PERF

Al usar perf, utilizamos un evento llamado *"fp_arith_inst_retired.256b_packed_double"* y cuya descripción nos dice *"Number of SSE/AVX computational 256-bit packed double precision floating-point instructions retired. Each count represents 4 computations. Applies to SSE* and AVX* packed double precision floating-point instructions: ADD SUB MUL DIV MIN MAX SQRT DPP FM(N)ADD/SUB. DPP and FM(N)ADD/SUB instructions count twice as they perform multiple calculations per element"*

Es decir, cada medida del evento *fp_arith_inst_retired.256b_packed_double* indica que se han realizado cuatro instrucciones de punto flotante (pueden ser sumas, restas, multiplicaciones, etc.).

En la siguiente imagen se han medido las ejecuciones de dos programas: *mat_mul.py* y *mat_mul_zeros.py*.

```
jlpadillas01@PH315-51:~/TFG$ make perf
sudo sysctl -w kernel.nmi_watchdog=0
kernel.nmi_watchdog = 0
sudo /usr/bin/perf stat --event instructions,cycles,fp_arith_inst_retired.128b_packed_double,fp_arith_inst_retired.128b_packed_single,fp_arit
ngle,fp_arith_inst_retired.scalar_double,fp_arith_inst_retired.scalar_single,fp_assist.any --cpu=0 taskset -c 0 python3 src/mat_mul.py

 Performance counter stats for 'CPU(s) 0':

    60.174.510.301      instructions              #    3,48  insn per cycle           (55,37%)
    17.307.943.403      cycles                                                        (66,53%)
                 6      fp_arith_inst_retired.128b_packed_double                                 (66,63%)
                 0      fp_arith_inst_retired.128b_packed_single                                 (66,72%)
    62.560.912.768      fp_arith_inst_retired.256b_packed_double                                 (66,80%)
                 0      fp_arith_inst_retired.256b_packed_single                                 (66,80%)
         5.203.334      fp_arith_inst_retired.scalar_double                            (44,44%)
         2.659.916      fp_arith_inst_retired.scalar_single                            (44,35%)
                 0      fp_assist.any                                                  (44,27%)

       4,373287950 seconds time elapsed

sudo /usr/bin/perf stat --event instructions,cycles,fp_arith_inst_retired.128b_packed_double,fp_arith_inst_retired.128b_packed_single,fp_arit
ngle,fp_arith_inst_retired.scalar_double,fp_arith_inst_retired.scalar_single,fp_assist.any --cpu=0 taskset -c 0 python3 src/mat_mul_zeros.py

 Performance counter stats for 'CPU(s) 0':

    60.289.906.579      instructions              #    3,46  insn per cycle           (55,52%)
    17.416.990.640      cycles                                                        (66,64%)
                 3      fp_arith_inst_retired.128b_packed_double                                 (66,64%)
                 0      fp_arith_inst_retired.128b_packed_single                                 (66,64%)
    62.620.400.189      fp_arith_inst_retired.256b_packed_double                                 (66,64%)
                 0      fp_arith_inst_retired.256b_packed_single                                 (66,64%)
         4.756.251      fp_arith_inst_retired.scalar_double                            (44,48%)
         2.541.507      fp_arith_inst_retired.scalar_single                            (44,48%)
                 0      fp_assist.any                                                  (44,48%)

       4,388223459 seconds time elapsed

sudo sysctl -w kernel.nmi_watchdog=1
kernel.nmi_watchdog = 1
jlpadillas01@PH315-51:~/TFG$
```

En ambos programas se hace uso de la librería *Numpy* para *Python*; en concreto, se usa la función *empty()* que permite generar matrices con contenido aleatorio, y, la otra función *matmul()* que permite multiplicar matrices.

