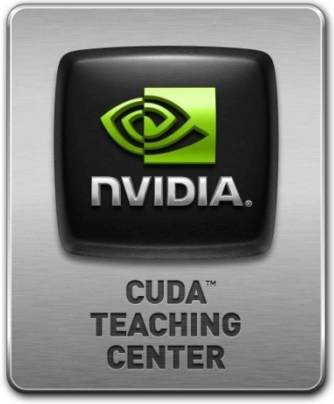
Programación masivamente paralela en procesadores gráficos

P. Ezzatti, M. Pedemonte, E. Dufrechou



Práctico 1: acceso a la memoria

Repaso: la jerarquía de memoria

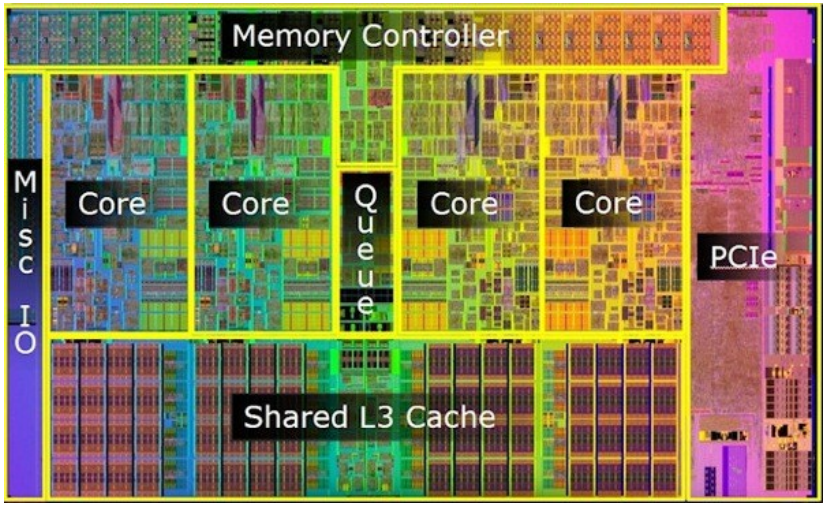
- **La velocidad de los procesadores crece más rápido que la del acceso a los datos**
- **Hacer cálculos es "gratis" comparado con acceder a memoria**
- **La solución es mantener los datos en memorias rápidas y pequeñas, y aprovechar al máximo el ancho de banda**

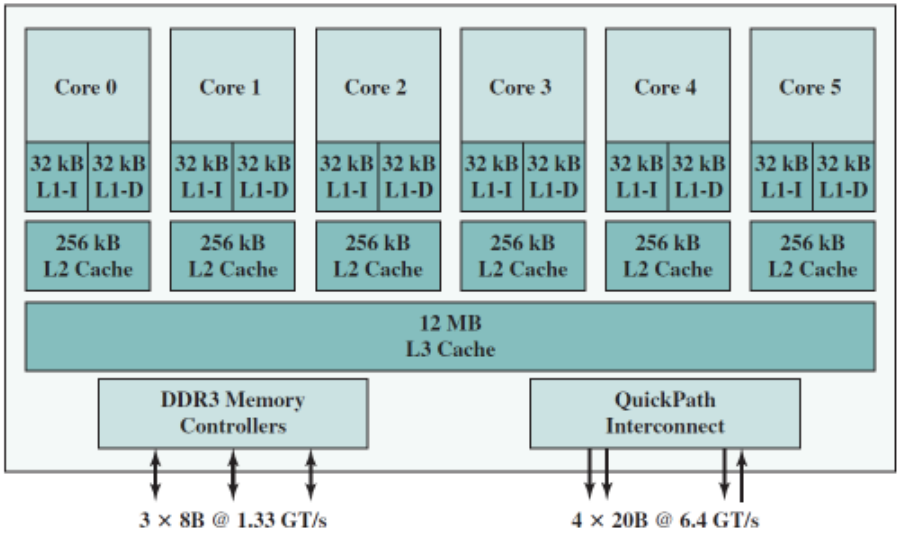
System Event	Actual Latency	Scaled Latency
One CPU cycle	0.4 ns	**1 s**
Level 1 cache access	0.9 ns	**2 s**
Level 2 cache access	2.8 ns	**7 s**
Level 3 cache access	28 ns	**1 min**
Main memory access (DDR DIMM)	~100 ns	**4 min**
Intel Optane memory access	<10 µs	**7 hrs**
NVMe SSD I/O	~25 µs	**17 hrs**
SSD I/O	50–150 µs	**1.5–4 days**
Rotational disk I/O	1–10 ms	**1–9 months**
Internet call: SF to NYC	65 ms	**5 years**
Internet call: SF to Hong Kong	141 ms	**11 years**

