|  |  |  |
| --- | --- | --- |
| 9 mar.  https://mail.google.com/mail/u/0/images/cleardot.gif |  | **https://mail.google.com/mail/u/0/images/cleardot.gif**  **https://mail.google.com/mail/u/0/images/cleardot.gif** |

Tal y como hablamos ayer, te paso información sobre el algoritmo a implementar en tu TFG.  
El algoritmo se llama Differential Box Counting y la versión más básica es la que se publicó en Sarkar 1994 (llevas adjunto el artículo). El algoritmo está descrito en el apartado III de ese artículo. Mirátelo e intenta entender cómo funciona el algoritmo. Un día de la semana que viene te pasas por tutorías y lo repasamos entre los dos a ver si lo tienes claro antes de empezar a implementarlo.  
También te paso el TFG de Mario para que te sirva de base para tu memoria del TFG.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | |  | | 27 mar.  https://mail.google.com/mail/u/0/images/cleardot.gif |  | **https://mail.google.com/mail/u/0/images/cleardot.gif**  **https://mail.google.com/mail/u/0/images/cleardot.gif** |

¿Cómo vas con la instalación y prueba de OpenCL?  
Yo he probado algunas demos OpenCL tanto en una GPU NVIDIA como en una CPU Intel en un ordenador del 2015 y me han funcionado sin necesidad de instalar drivers.  
  
Esta página está bastante bien, lo explica todo paso a paso, hasta la creación de un programa de prueba en Visual Studio: <https://streamcomputing.eu/blog/2015-03-16/how-to-install-opencl-on-windows/>  
  
Voy a probar la instalación en un ordenador más antiguo y ya te cuento.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | |  | | Archivos adjuntos30 mar.  https://mail.google.com/mail/u/0/images/cleardot.gif |  | **https://mail.google.com/mail/u/0/images/cleardot.gif**  **https://mail.google.com/mail/u/0/images/cleardot.gif** |

Respecto al código:  
- Organiza el código en módulos (.c y .h diferentes), cuanto más código tengas más difícil te será luego organizarlo bien. Los módulos que yo veo por el momento serían: 1) tratamiento de imágenes; 2) Algoritmo DBC; 3) main.c  
- Comenta los módulos, estructuras de datos, funciones y variables principales utilizando doxygen. Si lo haces poco a poco no te cuesta nada y luego tendrás una magnífica documentación automática para adjuntar a tu TFG.  
- Respecto al cálculo de la DF en base al box-counting, como hablamos, se hace mediante una regresión lineal de la nube de puntos obtenidas pintando los diferentes pares log(1/r) - log(Nr). Aquí te paso un enlace con un programa ya hecho en C que te puede servir de guía (la DF es la pendiente de la recta, a en el código de ejemplo): <http://ismaxsotoprs.blogspot.com.es/2011/11/regresion-lineal-en-lenguaje-c.html>. La manera de seleccionar los puntos con los cuales hacer la regresión lineal se hace pintándolos y cogiendo aquellos que ajustan mejor la recta (los que tienen mayor correlación, también se calcula en el código de ejemplo). Tú no tienes que pintar los puntos en tu programa, pero sí deberías pintarlos por tu cuenta para poder verlos y seleccionar el mejor rango, por ejemplo los puedes pintar fácil utilizando un gráfico de excel.  
  
Cuando tengas implementado el cálculo de la DF, tienes que probar el algoritmo implementado y validarlo con imágenes de las que conozcamos la dimensión fractal.  
En este enlace tienes una base de datos de imágenes que se utilizan habitualmente para probar algoritmos de análisis de imágenes:  
<http://www.ux.uis.no/~tranden/brodatz.html>  
Estas imágenes corresponden a esta publicación, para que la referencies en la memoria del TFG:  
P. Brodatz, Texture: A Photographic Album for Artists and Designers, New York, 1966.  
  
En el artículo que te adjunto (también tienes que referenciarlo en la memoria de tu TFG), se utilizan varias de estas imágenes para probar un algoritmo de Differential Box Counting. Lo que tienes que hacer es pasarle tu algoritmo a las mismas imágenes que se utilizan en ese artículo (las que se indican en la figura 7) y comprobar que salen los mismos resultados que los que se muestran en el artículo en la tabla 2 en la primera columna, que corresponde al algoritmo original DBC que tú has implementado.  
  
Ya me vas contando.

Archivo enviado : 30 marzo- 2014\_Liu\_Improved\_Differential\_Box\_counting

|  |  |  |
| --- | --- | --- |
| 10 may.  https://mail.google.com/mail/u/0/images/cleardot.gif |  | **https://mail.google.com/mail/u/0/images/cleardot.gif**  **https://mail.google.com/mail/u/0/images/cleardot.gif** |

Te adjunto los dos artículos con propuestas paralelas del DBC.

