Arquitecturas Avanzadas

Grupo AA-2-1

18/11/2019

Julio César Velasquez Cardenas 1397896
Sergio Prada Maeso 1459122
Juan Carlos Bermúdez Rodríguez 1455486

Evaluar Ejecución Single-Thread

1. Evaluar el rendimiento single-thread de las versiones clásica y divide&conquer con tamaños n= 1024 y 2048. Para el segundo caso, probad con los valores DQSZ= 16 y 256.

(Classic)

Performance counter stats for './MM 1024 0':

```
        2.118,62 msec task-clock                #    1,001 CPUs utilized
             163      context-switches          #    0,077 K/sec
              10      cpu-migrations            #    0,005 K/sec
           2.035      page-faults               #    0,961 K/sec
   6.906.549.298      cycles                    #    3,260 GHz
   4.242.883.930      instructions              #    0,61  insn per cycle
     321.826.453      branches                  #  151,904 M/sec
       1.155.799      branch-misses             #    0,36% of all branches

        2,117303816 seconds time elapsed

        2,108376000 seconds user
        0,010975000 seconds sys
```

Performance counter stats for './MM 2048 0':

```
       83.097,20 msec task-clock                #    0,998 CPUs utilized
           4.226      context-switches          #    0,051 K/sec
              21      cpu-migrations            #    0,000 K/sec
           2.052      page-faults               #    0,025 K/sec
 276.661.310.862      cycles                    #    3,329 GHz
  33.266.817.770      instructions              #    0,12  insn per cycle
   2.393.081.805      branches                  #   28,799 M/sec
       6.994.431      branch-misses             #    0,29% of all branches

       83,248934698 seconds time elapsed

       83,075626000 seconds user
        0,013970000 seconds sys
```

(Divide and Conquer)

Performance counter stats for './MM 1024 16':

```
2.203,28 msec task-clock                #    1,000 CPUs utilized
164      context-switches          #    0,074 K/sec
14       cpu-migrations            #    0,006 K/sec
2.034    page-faults               #    0,923 K/sec
7.297.377.571      cycles                    #    3,312 GHz
9.366.463.060      instructions              #    1,28  insn per cycle
1.196.398.948      branches                  #  543,008 M/sec
279.602       branch-misses             #    0,02% of all branches

2,203047821 seconds time elapsed

2,200846000 seconds user
0,002995000 seconds sys
```

Performance counter stats for './MM 1024 256':

```
3.010,69 msec task-clock                #    1,000 CPUs utilized
201      context-switches          #    0,067 K/sec
13       cpu-migrations            #    0,004 K/sec
2.034    page-faults               #    0,676 K/sec
10.014.199.483     cycles                    #    3,326 GHz
8.814.740.768      instructions              #    0,88  insn per cycle
1.128.410.330      branches                  #  374,801 M/sec
4.311.055       branch-misses             #    0,38% of all branches

3,011179281 seconds time elapsed

3,007291000 seconds user
0,003984000 seconds sys
```

Performance counter stats for './MM 2048 16':

```
17.202,23 msec task-clock                #    0,999 CPUs utilized
922      context-switches          #    0,054 K/sec
11       cpu-migrations            #    0,001 K/sec
2.052    page-faults               #    0,119 K/sec
57.662.044.927     cycles                    #    3,352 GHz
74.148.639.660     instructions              #    1,29  insn per cycle
9.363.390.376      branches                  #  544,313 M/sec
1.756.923       branch-misses             #    0,02% of all branches
```

17,222358218 seconds time elapsed

17,194935000 seconds user
0,011977000 seconds sys

Performance counter stats for './MM 2048 256':

22.995,18 msec task-clock # 0,999 CPUs utilized
1.237 context-switches # 0,054 K/sec
12 cpu-migrations # 0,001 K/sec
2.052 page-faults # 0,089 K/sec
77.263.485.371 cycles # 3,360 GHz
69.734.405.599 instructions # 0,90 insn per cycle
8.819.626.023 branches # 383,542 M/sec
34.026.633 branch-misses # 0,39% of all branches

23,024786811 seconds time elapsed

22,992827000 seconds user
0,007985000 seconds sys

Al aumentar el tamaño del bloque en la versión divide&conquer, es posible que necesitemos datos que pertenecen a bloques distintos en una misma iteración, lo que implica más accesos a memoria y un tiempo de ejecución más lento.

