2º curso / 2º cuatr.

Grado Ing. Inform.

Dobles Grados

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 4. Optimización de código

Estudiante (nombre y apellidos):
Grupo de prácticas y profesor de prácticas:

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo y se lista con lscpu): **AMD Ryzen 7 4700U with Radeon Graphics (8) @ 2.000GHz**

Sistema operativo utilizado: ***Kali GNU/Linux Rolling x86_64***

Versión de gcc utilizada: **gcc version 11.3.0 (Debian 11.3.0-3)**

Volcado de pantalla que muestre lo que devuelve `lscpu` en la máquina en la que ha tomado las medidas:

```
┌──(icaro㉿kali)-[~]
└─$ lscpu
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         48 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  8
  On-line CPU(s) list:   0-7
Vendor ID:               AuthenticAMD
  Model name:            AMD Ryzen 7 4700U with Radeon Graphics
    CPU family:          23
    Model:               96
    Thread(s) per core:  1
    Core(s) per socket:  8
    Socket(s):           1
    Stepping:            1
    Frequency boost:     enabled
    CPU max MHz:         2000,0000
    CPU min MHz:         1400,0000
    BogoMIPS:            3992.21
    Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr s
                         se sse2 ht syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm constant_tsc rep_good nopl nonstop
                         _tsc cpuid extd_apicid aperfmperf rapl pni pclmulqdq monitor ssse3 fma cx16 sse4_1 sse4_2
                         movbe popcnt aes xsave avx f16c rdrand lahf_lm cmp_legacy svm extapic cr8_legacy abm sse4a
                          misalignsse 3dnowprefetch osvw ibs skinit wdt tce topoext perfctr_core perfctr_nb bpext p
                         erfctr_llc mwaitx cpb cat_l3 cdp_l3 hw_pstate ssbd mba ibrs ibpb stibp vmmcall fsgsbase bm
                         i1 avx2 smep bmi2 cqm rdt_a rdseed adx smap clflushopt clwb sha_ni xsaveopt xsavec xgetbv1
                          xsaves cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_local clzero irperf xsaveerptr rdpru w
                         bnoinvd arat npt lbrv svm_lock nrip_save tsc_scale vmcb_clean flushbyasid decodeassists pa
                         usefilter pfthreshold avic v_vmsave_vmload vgif v_spec_ctrl umip rdpid overflow_recov succ
                         or smca
Virtualization features:
  Virtualization:        AMD-V
Caches (sum of all):
  L1d:                   256 KiB (8 instances)
  L1i:                   256 KiB (8 instances)
  L2:                    4 MiB (8 instances)
  L3:                    8 MiB (2 instances)
NUMA:
  NUMA node(s):          1
  NUMA node0 CPU(s):     0-7
Vulnerabilities:
  Itlb multihit:         Not affected
  L1tf:                  Not affected
  Mds:                   Not affected
  Meltdown:              Not affected
  Spec store bypass:     Mitigation; Speculative Store Bypass disabled via prctl and seccomp
  Spectre v1:            Mitigation; usercopy/swapgs barriers and __user pointer sanitization
  Spectre v2:            Mitigation; Full AMD retpoline, IBPB conditional, IBRS_FW, STIBP disabled, RSB filling
  Srbds:                 Not affected
  Tsx async abort:       Not affected
```

1. Modificar el código secuencial para la multiplicación de matrices disponible en SWAD (solo el trozo que calcula la multiplicación) para reducir el tiempo de ejecución. Justificar los tiempos obtenidos (usando siempre -O2) a partir de la modificación realizada. Incorporar los códigos modificados en el cuaderno.

MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación A) –explicación-:

He intercambiado los bucles j y k que calculan el producto de matrices , de forma que se pueda salvar provecho a la contigüidad de los datos en memoria (Aprovechar la localidad espacial)

Modificación B) –explicación-:

En el calculo del producto de matrices he realizado un desenrollado de bucle en el que por cada iteracion se va a calcular el valor de cuatro elementos de la matriz resultado.
Con el desenrollado se elimina la dependencia RAW con los retardos provocados por los LOAD/STORES consecutivos disminuyen

CÓDIGOS FUENTE MODIFICACIONES

A) Captura de `pmm-secuencial-modificado_A.c`

```
    /*
     * ZONA MULTIPLICACION PARA MODIFICAR
     * MODIFICACION A
     */

    // Calcular m3 = m1 * m2

    clock_gettime(CLOCK_REALTIME,&cgt1);
    /*
    for(i = 0; i < N; i++){
        for (j = 0; j < N; j++)
            for (k = 0; k < N; k++)
                m3[i][j] += m1[i][k] * m2[k][j];
    }
    */

    for (i = 0 ; i < N; ++i){
         for (k  = 0; k < N; ++k)
                 for (j = 0; j < N; ++j)
                         m3[i][j] = m1[i][k] + m2[k][j];
    }

    //

    clock_gettime(CLOCK_REALTIME,&cgt2);
    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
        (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
```

Capturas de pantalla (que muestren la compilación y que el resultado es correcto):

```
┌──(icaro㉿kali)-[~/…/Practica/Seminario4/CODIGO/bp4]
└─$ gcc  -O2  pmm-secuencial-modificado-a.c -o pmm-secuencial-a -lrt


┌──(icaro㉿kali)-[~/…/Practica/Seminario4/CODIGO/bp4]
└─$ ./pmm-secuencial-a 1000
Tiempo: 0.731830570      Tamaño: 1000    m3[0][0]: 1098.900000   m3[N-1][N-1]: 1099998.900000

┌──(icaro㉿kali)-[~/…/Practica/Seminario4/CODIGO/bp4]
└─$ ./pmm-secuencial 1000
Tiempo: 1.001622608      Tamaño: 1000    m3[0][0]: 33283350.000000       m3[N-1][N-1]: 99899933383350.000000
```

B)

```
/*
 *  ZONA MULTIPLICACION ZONA A MODIFICAR
 *  DESENROLLAR
 */

// Calcular m3 = m1 * m2
clock_gettime(CLOCK_REALTIME,&cgt1);
/*
for(i = 0; i < N; i++){
    for (j = 0; j < N; j++)
        for (k = 0; k < N; k++)
            m3[i][j] += m1[i][k] * m2[k][j];
}
*/

for (i = 0; i < N; i+= 4){
    for (j = 0; j < N; j+=4){
            for (k = 0; k < N; k++){
                    m3[i][j] += m1[i][k] * m2[k][j];
                    m3[i+1][j+1] += m1[i+1][k] * m2[k][j+1];
                    m3[i+2][j+2] += m1[i+2][k] * m2[k][j+2];
                    m3[i+3][j+3] += m1[i+3][k] * m2[k][j+3];

            }
    }
}
```

```
┌──(icaro㉿kali)-[~/…/Practica/Seminario4/CODIGO/bp4]
└─$ gcc  -O2  pmm-secuencial-modificado-b.c -o pmm-secuencial-b -lrt


┌──(icaro㉿kali)-[~/…/Practica/Seminario4/CODIGO/bp4]
└─$ ./pmm-secuencial-b 1000
Tiempo: 0.273129798      Tamaño: 1000    m3[0][0]: 33283350.000000       m3[N-1][N-1]: 99899933383350.000000

┌──(icaro㉿kali)-[~/…/Practica/Seminario4/CODIGO/bp4]
└─$ ./pmm-secuencial 1000
Tiempo: 1.010135195      Tamaño: 1000    m3[0][0]: 33283350.000000       m3[N-1][N-1]: 99899933383350.000000
```

TIEMPOS: para n = 1000

Modificación	**Breve descripción de las modificaciones**	**-O2**
Sin modificar		*1.001622608*
Modificación A)	Intercambio los bucles j y k	0.731830570
Modificación B)	Desenrollo el bucle, calculo de 4 elementos/iteración	0.273129798
...		

COMENTARIOS SOBRE LOS RESULTADOS Y JUSTIFICACIÓN DE LAS MEJORAS EN TIEMPO:
Los codigos se han implementado tomando como ejemplo n = 1000.

