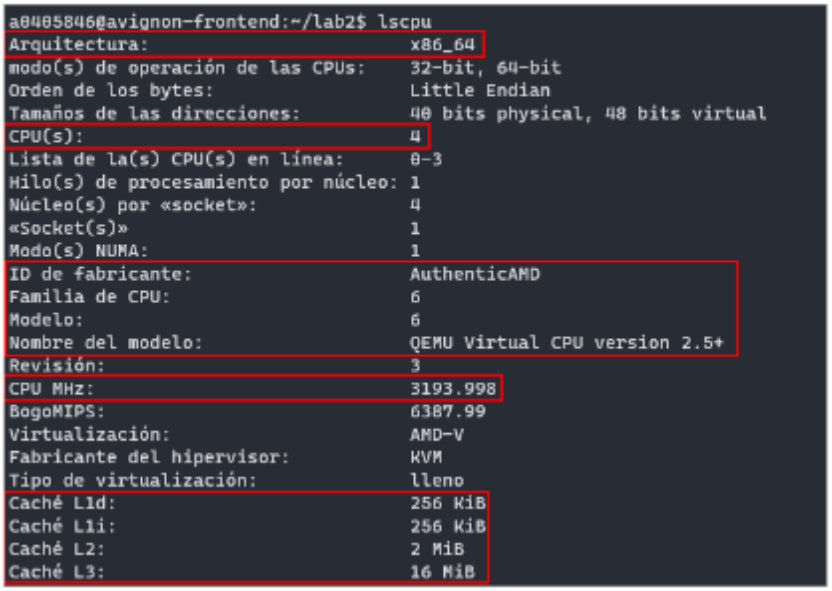
Álvaro Morata (100405846) y Javier Moreno (100428998)

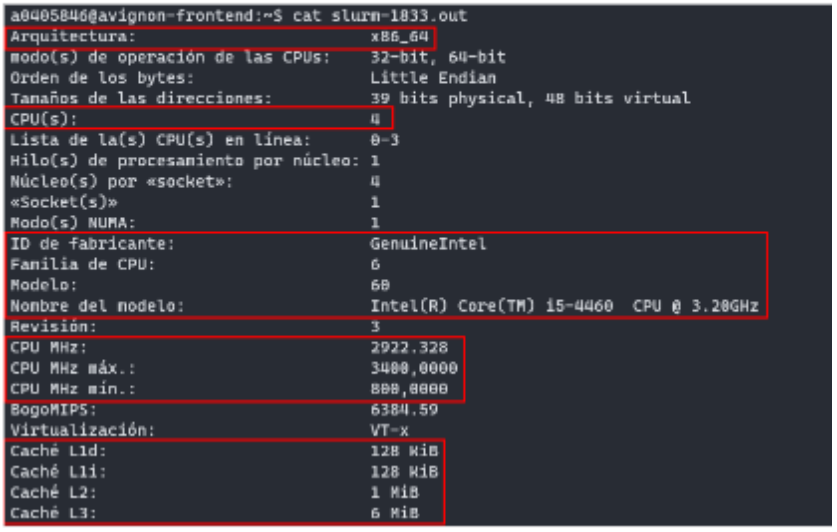
1. Comparación hardware frontend y nodos de cómputo

En las siguientes imágenes podemos ver las diferentes características entre el frontend y el nodo de cómputo. Las más importantes son la de la arquitectura, el número de CPUs, el modelo de las CPU, el rango de las frecuencias del reloj y el tamaño de los distintos niveles de caché.

Frontend lscpu



Nodo de cómputo lscpu



Como podemos observar en ambas imágenes, la arquitectura es la misma tanto en el nodo de cómputo como en el frontend además de tener el mismo número de CPUs. Los primeros cambios que hay son en cuanto al modelo, ya que no solo tienen distintos fabricantes, si no que los modelos también son diferentes.

Los cambios más reseñables están relacionados con las frecuencias y con las memorias caché. En el nodo de cómputo, podemos observar los rangos de frecuencia que alcanza el procesador (mínimo y máximo), además de la frecuencia actual de la CPU. Sin embargo, en el frontend solo podemos observar la frecuencia actual. En cuanto a la caché, el frontend es el doble de grande en los dos primeros niveles y casi el triple de grande en el tercer nivel.