Al utilizar la función optimizada, dividimos el problema en 8 subtareas que trabajarán cada una con una parte del problema e irán llamando de nuevo a la misma función (de forma recursiva) con un rango cada vez más pequeño. Iremos dividiendo este problema hasta que lleguemos a un rango mínimo definido, para el cual al programa le sea fácil trabajar con él.

2. Optimizar las dos versiones del programa intercambiando los dos bucles internos que realizan la multiplicación de matrices. Evaluad la mejora del rendimiento con los tamaños n= 1024, 2048 y 4096. Explicad las razones de esta mejora e indicad cuál es el cuello de botella del rendimiento en cada caso.

(Classic)

Performance counter stats for './MM 1024 0':

288,72 msec task-clock # 1,017 CPUs utilized
46 context-switches # 0,159 K/sec

```
        10      cpu-migrations          #       0,035 K/sec
    2.034   page-faults              #       0,007 M/sec
    842.443.794   cycles                    #       2,918 GHz
    1.669.716.009         instructions              #       1,98  insn per cycle
    184.331.463   branches                #  638,435 M/sec
    603.662         branch-misses           #       0,33% of all branches

    0,283953648 seconds time elapsed

    0,284737000 seconds user
    0,004995000 seconds sys
```

Performance counter stats for './MM 2048 0':

```
    3.149,72 msec task-clock                #       1,002 CPUs utilized
    213     context-switches        #       0,068 K/sec
        10      cpu-migrations          #       0,003 K/sec
    2.054   page-faults              #       0,652 K/sec
    10.413.182.846        cycles                  #       3,306 GHz
    12.543.708.147        instructions            #       1,20  insn per cycle
    1.265.085.533         branches                #  401,650 M/sec
    2.681.248       branch-misses           #       0,21% of all branches

    3,143053166 seconds time elapsed

    3,146416000 seconds user
    0,003990000 seconds sys
```

Performance counter stats for './MM 4096 0':

```
    28.882,05 msec task-clock               #       0,999 CPUs utilized
    1.746   context-switches        #       0,060 K/sec
        17      cpu-migrations          #       0,001 K/sec
    2.128   page-faults              #       0,074 K/sec
    96.842.639.324        cycles                  #       3,353 GHz
    97.426.697.471        instructions            #       1,01  insn per cycle
    9.354.255.235         branches                #  323,878 M/sec
    12.974.144      branch-misses           #       0,14% of all branches

    28,920810587 seconds time elapsed

    28,835282000 seconds user
```

0,041895000 seconds sys

(Divide and Conquer)

Performance counter stats for './MM 1024 16':

778,89 msec task-clock # 1,005 CPUs utilized
68 context-switches # 0,087 K/sec
8 cpu-migrations # 0,010 K/sec
2.034 page-faults # 0,003 M/sec
2.312.300.935 cycles # 2,969 GHz
4.291.072.566 instructions # 1,86 insn per cycle
591.398.370 branches # 759,287 M/sec
166.405 branch-misses # 0,03% of all branches

0,774792360 seconds time elapsed

0,770937000 seconds user
0,009010000 seconds sys

Performance counter stats for './MM 1024 256':

384,73 msec task-clock # 1,011 CPUs utilized
43 context-switches # 0,112 K/sec
10 cpu-migrations # 0,026 K/sec
2.034 page-faults # 0,005 M/sec
1.179.847.886 cycles # 3,067 GHz
2.442.113.047 instructions # 2,07 insn per cycle
333.749.188 branches # 867,490 M/sec
96.171 branch-misses # 0,03% of all branches

