

Diplomado BigData

# Unidad 7: Computación de alto rendimiento para Big Data.

## Informe de laboratorio Nº2

Junio/2018

Alumnos:

1. Humberto Andrés Alvarez Vilches
2. Francisco Rodrigo Marín Küllmer
3. Francisco

Tabla de contenido

[Unidad 7: Computación de alto rendimiento para Big Data. 1](#_Toc517972820)

[Informe de laboratorio Nº2 1](#_Toc517972821)

[Ambientación de las pruebas 3](#_Toc517972822)

[Selección de las imágenes 4](#_Toc517972823)

[Resultados experimentales, Imagen GRANDE. 6](#_Toc517972824)

[Resultados experimentales, Imagen MEDIANA. 7](#_Toc517972825)

[Resultados experimentales, Imagen PEQUEÑA. 9](#_Toc517972826)

[Conclusiones experimentos 11](#_Toc517972827)

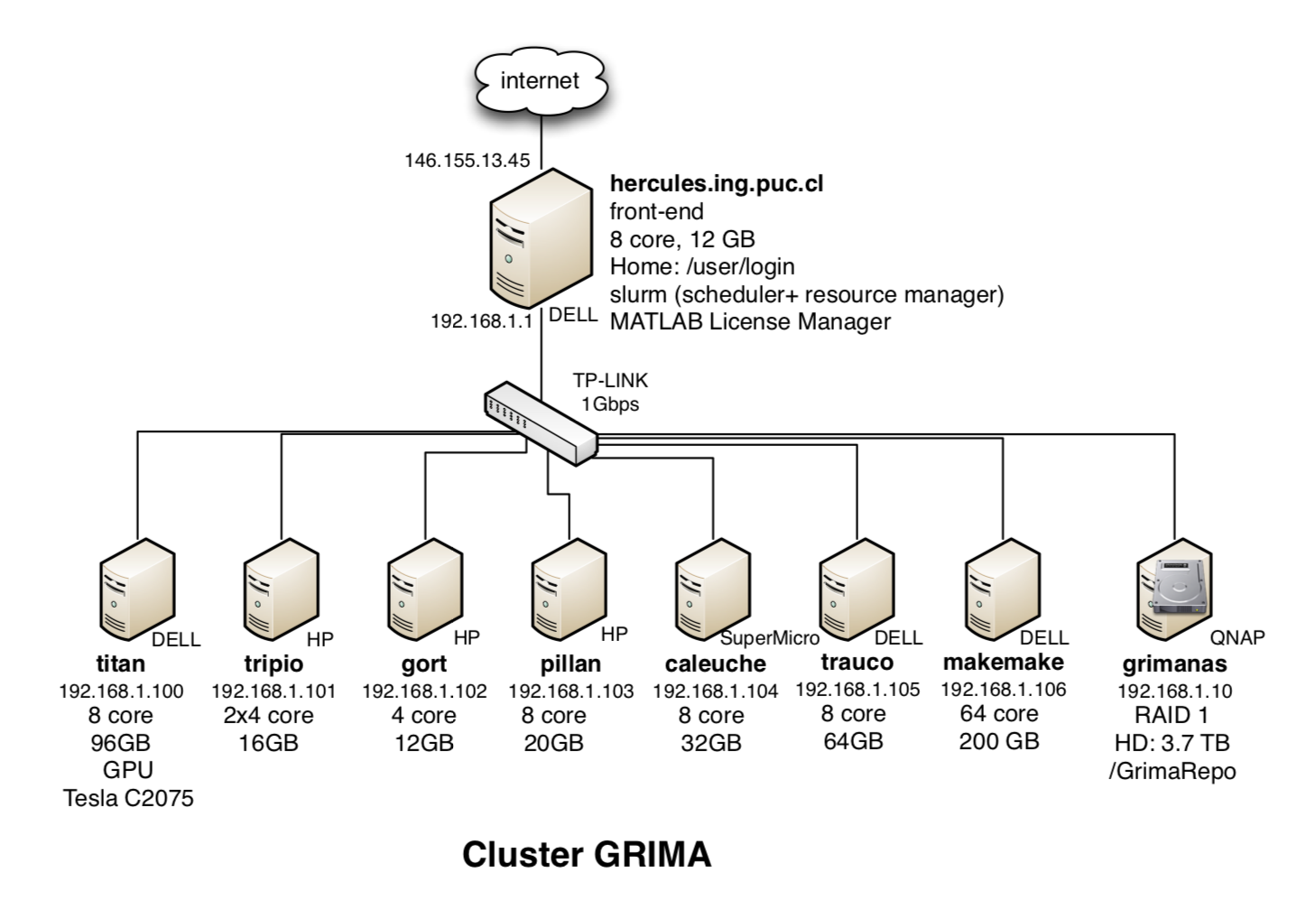
[Preguntas 11](#_Toc517972828)

### Ambientación de las pruebas

La actividad se ejecuta en el Cluster Grima /Nodo Caleuche.