Álvaro Morata (100405846) y Javier Moreno (100428998)

2. Análisis del rendimiento de mandel

mandel.py

```
Performance counter stats for 'system wide':

      149.982,51 msec cpu-clock                 #    3,999 CPUs utilized
         104.644      context-switches          #    0,698 K/sec
          14.823      cpu-migrations            #    0,099 K/sec
           6.226      page-faults               #    0,042 K/sec
 466.945.221.280      cycles                    #    3,113 GHz
1.391.183.564.552     instructions              #    2,98  insn per cycle
 278.582.753.083      branches                  # 1857,435 M/sec
     426.153.988      branch-misses             #    0,15% of all branches

    37,503728210 seconds time elapsed

Ejecución de mandel.py finalizada
```

release/mandel

```
Performance counter stats for 'system wide':

       26.766,92 msec cpu-clock                 #    4,000 CPUs utilized
           1.314      context-switches          #    0,049 K/sec
              11      cpu-migrations            #    0,000 K/sec
             446      page-faults               #    0,017 K/sec
  21.416.065.545      cycles                    #    0,800 GHz
  32.537.917.259      instructions              #    1,52  insn per cycle
   4.935.879.083      branches                  #  184,402 M/sec
      34.465.337      branch-misses             #    0,70% of all branches

     6,692033222 seconds time elapsed

Ejecución de release/mandel finalizada
```

release/mandel-par

```
Performance counter stats for 'system wide':

        3.972,21 msec cpu-clock                 #    3,999 CPUs utilized
             887      context-switches          #    0,223 K/sec
              16      cpu-migrations            #    0,004 K/sec
             636      page-faults               #    0,160 K/sec
   7.304.646.899      cycles                    #    1,839 GHz
  19.839.723.656      instructions              #    2,72  insn per cycle
     121.326.991      branches                  #   30,544 M/sec
       1.272.469      branch-misses             #    1,05% of all branches

     0,993380578 seconds time elapsed

Ejecución de relase/mandel-par finalizada
```

debug/mandel

```
Performance counter stats for 'system wide':

          14,14 msec cpu-clock                 #    3,690 CPUs utilized
             38      context-switches          #    0,003 M/sec
              4      cpu-migrations            #    0,283 K/sec
            121      page-faults               #    0,009 M/sec
      5.434.041      cycles                    #    0,384 GHz
      4.682.750      instructions              #    0,86  insn per cycle
        849.979      branches                  #   60,117 M/sec
         29.544      branch-misses             #    3,48% of all branches

     0,003832036 seconds time elapsed

Ejecución de debug/mandel finalizada
```

Podemos observar que prácticamente en todas las ejecuciones se utilizan 4 núcleos.

En cuanto al número de ciclos, los que tienen un mayor número son mandel.py y debug/mandel-par con una gran diferencia. Están seguidas de las ejecuciones de release/mandel y release/mandel-par. Finalmente las que menor número de ciclos tienen es la de debug/mandel.

En cuanto al número de instrucciones, el que tiene un mayor número es mandel.py. con una gran diferencia. Está seguida de las ejecuciones de debug/mandel ,release/mandel y release/mandel-par y finalmente debug/mandel que nuevamente vuelve a ser la que menor número de instrucciones tiene.

En cuanto a la frecuencia de reloj, sigue el mismo patrón que en el caso de las instrucciones.

En cuanto al tiempo transcurrido en la ejecución, vuelve a mantener la misma línea general a excepción de que esta vez, debug/mandel-par tarda un poco más que mandel.py.

En cuanto a las instrucciones por ciclo el orden sería, de mayor a menor, mandel.py, release/mandel-par, debug/mandel-par, release/mandel y debug/mandel.

debug/mandel-par

```
Performance counter stats for 'system wide':

       155.681,94 msec cpu-clock                 #    4,000 CPUs utilized
              2.972      context-switches          #    0,019 K/sec
                139      cpu-migrations            #    0,001 K/sec
                700      page-faults               #    0,004 K/sec
    491.849.602.443      cycles                    #    3,159 GHz
    823.892.787.484      instructions              #    1,68  insn per cycle
    117.708.003.792      branches                  #  756,080 M/sec
        268.539.088      branch-misses             #    0,23% of all branches

      38,920292951 seconds time elapsed

Ejecución de debug/mandel-par finalizada
```

Teniendo en cuenta los resultados anteriores, podemos concluir que la ejecución más eficiente sería la de debug/mandel y la menos la de mandel.py con mucha diferencia.