0,380542671 seconds time elapsed

0,382635000 seconds user
0,003004000 seconds sys

Performance counter stats for './MM 2048 16':

5.364,46 msec task-clock # 1,001 CPUs utilized
306 context-switches # 0,057 K/sec
17 cpu-migrations # 0,003 K/sec

```
     2.052   page-faults               #        0,383 K/sec
    17.781.346.962      cycles                    #        3,315 GHz
    33.548.113.641      instructions              #        1,89  insn per cycle
     4.523.952.478      branches                  #  843,319 M/sec
       771.199          branch-misses             #        0,02% of all branches

     5,357744013 seconds time elapsed

     5,355429000 seconds user
     0,009993000 seconds sys
```

Performance counter stats for './MM 2048 256':

```
     2.801,28 msec task-clock                #        1,004 CPUs utilized
          192      context-switches          #        0,069 K/sec
           14      cpu-migrations            #        0,005 K/sec
        2.052      page-faults               #        0,733 K/sec
    9.304.117.676      cycles                    #        3,321 GHz
   18.768.112.575      instructions              #        2,02  insn per cycle
    2.465.753.648      branches                  #  880,225 M/sec
          308.848      branch-misses             #        0,01% of all branches

     2,790388737 seconds time elapsed

     2,788151000 seconds user
     0,012968000 seconds sys
```

Performance counter stats for './MM 4096 16':

```
    42.058,60 msec task-clock                #        0,999 CPUs utilized
        2.203      context-switches          #        0,052 K/sec
           15      cpu-migrations            #        0,000 K/sec
        2.124      page-faults               #        0,051 K/sec
  141.571.071.754      cycles                    #        3,366 GHz
  265.664.791.829      instructions              #        1,88  insn per cycle
   35.453.172.006      branches                  #  842,947 M/sec
        5.523.404      branch-misses             #        0,02% of all branches

    42,086640899 seconds time elapsed

    42,036026000 seconds user
     0,026947000 seconds sys
```

Performance counter stats for './MM 4096 256':

```
21.615,48 msec task-clock                #    1,000 CPUs utilized
1.133   context-switches          #    0,052 K/sec
      10      cpu-migrations           #    0,000 K/sec
2.124   page-faults               #    0,098 K/sec
72.469.671.037      cycles                     #    3,353 GHz
147.405.554.176     instructions               #    2,03  insn per cycle
18.982.855.235      branches                   #  878,206 M/sec
1.899.620       branch-misses            #    0,01% of all branches

21,608221798 seconds time elapsed

21,590130000 seconds user
0,026950000 seconds sys
```

Los resultados resumidos son estos:

		Versión Inicial				Bucles Intercambiados			
		CPU's Used	Time Elapsed	IPC	Result	CPU's Used	Time Elapsed	IPC	Result
1024	0	1	2,108	0,61	1	1	0,285	1,98	1
2048	0	0,998	83,249	0,12	1	1,002	3,143	1,2	1
4096	0	-	-	-	1	0,999	28,835	1,01	1
1024	16	1	2,201	1,28	1	1,005	0,771	1,86	1
1024	256	1	3,007	0,88	1	1,011	0,383	2,07	1
2048	16	0,999	17,195	1,29	1	1,001	5,355	1,89	1
2048	256	0,999	23,025	0,9	1	1,004	2,789	2,02	1
4096	16	-	-	-	1	0,999	42,036	1,88	1
4096	256	-	-	-	1	1	21,59	2,03	1

En intercambiar los dos bucles internos del programa hemos conseguido mejorar el IPC del programa al mejorar el acceso a los datos necesarios, aumentando la localidad espacial de los datos, es decir, hacemos referencia a posiciones de memoria cercanas en una misma iteración, accediendo de una manera más óptima a la memoria, ya que antes se accedia a los elementos por columnas y se ha cambiado este acceso a uno por filas. Así conseguimos perder menos tiempo al no necesitar referenciar a bloques distintos en cada iteración, siendo los accesos a memoria el cuello de botella en la versión inicial del programa.

