Informe de Segunda Práctica de Arquitectura de Computadores

Máximo García Aroca

22 de octubre, 2023

Medición del rendimiento: se analiza el rendimiento de un programa de multiplicación de matrices. Se utiliza la herramienta "perf" disponible en el kernel de Linux a partir de la versión 2.6.31 para medir eventos de rendimiento, incluyendo eventos hardware, software y de traza. Esta práctica busca comprender y mejorar el rendimiento del código a través de la medición detallada de eventos de rendimiento del sistema.

1 Ejecución y análisis de *perf list*

Ejecuta **perf list** y analiza qué tipos de eventos proporciona.

Respuesta:

perf es una herramienta de análisis de rendimiento en sistemas Linux. Sirve para identificar cuellos de botella, problemas, optimizar programas, etc. Esta herramienta detalla información de uso de la CPU, el uso de la memoria y más características que nos proporcionan una visión global del rendimiento de la máquina en la ejecución de una aplicación o programa.

Puede muestrear eventos, seguir hilos y procesos, rastrear llamadas al sistema, etc.

Con **perf list** se muestra los tipos de eventos de hardware y software que hay disponibles. Es una forma de listarlos y saber cuáels eventos de rendimiento se pueden analizar con la herramienta.

Tenemos eventos hardware, software, de caché, etc.

Para filtrar y saber cuales son los tipos de eventos podemos hacerlo con **perf list –details** y con la flag del evento que queremos filtrar como **hw** para hardware o **sw** para software, **cache** para los eventos caché, etc.

```
branch-instructions OR cpu/branch-instructions/    [Kernel PMU event]
branch-misses OR cpu/branch-misses/                [Kernel PMU event]
bus-cycles OR cpu/bus-cycles/                      [Kernel PMU event]
cache-misses OR cpu/cache-misses/                  [Kernel PMU event]
cache-references OR cpu/cache-references/          [Kernel PMU event]
cpu-cycles OR cpu/cpu-cycles/                      [Kernel PMU event]
instructions OR cpu/instructions/                  [Kernel PMU event]
mem-loads OR cpu/mem-loads/                        [Kernel PMU event]
mem-stores OR cpu/mem-stores/                      [Kernel PMU event]
ref-cycles OR cpu/ref-cycles/                      [Kernel PMU event]
topdown-fetch-bubbles OR cpu/topdown-fetch-bubbles/ [Kernel PMU event]
topdown-recovery-bubbles OR cpu/topdown-recovery-bubbles/ [Kernel PMU event]
topdown-slots-issued OR cpu/topdown-slots-issued/  [Kernel PMU event]
topdown-slots-retired OR cpu/topdown-slots-retired/ [Kernel PMU event]
topdown-total-slots OR cpu/topdown-total-slots/    [Kernel PMU event]
cstate_core/c3-residency/                          [Kernel PMU event]
cstate_core/c6-residency/                          [Kernel PMU event]
cstate_core/c7-residency/                          [Kernel PMU event]
cstate_pkg/c2-residency/                           [Kernel PMU event]
cstate_pkg/c3-residency/                           [Kernel PMU event]
cstate_pkg/c6-residency/                           [Kernel PMU event]
cstate_pkg/c7-residency/                           [Kernel PMU event]
msr/aperf/                                         [Kernel PMU event]
msr/cpu_thermal_margin/                            [Kernel PMU event]
msr/mperf/                                         [Kernel PMU event]
msr/smi/                                           [Kernel PMU event]
msr/tsc/                                           [Kernel PMU event]
power/energy-pkg/                                  [Kernel PMU event]
power/energy-ram/                                  [Kernel PMU event]
uncore_imc_0/cas_count_read/                       [Kernel PMU event]
uncore_imc_0/cas_count_write/                      [Kernel PMU event]
uncore_imc_0/clockticks/                           [Kernel PMU event]
uncore_imc_1/cas_count_read/                       [Kernel PMU event]
uncore_imc_1/cas_count_write/                      [Kernel PMU event]
uncore_imc_1/clockticks/                           [Kernel PMU event]
uncore_imc_4/cas_count_read/                       [Kernel PMU event]
uncore_imc_4/cas_count_write/                      [Kernel PMU event]
uncore_imc_4/clockticks/                           [Kernel PMU event]
```

```
List of pre-defined events (to be used in -e):

  branch-instructions OR branches                    [Hardware event]
  branch-misses                                      [Hardware event]
  bus-cycles                                         [Hardware event]
  cache-misses                                       [Hardware event]
  cache-references                                   [Hardware event]
  cpu-cycles OR cycles                               [Hardware event]
  instructions                                       [Hardware event]
  ref-cycles                                         [Hardware event]

  alignment-faults                                   [Software event]
  bpf-output                                         [Software event]
  cgroup-switches                                    [Software event]
  context-switches OR cs                             [Software event]
  cpu-clock                                          [Software event]
  cpu-migrations OR migrations                       [Software event]
  dummy                                              [Software event]
  emulation-faults                                   [Software event]
  major-faults                                       [Software event]
  minor-faults                                       [Software event]
  page-faults OR faults                              [Software event]
  task-clock                                         [Software event]

  duration_time                                      [Tool event]

  L1-dcache-load-misses                              [Hardware cache event]
  L1-dcache-loads                                    [Hardware cache event]
  L1-dcache-stores                                   [Hardware cache event]
  L1-icache-load-misses                              [Hardware cache event]
  LLC-load-misses                                    [Hardware cache event]
  LLC-loads                                          [Hardware cache event]
  LLC-store-misses                                   [Hardware cache event]
  LLC-stores                                         [Hardware cache event]
  branch-load-misses                                 [Hardware cache event]
  branch-loads                                       [Hardware cache event]
```

```
Usage: perf list [<options>] [hw|sw|cache|tracepoint|pmu|sdt|metric|metricgroup|event_glob]
```

