FASE II

Proyecto de evaluación del rendimiento

JUAN SÁEZ
LAURA MONROIG
ALEJANDRO AYELO
JULIA DE LA CRUZ
DANIEL ASENSI
ANDER DELGADO

Índice

1. Introducción

En la fase II de esta práctica desarrollaremos una serie de Benchmarks reducidos para poner a prueba y evaluar el rendimiento de los distintos ordenadores de los componentes del equipo, en total utilizaremos 6 ordenadores distintos.
Se van a realizar 3 Benchmarks basados en el mismo algoritmo, pero implementados en los siguientes lenguajes:

- C++
- Ensamblador x86
- Ensamblador con instrucciones SSE

El objetivo principal es comparar el tiempo, medido en segundos, que tarda cada terminal en ejecutar los programas y explicar los resultados, empleando diferentes recursos gráficos.

La segunda parte del proyecto consiste en ejecutar en todos los ordenadores el benchmark, proporcionado por la universidad, SPEC CPU2000 producido por SPEC (Standard Performance Evaluation Corporation).
El cual está formado a su vez por dos benchmarks:

- CINT2000-> Para medir y comparar el rendimiento de computación intensiva de enteros.
- CFP2000-> Para medir y comparar el rendimiento de computación intensiva en flotantes.

Con el SPEC CPU2000 solamente evaluaremos el rendimiento del procesador de nuestros ordenadores.

1.1. Ejemplo algoritmos para la evaluación del rendimiento

Hay 5 niveles de programas utilizados para evaluar el rendimiento de los ordenadores:

1. Real programs: como los compiladores de C o herramientas como photoshop
2. Scripted: programas reales modificados para o mejorar la portabilidad o para centrarse en un parte del funcionamiento del sistema
3. Kernels: programas compuestos por piezas clave de programas reales. Ejemplos «Livermore Loops» y «Linpack»
4. Benchmarks Reducidos: se componen de pocas líneas de código y antes de ejecutarlos ya se conoce el resultado. Ejemplos QuickSort (empleando divide y vencerás)
5. Benchmarks Sintéticos: Están diseñados para medir el rendimiento de un componente individual de un ordenador. Cabe destacar como ejemplos Whetstone (mide la capacidad del procesador de trabajar con números de punto flotante) y Dhrystone (mide la capacidad del procesador de trabajar con números enteros).

1.2. Características de los ordenadores

Ordenador 1 (Julia):
- Windows 10
- Intel(R) Core(TM) i7-8700T CPU @ 2.40GHz turbo: 4.00GHz
- 6 núcleos, 12 hilos
- 8,00 GB (7,79 GB usable)

Ordenador 2 (Ander):
- Windows 10
- Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz turbo: 4.00GHz
- 4 núcleos, 8 hilos
- 16,00 GB (15,9 GB usable)

Ordenador 3 (Laura):
- Windows 10
- Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz turbo: 3,9GHz
- 4 núcleos, 8 hilos
- 8,00 GB (7,9 GB usable)

Ordenador 4 (Juan):
- Windows 10
- Intel(R) Core(TM) i7-9750H CPU @ 2,6 GHz
- 6 núcleos, 12 hilos
- 16,00 GB (15,9 GB usable)

Ordenador 5 (Alejandro):
- Windows 10
- Intel(R) Core(TM) i5-6300HQ CPU @ 2,30 GHz
- 4 núcleos, 8 hilos
- 8,00 GB (7,9 GB usable)

Ordenador 6(Daniel):
- Windows 10
- Intel(R) Core(TM) i7-10700K CPU @ 3,8 GHz
- 8 núcleos, 16 hilos
- 20,00 GB (19,9 GB usable)

2. Benchmark

El algoritmo sobre el que vamos a basar nuestro benchmark realiza la inversa y el determinante de la matriz dada, a la vez. La implementación de dicho algoritmo para ordenadores es extremadamente simple y bastante eficiente en la utilización de memoria y tiempo. El número de multiplicaciones/divisiones que realiza es de n^3. Sin embargo, su eficiencia radica en la sencillez del código y la utilización mínima de memoria.

Ejemplo comparación con método Gauss Jordan

Size of Matrix	Gauss Jordan	New Technique
30*30	0.016	0
50*50	0.032	0
100*100	0.26	0.015
150*150	0.54	0.047
200*200	1.13	0.125
250*250	2.05	0.234

El algoritmo toma una matriz cuadrada $A=[a_{i,j}]$ de dimensión n. El inverso se calcula en n iteraciones. En cada iteración (p), todos los elementos existentes $a_{i,j}$ de A cambian a nuevos valores $a'_{i,j}$. Después de la última iteración, es decir, cuando $p=n$, $a'_{i,j}$ serán los elementos de la matriz inversa. El determinante de la matriz (d) también se calcula iterativamente a través de la multiplicación sucesiva del pivote seleccionado en cada iteración.