3. Análisis de la energía de ejecución

mandel.py

```
 Performance counter stats for 'system wide':

            733,74 Joules power/energy-cores/
              0,00 Joules power/energy-gpu/
            935,44 Joules power/energy-pkg/
             93,24 Joules power/energy-ram/

       38,581927561 seconds time elapsed

Ejecución de mandel.py finalizada
```

release/mandel

```
 Performance counter stats for 'system wide':

             21,95 Joules power/energy-cores/
              0,05 Joules power/energy-gpu/
             54,52 Joules power/energy-pkg/
             16,08 Joules power/energy-ram/

        6,661741170 seconds time elapsed

Ejecución de release/mandel finalizada
```

release/mandel-par

```
 Performance counter stats for 'system wide':

              9,31 Joules power/energy-cores/
              0,00 Joules power/energy-gpu/
             14,07 Joules power/energy-pkg/
              2,30 Joules power/energy-ram/

        0,967937541 seconds time elapsed

Ejecución de relase/mandel-par finalizada
```

debug/mandel

```
 Performance counter stats for 'system wide':

              0,01 Joules power/energy-cores/
              0,00 Joules power/energy-gpu/
              0,02 Joules power/energy-pkg/
              0,01 Joules power/energy-ram/

        0,003282712 seconds time elapsed

Ejecución de debug/mandel finalizada
```

Como se puede observar en las capturas de la salida, la cantidad de energía consumida va directamente relacionada con el tiempo de ejecución. Aún así se pueden observar ciertas curiosidades:

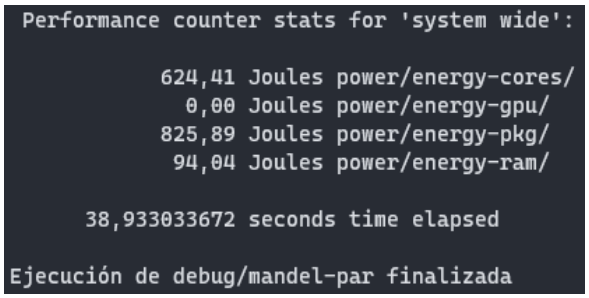
En cuanto a la estadística energy-cores, referida a la energía consumida por los núcleos, podemos observar que, a parte de ser directamente proporcional al tiempo de ejecución, ocurre algo interesante. Y es que en el caso de debug/mandel-par, a pesar de tardar un poco más en ejecución que mandel.py, el consumo de energía es unos 100 julios menor.

En cuanto a la estadística energy-gpu, referida a la energía consumida por los núcleos gráficos del procesador, podemos observar como en todos los casos es 0 julios, excepto en el caso de release/mandel, pero esto se podría deber a un error de estimación, puesto que los núcleos gráficos no se deberían utilizar para nada en esta tarea.

En cuanto al consumo del paquete, o procesador (como se puede observar aquí), podemos ver que sigue las mismas tendencias que la estadística energy-cores.

Energy-ram, a diferencia del resto, sí que es más alta en el caso de debug/mandel-par

debug/mandel-par



que en el de mandel.py. Esto puede deberse a que el consumo de la memoria RAM no depende del procesador, si no del tiempo que se tiene que acceder a ella, además de trabajar a frecuencias fijas, a diferencia del procesador.

4. Conclusiones

Álvaro:
A diferencia de probablemente bastantes compañeros, no me he encontrado con ningún problema a la hora de hacer el laboratorio o los scripts, puesto que ya tenía bastante conocimiento de Linux. Pero sí que es verdad que a bastante gente según pude observar le estaba costando seguir la clase, al no estar familiarizados con el entorno y, aunque es verdad que en Aula Global hay documentación de introducción a Linux, supongo que a bastante gente le hubiese gustado tener una clase introductoria, más que nada para familiarizarse con una CLI. Sobre el uso de ssh o un editor como Nano o Vim.
A parte de lo dicho, no conocía ninguno de los comandos vistos en este laboratorio (*perf*, *lscpu*, ni la generación de los archivos slurm al usar *sbatch*). Me ha parecido muy interesante el uso de estos comandos para ver el rendimiento de los diferentes programas, además de haberme quedado sorprendido por el buen rendimiento de *debug/mandel*, ya que suponía que iba a tardar más que las releases al estar en modo de depuración.

Javier:
En mi caso sí que he tenido problemas con las ejecuciones de los scripts y los comandos de la terminal ya que no tenía conocimientos previos y no había ninguna documentación disponible para consultar en el aula global. Por suerte, mi compañero me ha sabido explicar su funcionamiento haciendo que pudiese seguir las etapas del laboratorio de forma satisfactoria. Es por esto que concuerdo con lo previamente mencionado de que se debería hacer una clase introductoria para manejar el entorno. Dejando ese tema de lado, el resto del laboratorio me parece una buena introducción a la asignatura. El hecho de comparar resultados de computaciones de programas reales hace que veas más claro los conceptos vistos en clase. Además, también podemos ver de una forma clara que programas rinden mejor y peor y en qué características.