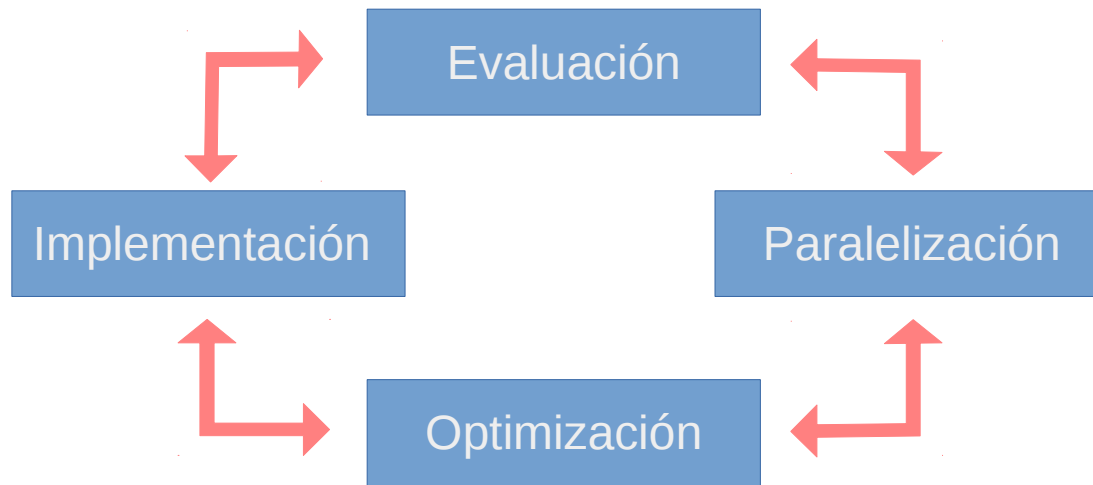


Herramientas de profiling



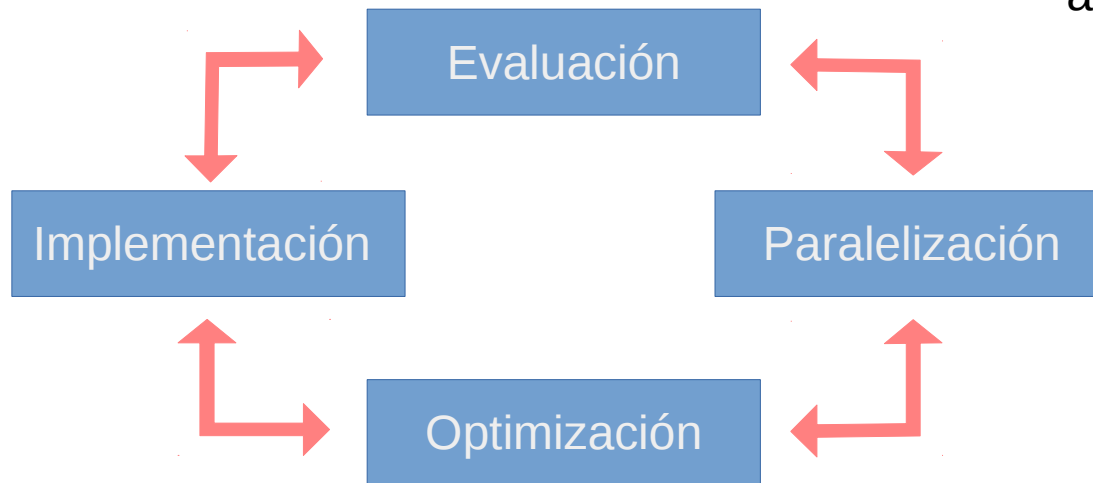


Identificar regiones críticas

- Regiones con altos requerimientos de cómputo
- Cuellos de botella

...