CPU. 8 CORE

RAM. 32 Gb



### Selección de las imágenes

Ubicados en la carpeta /user/cruz/DipBD/2017-2/AC03/images, listamos las imágenes con el comando: Ls -lsh

|  |
| --- |
|  |

Usaremos las siguientes

|  |
| --- |
| Grande 🡺 lake.png – 4,7 MB  Mediana 🡺 Yoshi.png – 444 KB  Pequeña 🡺 world.png – 56 KB |

### Resultados experimentales, Imagen GRANDE.

Se ejecuta el programa ./masker para la imagen GRANDE.

|  |  |
| --- | --- |
| Imagen original:  lake.png – 4,7 MB | Imagen modificada:  Img\_lake.png – 3,7MB |
|  |  |
| Justificación de la elección.  Elegimos la imagen lake.png, considerando que es la de mayor peso en Bytes. (4,7 Mb).  Se eligió considerando el formato PNG . | |

Con esto se obtiene los siguientes tiempos de ejecución:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Iteraciones | | |
| Threads | 10 | 50 | 100 |
| 1 | 9,645843 | 47,827253 | 98,142908 |
| 2 | 5,138992 | 24,608324 | 52,230964 |
| 4 | 2,634882 | 13,282095 | 30,231417 |
| 8 | 2,243051 | 9,5614050 | 27,126446 |
| 16 | 2,013196 | 9,0094940 | 18,849647 |

Rendimiento:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Iteraciones | | | | | |
|  | 10 | | 50 | | 100 | |
| Threads | Speedup | Eficiencia | Speedup | Eficiencia | Speedup | Eficiencia |
| 1 | 1,00000000 | 1,00000000 | 1,00000000 | 1,00000000 | 1,00000000 | 1,00000000 |
| 2 | 1,87699125 | 0,93849562 | 1,94353963 | 0,97176982 | 1,87901774 | 0,93950887 |
| 4 | 3,66082542 | 0,91520635 | 3,60088171 | 0,90022043 | 3,24638795 | 0,81159699 |
| 8 | 4,30032264 | 0,53754033 | 5,00211559 | 0,62526445 | 3,61797885 | 0,45224736 |
| 16 | 4,79130845 | 0,29945678 | 5,30853930 | 0,33178371 | 5,20661782 | 0,32541361 |

|  |  |
| --- | --- |
| Resultado gráfico | Conclusión del experimento |
|  | Observamos que durante la ejecución realizada por 4 a 8 core, el speedup se reduce sustancialmente. Posterior a ellos, la curva es decreciente.  La aplicación es que el cluster responde con un buen Speedup mientras hayan CORE disponibles para paralelizar. |
|  | La eficiencia posee buen indicador usando hasta 4 CORE. La eficiencia se reduce de 1 a 0.8, lo cual es un muy buen indicador.  Mientras que la eficiencia cae pasando el umbral de 4 a 8 CORE.  La diferencia entre la iteración=100 y iteración=10, creemos que es debido a un mayor porcentaje de codigo secuencial (no paralelo) que evita que a mayor cantidad de CORE, no sea capaz de procesar paralelamente. |

### Resultados experimentales, Imagen MEDIANA.

Se ejecuta el programa ./masker para la imagen MEDIANA.

|  |  |
| --- | --- |
| Imagen original:  Yoshi.png – 444 KB | Imagen modificada:  Img\_Yoshi.png – 532 KB |
|  |  |
| Justificación de la elección.  Elegimos la imagen lake.png, considerando que es la de mayor peso en Bytes. (444 Kb).  Se consideró la elección de imagen en formato PNG. | |

Con esto se obtiene los siguientes tiempos de ejecución:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Iteraciones | | |
| Threads | 10 | 50 | 100 |
| 1 | 3,342240 | 16,716926 | 32,623741 |
| 2 | 1,685830 | 9,701681 | 17,187236 |
| 4 | 0,908381 | 4,977571 | 9,3971940 |
| 8 | 0,822689 | 2,742120 | 8,1800870 |
| 16 | 0,683779 | 3,099466 | 7,3915100 |