2 perf stat

Ejecuta perf stat –e cpu-cycles,instructions ./benchmark6 y compara el tiempo proporcionado por perf con la salida del programa. ¿Hay diferencia? ¿por qué?

Respuesta:

El comando **stat** cuenta el total de eventos para un programa durante un periodo del tiempo y junto con **-e** se seleccionan los eventos que queremos contar. Se seleccionan el número de instrucciones y ciclos de CPU. Después se indica el ejecutable sobre el que se quiere aplicar el conteo.

El tiempo que queda registrado por el programa y por la herramienta **perf** es ligeramente diferente, algo mayor en el **perf stat** y esto es debido a que perf mide primero el tiempo que emplea el programa **benchmark** en ejecutarse y a este tiempo se le suma lo que tardó en hacer el conteo de los eventos que seleccionamos previamente.

```
arqcom016@aloe:~/Descargas/MultMat/Naive$ perf stat -e cpu-cycles,instructions ./benchmark
Description:    A naive C version on matrices of size 256 x 256

Time 3.010108e-02 s

 Performance counter stats for './benchmark':

       82.720.435      cpu-cycles
      134.233.219      instructions              #    1,62  insn per cycle

      0,037267249 seconds time elapsed

      0,036765000 seconds user
      0,000000000 seconds sys
```

3 Múltiples ejecuciones

Si ejecutas varias veces el programa, ¿da siempre el mismo número de eventos? Reporta los eventos, el tiempo que reporta perf, el tiempo que mide el programa y justifica la respuesta.

Respuesta:

Estas son las salidas que me ha mostrado al ejecución de **perf stat** mostrando los eventos de instrucciones y ciclos por CPU.

```
arqcom016@aloe:~/Descargas/MultMat/Naive$ ./benchmark
Description:    A naive C version on matrices of size 256 x 256

Time 1.011386e-01 s
arqcom016@aloe:~/Descargas/MultMat/Naive$ perf stat -e cpu-cycles,instructions ./benchmark
Description:    A naive C version on matrices of size 256 x 256

Time 1.089770e-01 s

 Performance counter stats for './benchmark':

      282.145.072      cpu-cycles
      407.427.486      instructions              #    1,44  insn per cycle

      0,117302976 seconds time elapsed

      0,116421000 seconds user
      0,000000000 seconds sys
```

```
arqcom016@aloe:~/Descargas/MultMat/Naive$ perf stat -e cpu-cycles,instructions ./benchmark
Description:    A naive C version on matrices of size 256 x 256

Time 1.099549e-01 s

 Performance counter stats for './benchmark':

      285.878.036      cpu-cycles
      407.395.368      instructions              #    1,43  insn per cycle

      0,118225222 seconds time elapsed

      0,117392000 seconds user
      0,000000000 seconds sys
```

```
arqcom016@aloe:~/Descargas/MultMat/Naive$ perf stat -e cpu-cycles,instructions ./benchmark
Description:    A naive C version on matrices of size 256 x 256

Time 9.606395e-02 s

 Performance counter stats for './benchmark':

      354.929.870      cpu-cycles
      407.440.574      instructions              #    1,15  insn per cycle

      0,101971467 seconds time elapsed

      0,101106000 seconds user
      0,000000000 seconds sys
```

```
arqcom016@aloe:~/Descargas/MultMat/Naive$ perf stat -e cpu-cycles,instructions ./benchmark
Description:    A naive C version on matrices of size 256 x 256

Time 9.434225e-02 s

 Performance counter stats for './benchmark':

      354.832.562      cpu-cycles
      407.429.973      instructions              #    1,15  insn per cycle

      0,100383808 seconds time elapsed

      0,099612000 seconds user
      0,000000000 seconds sys
```

Como se puede ver los resultados de tiempo son distintos debido a que el número de instrucciones en cada ejecución es el mismo y por ende también cambia el número de ciclos de CPU, lo que cambia el tiempo total.

Las variaciones de estas ejecuciones puede darse porque la CPU está ejecutando otros procesos, esto provoca cambios de contextos, por lo que las instrucciones varían de un caso a otro. También puede verse debido a fallos de acceso a la caché.