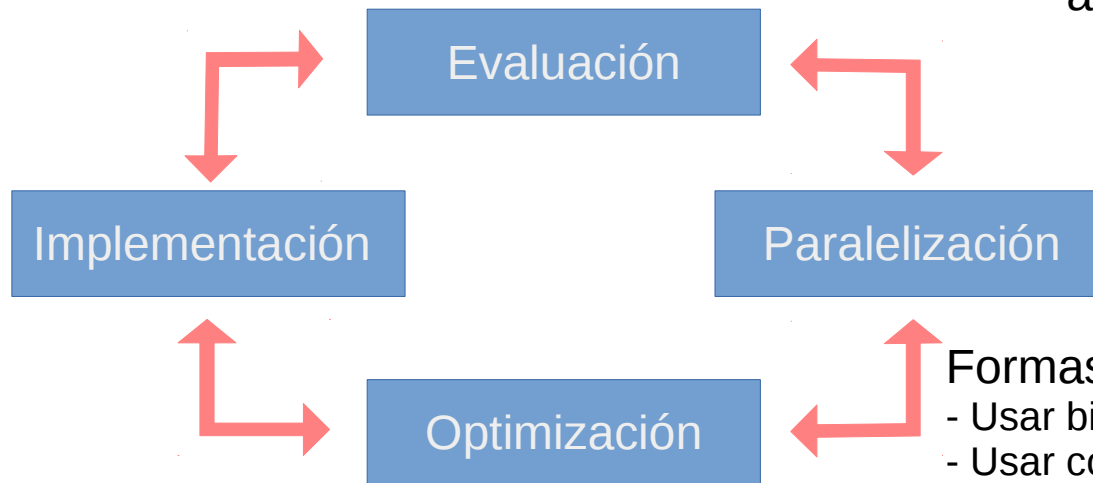
? Puedo usar GPU para acelerar el procesamiento?



Identificar regiones críticas

- Regiones con altos requerimientos de cómputo
- Cuellos de botella
- ...

? Puedo usar GPU para acelerar el procesamiento?



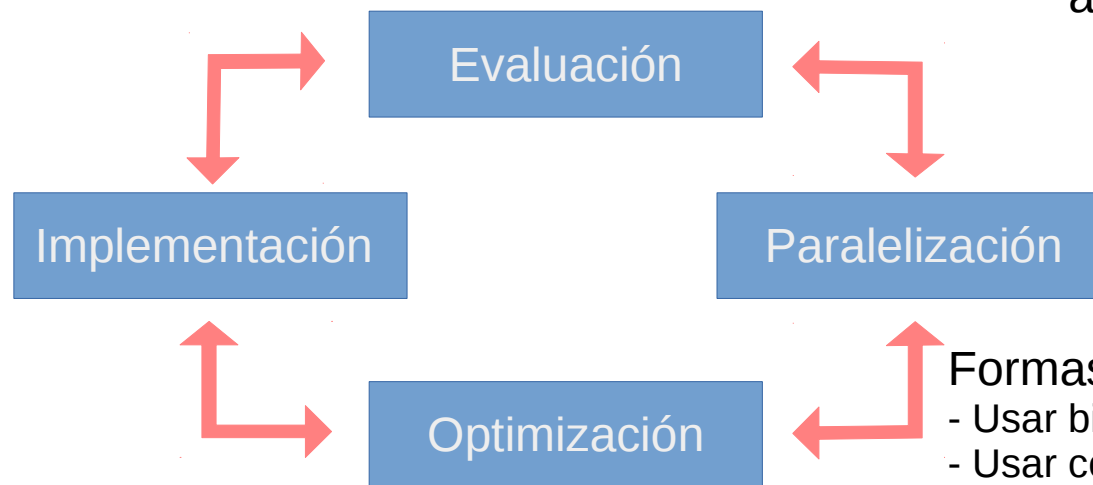
Formas de paralelizar el código:

- Usar bibliotecas de CUDA
- Usar compiladores que paralelicen código
- Desarrollar kernels CUDA

Identificar regiones críticas

- Regiones con altos requerimientos de cómputo
- Cuellos de botella
- ...

? Puedo usar GPU para acelerar el procesamiento?



Formas de paralelizar el código:

- Usar bibliotecas de CUDA
- Usar compiladores que paralelicen código
- Desarrollar kernels CUDA