Rendimiento:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Iteraciones | | | | | |
|  | 10 | | 50 | | 100 | |
| Threads | Speedup | Eficiencia | Speedup | Eficiencia | Speedup | Eficiencia |
| 1 | 1,00000000 | 1,00000000 | 1,00000000 | 1,00000000 | 1,00000000 | 1,00000000 |
| 2 | 1,98254866 | 0,99127433 | 1,72309582 | 0,86154791 | 1,89813772 | 0,94906886 |
| 4 | 3,67933719 | 0,91983430 | 3,35845054 | 0,83961263 | 3,47164707 | 0,86791177 |
| 8 | 4,06258015 | 0,50782252 | 6,09635100 | 0,76204387 | 3,98818998 | 0,49852375 |
| 16 | 4,88789507 | 0,30549344 | 5,39348585 | 0,33709287 | 4,41367745 | 0,27585484 |

|  |  |
| --- | --- |
| Resultado gráfico | Conclusión del experimento |
|  | Observamos que durante la ejecución con 4 a 8 core, el speedup se reduce sustancialmente. Posterior a ellos, la curva es decreciente.  La aplicación es que el cluster responde con un buen Speedup mientras hayan CORE disponibles para paralelizar. |
|  | La eficiencia posee buen indicador usando hasta 4 CORE. La eficiencia se reduce de 1 a 0.8, lo cual es un muy buen indicador.  Mientras que la eficiencia cae pasando el umbral de 4 a 8 CORE.  La diferencia entre la iteración=100 y iteración=10, creemos que es debido a un mayor porcentaje de codigo secuencial (no paralelo) que evita que a mayor cantidad de CORE, no sea capaz de procesar paralelamente. |

### Resultados experimentales, Imagen PEQUEÑA.

Se ejecuta el programa ./masker para la imagen PEQUEÑA.

|  |  |
| --- | --- |
| Imagen original: world.png – 56 KB | Imagen modificada: Img\_world.png – 990 KB |
|  |  |
| Justificación de la elección.  Elegimos la imagen world.png, considerando que es una de las imágenes con menor peso en Bytes. (56 Kb).  El formato considerado es PNG. | |

Con esto se obtiene los siguientes tiempos de ejecución:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Iteraciones | | |
| Threads | 10 | 50 | 100 |
| 1 | 13,139919 | 65,714986 | 131,531881 |
| 2 | 6,997945 | 36,033334 | 71,035224 |
| 4 | 4,084180 | 23,400934 | 41,650713 |
| 8 | 2,657337 | 12,784658 | 26,266155 |
| 16 | 2,613382 | 12,061834 | 23,731415 |

Rendimiento:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Iteraciones | | | | | |
|  | 10 | | 50 | | 100 | |
| Threads | Speedup | Eficiencia | Speedup | Eficiencia | Speedup | Eficiencia |
| 1 | 1,00000000 | 1,00000000 | 1,00000000 | 1,00000000 | 1,00000000 | 1,00000000 |
| 2 | 1,87768252 | 0,93884126 | 1,82372761 | 0,91186380 | 1,85164308 | 0,92582154 |
| 4 | 3,21727226 | 0,80431806 | 2,80822065 | 0,70205516 | 3,15797429 | 0,78949357 |
| 8 | 4,94476952 | 0,61809619 | 5,14014423 | 0,64251803 | 5,00765647 | 0,62595706 |
| 16 | 5,02793660 | 0,31424604 | 5,44817529 | 0,34051096 | 5,54252163 | 0,34640760 |

|  |  |  |
| --- | --- | --- |
|  | Conclusión del experimento |  |
|  | Observamos que durante la ejecución con 4 a 8 core, el speedup se reduce sustancialmente. Posterior a ellos, la curva es decreciente.  La aplicación es que el cluster responde con un buen Speedup mientras hayan CORE disponibles para paralelizar. |  |
|  | La eficiencia posee buen indicador usando hasta 4 CORE. La eficiencia se reduce de 1 a 0.8, lo cual es un muy buen indicador.  Mientras que la eficiencia cae pasando el umbral de 4 a 8 CORE.  La diferencia entre la iteración=100 y iteración=10, creemos que es debido a un mayor porcentaje de codigo secuencial (no paralelo) que evita que a mayor cantidad de CORE, no sea capaz de procesar paralelamente. |  |

### Conclusiones experimentos

Preguntas  
En esta sección se debe dar una respuesta fundamentada a las siguientes preguntas:

• ¿Hasta cuánto aumenta el speedup en el nodo escogido? ¿Por qué?

* Dsdsd

• ¿Es eficiente la forma de paralelizar del programa?

* Dsdsd

• ¿Cómo se puede relacionar OpenMP al ecosistema Hadoop?

* Dsdsd

• ¿Qué hace la instrucción collapse(2)? ¿Qué sucede si la elimina del código de masker?

* Dsdsd