En este algoritmo, los pivotes se seleccionan diagonalmente empezado por $a_{1,1}$ hasta $a_{n,n}$. Si se encuentra que cualquier pivote es cero, $a_{p,p}=0$, no se puede calcular la inversa. Si se calcula una inversa, 'd' contendrá el determinante de A.

1. Código en C++

MatB representa la matriz original pasada a la función que iterativamente se convertirá en la matriz inversa. La variable size representa representa la dimensión de la matriz. LA función devuelve un float, un valor distinto de cero indica el determinante de la matriz mientras que si devuelve un 0 significa que no se puede calcular el determinante de la matriz y a su vez MatB no contendrá la inversa.

```
int inversaMatCBench()
{
    float MatB[MAX][MAX] =
    { {1, 3, 7, 8, 7, 0, 3, 0, 1, 8},
    {0, 9, 4, 7, 3, 7, 6, 3, 9, 4},
    {2, 8, 8, 8, 6, 9, 6, 0, 9, 9},
    {3, 8, 9, 1, 6, 2, 3, 3, 7, 4},
    {8, 8, 2, 8, 4, 9, 9, 4, 9, 2},
    {5, 2, 3, 7, 1, 5, 7, 3, 2, 7},
    {0, 8, 0, 7, 1, 2, 3, 6, 0, 6},
    {9, 4, 2, 3, 9, 0, 3, 4, 6, 3},
    {3, 7, 5, 6, 2, 4, 7, 9, 8, 9},
    {1, 4, 9, 4, 0, 4, 3, 1, 6, 7}
    };

    int size = MAX-1;

    float pivot, det = 1.0;
    int i, j, p;

    float compCero = 0.0;

    for (p = 0; p < size; p++)
    {
        pivot = MatB[p][p];
        det = det * pivot;

        if (fabs(pivot) < 1e-5) return 0;

        for (i = 0; i <= size; i++) // i1
            MatB[i][p] = -MatB[i][p] / pivot;

        for (i = 0; i < size; i++) // i2
            if (i != p)
                for (j = 0; j < size; j++) // j1
                    if (j != p)
                        MatB[i][j] = MatB[i][j] + MatB[p][j] * MatB[i][p];

        for (j = 0; j < size; j++)
            MatB[p][j] = MatB[p][j] / pivot;
        MatB[p][p] = 1 / pivot;
    }

    return 1;
}
```

2. Código en Ensamblador x86

En primer lugar, definiremos MatB que representa la matriz original de la cual queremos calcular la inversa, como hemos dicho anteriormente. En este caso "esi" representa la variable "p" y "edi" la variable "size" del código en C. En "_p", guardamos en xmm0 una posición de la matriz y seguidamente la moveremos a aux1 y a pivot. Después hará la operación

det = det * pivot en valor absoluto en float. Comprobará que no sea menor que "_precision" sino terminará el bucle. Y moveremos la "p" a ebx, "i" a esi y finalmente la pila.

En _i1, guardaremos la primera y la segunda posición de la matriz, después realiza la operación -MatB[i][p] / pivot, y una vez realizada salta a _i2, realizando los correspondientes bucles.

Y finalmente saltando a _lo haciendo las operaciones finales.

```
__asm
{
    mov esi, 0 // esi = p
    mov edi, _size // size

    jmp _p

    _p :
    mov eax, MAX
        mul esi
        add eax, esi

        movss xmm0, [MatB + 4 * eax]
        movss aux1, xmm0

        movss pivot, xmm0

        // det = det * pivot
        fld pivot
        fmul det
        fstp det

        // compCero = |pivot|;
        fld pivot
        fabs
        fstp compCero

        // if (pivot < 1e-5) -> FIN:
        mov eax, _precision
        cmp compCero, eax
        jl fin

        mov ebx, esi    // ebx = p
        push esi        // esi = i
        mov esi, 0
```

```
    _i1 :
mov eax, MAX
    mul esi // [*][]
    add eax, ebx // [][*]

    movss xmm0, [MatB + 4 * eax]
    movss aux1, xmm0

    //  -MatB[i][p] / pivot
    fld aux1
    fdiv pivot
    fstp aux1

    fld aux1
    fmul unoNegativo
    fstp[MatB + 4 * eax]

    jmp incremento_i_1
```

```
    _cp1 :
mov ebx, esi        // ebx = p
    push esi
    mov esi, 0          // esi = i
    jmp _i2

    _cp2 :
mov ebx, esi // ebx == p
    push esi     // esi = j
    mov esi, 0
    jmp _j2
```

```
    _lo :
// pivot = MatB[p][p]; p = esi
mov eax, MAX
    mul esi
    add eax, esi

    movss xmm0, [MatB + 4 * eax]
    movss aux1, xmm0

    fld unoPositivo
    fdiv pivot
    fstp[MatB + 4 * eax]

    jmp _cp4
```