Números aproximados (dependen de la arquitectura y otros factores…) la tendencia es que cada nivel es 3-10X más lento que el nivel superior.

Repaso: la jerarquía de memoria

- **Los procesadores dedican un área importante a la memoria caché**
  - **En procesadores de propósito general suele estar organizada en 3 niveles.**
  - **L1 es pequeño y se divide en instrucciones y datos**
  - **L2 está dentro del core y es compartido por inst. y datos.**
  - **L3 es compartido por todos los cores**





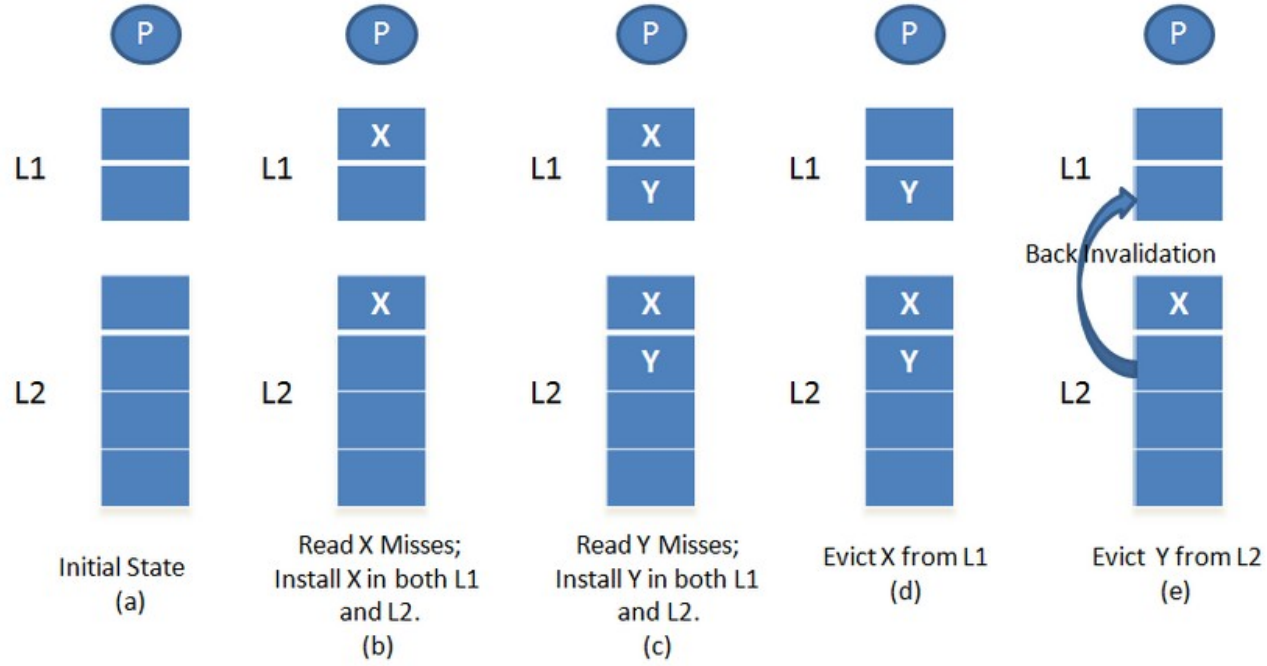
Intel Core i7-990X Block Diagram

Repaso: tipos de caché

- Cuando accedemos a 1 byte en la memoria principal en realidad se carga un bloque en la caché del tamaño de una línea (64 B)
- En qué dirección de la caché se carga ese bloque?
- Fully-associative
  - El bloque se puede almacenar en cualquier línea
  - Si la línea está ocupada se necesita una política de remplazo
  - Debo comparar el tag del bloque contra todas las líneas de la caché (lento y caro)
- Direct-mapped
  - Cada bloque tiene asociada una línea
  - No se necesita una política de remplazo (rápido y barato)
- Set-associative
  - La caché se divide en “pequeñas cachés fully-associative” (sets)
  - Balance entre las estrategias anteriores