Modificacion A:
He realizado un cambio en la forma de acceder a los datos “Localidad de acceso”. Ahora se aprovecha mejor como el compilador los almacena, aprovecha la localidad. Podríamos entender una matriz como si fuese un array para el cual existen una serie de punteros donde indican donde comienza cada fila. Con la modificación realizada lo que conseguimos es movernos por las columnas de la matriz y sacar provecho de la localidad de estos datos
Si se hubiese mantenido como en el original, lo que conseguiría sera un mayor numero de cambios de filas y por consiguiente un tiempo mayor para obtener cada resultado.

Modificacion B:
Reducimos el numero de saltos/iteraciones aplicando el desenrollado de bucles. Las variables als incremento en 4 en 4 . Por un lado el tamaño del codigo aumenta, pero por otro lado al eliminar las dependencias RAW, los stalls (atascos memoria) provocado por los load y store localizados cerca disminuyen.

2. Usar en este ejercicio el programa secuencial disponible en SWAD que utiliza como base el código de la Figura 1. Modificar en el programa el código mostrado en la Figura 1 para reducir el tiempo de ejecución. Justificar los tiempos obtenidos (usando siempre -O2) a partir de la modificación realizada. En las ejecuciones de evaluación usar valores de N y M mayores que 1000. Incorporar los códigos modificados en el cuaderno.

Figura 1 . Código C++ que suma dos vectores. `M` y `N` deben ser parámetros de entrada al programa, usar valores mayores que 1000 en la evaluación.

```
struct nombre {
        int a;
        int b;
}  s[N];

main()
{
   …
   for (ii=0; ii<M;ii++) {
      X1=0; X2=0;
      for(i=0; i<N;i++)  X1+=2*s[i].a+ii;
      for(i=0; i<N;i++)  X2+=3*s[i].b-ii;

      if (X1<X2)  R[ii]=X1  else  R[ii]=X2;
   }
   …
}
```

MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación A) –explicación-:

Descartamos el segundo for que realiza el calculo de X2 deido a que se ejecuta las mismas veces que el de X1 y no hay dependencias entre esas dos variables, de forma que junto el mismo bucle y calculo.

Se unen los bucles por que el struct esta definido con a y b dispuestos en memoria de manera correlativa (menos fallos de caché)

Modificación B) –explicación-:

Aplico un desenrollado de bucle. Reducimos el tiempo de acceso a los datos aprovechando su localidad espacial y accediendo a aquellos que están más contiguos. Se sacrifican el tamaño del codigo, pero los resultados son mejores, elimino la dependencia RAW.

CÓDIGOS FUENTE MODIFICACIONES

A) Captura figura1-modificado_A.c

```
// ZONA A MODIFICAR

clock_gettime(CLOCK_REALTIME,&cgt1);
    for (ii=0; ii<M;ii++){
            X1=0; X2=0;
            //for(i=0; i<N;i++) X1 += 2*s[i].a + ii;
            //for(i=0; i<N;i++) X2 += 3*s[i].b - ii;
            for (i = 0; i < N; ++i){
                    X1+=2*s[i].a + ii;
                    X2+=2*s[i].b - ii;
            }

            if (X1<X2) {R[ii]=X1;} else {R[ii]=X2;}
    }
clock_gettime(CLOCK_REALTIME,&cgt2);
```

Capturas de pantalla (que muestren la compilación y que el resultado es correcto):

```
┌──(icaro㉿kali)-[~/…/Practica/Seminario4/CODIGO/bp4]
└─$ ./figura-original 1000 1000
Tiempo: 0.002782018
Elemento 0 y 999 de R: -202208, -844581

┌──(icaro㉿kali)-[~/…/Practica/Seminario4/CODIGO/bp4]
└─$ ./figura-modificado-a 1000 1000
Tiempo: 0.002184905
Elemento 0 y 999 de R: -198844, -896524
```