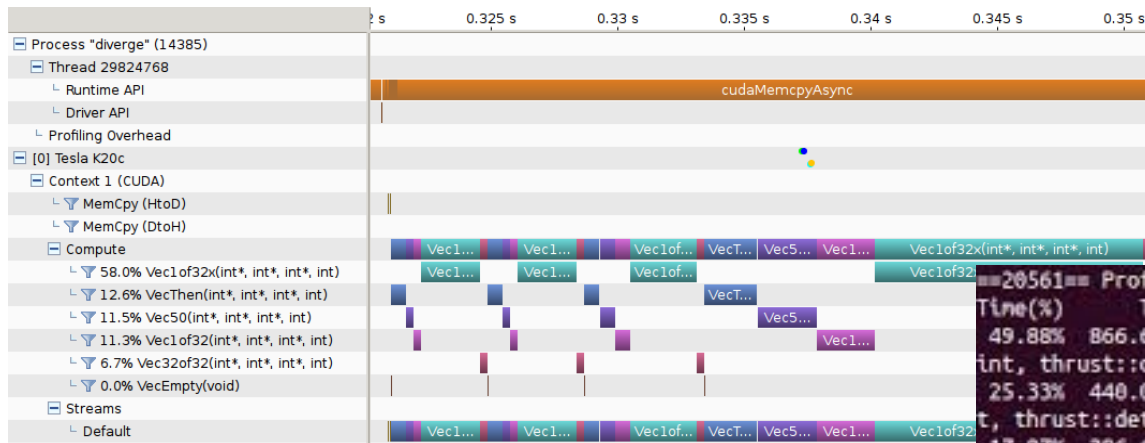
Niveles de optimización:

- A nivel de grid: enfoca en la utilización de la GPU y eficiencia
 - lanzar varios kernels al mismo tiempo
 - solapar ejecución de kernel con transferencia de datos.
- A nivel de kernel:
 - uso eficiente de ancho de banda de las memorias de la GPU
 - uso eficiente de los cores.
 - Reducir / ocultar latencias de instrucciones o de memorias.

CUDA:

– Herramientas de profiling :

- nvprof: NVIDIA profiling
- nvvp; NVIDIA Visual Profiling



==20561== Profiling result:

Time(%)	Time	Calls	Avg	Min	Max	Name
49.88%	866.69ms	504758	1.7170us	1.5040us	2.0160us	void th
25.33%	440.05ms	252662	1.7410us	1.5360us	2.3680us	void th
17.07%	296.60ms	200	1.4830ms	1.2040ms	1.7253ms	kerComp
2.98%	51.819ms	200	259.09us	246.97us	264.83us	kerMake
1.16%	20.173ms	501	40.265us	928ns	17.677ms	[CUDA m
0.93%	16.198ms	200	80.991us	71.840us	90.751us	kerColl
0.73%	12.636ms	400	31.589us	14.720us	50.432us	[CUDA m
0.69%	12.075ms	200	60.376us	59.680us	62.304us	kerRena
0.63%	10.993ms	200	54.963us	52.608us	58.208us	kerMake
0.32%	5.5524ms	200	27.761us	22.559us	33.152us	[CUDA m
0.12%	2.1342ms	1	2.1342ms	2.1342ms	2.1342ms	void th

nvprof

- Usando nvprof se pueden recolectar información de dos tipos:
 - Una linea de tiempo de actividades de CUDA, en CPU y GPU
 - Eventos y métricas para los kernels.
- Se llama desde linea de comando con la sintaxis:
 - `nvprof [opciones de nvprof] <aplicacion> [argumentos]`

nvprof

- Modos:
 - Por defecto viene en modo resumen: `nvprof ./aplicacion`
 - Modo traza: con las opciones
 - `-print-gpu-trace`: muestra cronológicamente la actividad en la GPU
 - `-print-api-trace`: CUDA runtime y API calls invocadas en host, orden cronológico.
Probar!! `nvprof -print-gpu-trace ./ejemplo`
 - Modo event/metric resumen:
 - **Eventos**: se basan en contadores de hardware observados durante la ejecución de una aplicación. `nvprof -events <event names>`
 - **Métricas**: se calculan en base a los eventos: por ejemplo, cantidad de accesos a memoria global y aciertos en cache L1 son eventos, y de ellos se puede derivar la métrica del uso de cache de la aplicación. `nvprof -metrics <metric names>`
 - Para conocer todos los eventos y métricas posibles: `--query-events` y `-query-metrics`.
- Probar:

`nvprof -query-events`

`nvprof -query-metrics`

nvprof

- `nvprof -query-events`

```
Available Events:
      Name  Description
Device 0 (GeForce GTX 550 Ti):
  Domain domain_a:
    sm_cta_launched: Number of thread blocks launched on a multiprocessor.
    l1_global_load_hit: Number of cache lines that hit in L1 cache for global memory load access.
    l1_global_load_miss: Number of cache lines that miss in L1 cache for global memory load accesses.
uncached_global_load_transaction: Number of uncached global load transactions. Increments by 1 per transaction. Transaction can be 32/64/96/128B.
  global_store_transaction: Number of global store transactions. Increments by 1 per transaction. Transaction can be 32/64/96/128B.
    l1_shared_bank_conflict: Number of shared bank conflicts caused due to addresses for two or more shared memory requests fall in the same memory bank.
    shared_load: Number of executed load instructions where state space is specified as shared, increments per warp on a multiprocessor.
    shared_store: Number of executed store instructions where state space is specified as shared, increments per warp on a multiprocessor.
    branch: Number of branch instructions executed per warp on a multiprocessor.
    divergent_branch: Number of divergent branches within a warp. This counter will be incremented by one if at least one thread in a warp diverges (that is, follows a different execution path).
```

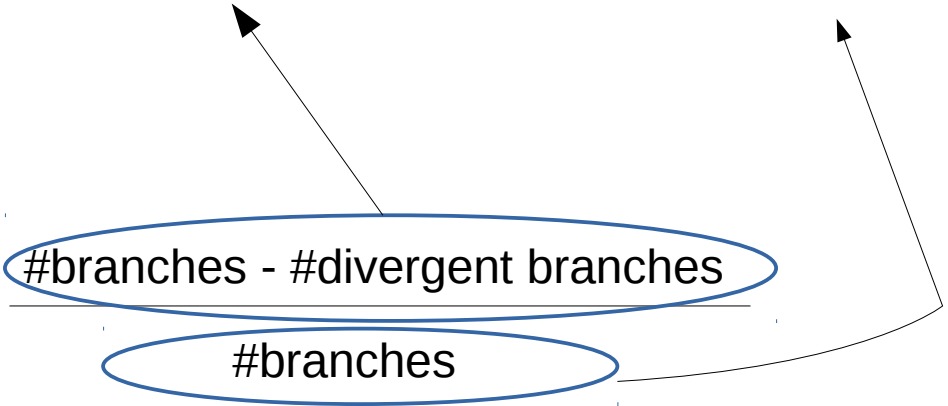
Probar: `nvprof -events sm_cta_launched,threads_launched ./ejemplo`

↓
Bloques repartidos a los SM

↓
threads lanzados

nvprof -metrics branch_efficiency

- Se define como la tasa de ramas no divergentes y la cantidad total de ramas:

$$\text{Branch efficiency} = 100 \times \frac{\# \text{branches} - \# \text{divergent branches}}{\# \text{branches}}$$


nvprof

Eventos y métricas de operaciones de memoria: para medir la eficiencia de un kernel operando con las diferentes tipos de memoria.

1) Accesos a memoria global:

- Si no se accede de manera “correcta” a la memoria global, un pedido de acceso se puede traducir en 2 o más accesos reales (replay memory requests)
- Métricas para chequear eficiencia de lecturas y escrituras en memoria global:
 - `gld_efficiency` → `nvprof -metrics gld_efficiency ./app`
 - `gst_efficiency` → `nvprof -metrics gst_efficiency ./app`

muestran la relación entre los pedidos de accesos a memoria con los accesos requeridos realmente (el primero no incluye repeticiones y el segundo si).

Use nvprof to Profile Anything

nvprof knows how to profile CUDA kernels running on NVIDIA GPUs, no matter what language they are written in (as long as they are launched using the CUDA runtime API or driver API). This means that I can use nvprof to profile OpenACC programs (which have no explicit kernels), or even programs that generate PTX assembly kernels internally. Mark Ebersole showed a great example of this in his recent CUDACast ([Episode #10](#)) about CUDA Python, in which he used the [NumbaPro](#) compiler (from Continuum Analytics) to Just-In-Time compile a Python function and run it in parallel on the GPU.

During initial implementation of OpenACC or CUDA Python programs, it may not be obvious whether or not a function is running on the GPU or the CPU (especially if you aren't timing it). In Mark's example, he ran the Python interpreter inside of nvprof, capturing a trace of the application's CUDA function calls and kernel launches, showing that the kernel was indeed running on the GPU, as well as the `cudaMemcpy` calls used to transfer data from the CPU to the GPU. This is a great example of the "sanity check" ability of a lightweight command line GPU profiler like nvprof.

<https://devblogs.nvidia.com/parallelforall/cuda-pro-tip-nvprof-your-handly-universal-gpu-profiler/>

nvprof

Eventos y métricas de operaciones de memoria: para medir la eficiencia de un kernel operando con las diferentes tipos de memoria.