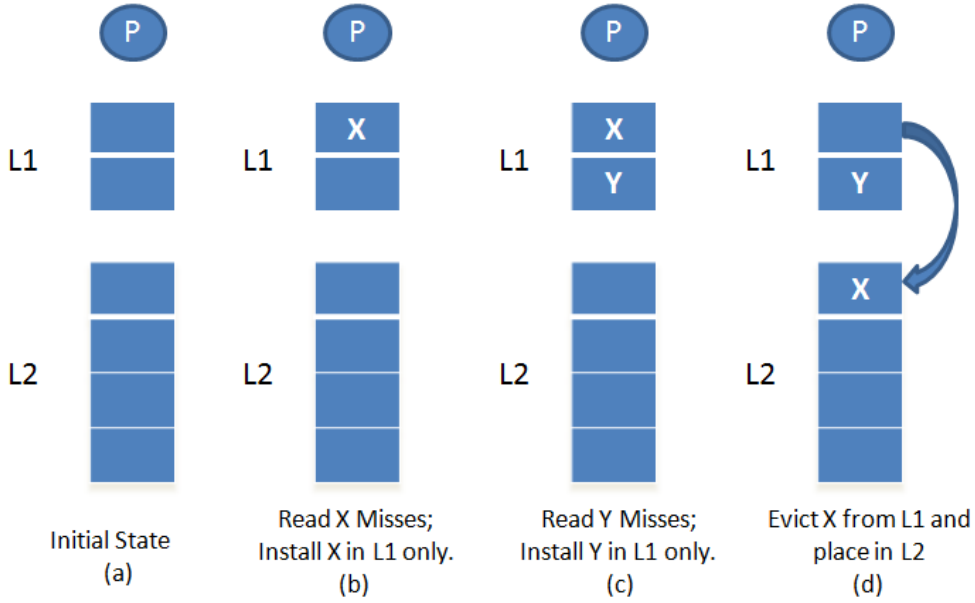
Repaso: inclusión de cachés

- Cómo se distribuyen los datos entre los distintos niveles?
- Caché inclusiva (incluye a la cache de nivel superior)
  - Miss: el dato se escribe en L1 y L2
  - Reemplazo en L1: no se hace nada
  - Reemplazo en L2: se invalida la línea correspondiente en L1