3. Código en Ensamblador con instrucciones SSE

Como en los lenguajes anteriores, en primer lugar definimos la matriz MatB, guardamos las variables p en "esi", size en "edi". En _p, guardamos las posiciones de la matriz en "xmm0" y las pasamos a "pivot", hará compCero = |pivot|, compCero con valor 0.0. Comprobará que no sea menor que "_precision" sino terminará el bucle. Una vez realizadas las operaciones moverá la "p" a ebx y la "i" a esi.

```
__asm
{
    mov esi, 0 // esi = p
    mov edi, _size // size
    movss xmm4, unoPositivo // xmm4 = det -> Return determiannte
    //movss xmm7, pivot // 1
    mov ecx, MAX
    jmp _p

    _p :
    mov eax, ecx
        mul esi
        add eax, esi

        VZEROALL

        movss xmm0, [MatB + 4 * eax]
        movss pivot, xmm0
        mulss xmm4, pivot

        // compCero = |pivot|;
        fld pivot
        fabs
        fstp compCero

        // if (pivot < 1e-5) -> FIN:
        mov eax, _precision
        cmp compCero, eax
        jl fin

        mov ebx, esi    // ebx = p
        push esi        // esi = i
        mov esi, 0
```

Pasará al bucle "_i1" donde guardaremos la primera y segunda posición de la matriz y realizará MatB[i][p] = -MatB[i][p] / pivot, incrementará 1 y seguirá con el bucle.

```
    _i1 :
mov eax, ecx
    mul esi // [*][]
    add eax, ebx // [][*]

    movss xmm6, [MatB + 4 * eax]
    divss xmm6, pivot
    mulss xmm6, unoNegativo

    movss[MatB + 4 * eax], xmm6

    jmp incremento_i_1
```

```
    _cp1 :
mov ebx, esi        // ebx = p
    push esi
    mov esi, 0          // esi = i
    jmp _i2

    _cp2 :
mov ebx, esi // ebx == p
    push esi     // esi = j
    mov esi, 0
    jmp _j2
```

El programa terminará saltando a "_lo" que realizará las operaciones correspondientes para calcular la inversa.

```
    _lo :
mov eax, ecx
    mul esi
    add eax, esi


    movss xmm6, unoPositivo
    divss xmm6, pivot
    movss[MatB + 4 * eax], xmm6

    jmp _cp4
```

3. SPEC CPU2000

Cada uno de los Benchmarks descritos a continuación se ejecutará 3 veces, para mayor fiabilidad de los resultados, los cuales explicaremos más adelante.

CINT2000

El benchmark Cint2000, contiene 11 aplicaciones escritas en C y una en C++ las cuales serán usadas como benchmarks independientes en el testeo y, realizará pruebas en enteros. Estas aplicaciones, son las siguientes:

- 164.gzip: Utilidad de compresión de datos.
- 175.vpr: Direccionamiento y ubicación de circuitos FPGA.
- 176.gcc: Compilador C.
- 181.mcf: Costo mínimo de flujo de red.
- 186.crafty: Programa de ajedrez.
- 197.parser: Procesamiento de lenguaje natural.
- 252.eon: Efectos producidos por distintas fuentes de luz.
- 253.perlbmk: Perl.
- 254.gap: Teoría de grupo computacional.
- 255.vortex: Base de datos orientada a objetos.
- 256.bzip2: Utilidad de compresión de datos.
- 300.twolf: Simulador de ubicación y ruteo.

Comando de ejecución

"runspec --reportable --config=win32-x86-vc7.cfg -T base int"

Fichero con la solución de la ejecución .asc (results)

CFP2000

El benchmark Cfp2000, contiene 14 aplicaciones (6 en Fortran77, 4 en FORTRAN90 y 4 en C), las cuales serán usadas como benchmarks independientes en el testeo y, realizará pruebas en base coma flotante. Estas aplicaciones, son las siguientes:

- 168.wupwise: Cromodinámica de cuantos.
- 171.swim: Modelado de aguas poco profundas.
- 172.mgrid: Multi-grilla en campos potenciales 3D.
- 173.applu: Ecuaciones diferenciales parciales parabólicas/elípticas.
- 177.mesa: Biblioteca de gráficos 3D.
- 178.galgel: Dinámica de fluidos: análisis de inestabilidad oscilatoria.
- 179.art: Simulación de red neuronal: teoría de la resonancia adaptativa.
- 183.equake: Simulación de elementos finitos: modelado de terremotos.
- 187.facerec: Reconocimientos de imágenes: reconocimiento de rostros.