Así, el programa *mat_mul.py* genera dos matrices cuadradas de orden 5.000 y las multiplica consiguiendo un número de instrucciones (estimados por *perf*) de unos 60,17E9; y, unos 62,56E9 eventos (medidos) de *fp_arith_inst_retired.256b_packed_double.*

Por tanto, tenemos que

$$\textbf{\textit{numOperaciones}}\left(\textbf{\textit{perf}}\right) = 62{,}56E9 \cdot 4 = \mathbf{250{,}24E9}$$

el número de operaciones medidos durante la ejecución del programa es de unos 250,24E9.

Veamos un ejemplo de una multiplicación de dos matrices de orden 3:

$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} \cdot \begin{bmatrix} 8 & 7 & 6 \\ 5 & 4 & 3 \\ 2 & 1 & 0 \end{bmatrix} = \begin{bmatrix} (0\cdot 8+1\cdot 5+2\cdot 2) & (0\cdot 7+1\cdot 4+2\cdot 1) & (0\cdot 6+1\cdot 3+2\cdot 0) \\ (3\cdot 8+4\cdot 5+5\cdot 2) & (3\cdot 7+4\cdot 4+5\cdot 1) & (3\cdot 6+4\cdot 3+5\cdot 0) \\ (6\cdot 8+7\cdot 5+8\cdot 2) & (6\cdot 7+7\cdot 4+8\cdot 1) & (6\cdot 6+7\cdot 3+8\cdot 0) \end{bmatrix} = \begin{bmatrix} 9 & 6 & 3 \\ 54 & 1 & 2 \\ 99 & 78 & 57 \end{bmatrix}$$

Por cada elemento, podemos ver que se hacen 3 multiplicaciones y 2 sumas. Teniendo unos 3^2 elementos, tenemos $3^2(3+2)=45$ operaciones.

Si lo extrapolamos para una matriz de tamaño N, tenemos que el número de operaciones es igual a

$$numOperaciones = N^2(2N-1)$$

Aplicamos dicha fórmula para nuestra matriz de orden 5.000:

$$\boldsymbol{numOperaciones}\left(\boldsymbol{N=5.000}\right)=5.000\cdot 5.000\left(9.999\right)=249.975.000.000=\mathbf{249,98E9}$$

Por tanto, con todo ello conseguimos que lo medido con *perf* se asemeja a lo que esperamos obtener

$$numOperaciones\left(perf\right)=250,24E9\approx numOperaciones\left(N=5.000\right)=249,98E9$$

Si bien, lo medido supera en unas **268,65E6** operaciones, ello se puede achacar a las preparaciones anteriores y posteriores a la multiplicación de la matriz. Así como el tener otros programas/procesos en ejecución. Se puede refinar más las medidas si se hacen en modo CLI (para más adelante quizás se haga).

La creación del programa *mat_mul_zeros.py* fue con la intención de despejar la duda de si *Python* aplicaba algún tipo de optimización (como se puede hacer en C mediante el -O0, -O1, etc.) y evitar hacer las multiplicaciones si ve que uno de sus operandos tiene un cero. A la vista salta, por los resultados obtenidos, que no se aplican dichas optimizaciones.