4 Análisis del archivo 'Makefile'

Utiliza ahora el comando **perf record** (en lugar de **stat**) y examina la salida de una ejecución usando **perf report**. Utiliza la opción **perf annotate** para relacionar los eventos con el código. Explica brevemente qué hace cada comando, ilustrándolo con tu código. ¿Qué instrucción es la que tiene mayor impacto en los eventos examinados?

Respuesta:

Comandos de perf en Linux para el Análisis de Rendimiento:

- **perf record -e cpu-cyles,instructions ./benchmark**: Este comando se usa para registrar eventos de rendimiento mientras se ejecuta un programa o comando, en este caso, el ejecutable "benchmark" y hace recuento de número de instrucciones y número de ciclos de CPU. "perf record" captura información sobre eventos de hardware y software, como ciclos de CPU, llamadas al sistema, fallos de caché, etc., que ocurren durante la ejecución del programa. Los datos recopilados

se almacenan en un archivo que luego se utiliza para el análisis.

- **perf report**: Después de haber ejecutado "perf record", "perf report" se utiliza para analizar y visualizar los datos de rendimiento recopilados. Proporciona un resumen detallado de los eventos registrados, mostrando estadísticas y gráficos que te permiten identificar cuellos de botella y áreas de mejora en la ejecución del programa. Puedes ver qué partes del código consumen más recursos y qué eventos son más relevantes.

- **annotate** (utilizado dentro de "perf report"): "annotate" es una función dentro de "perf report" que te permite ver el código fuente del programa anotado con información de rendimiento. Puedes identificar las líneas específicas de código que están asociadas con los eventos de rendimiento, lo que facilita la optimización. "annotate" muestra qué líneas de código contribuyen a un mayor consumo de recursos, como ciclos de CPU o llamadas al sistema, lo que te ayuda a enfocarte en las áreas críticas para la mejora de rendimiento.

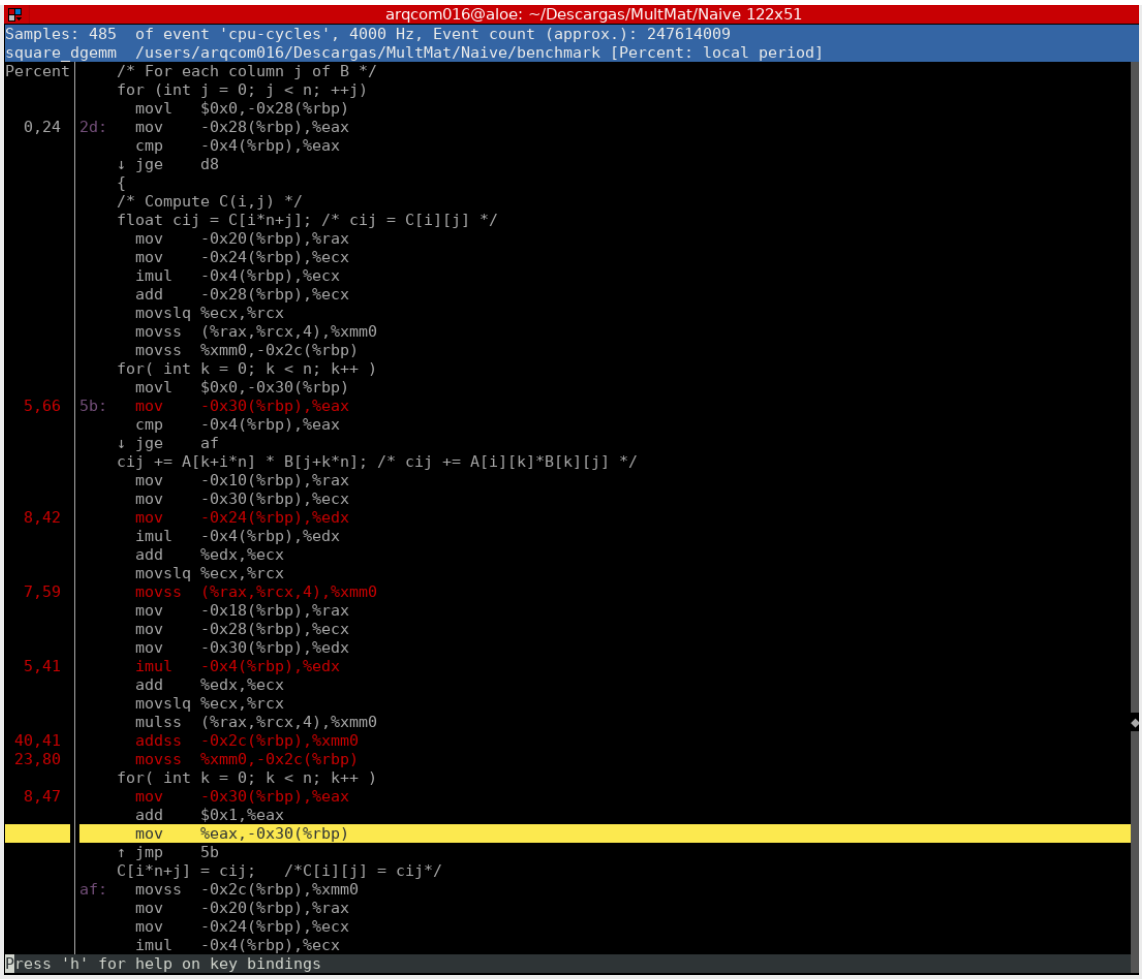
  Aquí muestro las llamadas a los comandos:

```
arqcom016@aloe:~/Descargas/MultMat/Naive$ perf record -e cpu-cycles,instructions ./benchmark
Description:    A naive C version on matrices of size 256 x 256

Time 1.128297e-01 s
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0,063 MB perf.data (972 samples) ]
arqcom016@aloe:~/Descargas/MultMat/Naive$ perf report -i perf.data
arqcom016@aloe:~/Descargas/MultMat/Naive$ perf annotate -i perf.data
arqcom016@aloe:~/Descargas/MultMat/Naive$
```