- 188.ammp: Química computacional.
- 189.lucas: Teoría de los números: prueba de primalidad.
- 191.fma3d: Simulación de elementos finitos en choque.
- 200.sixtrack: Modelo de acelerador de partículas.
- 301.apsi: Problemas de temperatura, viento y distribución de contaminantes

Comando de ejecución

"runspec --reportable --config=win32-x86-vc7.cfg -T base fp"

Expresión de los resultados

- Base Ref Time: son los valores de la máquina de referencia, que es una Sun Ultra 5/10 de estación de trabajo con un procesador de 300 MHz SPARC y 256 MB de memoria, que proporciona el propio SPEC.
- Base Run Time: será el tiempo de ejecución de la opción base.
- Base Ratio: esta columna indica la ganancia de rendimiento respecto del valor de referencia. Su cálculo se realiza de la siguiente manera:

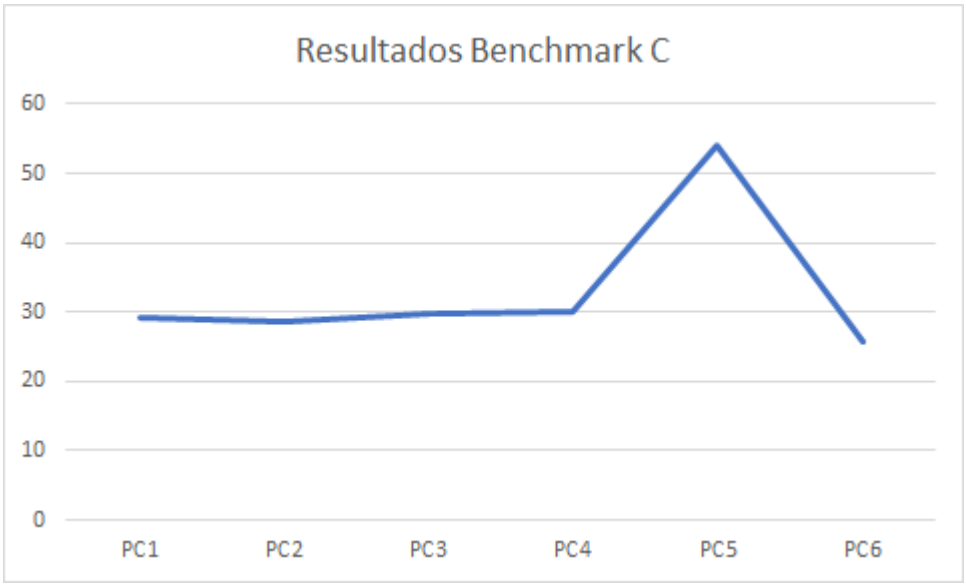
  $$Ratio = 100 \times RefTime / Run\ Time$$
- Media geométrica: Realizada para comparar los resultados entre los ordenadores realizados.

4. Análisis y evaluación de resultados

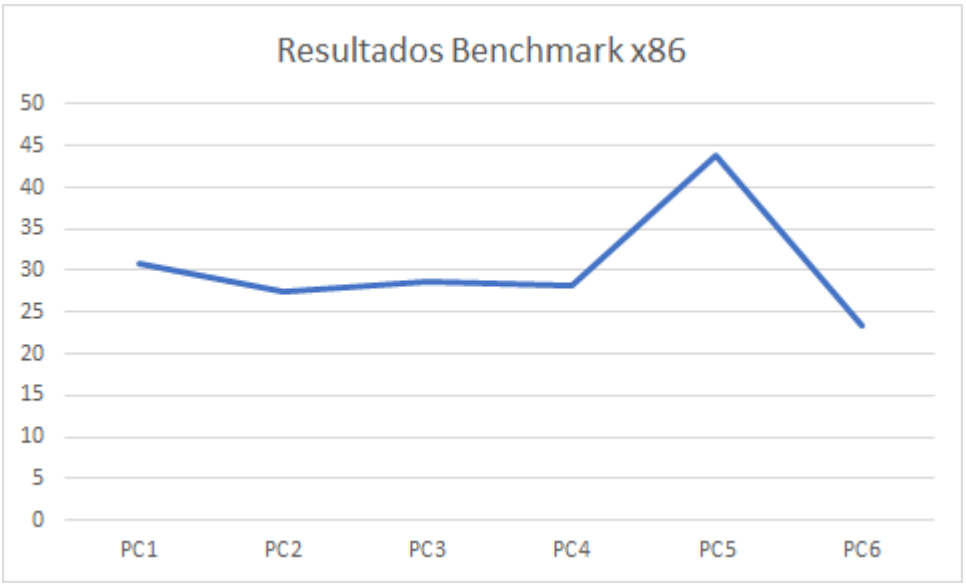
Nuestro grupo ha realizado un fichero llamado benchmark_final.cpp que imprime por pantalla los 3 tiempos de ejecución de los benchmark.

El programa realiza primero el algoritmo en C, luego en x86 y por último en SSE. En ese orden aparecen por pantalla los tiempos también.