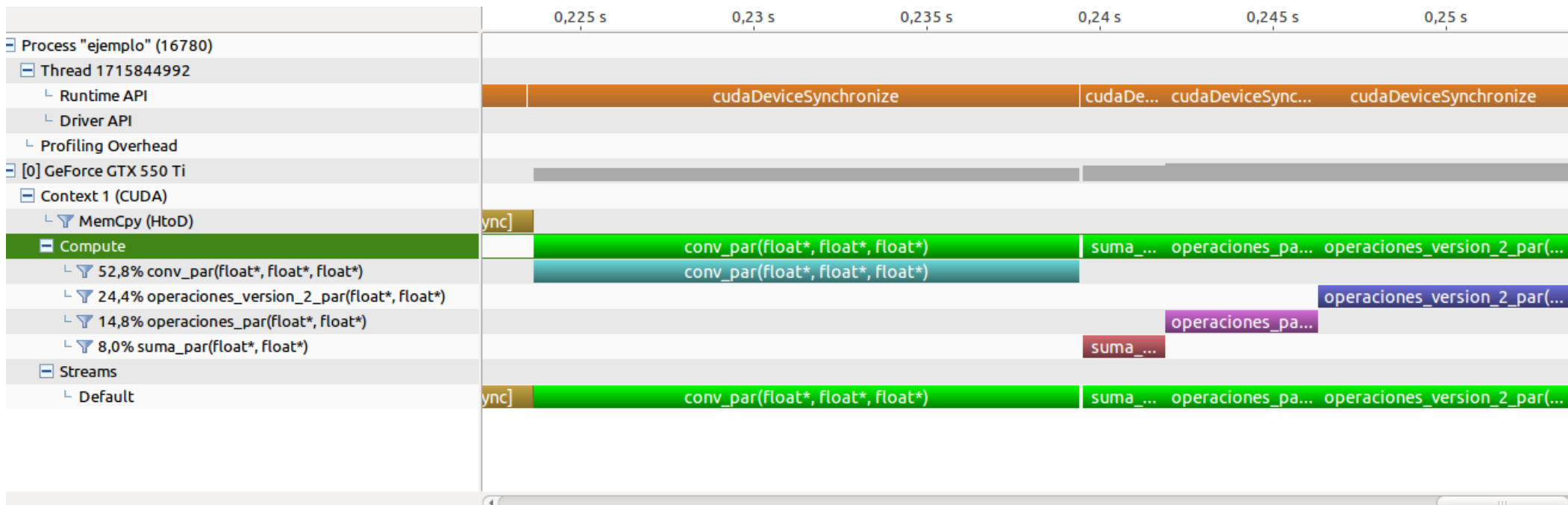
2) Más sobre accesos a memoria global:

- Métricas para chequear eficiencia de lecturas y escrituras en memoria global:
 - `gld_transactions_per_request` → promedio de lecturas realizadas por cada pedido lectura
 - `gst_transactions_per_request` → promedio de escrituras realizadas por cada pedido de w