TOPLEV

Con toplev.py no puedo obtener el número exacto de operaciones que se han realizado. Sólo porcentajes:

```
jlpadillas01@PH315-51:~/TFG$ make tlev
sudo sysctl -w kernel.nmi_watchdog=0
kernel.nmi_watchdog = 0
sudo /home/jlpadillas01/pmu-tools/toplev.py --core C0 -l4 --no-desc taskset -c 0 python3 src/mat_mul.py
Will measure complete system.
Using level 4.
# 4.11-full-perf on Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz [cfl/skylake]
C0    RET            Retiring                  % Slots                    86.6
C0    RET            Retiring.Light_Operations  % Slots                   86.3
C0-T0 RET            Retiring.Light_Operations.FP_Arith           % Uops                   103.7
C0-T0 RET            Retiring.Light_Operations.FP_Arith.FP_Vector  % Uops                   103.7  <==
C0-T0 MUX                                                         %                        2.9
C0-T1 MUX                                                         %                        2.9
Mismeasured (out of bound values): FP_Vector MEM_Latency
Run toplev --describe FP_Vector^ to get more information on bottleneck
sudo /home/jlpadillas01/pmu-tools/toplev.py --core C0 -l4 --no-desc taskset -c 0 python3 src/mat_mul_zeros.py
Will measure complete system.
Using level 4.
# 4.11-full-perf on Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz [cfl/skylake]
C0    RET            Retiring                  % Slots                    85.8
C0    RET            Retiring.Light_Operations  % Slots                   85.5
C0-T0 RET            Retiring.Light_Operations.FP_Arith           % Uops                   104.2
C0-T0 RET            Retiring.Light_Operations.FP_Arith.FP_Vector  % Uops                   104.2  <==
C0-T0 MUX                                                         %                        2.9
C0-T1 MUX                                                         %                        2.9
Mismeasured (out of bound values): FP_Vector MEM_Latency
Run toplev --describe FP_Vector^ to get more information on bottleneck
sudo sysctl -w kernel.nmi_watchdog=1
kernel.nmi_watchdog = 1
jlpadillas01@PH315-51:~/TFG$
```