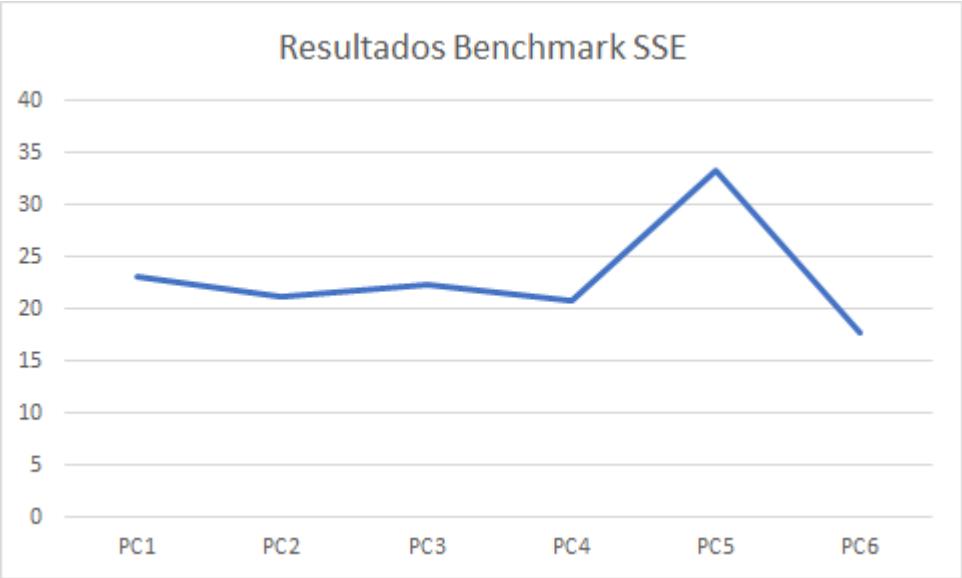
Resultados C:



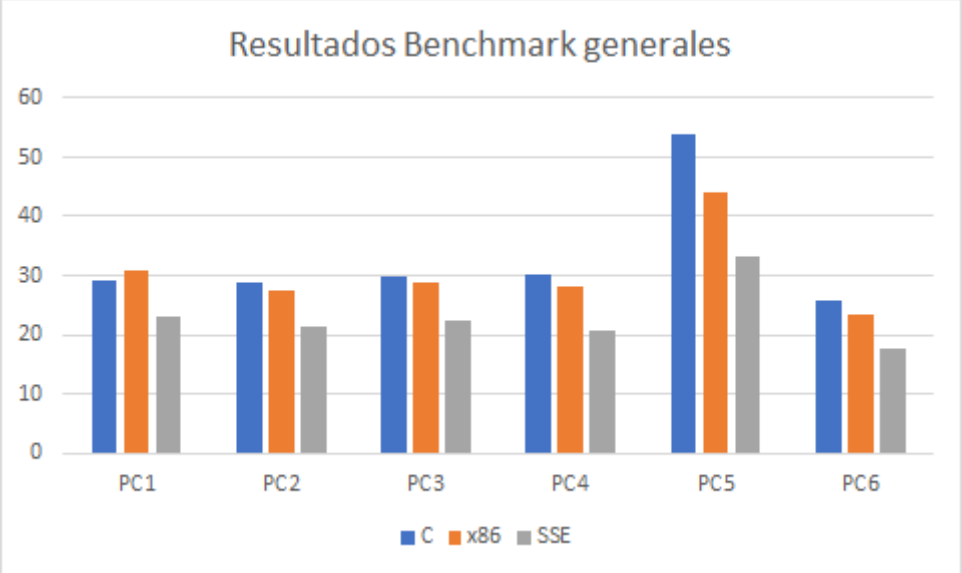
Resultado x86



Resultados SSE:



Resultado General:



En las gráficas anteriores podemos observar que según el lenguaje utilizado (c, x86, sse…) el tiempo en el cual se ejecuta el programa es diferente, si lo analizamos podemos darnos cuenta de que en el lenguaje c generalmente el tiempo de ejecución es más lento que en ensamblador. Aplicando instrucciones SSE podemos conseguir bajar aún más el tiempo de ejecución ya que estamos ejecutando varias instrucciones como si fuesen una única.

Podemos darnos cuenta perfectamente que el ordenador 5 es el más lento en ejecutar los benchmark respecto a los otros, esto es claramente observable viendo el pico de las gráficas (el ordenador 5 es el que "peores" componentes tiene), por el contrario, el ordenador 6 (el que mejor prestación tiene) es el más rápido en ejecutar todos los benchmark. El resto de ordenador tardan más o menos lo mismo en realizar los benchmark, por ello en las gráficas podemos observar que se forma casi una línea recta sin muchos picos.

Es curioso también, pero a la vez lógico que las tres gráficas a pesar de ser lenguajes diferentes la forma es bastante similar, aunque el tiempo de ejecución es completamente diferente debido a lo que hemos explicado anteriormente.