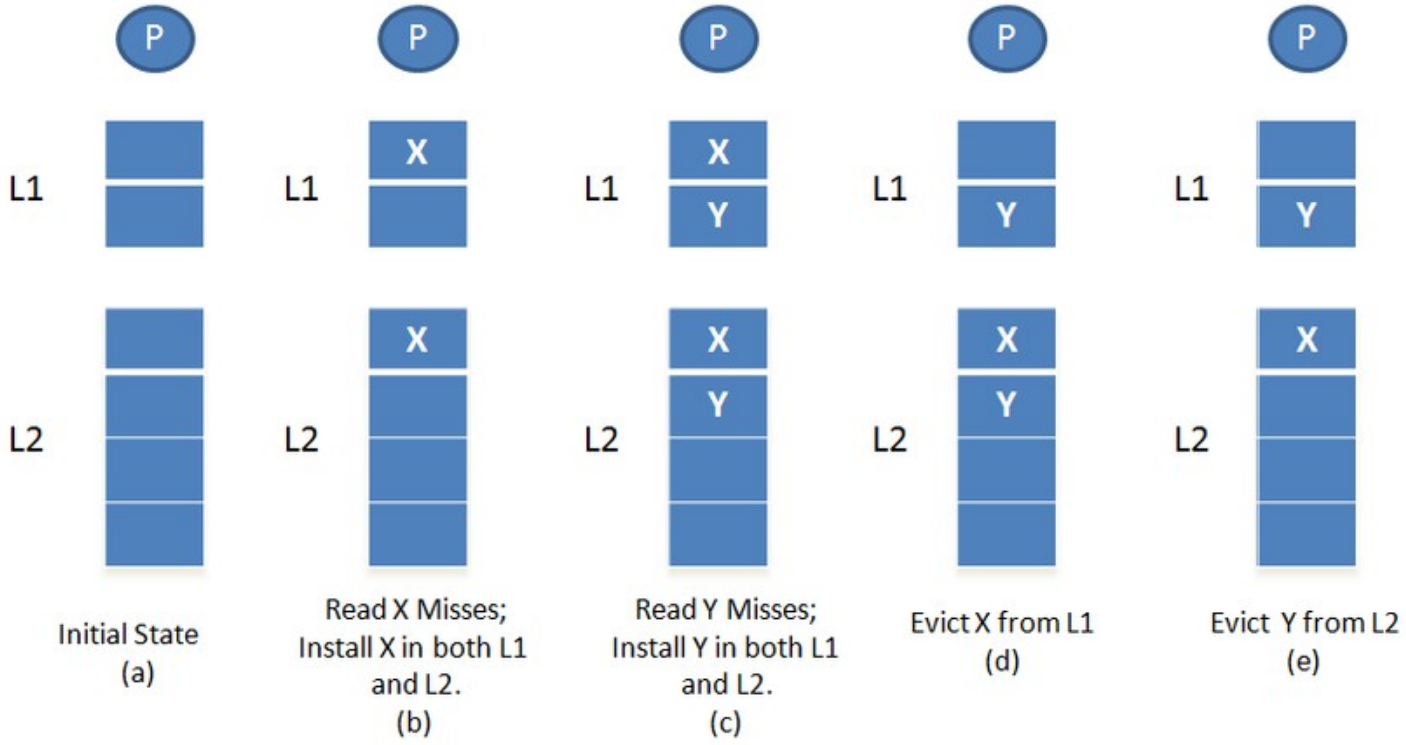
Repaso: inclusión de cachés

- Cómo se distribuyen los datos entre los distintos niveles?
- Caché exclusiva (excluye a la cache de nivel superior)
  - Miss: el dato se escribe en L1 y no en L2
  - Reemplazo en L1: la línea se copia a L2
  - Reemplazo en L2: no se hace nada



Repaso: inclusión de cachés

- Cómo se distribuyen los datos entre los distintos niveles?
- Caché no-inclusiva y no-exclusiva (NINE)
  - Miss: el dato se escribe en L1 y en L2
  - Reemplazo en L1 y L2: no se hace nada con la línea reemplazada



Repaso: inclusión de cachés

- Cómo se distribuyen los datos entre los distintos niveles?
- Caché inclusiva (incluye a la cache de nivel superior)
  - Pro: Menor latencia en miss para sistemas multicore con caches privadas
  - Con: Reduce la capacidad de la caché
- Caché exclusiva (excluye a la cache de nivel superior)
  - Pro: la capacidad de la caché es la suma de las cachés.
  - Con: las líneas reemplazadas en L1 van a L2 (mayor tráfico en el bus y tiempo en caché miss)
- No-inclusiva y no-exclusiva (NINE)
  - Pro: las líneas reemplazadas en L1 no se escriben en L2 (solo se escribe si hay miss en L2), las líneas reemplazadas en L2 no invalidan líneas en L1 (como en la excl.)

Repaso: alineamiento

- El acceso desalineado a los datos puede penalizar el rendimiento de un programa
- Es importante que el acceso a una estructura de datos no se reparta en más de una línea de cache
  - Reorganizar los datos y el cómputo para maximizar el uso de las líneas de caché accedidas en loops interiores
- Tener especial cuidado con tipos de datos “grandes” como punto flotante de doble precisión (8 bytes) o structs.
- Usar bibliotecas especializadas para reservar memoria alineada
  - Por ejemplo: aligned_alloc