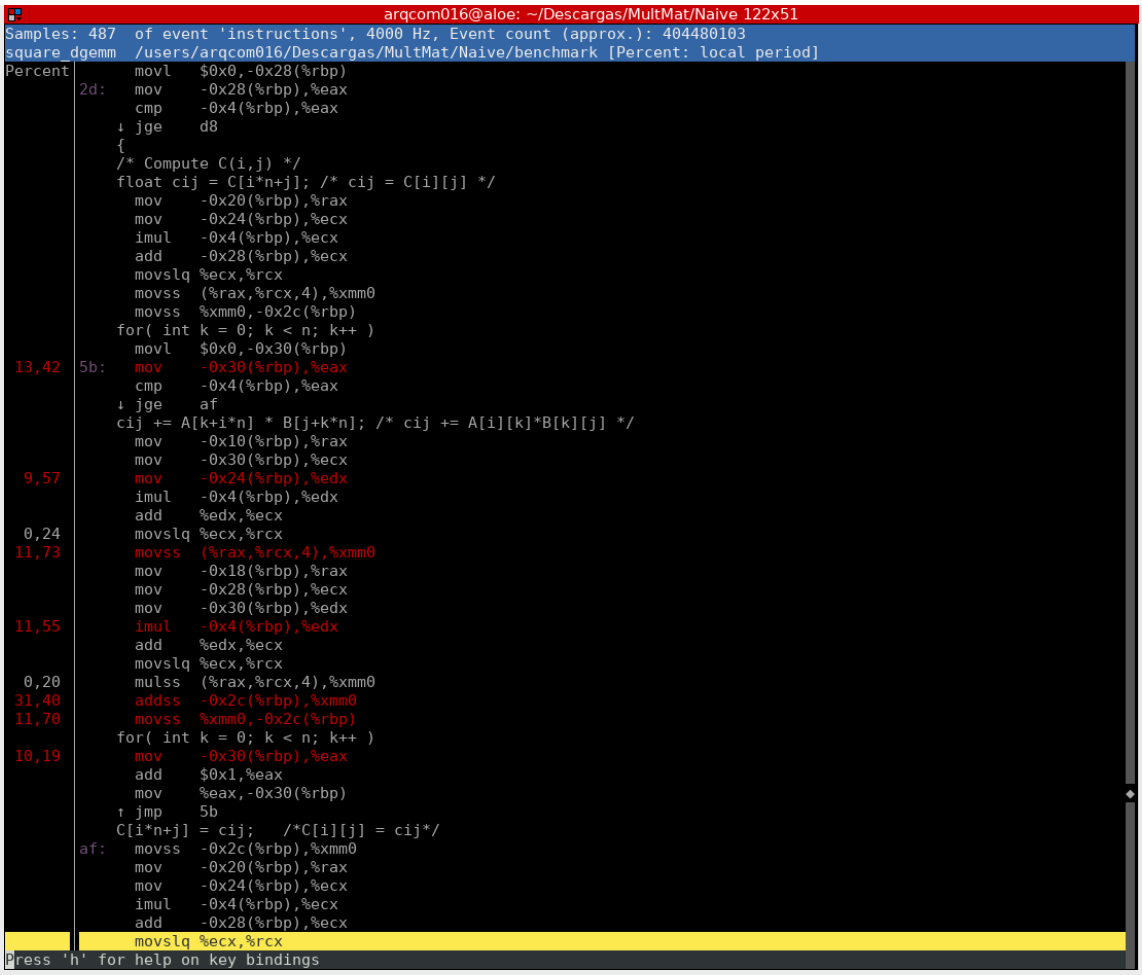
  Y estas son las salidas que me dan para el ejecutable **benchmark** sin ninguna optimización en la compilación.

  El reporte:

  Y las instrucciones ejecutadas con información de aquellas con más impacto y las que menos. En la siguiente imagen se muestran las instrucciones con más impacto en cuanto a ciclos de cpu en porcentaje.