La velocidad de los ordenadores es proporcional a los componentes de los mismos siendo los más rápidos los que mejor procesador tienen (más núcleos e hilos) y más RAM (más cantidad de gb). Es importante recordar que todas las pruebas han sido ejecutadas en el sistema operativo Windows 10. Un dato curioso es que los ordenadores 2 y 3 los cuales tienen exactamente el mismo procesador no tardan lo mismo en cuanto al tiempo de ejecución del benchmark esto es debido a que uno de ellos tiene el doble de RAM y no todos los componentes son iguales.

Resultados Spec CPU2000

Procesador:	Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.60 GHz			
	ORDENADOR 4			
Benchmarks	Base Ref Time	Base Run Time	Estimated Base Ratio	Media Geométrica:
164.gzip	1400	57,3	24,43	40,80
164.gzip	1400	56	25,00	
164.gzip	1400	56,7	24,69	
175.vpr	1400	39,4	35,53	
175.vpr	1400	43,1	32,48	
175.vpr	1400	44,8	31,25	
176.gcc	1100	22,7	48,46	
176.gcc	1100	21,6	50,93	
176.gcc	1100	25,9	42,47	
181.mcf	1800	30,3	59,41	
181.mcf	1800	28,7	62,72	
181.mcf	1800	25,4	70,87	
186.crafty	1000	24,1	41,49	
186.crafty	1000	23,3	42,92	
186.crafty	1000	23,7	42,19	
197.parser	1800	57,9	31,09	
197.parser	1800	57,8	31,14	
197.parser	1800	57,2	31,47	
252.eon	1300	22,5	57,78	
252.eon	1300	22	59,09	
252.eon	1300	23,2	56,03	
253.perlbmk	1800	--	--	
254.gap	1100	26,8	41,04	
254.gap	1100	25,5	43,14	
254.gap	1100	26,9	40,89	
255.vortex	1900	41,7	45,56	
255.vortex	1900	35,7	53,22	
255.vortex	1900	35,7	53,22	
256.bzip2	1500	46,1	32,54	
256.bzip3	1500	46,6	32,19	
256.bzip4	1500	43,1	34,80	
300.twolf	3000	77	38,96	
300.twolf	3000	76,9	39,01	
300.twolf	3000	72,3	41,49	

Procesador:	Intel(R) Core(TM) i7-10700K CPU @ 3.80GHz 3.79 GHz			
	ORDENADOR 6			
Benchmarks	Base Ref Time	Base Run Time	Estimated Base Ratio	Media Geométrica:
164.gzip	1400	45,4	30,84	57,35
164.gzip	1400	45	31,11	
164.gzip	1400	45,1	31,04	
175.vpr	1400	30,6	45,75	
175.vpr	1400	30,5	45,90	
175.vpr	1400	30,6	45,75	
176.gcc	1100	14,9	73,83	
176.gcc	1100	14,8	74,32	
176.gcc	1100	14,8	74,32	
181.mcf	1800	19,3	93,26	
181.mcf	1800	19,5	92,31	
181.mcf	1800	19,4	92,78	
186.crafty	1000	17,5	57,14	
186.crafty	1000	17,4	57,47	
186.crafty	1000	17,4	57,47	
197.parser	1800	44	40,91	
197.parser	1800	44,1	40,82	
197.parser	1800	44,1	40,82	
252.eon	1300	17	76,47	
252.eon	1300	16	81,25	
252.eon	1300	16,9	76,92	
253.perlbmk	1800	--	--	
254.gap	1100	17,7	62,15	
254.gap	1100	17,5	62,86	
254.gap	1100	17,5	62,86	
255.vortex	1900	22,5	84,44	
255.vortex	1900	22,3	85,20	
255.vortex	1900	22,3	85,20	
256.bzip2	1500	33,3	45,05	
256.bzip3	1500	34,1	43,99	
256.bzip4	1500	34	44,12	
300.twolf	3000	57	52,63	
300.twolf	3000	57,6	52,08	
300.twolf	3000	57,6	52,08	

Procesador:	Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz			
		ORDENADOR 3		
Benchmarks	Base Ref Time	Base Run Time	Estimated Base Ratio	Media Geométrica:
164.gzip	1400	63,4	22,08	34,20
164.gzip	1400	65,1	21,51	
164.gzip	1400	66,1	21,18	
175.vpr	1400	52,2	26,82	
175.vpr	1400	53	26,42	
175.vpr	1400	52,3	26,77	
176.gcc	1100	26,4	41,67	
176.gcc	1100	27,4	40,15	
176.gcc	1100	25,3	43,48	
181.mcf	1800	26	69,23	
181.mcf	1800	28,5	63,16	
181.mcf	1800	26,1	68,97	
186.crafty	1000	31	32,26	
186.crafty	1000	31	32,26	
186.crafty	1000	30,9	32,36	
197.parser	1800	79,7	22,58	
197.parser	1800	79	22,78	
197.parser	1800	76,3	23,59	
252.eon	1300	31,2	41,67	
252.eon	1300	29,3	44,37	
252.eon	1300	29,4	44,22	
253.perlbmk	1800	--	--	
254.gap	1100	29	37,93	
254.gap	1100	29,8	36,91	
254.gap	1100	29,9	36,79	
255.vortex	1900	40,2	47,26	
255.vortex	1900	40,1	47,38	
255.vortex	1900	40,5	46,91	
256.bzip2	1500	56,4	26,60	
256.bzip3	1500	55,9	26,83	
256.bzip4	1500	57,9	25,91	
300.twolf	3000	96,2	31,19	
300.twolf	3000	97,2	30,86	
300.twolf	3000	96,3	31,15	