Repaso: prefetch

- **Los procesadores modernos son capaces de detectar patrones regulares en el acceso a los datos.**
  - **Esto les permite cargar una línea de caché antes de que sea requerida por una instrucción.**
  - **Los compiladores (y programadores) también pueden incluir instrucciones que realizan prefetch de datos.**
- **Pueden afectar negativamente el desempeño:**
  - **Traer datos que no son necesarios**
  - **Aumentar la carga del bus de datos**
- **Para favorecer el prefetch:**
  - **Alinear los datos.**
  - **Acceder a los datos linealmente y usar saltos pequeños de ser posible.**
  - **Organizar los datos. Por ejemplo: en procesadores Intel Core, organizar los datos en bloques de 128 B para favorecer el pref. de L2. Acceder a una línea del bloque hace cargar la otra.**

Técnicas para mejorar el acceso

- **Los caché miss tienen un costo alto**
  - **Incluso si hay pocos misses en relación al total de accesos (bajo miss rate) los procesadores pasan bastante tiempo esperando por resolución de caché misses.**
  - **Es importante reducir al máximo la cantidad de caché misses aumentando la localidad temporal y espacial de los datos.**
- **Algunas técnicas:**
  - **Strip-Mining o Blocking (algoritmos “por bloques”): acceder a una porción de los datos que entre en caché repetidamente, antes de pasar a la siguiente porción.**
  - **Intercambio de loops: reordenar los loops para que los datos se recorran lo más secuencialmente posible.**
  - **Desplazamiento de loops: elimina dependencias de datos entre loops**

Técnicas para mejorar el acceso

- El compilador hace muchas de estas optimizaciones por nosotros... igual hay que ayudarlo:
  - Diseñar las estructuras de datos para que entren en la mitad de la caché L1 o en L2.
  - Optimizar para un punto medio (como para todo L1) no mejora respecto a optimizar para L2.
  - Favorecer siempre el acceso lineal a los datos con saltos pequeños (ayuda al prefetch automático)
  - Usar las flags del compilador: -O3, --unroll-loops, flags específicas para cada arquitectura, etc...

Strip-mining / blocking

- **Strip-mining: dividir loops lineales para que los datos queden en caché y puedan ser reutilizados (también ayuda a vectorizar)**
  - **Ejemplo - recorrida de puntos en una geometría:**

```
typedef struct _VERTEX {
    float x, y, z, nx, ny, nz, u, v;
} Vertex_rec;

main()
{
    Vertex_rec v[Num];
    ....
    for (i=0; i<Num; i++) {
        Transform(v[i]);
    }
    for (i=0; i<Num; i++) {
        Lighting(v[i]);
    }
    ....
}
```

```
main()
{
    Vertex_rec v[Num];
    ....
    for (i=0; i < Num; i+=strip_size) {
        FOR (J=I; J < MIN(NUM, I+STRIP_SIZE); J++) {
            TRANSFORM(V[J]);
        }
        FOR (J=I; J < MIN(NUM, I+STRIP_SIZE); J++) {
            LIGHTING(V[J]);
        }
    }
}
```

Strip-mining / blocking

- **Blocking: dividir loops bidimensionales en tiles para que los datos queden en caché y puedan ser reutilizados**
  - **Cada tile debería ser lo suficientemente pequeño para entrar en caché**
- **Ejemplo: A se recorre por filas y B se recorre por columnas**
  - **Si las matrices se almacenan por filas cada acceso a B es un cache miss**

```
float A[MAX, MAX], B[MAX, MAX]
for (i=0; i< MAX; i++) {
    for (j=0; j< MAX; j++) {
        A[i,j] = A[i,j] + B[j, i];
    }
}
```

```
float A[MAX, MAX], B[MAX, MAX];
for (i=0; i< MAX; i+=block_size) {
    for (j=0; j< MAX; j+=block_size) {
        for (ii=i; ii<i+block_size; ii++) {
            for (jj=j; jj<j+block_size; jj++) {
                A[ii,jj] = A[ii,jj] + B[jj, ii];
            }
        }
    }
}
```

Ejercicio 1

- **Parte a:**
  - **Obtenga los datos de la jerarquía de memoria de su computadora (cantidad de niveles de caché, tamaños, política de inclusión, asociatividad, etc.)**
  - **Teniendo en cuenta estos datos escriba un programa sencillo que mida la velocidad máxima de acceso (latencia o ancho de banda) de cada nivel de caché.**
  - **Explique el funcionamiento del programa y analice los resultados.**
- **Parte b:**
  - **Desarrolle un programa sencillo donde se aprecie el efecto del prefetch.**
  - **Explique el programa y los resultados**
- **Parte c:**
  - **Desarrolle un programa sencillo donde se aprecie el efecto del acceso desalineado a la memoria y dividir datos entre dos líneas de caché.**