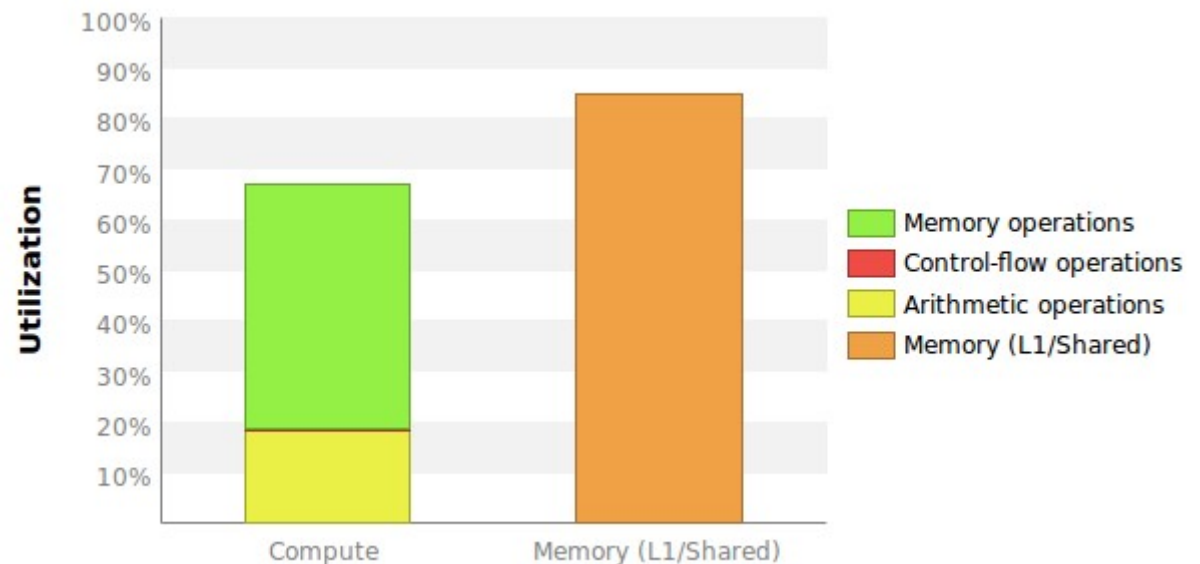
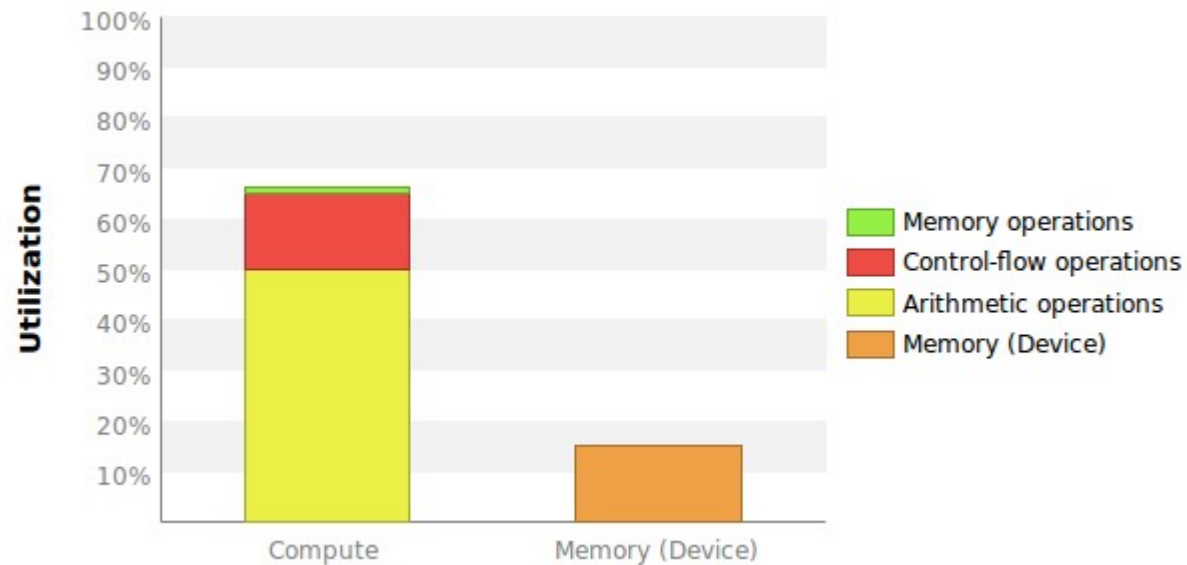
nvvp

- NVIDIA Visual Profiler es una herramienta gráfica con 2 características que la diferencian de nvprof:
 - Una línea de tiempo que muestra gráficamente las actividades de CPU y de la GPU.
 - Análisis automático de performance que ayuda a optimizar las aplicaciones.
- nvvp está disponible como una aplicación por sí misma (nvvp) o como parte de Nsight Eclipse Edition (entorno de desarrollo).
- Desde línea de comando: `nvvp ./ejemplo`

nvvp: time line de la aplicación



nvvp: análisis de performance de cada kernel...