A continuación, tenemos los diferentes códigos ensambladores de las nuevas versiones del programa (Con los bucles intercambiados):

-Classic 1024:

```
 0,04         pxor   %xmm2,%xmm2
13,39  2c7:   movss  (%rdx,%r9,1),%xmm1
15,16         movss  (%rdx),%xmm5
 0,01         add    $0x10,%rcx
17,17         movss  (%rdx,%rsi,1),%xmm3
16,20         movss  (%rdx,%r9,2),%xmm4
 2,39         unpcklps %xmm1,%xmm5
 0,13         movups -0x10(%rcx),%xmm6
 2,14         add    0x18(%rsp),%rdx
 4,07         unpcklps %xmm3,%xmm4
 0,95         movaps %xmm5,%xmm1
 2,55         movlhps %xmm4,%xmm1
12,68         mulps  %xmm6,%xmm1
 9,02         addps  %xmm1,%xmm2
 3,67         cmp    0x8(%rsp),%rcx
 0,12       ↑ jne    2c7
              movaps %xmm2,%xmm1
```

-Classic 2048:

```
        nop
 0,62   movss  (%rcx),%xmm1
 0,07   mov    %r8,%rdx
        mov    %r10,%rax
        nop
 0,21   movss  (%rax),%xmm0
44,44   mulss  (%rdx),%xmm0
 0,03   add    $0x4,%rax
10,42   add    %rsi,%rdx
15,89   addss  %xmm0,%xmm1
11,96   movss  %xmm1,(%rcx)
14,32   cmp    %rax,%rdi
 0,95 → jne    401430 <void MM_DQ<float>(float const*, float const*, float*, int, int)+0x1b0>
        add    $0x4,%rcx
        add    $0x4,%r8
 0,88   cmp    %rcx,%r9
 0,01 → jne    401420 <void MM_DQ<float>(float const*, float const*, float*, int, int)+0x1a0>
```

-Classic 4096:

```
        nop
 0,07   movss  (%rcx),%xmm1
 0,01   mov    %r8,%rdx
        mov    %r10,%rax
        nop
 0,03   movss  (%rax),%xmm0
53,66   mulss  (%rdx),%xmm0
 0,03   add    $0x4,%rax
 5,29   add    %rsi,%rdx
18,52   addss  %xmm0,%xmm1
11,13   movss  %xmm1,(%rcx)
11,06   cmp    %rax,%rdi
 0,11 → jne    401430 <void MM_DQ<float>(float const*, float const*, float*, int, int)+0x1b0>
        add    $0x4,%rcx
        add    $0x4,%r8
 0,07   cmp    %rcx,%r9
```

Divide&Conquer 1024:

```
 0,11   movaps %xmm2,%xmm4
        shufps $0x0,%xmm4,%xmm4
 5,14 2db:  movups (%r10,%rax,1),%xmm0
 7,99   movups (%r11,%rax,1),%xmm1
 0,37   movups (%rdx,%rax,1),%xmm6
21,08   mulps  %xmm5,%xmm0
 4,53   mulps  %xmm4,%xmm1
 6,30   addps  %xmm6,%xmm0
18,22   addps  %xmm1,%xmm0
16,34   movups %xmm0,(%rdx,%rax,1)
        add    $0x10,%rax
18,95   cmp    0x18(%rsp),%rax
 0,21 ↑ jne    2db
 0,15   mov    %r14d,%eax
        cmp    %r14d,0x10(%rsp)
```