Ejercicio 2

- **Parte a:**
  - **El siguiente código multiplica dos matrices A (mxp) y B (pxn), almacenando el resultado en C (mxn):**

```
VALT sum;
for (int row = 0; row < m; row++)
    for (int col = 0; col < n; col++){
        sum=0;
        for (int it = 0; it < p; it++)
            sum += A[row * p + it] * B[it * n + col];
            C[row * n + col]=sum;
        }
```

  - **Reordene los loops para maximizar los hits en caché y compare el rendimiento en GFlops con el código anterior para distintos tamaños de matriz (que no entren completamente en caché).**
- **Parte b:**
  - **Aplique la técnica de "blocking" y compare el desempeño de la versión anterior (después del reordenamiento de loops)**
  - **Obtenga resultados para distintos tamaños de matriz y bloque**

Ejercicio 2

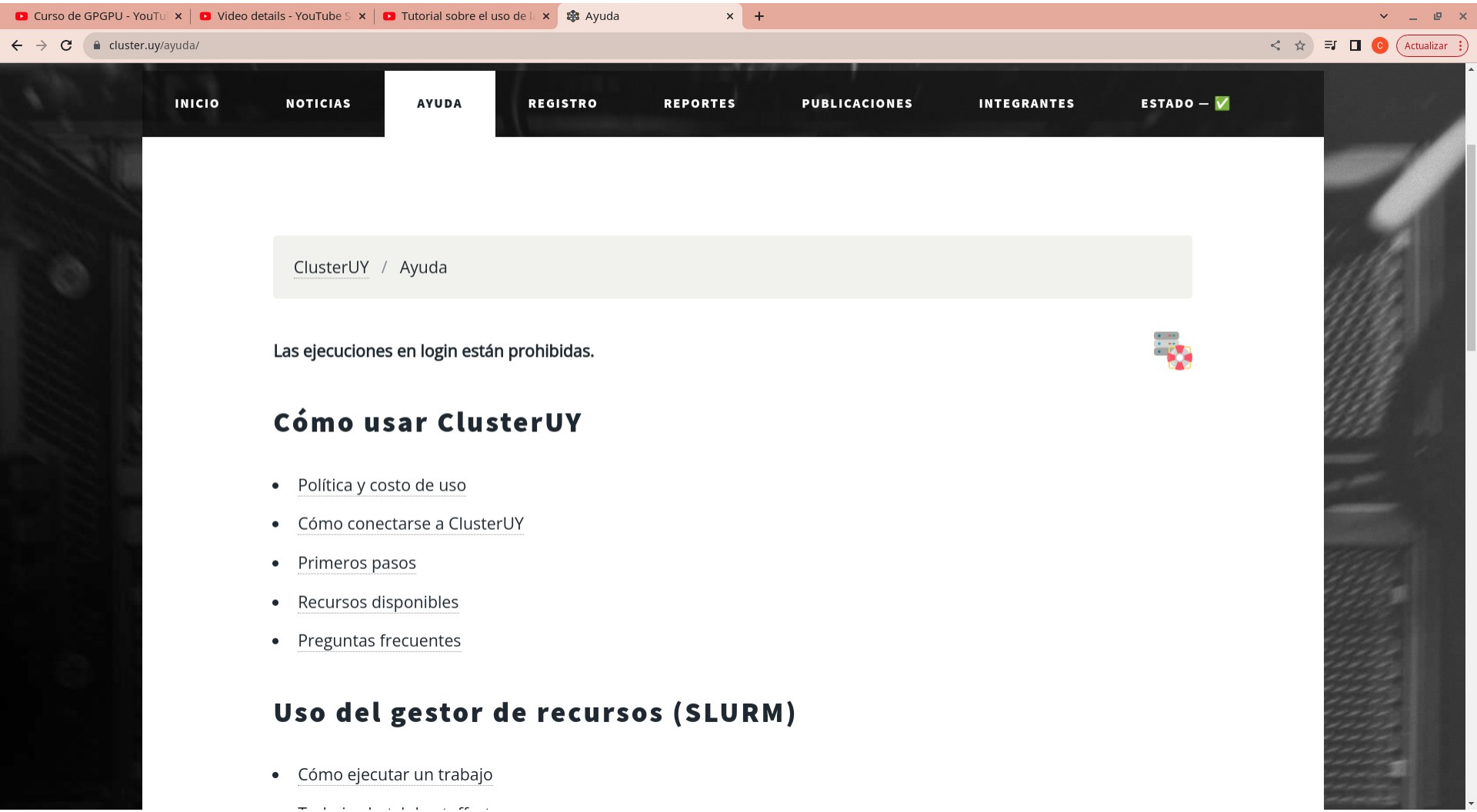
- **Parte c:**
  - **Muestre un tamaño de matriz y bloque donde debería producirse una tasa alta de caché miss debido a la competencia de los accesos por un set específico de la caché.**
  - **Compare y analice el rendimiento en Gflops con las partes anteriores**