nvvp: utilización de distintas FU

i Function Unit Utilization

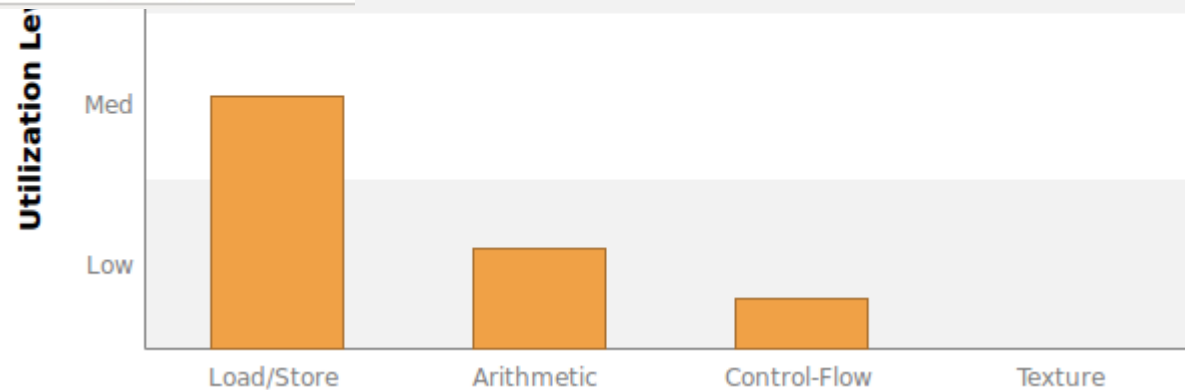
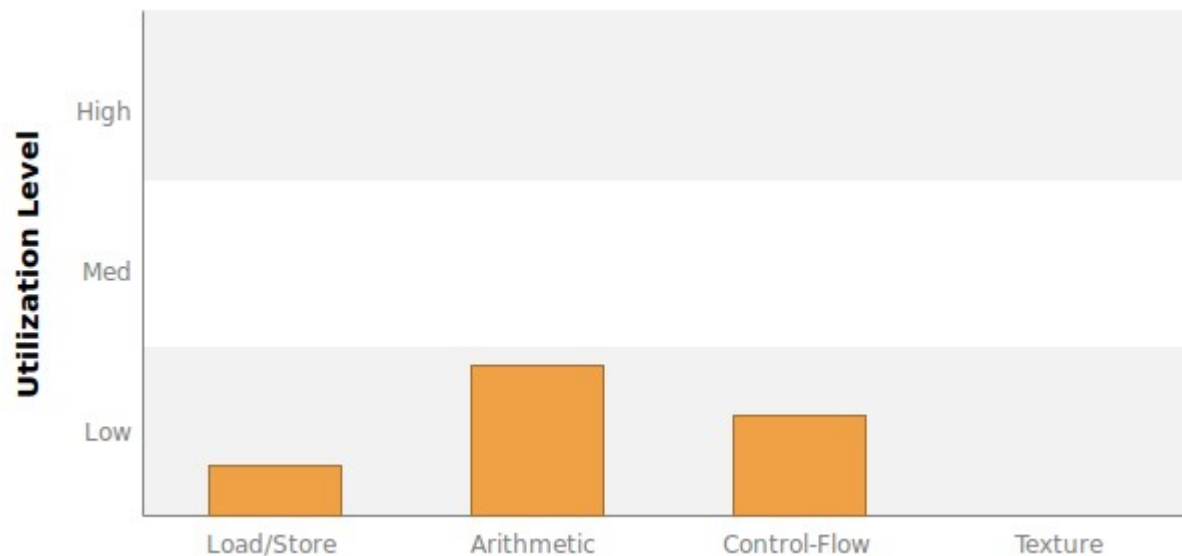
Different types of instructions are executed on different function units within each SM. Performance can be limited if a function unit is over-used by the instructions executed by the kernel. The following results show that the kernel's performance is not limited by overuse of any function unit.

Load/Store - Load and store instructions for local, shared, global, constant, etc. memory.

Arithmetic - All arithmetic instructions including integer and floating-point add and multiply, logical and binary operations, etc.

Control-Flow - Direct and indirect branches, jumps, and calls.

Texture - Texture operations.



- Se puede generar un archivo con la salida de nvprof para visualizarla con nvvp.

Simplemente:

```
[mdenham@fisica]$ nvprof -o profile.result ./aplicacion
```

Y despues se puede abrir profile.result desde nvvp.

Probando el código

1) Copiar el contenido de la carpeta con el código de ejemplo:

```
[mdenham@fisica]$ cp -a /share/apps/codigos/alumnos_icnpg2016/clase_02/profiling/ .
```

2) Cargar el módulo cuda-7.5 y compilar:

```
[mdenham@fisica]$ module load cuda-7.5  
[mdenham@fisica]$ make
```

3) Ejecutar utilizando nvprof: para esto se modifica el script submit_gpuh.h:

```
nvprof ./main
```

```
nvprof -print-gpu-trace ./main
```

```
nvprof --metrics gld_efficiency,gst_efficiency ./main
```