-Divide&Conquer 2048:

```
 2,77   cmp    0x18(%rsp),%rdx
 0,04   setae  %dil
 0,26   or     %edi,%r8d
 2,07   cmp    0x10(%rsp),%rcx
 0,07   setae  %dil
 2,60   cmp    %r10,%rax
        setae  %r11b
 3,96   or     %r11d,%edi
 0,26   test   %dil,%r8b
      → je     4016c0 <void MM_DQ<float>(float const*, float const*, float*, int, int)+0x370>
 0,21   cmpl   $0x2,0x20(%rsp)
 0,32 → jbe    4016c0 <void MM_DQ<float>(float const*, float const*, float*, int, int)+0x370>
 6,22   movss  (%rcx),%xmm1
        xor    %edi,%edi
 1,44   shufps $0x0,%xmm1,%xmm1
        nop
14,85   movups (%rdx,%rdi,1),%xmm0
 2,55   movups (%rax,%rdi,1),%xmm2
16,91   mulps  %xmm1,%xmm0
20,14   addps  %xmm2,%xmm0
 4,71   movups %xmm0,(%rax,%rdi,1)
        add    $0x10,%rdi
10,57   cmp    %rsi,%rdi
 0,11 → jne    4015e0 <void MM_DQ<float>(float const*, float const*, float*, int, int)+0x290>
 3,50   cmp    %ebx,%r15d
 0,04 → je     401667 <void MM_DQ<float>(float const*, float const*, float*, int, int)+0x317>
```

-Divide&Conquer 4096

```
        cmpl   $0x2,0x20(%rsp)
      → jbe    4016c0 <void MM_DQ<float>(float const*, float const*, float*, int, int)+0x370>
 0,53   movss  (%rcx),%xmm1
        xor    %edi,%edi
        shufps $0x0,%xmm1,%xmm1
        nop
21,13   movups (%rdx,%rdi,1),%xmm0
 0,23   movups (%rax,%rdi,1),%xmm2
12,02   mulps  %xmm1,%xmm0
31,39   addps  %xmm2,%xmm0
 6,38   movups %xmm0,(%rax,%rdi,1)
        add    $0x10,%rdi
25,44   cmp    %rsi,%rdi
 0,23   jne    4015e0 <void MM_DQ<float>(float const*, float const*, float*, int, int)+0x290>
 0,61   cmp    %ebx,%r15d
      → je     401667 <void MM_DQ<float>(float const*, float const*, float*, int, int)+0x317>
        lea    (%r9,%r15,1),%edi
```

3. Explorar la estrategia de "Register blocking". Consiste en desenrollar simultáneamente los dos bucles externos para poder aprovechar la localidad temporal del algoritmo, reutilizar datos en registros y así reducir el número total de accesos a las matrices (y por tanto reducir el número total de accesos a memoria).

```
c[i*N+j] += a[i*N+k]*b[k*N+j] + a[i*N+(k+1)]*b[(k+1)*N+j];
c[(i+1)*N+j] += a[(i+1)*N+k]*b[k*N+j] + a[(i+1)*N+(k+1)]*b[(k+1)*N+j];
```

Bucles de k i j desenrollados, accedemos a varios elementos por iteración.

Performance counter stats for './MM 1024 0':

```
277,25 msec task-clock                #    1,022 CPUs utilized
        43      context-switches      #    0,155 K/sec
        11      cpu-migrations        #    0,040 K/sec
2.034   page-faults                   #    0,007 M/sec
804.373.159     cycles                #    2,901 GHz
1.481.642.427           instructions          #    1,84  insn per cycle
116.857.483     branches              #  421,482 M/sec
338.838         branch-misses         #    0,29% of all branches

0,271316639 seconds time elapsed

0,271256000 seconds user
0,007032000 seconds sys
```