A no ser que añada el comando “--raw”, que nos muestra todos los datos obtenidos:

```
jlpadillas01@PH315-51: ~/TFG
raw 0 event cpu/event=0xb1,umask=0x10/ val 70402.0 ename uops_executed.x87 index 74 group 15 nodes FP_Arith
raw 6 event cpu/event=0xb1,umask=0x10/ val 46175.0 ename uops_executed.x87 index 74 group 15 nodes FP_Arith
raw 0 event cpu/event=0xb1,umask=0x1/ val 60147687904.0 ename uops_executed.thread index 75 group 15 nodes FP_Arith
raw 6 event cpu/event=0xb1,umask=0x1/ val 42882130.0 ename uops_executed.thread index 75 group 15 nodes FP_Arith
raw 0 event cpu/event=0xc7,umask=0x2/ val 2380925.0 ename fp_arith_inst_retired.scalar_single index 76 group 15 nodes FP_Arith
raw 6 event cpu/event=0xc7,umask=0x2/ val 0.0 ename fp_arith_inst_retired.scalar_single index 76 group 15 nodes FP_Arith
raw 0 event cpu/event=0xc7,umask=0x1/ val 4774662.0 ename fp_arith_inst_retired.scalar_double index 77 group 15 nodes FP_Arith
raw 6 event cpu/event=0xc7,umask=0x1/ val 5302.0 ename fp_arith_inst_retired.scalar_double index 77 group 15 nodes FP_Arith
raw 0 event cpu/event=0xc2,umask=0x2/ val 59598107703.0 ename uops_retired.retire_slots index 78 group 16 nodes FP_Arith
raw 6 event cpu/event=0xc2,umask=0x2/ val 41903247.0 ename uops_retired.retire_slots index 78 group 16 nodes FP_Arith
raw 0 event cpu/event=0xc7,umask=0x4/ val 0.0 ename fp_arith_inst_retired.128b_packed_double index 79 group 16 nodes FP_Arith
raw 6 event cpu/event=0xc7,umask=0x4/ val 0.0 ename fp_arith_inst_retired.128b_packed_double index 79 group 16 nodes FP_Arith
raw 0 event cpu/event=0xc7,umask=0x8/ val 0.0 ename fp_arith_inst_retired.128b_packed_single index 80 group 16 nodes FP_Arith
raw 6 event cpu/event=0xc7,umask=0x8/ val 139864.0 ename fp_arith_inst_retired.128b_packed_single index 80 group 16 nodes FP_Arith
raw 0 event cpu/event=0xc7,umask=0x10/ val 62215195395.0 ename fp_arith_inst_retired.256b_packed_double index 81 group 16 nodes FP_Arith
raw 6 event cpu/event=0xc7,umask=0x10/ val 0.0 ename fp_arith_inst_retired.256b_packed_double index 81 group 16 nodes FP_Arith
raw 0 event cpu/event=0xb1,umask=0x10/ val 53276.0 ename uops_executed.x87 index 82 group 17 nodes Other
raw 6 event cpu/event=0xb1,umask=0x10/ val 39525.0 ename uops_executed.x87 index 82 group 17 nodes Other
raw 0 event cpu/event=0xb1,umask=0x1/ val 60195262873.0 ename uops_executed.thread index 83 group 17 nodes Other
raw 6 event cpu/event=0xb1,umask=0x1/ val 31666814.0 ename uops_executed.thread index 83 group 17 nodes Other
raw 0 event cpu/event=0xc7,umask=0x2/ val 1985127.0 ename fp_arith_inst_retired.scalar_single index 84 group 17 nodes Other
raw 6 event cpu/event=0xc7,umask=0x2/ val 0.0 ename fp_arith_inst_retired.scalar_single index 84 group 17 nodes Other
raw 0 event cpu/event=0xc7,umask=0x1/ val 3975907.0 ename fp_arith_inst_retired.scalar_double index 85 group 17 nodes Other
raw 6 event cpu/event=0xc7,umask=0x1/ val 2791.0 ename fp_arith_inst_retired.scalar_double index 85 group 17 nodes Other
raw 0 event cpu/event=0xc2,umask=0x2/ val 59749179303.0 ename uops_retired.retire_slots index 86 group 18 nodes Other Ports_Utilization
raw 6 event cpu/event=0xc2,umask=0x2/ val 32911611.0 ename uops_retired.retire_slots index 86 group 18 nodes Other Ports_Utilization
raw 0 event cpu/event=0xc7,umask=0x4/ val 0.0 ename fp_arith_inst_retired.128b_packed_double index 87 group 18 nodes Other
raw 6 event cpu/event=0xc7,umask=0x4/ val 0.0 ename fp_arith_inst_retired.128b_packed_double index 87 group 18 nodes Other
raw 0 event cpu/event=0xc7,umask=0x8/ val 0.0 ename fp_arith_inst_retired.128b_packed_single index 88 group 18 nodes Other
raw 6 event cpu/event=0xc7,umask=0x8/ val 93161.0 ename fp_arith_inst_retired.128b_packed_single index 88 group 18 nodes Other
raw 0 event cpu/event=0xc7,umask=0x10/ val 62272994316.0 ename fp_arith_inst_retired.256b_packed_double index 89 group 18 nodes Other
raw 6 event cpu/event=0xc7,umask=0x10/ val 0.0 ename fp_arith_inst_retired.256b_packed_double index 89 group 18 nodes Other
raw 0 event cpu/event=0x3c,umask=0x0,any=1/ val 17395266700.0 ename cpu_clk_unhalted.thread_any index 90 group 18 nodes Ports_Utilization
raw 6 event cpu/event=0x3c,umask=0x0,any=1/ val 17392063924.0 ename cpu_clk_unhalted.thread_any index 90 group 18 nodes Ports_Utilization
raw 0 event cpu/event=0x14,umask=0x1,cmask=1/ val 243668.0 ename arith.divider_active index 91 group 19 nodes Ports_Utilization
raw 6 event cpu/event=0x14,umask=0x1,cmask=1/ val 404913.0 ename arith.divider_active index 91 group 19 nodes Ports_Utilization
raw 0 event cpu/event=0xa6,umask=0x1/ val 27051802.0 ename exe_activity.exe_bound_0_ports index 92 group 19 nodes Ports_Utilization
raw 6 event cpu/event=0xa6,umask=0x1/ val 7881992.0 ename exe_activity.exe_bound_0_ports index 92 group 19 nodes Ports_Utilization
```

Podemos ver que el valor obtenido es similar al que conseguimos con perf: unos 62,23E9 eventos.

PAPI