Documentos enviados:

1998\_Biswas\_Parallel\_Fractal\_Dimension

2012\_Tzeng\_DifferentialBoxCounting\_Multinucleo

|  |  |  |
| --- | --- | --- |
| 25 may.  https://mail.google.com/mail/u/0/images/cleardot.gif |  | **https://mail.google.com/mail/u/0/images/cleardot.gif**  **https://mail.google.com/mail/u/0/images/cleardot.gif** |

Te adjunto el artículo del que hemos hablado. Lo importante es lo que se cuenta en el apartado 3.2., sobre cómo implementar en CPU, dividiendo el procesamiento en base al número de núcleos que se tengan, en tu caso 4.  
Recuerda ir guardando versiones claramente identificadas de las diferentes optimizaciones que vayas aplicando y que supongan una mejora del tiempo.  
Saludos,

: ARTÍCULO\_PUBLICADO\_REVISTA.pdf

|  |  |  |
| --- | --- | --- |
| 2 jun.  https://mail.google.com/mail/u/0/images/cleardot.gif |  | **https://mail.google.com/mail/u/0/images/cleardot.gif**  **https://mail.google.com/mail/u/0/images/cleardot.gif** |

Bueno, parece que el algoritmo ya lo tienes fino, aunque no salgan los  
mismos valores de DBC que en el artículo. En cualquier caso, lo importante  
es acelerar con OpenCL el algoritmo.  
  
Respecto a esto, algo que no te he comentado y es importante, lo que hay  
que acelerar es sólo el algoritmo de DBC, es decir, el que devuelve el  
conteo de Nr para cada tamaño r. El cálculo del valor final de DBC no hay  
que incluirlo en la versión OpenCL ni en las comparativas de tiempos.  
  
Otra cosa, si algún día quieres enseñarme algo en el despacho y no ten  
encajan bien las horas de tutorías que he puesto, me mandas un email y  
quedamos a otra hora.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | |  | | Archivos adjuntos7 jun.  https://mail.google.com/mail/u/0/images/cleardot.gif |  | **https://mail.google.com/mail/u/0/images/cleardot.gif**  **https://mail.google.com/mail/u/0/images/cleardot.gif** |
| |  | | --- | | para mí  https://mail.google.com/mail/u/0/images/cleardot.gif | | | |

Hola Emilio,  
Te adjunto las transparencias de CUDA y el código de los problemas que utiliza Antonio Rueda en su asignatura de programación hardware.  
Te adjunto también el artículo con las optimizaciones CUDA, para que una vez que tengas funcionando la versión GPU la intentes mejorar algo.  
La web que hemos estado viendo con la correspondencia CUDA-OpenCL es esta: <http://developer.amd.com/tools-and-sdks/opencl-zone/opencl-resources/programming-in-opencl/porting-cuda-applications-to-opencl/>  
  
Lo que hemos hablado:

1. Ve añadiendo comentarios doxygen a medida que vas programando
2. Implementa con OpenCL sólo el conteo de Nr para cada r, pero no el cálculo de D
3. El algoritmo no debe hacer estimaciones del conteo. Prueba con los tamaños dividiendo por 2 el tamaño máximo y con todos los divisores, y te quedas con el algoritmo que devuelva los resultados más parecidos a los de Sharkar.
4. Los resultados para los algoritmos en C, OpenCL-CPU y OpenCL-GPU que implementes tienen que dar exactamente el mismo resultado
5. Algoritmo OpenCL-CPU: evitar que las hebras accedan de partida a las mismas posiciones de memoria. Por ejemplo, que cada hebra inicie en un cuadrante diferente y que vayan rotando.
6. Tener mucho cuidado dónde se ponen las instrucciones para la toma de tiempos, no debe incluirse la salida a pantalla ni el acceso a disco.

Elementos enviados: 3 archivos de cuda

YO:

|  |  |  |
| --- | --- | --- |
| 30 jun. (hace 6 días)  https://mail.google.com/mail/u/0/images/cleardot.gif |  | **https://mail.google.com/mail/u/0/images/cleardot.gif**  **https://mail.google.com/mail/u/0/images/cleardot.gif** |
| |  | | --- | | para Juan  https://mail.google.com/mail/u/0/images/cleardot.gif | | | |

Buenas Juan,

Correcciones:

 -OpenCL sólo realiza el conteo de Nr para cada r.

 -He conseguido que todos los algoritmos den el mismo resultado utilizando un tamaño de grid de potencias de 2.

Implementaciones:

 -Implementación en C. (0.16 segundos)

 -Implementación en CPU, versiones:

   +primera versión: cada hebra calcula el conteo N para un r. (0.003 segundos)

   +segunda versión: se subdivide la imagen en 4 partes u cada núcleo calcula el valor de N para esa subimagen para todos los valores de r. (0.003 segundos)

-Implementacion en GPU, optimizaciones:

   +1.0 Primera implementacion (0.084 segundos)

   +2.0 Utilizando el máximo de thread con 2 dimensiones. (0.01 segundos)

   +2.0.1 Utilizando 2 dimensiones y cargando la imagen en memoria compartida (0.011 segundos)

   +2.1 Utilizando 3 dimensiones (0.007 segundos)

   +2.2 Utilizando 3 dimensiones y guardando la salida en memoria compartida (0.008 segundos)

|  |  |  |
| --- | --- | --- |
| 1 jul. (hace 5 días)  https://mail.google.com/mail/u/0/images/cleardot.gif |  | **https://mail.google.com/mail/u/0/images/cleardot.gif**  **https://mail.google.com/mail/u/0/images/cleardot.gif** |

Una cosa que se me olvidó comentarte. Para la toma de tiempos en GPU, toma  
por un lado el tiempo que se tarda en cargar los datos en la GPU y en  
devolver los resultados a la CPU y por otro lado el tiempo de ejecución  
sólo del kernel. El tiempo real es el total de la suma de esos 3, que  
supongo que es el que tú me has pasado, pero es bueno tenerlos por  
separado, para que al menos se vea que el kernel GPU hace el cálculo más  
rápido que la versión CPU y CPU multi-hebra.