Performance counter stats for './MM 1024 16':

```
707,79 msec task-clock                #    1,005 CPUs utilized
        70      context-switches      #    0,099 K/sec
        10      cpu-migrations        #    0,014 K/sec
2.034   page-faults                   #    0,003 M/sec
```

```
     2.268.295.304      cycles                    #    3,205 GHz
     3.277.030.075      instructions              #    1,44  insn per cycle
     186.406.297   branches                 #  263,365 M/sec
     459.580       branch-misses            #    0,25% of all branches

     0,704246308 seconds time elapsed

     0,702729000 seconds user
     0,005989000 seconds sys

 Performance counter stats for './MM 1024 256':

     304,32 msec task-clock                #    1,015 CPUs utilized
          42      context-switches         #    0,138 K/sec
          12      cpu-migrations           #    0,039 K/sec
     2.035   page-faults               #    0,007 M/sec
     894.458.601   cycles                    #    2,939 GHz
     1.558.007.915      instructions              #    1,74  insn per cycle
     119.769.077   branches                 #  393,566 M/sec
     87.540        branch-misses            #    0,07% of all branches

     0,299879350 seconds time elapsed

     0,299136000 seconds user
     0,005982000 seconds sys


 Performance counter stats for './MM 2048 0':

     2.154,30 msec task-clock              #    1,005 CPUs utilized
     163     context-switches          #    0,076 K/sec
          13      cpu-migrations           #    0,006 K/sec
     2.052   page-faults               #    0,953 K/sec
     6.887.490.804      cycles                    #    3,197 GHz
     10.976.899.574     instructions              #    1,59  insn per cycle
     724.478.669   branches                 #  336,294 M/sec
     1.452.220     branch-misses            #    0,20% of all branches

     2,144345539 seconds time elapsed

     2,143864000 seconds user
     0,010958000 seconds sys

Performance counter stats for './MM 2048 16':

     5.241,98 msec task-clock              #    1,001 CPUs utilized
```

```
369      context-switches          #          0,070 K/sec
         14       cpu-migrations           #          0,003 K/sec
2.052    page-faults               #          0,391 K/sec
17.517.806.219          cycles                    #          3,342 GHz
25.445.888.283          instructions              #          1,45  insn per cycle
1.286.497.743           branches                  #  245,422 M/sec
2.771.257       branch-misses             #          0,22% of all branches

5,238593594 seconds time elapsed

5,229774000 seconds user
0,011969000 seconds sys
```

```
Performance counter stats for './MM 2048 256':

2.055,17 msec task-clock                #          1,006 CPUs utilized
147      context-switches          #          0,072 K/sec
         15       cpu-migrations           #          0,007 K/sec
2.052    page-faults               #          0,998 K/sec
6.666.775.582           cycles                    #          3,244 GHz
11.693.795.015          instructions              #          1,75  insn per cycle
753.589.089     branches                  #  366,680 M/sec
242.579         branch-misses             #          0,03% of all branches

2,043660121 seconds time elapsed

2,040538000 seconds user
0,014952000 seconds sys
```

Paralelización MIMD

1. Paralelizar la versión clásica del programa usando las directivas parallel y for. Medid la mejora de rendimiento para n= 2048 y 4096. Verificar que el resultado del programa paralelo coincide con el del programa secuencial (asumiendo que se pueden producir pequeñas diferencias debidas a errores de redondeo).

Hemos verificado que los resultados dan el mismo valor, por lo que al paralelizar se ha hecho de forma correcta y hemos logrado que todos los trabajos se coordinen entre sí para dar lugar a la solución como la de la ejecución Secuencial del programa.

