, y brinda soporte para CUDA, OpenCL, y DirectCompute 2.1.

Software

Se instaló la versión CUDA_5.5.20 para Windows 7 que incluye NVIDIA Nsight Visual Studio Edition_Win64_3.2.2 y la versión 332.21 de los Drivers Nvidia. La instalación incluye además el kit de herramientas CUDA y el desarrollador SDK de CUDA, con ejemplos y documentación útil para una amplia gama de aplicaciones. El kit de herramientas CUDA es un entorno de desarrollo que incluye:

- Compilador NVCC (**NV**idia **C**uda **C**ompiler)
- Bibliotecas CUBLAS, CURAND y CUFFT.
- Herramientas para optimización y debugging.
- Guías de programación, manuales de usuario, etc.

Comparativa de desempeño: Paralelo vs Secuencial

La medida utilizada para evaluar el rendimiento del algoritmo será el speedup calculado a partir de los tiempos de ejecución, en segundos. Se tomará como línea base para este cálculo, el tiempo de ejecución de la versión serie de la aplicación. Así, es:

Speedup

El tiempo de ejecución de un algoritmo secuencial es evaluado como una función del tamaño del problema (el comportamiento asintótico del tiempo de ejecución será idéntico en cualquier plataforma secuencial). En cambio, el tiempo de ejecución de un programa paralelo depende del tamaño del problema y del número de procesadores utilizados. Es por ello que los algoritmos paralelos deben ser evaluados y analizados teniendo en cuenta también la plataforma.

La siguiente tabla muestra los tiempos de ejecución serie y paralela del 2do algoritmo.

Tamaño de la búsqueda	Tiempo de ejec. secuencial (seg)	Tiempo de ejec. paralela (seg)	Speedup
2^{10}=1024	0.016	- - - -	- - - -
2^{12}=4096	0.031	0.015	2.066
2^{14}= 16.384	0.109	0.031	3.516
2^{16}= 65.536	0.452	0.078	5.794
2^{18}= 262.144	1.872	0.327	5.724
2^{20}= 1.048.576	7.410	1.326	5.588
2^{22}= 4.194.304	31.185	5.506	5.663
2^{24}= 16.777.216	134.818	22.526	5.985
2^{26}= 67.108.864	1102.502 0hs 18min 22seg	121.619 0hs 02min 01seg	9.065
2^{28}= 268.435.456	5060.158 1hs 24min 20seg	755.484 0hs 12min 35seg	6.697
2^{30}= 1.073.741.824	33498.039 9hs 18min 18seg	5681.336 1hs 34min 41seg	5.896

Cumplimiento de los Objetivos Trazados

Los objetivos de investigación y ejecución propuestos y logrados en este trabajo son:

Objetivos Generales

Acopiar evidencia computacional acerca de la validez de una nueva conjetura. Para ello fue necesario:

- Desarrollar algoritmos que, mediante la búsqueda de contraejemplos, ponen a prueba dicha conjetura.

- Implementar dichos algoritmos usando técnicas y recursos de computación que posibilitan el manejo eficiente de grandes volúmenes de datos.

Objetivos Específicos

- Diseñar algoritmos aptos para buscar coprimos de todos los números naturales menores o iguales a un número k. Se desarrollaron dos algoritmos diferentes para alcanzar este objetivo.
- Dar a dichos algoritmos capacidades adicionales, como la de generar listados de primos, hallar factores primos, etc.
- Implementar los algoritmos de búsqueda empleando hardware gráfico y software de programación CUDA.
- Analizar y mejorar, con la ayuda de herramientas especializadas, el desempeño de los algoritmos.

Conclusiones, Aportes y Trabajo Futuro

Este trabajo muestra, sin haberlo previsto, como la computación de alta performance (HPC), puede contribuir a la investigación matemática. Una evidencia de la simbiosis y retroalimentación que puede producirse entre las Ciencias de la Computación y la Matemática queda de manifiesto con la ejecución del segundo algoritmo de búsqueda de contraejemplos. Con este algoritmo se aplicaron a la búsqueda más restricciones que las establecidas por la conjetura original, asumiendo el riesgo de toparse con "pseudo-contraejemplos", pero, en la práctica esto no sucedió, surgiendo así, mediante la exploración computacional, la oportunidad de plantear otra conjetura, aún más estricta que la defendida en esta tesis.

En lo que respecta a GPU Computing, es difícil elaborar conclusiones novedosas sobre un tema tan documentado y tantas veces abordado. No obstante, enunciar algunas opiniones sobre el tema, aun siendo estas subjetivas, puede ser de utilidad para otros que quieran "vivir la experiencia CUDA".

- GPU Computing no es aplicable a cualquier problema.
- CUDA es relativamente fácil de aprender.
- Generar aplicaciones eficientes bajo la arquitectura CUDA no es tarea sencilla.
- Hay demasiados aspectos a considerar para paralelizar adecuadamente sobre GPU.
- Aún con un equipo de bajo costo se puede experimentar y trabajar con CUDA.
- Particularmente, resulta asombroso poder explorar, en un equipo dotado con una pequeña GPU, más de 67 millones de números, resolver la primalidad de cada uno de ellos, buscar los factores primos de cada número compuesto, medir la brecha entre coprimos y

varias cosas más... en tan sólo dos minutos.

Las conclusiones y aportes vinculados a los objetivos específicos de este trabajo son:

- Existe, y se obtuvo, suficiente evidencia factual como para plantear otra conjetura, aún más restrictiva que la conjetura propuesta en este trabajo.
- La algoritmia desarrollada puede ser útil a quienes deseen aplicarla sobre espacios de búsqueda y equipos más grandes, o aplicarla a otros fines relacionados con la teoría de números.
- Resta, como trabajo a futuro, mostrar de qué manera la conjetura planteada se vincula con otras importantes conjeturas de la Teoría de Números.

Formación de Recursos Humanos

En el marco de las actividades de Posgrado de la UNSJ, se ha planificado el dictado de un curso sobre "GPU-Computing y Programación bajo CUDA"

Referencias

[1] CUDA C Programming Guide. Versión 5.0 y 5.5. NVIDIA 2012, 2013.

[2] CUDA C Best Practices Guide. Versión 4.1 y 5.5. NVIDIA 2012, 2013.

[3] CUDA TOOLKIT Reference Manual. Versión 5.5. NVIDIA 2013.

[4] CUDA SAMPLES Reference Manual. Versión 5.5. NVIDIA 2013.

[5] CUBLAS Library. Versión 5.5. User Guide. NVIDIA 2013.

[6] CUDA Getting Started Guide for Microsoft Windows. Versión 5.5. NVIDIA 2013.

[7] CUDA by Example, Jason Sanders & Edward Kandrot. Addison Wesley 2010.

[8] CUDA Developer Guide for NVIDIA Optimus Platforms. Version 1.0. 2010

[9] CUDA GDB Debugger. Versión 5.5. User Manual. NVIDIA 2013.

[10] CUDA Compiler Driver NVCC. Versión 5.5. Reference Guide. NVIDIA 2013.

[11] CUDA-MEMCHECK. Versión 5.5. User Manual. NVIDIA 2013.

[12] Compute Command Line Profiler. Versión 03. User Guide. NVIDIA 2011.

[13] PROFILER User's Guide. Versión 5.5. NVIDIA 2013.

Referencias Web

http://developer.nvidia.com/cuda.
http://developer.nvidia.com/cuda-toolkit
http://developer.nvidia.com/user
http://developer.download.nvidia.com/compute/cuda
http://developer.nvidia.com/nvidia-gpu-computing documentation