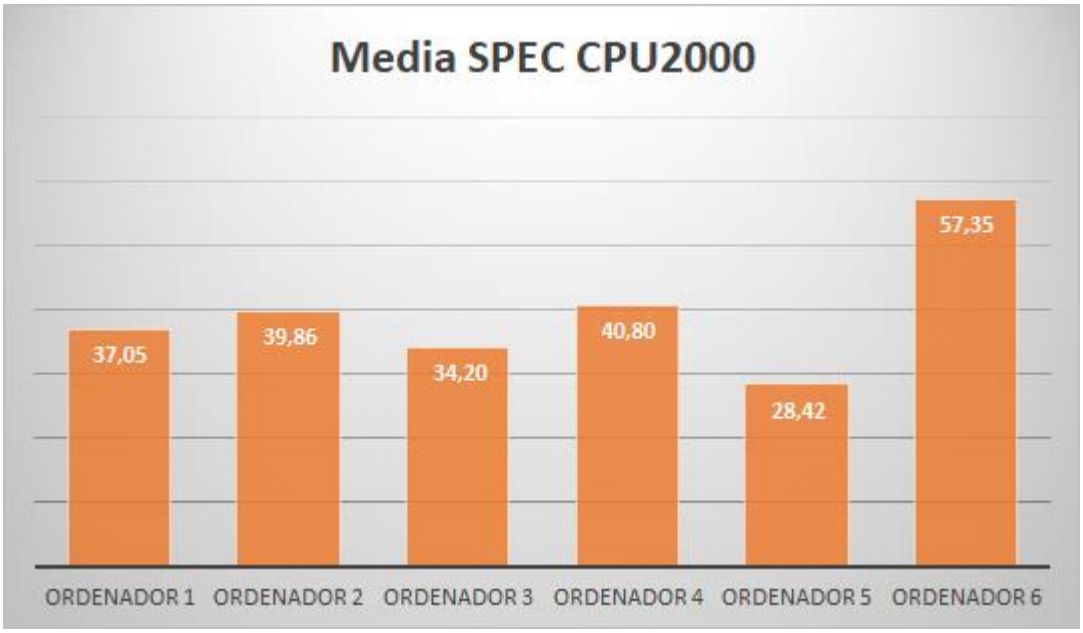
Procesador:	Intel(R) Core(TM) i5-6300HQ CPU @ 2.30GHz 2.30 GHz			
		ORDENADOR 5		
Benchmarks	Base Ref Time	Base Run Time	Estimated Base Ratio	Media Geométrica:
164.gzip	1400	85,5	16,37	28,42
164.gzip	1400	87,4	16,02	
164.gzip	1400	87,5	16,00	
175.vpr	1400	63,4	22,08	
175.vpr	1400	64,5	21,71	
175.vpr	1400	65	21,54	
176.gcc	1100	29,7	37,04	
176.gcc	1100	27,9	39,43	
176.gcc	1100	26,7	41,20	
181.mcf	1800	33,8	53,25	
181.mcf	1800	36,5	49,32	
181.mcf	1800	36,7	49,05	
186.crafty	1000	35,7	28,01	
186.crafty	1000	35,8	27,93	
186.crafty	1000	35,9	27,86	
197.parser	1800	89,7	20,07	
197.parser	1800	79,6	22,61	
197.parser	1800	89,7	20,07	
252.eon	1300	33,7	38,58	
252.eon	1300	33,5	38,81	
252.eon	1300	33,4	38,92	
253.perlbmk	1800	--	--	
254.gap	1100	32,8	33,54	
254.gap	1100	31,7	34,70	
254.gap	1100	34,7	31,70	
255.vortex	1900	46,9	40,51	
255.vortex	1900	59,5	31,93	
255.vortex	1900	47,3	40,17	
256.bzip2	1500	71,2	21,07	
256.bzip3	1500	75,3	19,92	
256.bzip4	1500	85,4	17,56	
300.twolf	3000	119	25,21	
300.twolf	3000	116	25,86	
300.twolf	3000	115	26,09	