Con parallel y for:

Performance counter stats for './MM 2048 0':

```
        3.941,08 msec task-clock                #    5,348 CPUs utilized
             803      context-switches          #    0,204 K/sec
              14      cpu-migrations            #    0,004 K/sec
           2.052      page-faults               #    0,521 K/sec
  12.422.475.773      cycles                    #    3,152 GHz
  10.996.548.965      instructions              #    0,89  insn per cycle
     729.575.404      branches                  #  185,121 M/sec
       1.285.539      branch-misses             #    0,18% of all branches

        0,736882610 seconds time elapsed

        3,926565000 seconds user
        0,015005000 seconds sys
```

Performance counter stats for './MM 4096 0':

```
       32.415,94 msec task-clock                #    7,041 CPUs utilized
           2.726      context-switches          #    0,084 K/sec
              22      cpu-migrations            #    0,001 K/sec
           2.125      page-faults               #    0,066 K/sec
 105.170.856.017      cycles                    #    3,244 GHz
  84.737.413.372      instructions              #    0,81  insn per cycle
   5.049.619.203      branches                  #  155,776 M/sec
       5.009.491      branch-misses             #    0,10% of all branches

        4,603747657 seconds time elapsed

       32,367983000 seconds user
        0,040882000 seconds sys
```

	Single Thread		Multi Thread		
	Time	IPC	Time	IPC	Speed Up
2048 / 0	83,249	0,12	3,92	0,89	x21,24
4096 / 0	-	-	4,6	0,81	

Observamos que el IPC aumenta (aproximadamente) 8 veces, lo que nos confirma que el uso de tareas ha resultado en una buena optimización que ha llegado a conseguir un speed up con una mejora de hasta de un x21,24 respecto la versión Inicial single Thread.

2. Paralelizar la versión divide&conquer del programa usando las directivas task y taskwait. Medid la mejora de rendimiento para n= 2048 y 4096, y para DQSZ= 16 y 256. Verificar que el resultado del programa paralelo coincide con el del programa secuencial (asumiendo que se pueden producir pequeñas diferencias debidas a errores de redondeo).

```
if (DQSZ) // Divide & Conquer Version
{
  #pragma omp parallel
  #pragma omp master
  MM_DQ<REAL> ( A, B, C, N, N);
}
static int DQSZ; // Smaller Size for a subproblem

// c[][] = c[][] + a[][] * b[][]
template <class real>
void MM_DQ ( const real *a, const real *b, real *c, int SZ, const int N)
{
  // SZ: dimension of submatrices a, b and c.
  // N:  size of original input matrices (size of a row)

  if (SZ <= DQSZ)
  { // Classical algorithm for base case
    #pragma omp task
    for (int i=0; i<SZ; i+=2)
      for (int k=0; k<SZ; k+=2)
        for (int j=0; j<SZ; j++)
        {
          c[i*N+j] += a[i*N+k]*b[k*N+j] + a[i*N+(k+1)]*b[(k+1)*N+j];
          c[(i+1)*N+j] += a[(i+1)*N+k]*b[k*N+j] + a[(i+1)*N+(k+1)]*b[(k+1)*N+j];
        }
  #pragma omp taskwait
    return;
  }
```

Los resultados coinciden, señal de que el programa ejecuta correctamente las tareas (El master pone las tareas en cola y los demás van ejecutando el trabajo que el master pone en cola. Al final, se hace un taskwait para que el master espere a que acaben todas las tareas y una ve concluidas, se elimina la zona paralela y se vuelve a la ejecución secuencial.

Performance counter stats for './MM 2048 16':

```
     76.180,95 msec task-clock                #    7,859 CPUs utilized
     6.169      context-switches          #    0,081 K/sec
           20      cpu-migrations            #    0,000 K/sec
     2.063      page-faults               #    0,027 K/sec
247.291.618.317      cycles                    #    3,246 GHz
106.247.893.694      instructions              #    0,43  insn per cycle
     24.085.228.557      branches                  #  316,158 M/sec
     72.516.672      branch-misses             #    0,30% of all branches

     9,693897044 seconds time elapsed

     75,848039000 seconds user
```

0,343339000 seconds sys

Performance counter stats for './MM 2048 256':