Y en la siguiente imagen se muestran las instrucciones con más impacto en cuanto a llamadas. La que más llamadas ha tenido ha sido **adds -0x2c(%rbp), %xmm0** con un 31,40% de las 404.480.103 instrucciones ejecutadas.



5 Optimización

Cambia el valor de optimización del compilador, controlado con –**O0** en **OPT**, y vuelve a obtener el número de eventos y el tiempo que mide el programa para cada opción **(-O1,-O2,-O3)**. Rellena una tabla con los resultados obtenidos y discútelos, analizando qué cambios introduce cada opción de compilación, cómo afectan a los eventos hardware y qué impacto tienen en los tiempos. Puedes usar también **record** y **annotate** para tener pistas sobre los cambios que han tenido más impacto en cada ejecutable.

Respuesta:

Estos son los valores obtenidos de la compilación de **benchmark** con distintas flags de optimización:

Optimización	Tiempo de ejecución	Instrucciones	Ciclos de CPU
-O0	0,08	407.397.931	265.894.334
-O1	0,03	136.295.230	89.474.435
-O2	0,004	48.936.234	30.446.090
-O3	0,005	48.985.058	27.410.573

Se puede observar con los datos calculados que hay una gran diferencia entre no optimizar nada el código y las otras tres opciones. La opción con más cabida sería la flag **-O2** ya que la diferencia entre ésta y la flag **-O3** en mi caso es redundante y lo más seguro es que con otros compiladores y estructuras la mejora sera ínfima.

- **-O0**: se muestrearon 519 ciclos de CPU y 520 instrucciones. La instrucción **addss -0x2c(%rbp), %xmm0** fue la que tuvo más impacto.
- **-O1**: de los 188 ciclos de CPU muestreados me proporciona una aproximación de 81038487 ciclos donde la instrucción **inc %r15** se lleva el 51% de ellos y el 59% de las instrucciones ejecutadas.
- **-O2**: se muestrearon 44 ciclos de CPU y 47 instrucciones. Esta vez fue la instrucción **add $0x4, %r12** la que se lleva el 36,38% de ciclos y un 45% de ejecuciones.
- **-O3**: se muestrearon 47 ciclos de CPU y 51 instrucciones, quedandose por debajo de la optimización **-O2**. Volvió a ser más impactante la instrucción **add $0x4, %r12** con 42,84% de ciclos y 38,28% de ejecuciones, pero también se ejecutó considerablemente otras funciones, entre la que se encuentra **mulps %xmm2, %xmm3**.

6 Modelo Roofline

El modelo Roofline visto en clase permite describir el rendimiento de un procesador de forma un poco más precisa. Usa la herramienta Intel Advisor8 (advixe-gui) para calcular la gráfica del modelo Roofline y situar nuestra rutina principal dentro de esa gráfica. Explica la información que proporciona, cuál es el cuello de botella y posibles opciones de optimización para mejorar el rendimiento

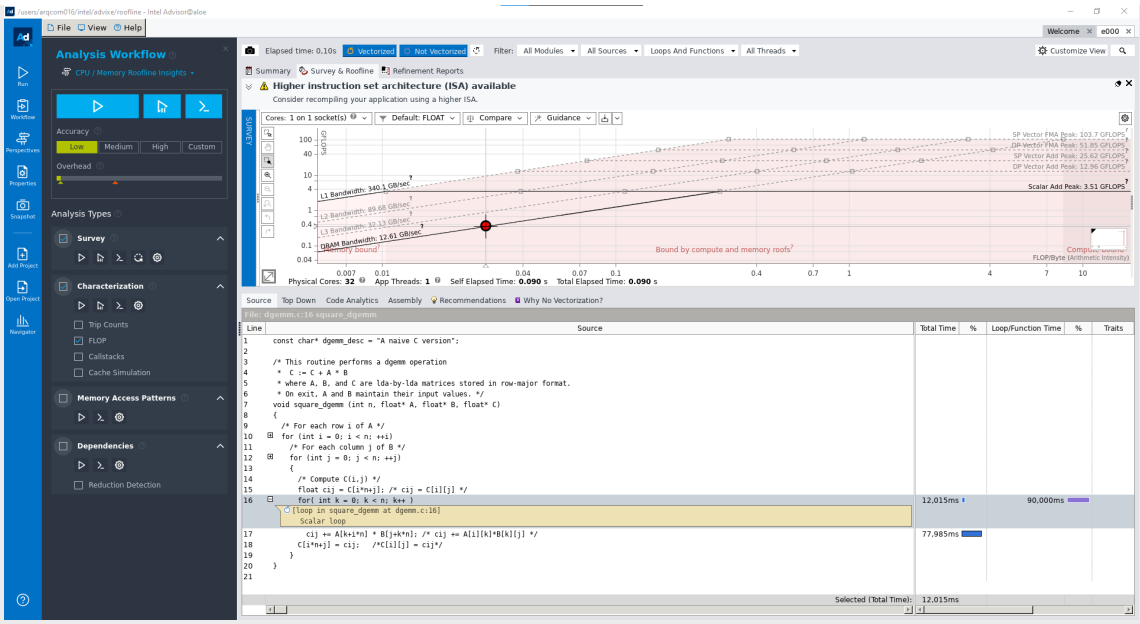
Repuesta:

El modelo Roofline es una herramienta gráfica permite el análisis de rendimiento sobre una aplicación. Evalua y representa de manera gráfica el rendimiento de las aplicaciones en relación con las capacidades y limitaciones del hardware y el sistema donde se ejecutan.