Procesador:	Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz			
	ORDENADOR 2			
Benchmarks	Base Ref Time	Base Run Time	Estimated Base Ratio	Media Geométrica:
164.gzip	1400	61,8	22,65	39,86
164.gzip	1400	56,8	24,65	
164.gzip	1400	56,7	24,69	
175.vpr	1400	39,8	35,18	
175.vpr	1400	37,5	37,33	
175.vpr	1400	37,3	37,53	
176.gcc	1100	19,9	55,28	
176.gcc	1100	20,5	53,66	
176.gcc	1100	19,5	56,41	
181.mcf	1800	21,9	82,19	
181.mcf	1800	21,6	83,33	
181.mcf	1800	21,6	83,33	
186.crafty	1000	21,7	46,08	
186.crafty	1000	21,4	46,73	
186.crafty	1000	21,3	46,95	
197.parser	1800	93,8	19,19	
197.parser	1800	84,8	21,23	
197.parser	1800	83,8	21,48	
252.eon	1300	32,5	40,00	
252.eon	1300	30,8	42,21	
252.eon	1300	26,5	49,06	
253.perlbmk	1800	--	--	
254.gap	1100	26,4	41,67	
254.gap	1100	27,1	40,59	
254.gap	1100	25,7	42,80	
255.vortex	1900	36,4	52,20	
255.vortex	1900	35,8	53,07	
255.vortex	1900	36,5	52,05	
256.bzip2	1500	49,9	30,06	
256.bzip3	1500	48,5	30,93	
256.bzip4	1500	49	30,61	
300.twolf	3000	85,3	35,17	
300.twolf	3000	82,3	36,45	
300.twolf	3000	82,5	36,36	

Procesador:	Intel(R) Core(TM) i7-8700T CPU @ 2.40GHz 2.40 GHz			
	ORDENADOR 1			
Benchmarks	Base Ref Time	Base Run Time	Estimated Base Ratio	Media Geométrica:
164.gzip	1400	72,4	19,34	37,05
164.gzip	1400	69,6	20,11	
164.gzip	1400	67,7	20,68	
175.vpr	1400	64	21,88	
175.vpr	1400	55,7	25,13	
175.vpr	1400	63,7	21,98	
176.gcc	1100	28,5	38,60	
176.gcc	1100	21,8	50,46	
176.gcc	1100	23,2	47,41	
181.mcf	1800	28,6	62,94	
181.mcf	1800	29,1	61,86	
181.mcf	1800	28,9	62,28	
186.crafty	1000	25,3	39,53	
186.crafty	1000	25	40,00	
186.crafty	1000	25,3	39,53	
197.parser	1800	71,9	25,03	
197.parser	1800	66,7	26,99	
197.parser	1800	61,9	29,08	
252.eon	1300	22,7	57,27	
252.eon	1300	23,1	56,28	
252.eon	1300	22,7	57,27	
253.perlbmk	1800	--	--	
254.gap	1100	26,5	41,51	
254.gap	1100	27,9	39,43	
254.gap	1100	26	42,31	
255.vortex	1900	37,1	51,21	
255.vortex	1900	42,7	44,50	
255.vortex	1900	46,4	40,95	
256.bzip2	1500	50,9	29,47	
256.bzip3	1500	52,8	28,41	
256.bzip4	1500	46,8	32,05	
300.twolf	3000	73,9	40,60	
300.twolf	3000	73	41,10	
300.twolf	3000	72,8	41,21	

En la siguiente gráfica mostraremos de forma resumida los datos obtenidos por cada uno de los computadores de los componentes del grupo, como podemos observar se ve el pico de rendimiento asociado al ordenador 6, lo que se asocia a una mayor velocidad de computación, también podemos observar resultados muy similares en los ordenadores 1,2 y 4 ya que todos ellos poseen un hardware muy parecido, ya que aunque las máquinas 1 y 4 poseen la misma cantidad de hilos y núcleos pero compensados con la mayor cantidad de RAM del segundo ordenador hace que sus resultados sean muy similares, Es normal obtener resultados diferentes de rendimiento ya que el benchmark utilizado es sintético.

Una de las conclusiones a la que hemos llegado después de realizar dichos benchmark es que uno de los factores más influyentes en las velocidades de los computadores son los núcleos y los hilos que poseen nuestros procesadores, siendo el ordenador 6 el que más cantidad de núcleos e hilos respectivamente.



4. Referencias

- http://mixteco.utm.mx/~merg/AC/2009/2.4-Programas_evaluacion_rendimiento.html
- http://spec.org/
- Najafi, H. S. and Solary, M. S. (2006), 'Computational algorithms for computing the inverse of a square matrix, quasi-inverse of a non-square matrix and block matrices', Applied Mathematical Computation, 183 (2006), pp. 539-550.
- https://www.cs.buap.mx/~mgonzalez/asm_mododir2.pdf
-