ClusterUY: características

- 1216 núcleos de cómputo CPU
  - 1120 son núcleos Intel Xeon-Gold 6138 2.00GHz
  - 96 son núcleos AMD EPYC 7642 2.30GHz
- 3,8 TB de memoria RAM
- 100.352 núcleos de cómputo GPU
  - Nvidia Tesla P100 con 12Gb de memoria
  - Nvidia Ampere A100 de 40Gb (6912 núcleos CUDA FP32, 3456 CUDA FP64, y 422 núcleos Tensor)
- Red de alta velocidad Ethernet de 10 Gbps

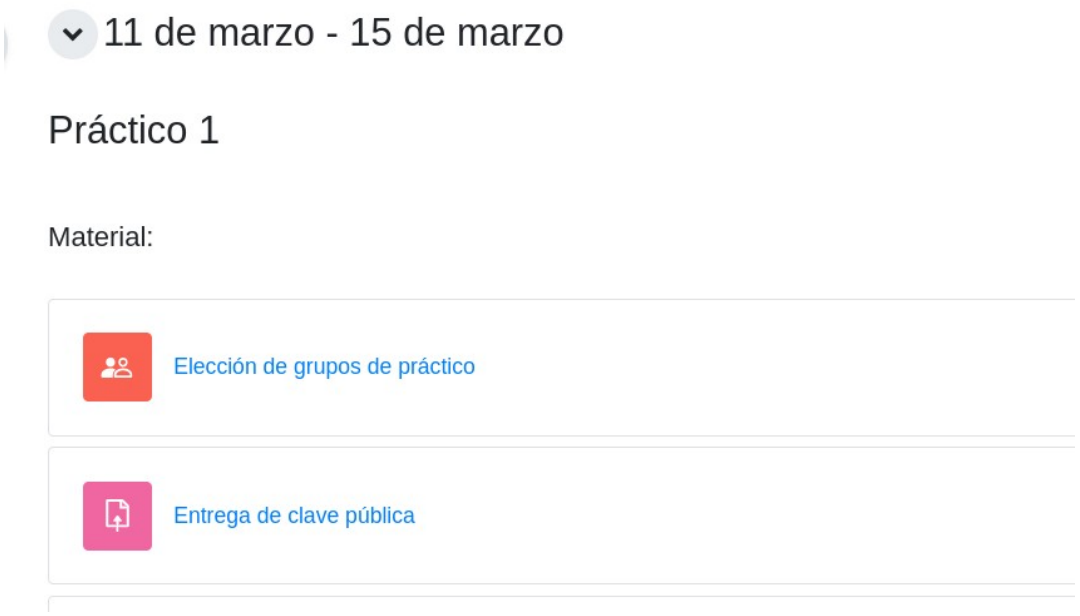
ClusterUY: web

- www.cluster.uy



ClusterUY: login

- Formar un grupo de práctico usando la actividad en el eva
- Crear un par de claves pública y privada para el grupo:
  - ssh-keygen -t rsa
  - Enviar la clave PÚBLICA a través de la tarea que está en eva con ese fin.