El concepto fundamental detrás del modelo Roofline es la representación visual del rendimiento en función de la intensidad operativa, expresada en unidades como GFLOPS/s o GB/s. A través de este enfoque, es posible comprender de manera efectiva cómo se utiliza el hardware en relación con su capacidad máxima.

El modelo Roofline proporciona una representación gráfica clara que facilita la identificación de cuellos de botella en las aplicaciones, la determinación de áreas de mejora de rendimiento y la comprensión de la eficiencia en el uso de los recursos del hardware. En última instancia, brinda a desarrolladores y científicos la información necesaria para tomar decisiones informadas sobre cómo maximizar el rendimiento de sus aplicaciones.

Aquí podemos ver el modelo Roofline conseguido con la herramienta *Intel Advisor* para el ejecutable **benchmark** compilado con el compilador de Intel **icx** y sin flags de optimización:



El punto rojo indica dónde se encuentra **benchmark**. Se encuentra bastante por debajo del **Scalar Add Peak**. El **Scalar Add Peak** es la capacidad de realizar operaciones de adición escalar a la máxima velocidad teórica que el hardware puede alcanzar, por lo que delimita un techo de rendimiento máximo teórico que no se puede sobrepasar. Benchmark registra 0,89 GFLOPS, mientras que el **SAP** está en 3,51 GFLOPS. Esta diferencia se puede deber a que la compilación no fue optimizada.

7 Cálculo del Modelo Roofline

Explica cómo se calcula la gráfica Roofline para el procesador de referencia en estas prácticas. Justifica los valores obtenidos para cada asíntota de la gráfica y compáralos con los valores que proporciona Intel Advisor. Puedes consultar información sobre la arquitectura del procesador y su jerarquía de memoria con los comandos: "more /proc/cpuinfo"; "lscpu"; "lstopo". Puedes usar la herramienta likwid-bench9 y los benchmarks copy, copy_mem o stream para medir los distintos anchos de banda de la jerarquía de memoria y estimar las asíntotas.

Repuesta:

Para calcular el modelo Roofline necesitaremos una serie de detalles acerca de las características de nuestra máquina. Con esta información del sistema podremos hallar la asíntota superior (Peak Performance), y la asíntota inferior (Memory Bound). Hay que tener en cuenta que estos datos son aproximadso en base a los cálculos que se hacen dependiendo de la precisión de los datos de la arquitectura y a que el rendimiento real de las aplicaciones puede verse afectado por diversos factores.

```
arqcom016@aloe:~/Descargas/MultMat$ likwid-bench -t copy_mem -w S0:1GB
Warning: Sanitizing vector length to a multiple of the loop stride 4 and thread count 16 from 62500000 elements (500000000 bytes) to 62499968 elements (499999744 bytes)
Allocate: Process running on hwthread 0 (Domain S0) - Vector length 62499968/499999744 Offset 0 Alignment 512
Allocate: Process running on hwthread 0 (Domain S0) - Vector length 62499968/499999744 Offset 0 Alignment 512
Initialization: First thread in domain initializes the whole stream
--------------------------------------------------------------------------------
LIKWID MICRO BENCHMARK
Test: copy_mem
--------------------------------------------------------------------------------
Using 1 work groups
Using 16 threads
--------------------------------------------------------------------------------
Running without Marker API. Activate Marker API with -m on commandline.
--------------------------------------------------------------------------------
Group: 0 Thread 6 Global Thread 6 running on hwthread 6 - Vector length 3906248 Offset 23437488
Group: 0 Thread 7 Global Thread 7 running on hwthread 7 - Vector length 3906248 Offset 27343736
Group: 0 Thread 8 Global Thread 8 running on hwthread 8 - Vector length 3906248 Offset 31249984
Group: 0 Thread 10 Global Thread 10 running on hwthread 10 - Vector length 3906248 Offset 39062480
Group: 0 Thread 11 Global Thread 11 running on hwthread 11 - Vector length 3906248 Offset 42968728
Group: 0 Thread 12 Global Thread 12 running on hwthread 12 - Vector length 3906248 Offset 46874976
Group: 0 Thread 13 Global Thread 13 running on hwthread 13 - Vector length 3906248 Offset 50781224
Group: 0 Thread 14 Global Thread 14 running on hwthread 14 - Vector length 3906248 Offset 54687472
Group: 0 Thread 3 Global Thread 3 running on hwthread 3 - Vector length 3906248 Offset 11718744
Group: 0 Thread 15 Global Thread 15 running on hwthread 15 - Vector length 3906248 Offset 58593720
Group: 0 Thread 2 Global Thread 2 running on hwthread 2 - Vector length 3906248 Offset 7812496
Group: 0 Thread 0 Global Thread 0 running on hwthread 0 - Vector length 3906248 Offset 0
Group: 0 Thread 1 Global Thread 1 running on hwthread 1 - Vector length 3906248 Offset 3906248
Group: 0 Thread 5 Global Thread 5 running on hwthread 5 - Vector length 3906248 Offset 19531240
Group: 0 Thread 9 Global Thread 9 running on hwthread 9 - Vector length 3906248 Offset 35156232
Group: 0 Thread 4 Global Thread 4 running on hwthread 4 - Vector length 3906248 Offset 15624992
--------------------------------------------------------------------------------
Cycles:                25430311417
CPU Clock:             2299991923
Cycle Clock:           2299991923
Time:                  1.105670e+01 sec
Iterations:            8192
Iterations per thread: 512
Inner loop executions: 976562
Size (Byte):           999999488
Size per thread:       62499968
Number of Flops:       0
MFlops/s:              0.00
Data volume (Byte):    511999737856
MByte/s:               46306.76
Cycles per update:     0.794698
Cycles per cacheline:  6.357581
Loads per update:      1
Stores per update:     1
Load bytes per element: 8
Store bytes per elem.: 8
Load/store ratio:      1.00
Instructions:          87999954960
UOPs:                  111999942656
```