3.112,61 msec task-clock # 1,417 CPUs utilized
1.243 context-switches # 0,399 K/sec
25 cpu-migrations # 0,008 K/sec
2.062 page-faults # 0,662 K/sec
9.647.584.760 cycles # 3,100 GHz
12.995.153.393 instructions # 1,35 insn per cycle
1.124.466.507 branches # 361,261 M/sec
434.472 branch-misses # 0,04% of all branches

2,196343817 seconds time elapsed

3,116100000 seconds user
0,021170000 seconds sys

Performance counter stats for './MM 4096 16':

605.075,04 msec task-clock # 7,746 CPUs utilized
78.020 context-switches # 0,129 K/sec
743 cpu-migrations # 0,001 K/sec
2.137 page-faults # 0,004 K/sec
1.964.732.062.321 cycles # 3,247 GHz
846.577.751.930 instructions # 0,43 insn per cycle
191.706.119.311 branches # 316,830 M/sec
580.285.410 branch-misses # 0,30% of all branches

78,114485326 seconds time elapsed

601,038878000 seconds user
4,168357000 seconds sys

Performance counter stats for './MM 4096 256':

21.219,83 msec task-clock # 1,363 CPUs utilized
7.124 context-switches # 0,336 K/sec
244 cpu-migrations # 0,011 K/sec
2.135 page-faults # 0,101 K/sec
69.459.559.572 cycles # 3,273 GHz
100.416.744.340 instructions # 1,45 insn per cycle
8.024.390.767 branches # 378,155 M/sec
2.418.121 branch-misses # 0,03% of all branches

15,565822403 seconds time elapsed

21,324527000 seconds user
0,083887000 seconds sys

3. Explicar las diferencias de rendimiento entre las dos versiones, y a partir de ellas encontrar el cuello de botella del rendimiento en cada caso. Predecid el tiempo de ejecución de cada versión para n= 8192 y luego verificar la predicción y sacar conclusiones.

Para examinar el cuello de botella de cada ejecución, tendríamos que echar mano del código ensamblador y de todas las modificaciones hechas en él desde que empezamos la optimización de este programa y hacer una búsqueda exhaustiva.
Desafortunadamente no hemos contado con el tiempo suficiente para hacerlo.

Result= 6.33891e+29

Performance counter stats for './MM 8192 0':

```
     145.762,76 msec task-clock              #    3,840 CPUs utilized
          2.498      context-switches        #    0,017 K/sec
             10      cpu-migrations          #    0,000 K/sec
          2.324      page-faults             #    0,016 K/sec
465.738.308.358      cycles                  #    3,195 GHz
253.798.966.477      stalled-cycles-frontend   #   54,49% frontend cycles idle
162.609.432.793      stalled-cycles-backend    #   34,91% backend cycles idle
665.288.890.746      instructions            #    1,43  insn per cycle
                                             #    0,38  stalled cycles per insn
 37.354.836.493      branches                #  256,271 M/sec
     24.032.894      branch-misses           #    0,06% of all branches

    37,959135907 seconds time elapsed

   145,584014000 seconds user
     0,167860000 seconds sys
```

Result= 6.33891e+29

Performance counter stats for './MM 8192 256':

```
     285.683,78 msec task-clock              #    1,322 CPUs utilized
          44.846      context-switches        #    0,157 K/sec
            510      cpu-migrations          #    0,002 K/sec
```

```
        2.328      page-faults              #    0,008 K/sec
953.785.144.441      cycles                   #    3,339 GHz
657.239.931.150      stalled-cycles-frontend   #   68,91% frontend cycles idle
512.806.730.602      stalled-cycles-backend    #   53,77% backend cycles idle
810.346.208.925      instructions             #    0,85  insn per cycle
                                              #    0,81  stalled cycles per insn
 66.285.516.126      branches                 #  232,024 M/sec
   562.325.614       branch-misses            #    0,85% of all branches

 216,114154930 seconds time elapsed

 285,965805000 seconds user
   1,061421000 seconds sys
```