ClusterUY: login

- Formar un grupo de práctico usando la actividad en el eva
- Crear un par de claves pública y privada para el grupo:
  - ssh-keygen -t rsa
  - Enviar la clave PÚBLICA a través de la tarea que está en eva con ese fin.
- Poner las claves generadas (al menos la privada) en el directorio ~/.ssh (linux)
  - De otra forma hay que especificar la ruta a la clave desde el comando ssh:
    - ssh -i /ruta/clave/id_rsa usuario@login.cluster.uy
- Si tengo más de una clave en .ssh puedo especificarlo en un archivo ~/.ssh/config usando la palabra clave IdentityFIle:
  - IdentityFile ~/.ssh/clave1
  - IdentityFile ~/.ssh/clave2

ClusterUY: lanzar trabajos

- Una vez que entramos al cluster estamos en el nodo login
  - NO EJECUTAR NINGÚN PROCESO PESADO EN LOGIN!!!
  - Para ejecutar trabajos (incluso compilar) debemos usar el sistema de colas de ejecución (SLURM)

- Se ejecuta a través del comando sbatch
  - El parámetro del comando es un script de bash que comienza con las directivas que se pasan a SLURM y los comandos a ejecutar en el nodo de cómputo.
  - Ejemplo: sbatch lanzar_trabajo.sh parámetro1 parámetro2 ...

ClusterUY: lanzar trabajos

```
#!/bin/bash
#SBATCH --job-name=mitrabajo
#SBATCH --ntasks=1
#SBATCH --mem=512
#SBATCH --time=00:01:00

#SBATCH --partition=besteffort
# SBATCH --partition=normal

#SBATCH --qos=besteffort_gpu
# SBATCH --qos=gpu

#SBATCH --gres=gpu:p100:1
# #SBATCH --mail-type=ALL
#SBATCH --mail-user=mi@correo
#SBATCH -o salida.out

export PATH=$PATH:/usr/local/cuda/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64

$1 $2 $3 $4 $5 $6 $7 $8 $9
```

ClusterUY: comandos útiles

- squeue -u usuario
  - Permite ver los trabajos encolados del usuario
- scancel número_trabajo
  - Permite cancelar un trabajo
- interactivo -gpu
  - Un trabajo interactivo permite conectarse directamente a un nodo de cómputo y trabajar con una consola.
- srun --job-name=mitrabajo --time=00:20:00 --partition=besteffort --qos=besteffort --ntasks=1 --mem=512 --gres=gpu:p100:1 --pty bash -l
  - Solicita una sesión interactiva de forma manual
  - En la partición besteffort los trabajos pueden pausarse para dejar los recursos a otros trabajos de mayor prioridad. En general es más rápida que la partición normal

ClusterUY: almacenamiento temporal

- Hay un espacio de almacenamiento rápido dentro de los nodos de cómputo que se ubica en /scratch/nombre_de_usuario
  - Para usarlo incluir --tmp=xxxG en el script de SLURM, donde xxx es la cantidad de GB que se reservan de ese espacio (max. 300).
  - Luego copiar los datos temporales al espacio /scratch/
  - Una vez terminada la ejecución, copiar el resultado de vuelta al home
- Ejecución de trabajos con CUDA que usen muchos archivos en HOME
  - Cuando un trabajo que ejecuta con CUDA utiliza una gran cantidad de archivos (por ejemplo, de tamaño pequeño) ubicados en el directorio home del usuario, puede ocurrir que ese proceso no responda y no haya forma de finalizarlo. Para evitar este problema, es necesario utilizar el almacenamiento de alta velocidad ubicado en /scratch/<nombre_de_usuario para estos archivos (ver Uso del espacio temporal de alta velocidad).

ClusterUY: ojo...

- Hay un problema conocido con la herramienta VisualStudio Code y ClusterUY
  - La herramienta deja varios procesos corriendo en el cluster que quedan vivos incluso luego de cerrar sesión (y consumen muchos recursos en el nodo login)
  - Solución: Evitar el uso de esta herramienta (o por lo menos de la extensión SFTP...)
- Mejor acceder al cluster directamente desde la terminal
- Para editar los archivos de forma remota:
  - Hacerlo directamente desde la terminal usando vi / vim / nano
  - (en linux) conectar el navegador de archivos al cluster yendo a “+ Otras ubicaciones” y tipeando “sftp://usuario@login.cluster.uy/”