Con el uso de **lstop** y la visualización del fichero **/proc/cpuinfo** se consigue información que se necesita para hacer un estimado de la asíntota superior.

$$ScalarAddPeak = Frecuenciadereloj(GHz) * Nnúcleos * OperacionesporFLOP * Ndeciclos/s$$

El número de ciclos por segundo equivale a la frecuencia de reloj y las operaciones por ciclo o $FLOPS = Frecuencia * Nciclos * FLOPS/ciclo$. Como los **FLOPS / ciclo** son 1 en la **Intel Xeon CPU E5-2698 v3**, y tenemos la frecuencia de reloj y el número de núcleos, podemos hallar la asíntota superior (**GFLOPS**).

$$ScalarAddPeak = 2,3 * 16 * 2,3 * 16 * 1 * 2,3 = \textbf{3.114,752} = \textbf{3,114 GFLOPS}$$

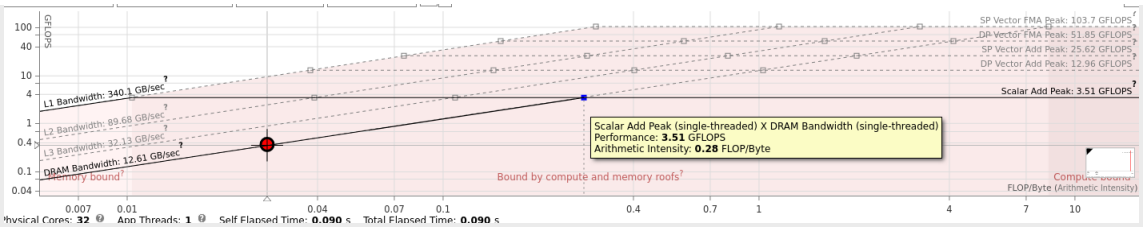
Este cálculo dista un poco de los 3,51 GFLOPS que nos proporciona la herramienta **Intel Advisor**, pero es un aproximado.

La asíntota inferior corresponde al ancho de banda del sistema. Esto se refiere a la velocidad a la que se pueden transferir datos entre la CPU y la memoria principal (RAM). Obtengo esta marca con la herramienta benchmarking likwid-bench con el benchmark stream, copy, copy_mem.

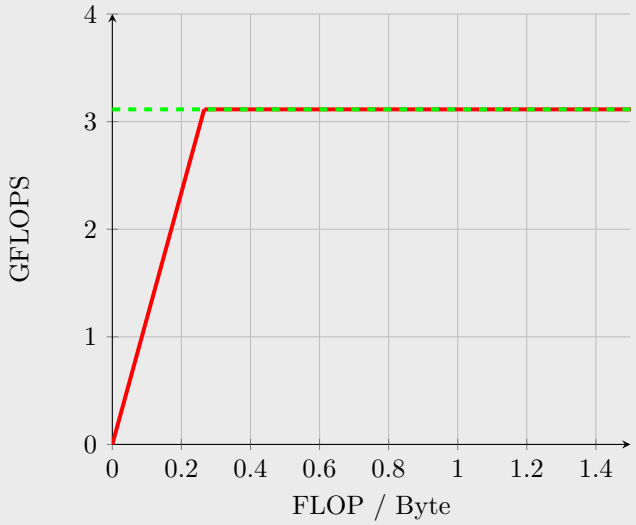
Los valores que me dieron son estos:

stream	copy	copy_mem
32,3 GB/s	34,1 GB/s	46,3 GB/s

Y con estos valores se saca de media **37,56 GB/s**, lo que se traduce a $FLOP/byte = FLOPs/(AnchodebandaenGB/s) = 1FLOPs/37,56666GB/s = 0.2661934...FLOP/byte$. Esta marca si se asemeja al valor dado por **Intel Advisor**, la cual nos daba **0,28 FLOP / byte**.



Y ahora sí con estas dos asíntotas podemos recrear el modelo Roofline aproximado.



8 Intensidad operacional

Para situar la multiplicación de matrices dentro de la gráfica anterior necesitas calcular la intensidad operacional, la cual depende del número de bytes transferidos entre la memoria principal y la cache. Revisa la lista de eventos del primer ejercicio y selecciona el más adecuado para calcular la intensidad operacional. Calcula también el rendimiento del programa y sitúalo en la gráfica anterior y compáralo con el que proporciona Intel Advisor.

Repuesta:

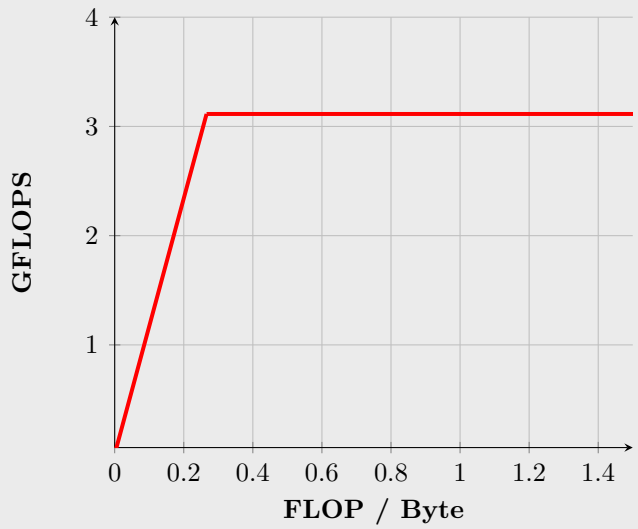
Como indica el enunciado para calcular la intensidad operacional a partir de la lista de eventos necesito elegir los eventos que me indiquen el número de bytes tarnsferidos entre la memoria principal y la caché, que son las veces que la memoria de último nivel ha fallado para encontrar los datos y accede al siguiente nivel que es la memoria principal (**DRAM**). $\text{longest}_l at_c ache.miss son los eventos que reflejan esto hecho, los fallos d$

Con **perf stat -e longest**$_l at_c ache.miss./benchmark consigo los datos necesarios para hallar los FLOPS/byte que indican$

```
l2_trans.l1d_wb
     [L1D writebacks that access L2 cache]
l2_trans.l2_fill
     [L2 fill requests that access L2 cache]
l2_trans.l2_wb
     [L2 writebacks that access L2 cache]
l2_trans.rfo
     [RFO requests that access L2 cache]
lock_cycles.cache_lock_duration
     [Cycles when L1D is locked]
longest_lat_cache.miss
     [Core-originated cacheable demand requests missed L3]
longest_lat_cache.reference
     [Core-originated cacheable demand requests that refer to L3]
mem_load_uops_l3_hit_retired.xsnp_hit
     [Retired load uops which data sources were L3 and cross-core snoop hits in on-pkg core cache Supports address when precise. Spec update: HSD29, HSD25, HSM26, HSM30 (Precise event)]
mem_load_uops_l3_hit_retired.xsnp_hitm
     [Retired load uops which data sources were HitM responses from shared L3 Supports address when precise. Spec update: HSD29, HSD25, HSM26, HSM30 (Precise event)]
mem_load_uops_l3_hit_retired.xsnp_miss
     [Retired load uops which data sources were L3 hit and cross-core snoop missed in on-pkg core cache Supports address when precise. Spec update: HSD29, HSD25, HSM26, HSM30 (Precise event)]
```

```
arqcom016@aloe:~/Descargas/MultMat/Naive$ perf stat -e longest_lat_cache.miss ./benchmark
Description:    A naive C version on matrices of size 256 x 256

Time 8.107127e-02 s

 Performance counter stats for './benchmark':

             1.645      longest_lat_cache.miss

       0,086672959 seconds time elapsed

       0,086217000 seconds user
       0,000000000 seconds sys
```

Y calculando el valor con la ecuación $(2 * tam\_matriz^2 + 2 * tam\_matriz^2)/(tamañobloque * vecesdelevento)$**,** $(2 * 256^2 + 2 * 256^2)/(64 * (1884 + 1296) = 1,288 FLOP/Byte$



No conseguí dar con un valor que cuadre con mi modelo.

9 Bibliografía y enlaces

- https://godbolt.org/
- https://es.stackoverflow.com/
- https://es.overleaf.com/learn/latex/Tutorials
- https://man7.org/linux/man-pages/man1/perf-report.1.html
- https://www.intel.com/