B)

```
┌──(icaro㉿kali)-[~/…/Practica/Seminario4/CODIGO/bp4]
└─$ gcc  -O2  figura1-modificado-b.c -o figura-modificado-b -lrt


┌──(icaro㉿kali)-[~/…/Practica/Seminario4/CODIGO/bp4]
└─$ ./figura-modificado-b 1000 1000
Tiempo: 0.000672279
Elemento 0 y 999 de R: -201902, -898422
```

```
// ZONA A MODIFICAR

int X1_1,X1_2,X1_3;
int X2_1,X2_2,X2_3;

clock_gettime(CLOCK_REALTIME,&cgt1);
    for (ii=0; ii<M;ii++){
            X1=X1_1=X1_2=X1_3=0; X2=X2_1=X2_2=X2_3=0;

            //for(i=0; i<N;i++) X1 += 2*s[i].a + ii;
            //for(i=0; i<N;i++) X2 += 3*s[i].b - ii;

            for(i=0; i<N;i+=4){
                    X1 += 2*s[i].a + ii;
                    X1_1+= 2*s[i+1].a + ii;
                    X1_2+= 2*s[i+2].a + ii;
                    X1_3+= 2*s[i+3].a + ii;
            }

            X1=X1 + X1_1 + X1_2 + X1_3;

            for(i=0; i<N;i+=4) {
                    X2 += 2*s[i].b - ii;
                    X2_1+= 2*s[i+1].b - ii;
                    X2_2+= 2*s[i+2].b - ii;
                    X2_3+= 2*s[i+3].b - ii;
            }

            X2=X2 + X2_1 + X2_2 + X2_3;

            if (X1<X2) {R[ii]=X1;} else {R[ii]=X2;}
    }
clock_gettime(CLOCK_REALTIME,&cgt2);
```

TIEMPOS:

Modificación	**Breve descripción de las modificaciones**	**-O2**
Sin modificar		*0.002782018*
Modificación A)	Calcular X1 y X2 en un mismo bucle	0.002184905
Modificación B)	Desenrollado de bucle	0.000672279
...		

COMENTARIOS SOBRE LOS RESULTADOS Y JUSTIFICACIÓN DE LAS MEJORAS EN TIEMPO:

Modificacion a:
Unimos ambos bucles para sacar partido a la localidad espacial de los accesos. La forma en la que esta declarado el struct permite que los datos a y b estén dispuestos en memoria de manera correlatica permitiendo que se den menos fallos de cache y sea optimo un acceso a ambos bajo el mismo bucle

Modificacion b:
Desarrollo un desenrollado de bucle como en pmm-secuencial y con ello reduzco el numero de saltos, aquí he necesitado variables locales X1_ 1 …. para evitar las dependencias RAW y mejorar el rendimiento

3. El benchmark Linpack ha sido uno de los programas más ampliamente utilizados para evaluar las prestaciones de los computadores. De hecho, se utiliza como base en la lista de los 500 computadores más rápidos del mundo (el Top500 Report). El núcleo de este programa es una rutina que opera con flotantes de doble precisión denominada `DAXPY` (***D**ouble precision- real **A**lpha **X** **P**lus **Y***) que multiplica un vector por una constante y los suma a otro vector (Lección 3/Tema 1):

```
for (i=0;i<N;i++) y[i]= a*x[i] + y[i];
```

A partir del programa `DAXPY` disponible en SWAD, generar los programas en ensamblador para cada una de las siguientes opciones de optimización del compilador: -O0, -Os, -O2, -O3. Explique las diferencias que se observan en el código justificando al mismo tiempo las mejoras en velocidad que acarrean. Incorporar los códigos al cuaderno de prácticas y destacar las diferencias entre ellos. Sólo se debe evaluar el tiempo del núcleo `DAXPY.N` deben ser parámetro de entrada al programa.

CAPTURA CÓDIGO FUENTE: `daxpy.c`

```c
int main(int argc, char** argv){

  int i;

  struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución

  //Leer argumento de entrada (nº de componentes del vector)
  if (argc<3){
    printf("Faltan argumentos de entrada (n. componentes, alpha)");
    exit(-1);
  }

  int N = atoi(argv[1]);        // Máximo N =2^32-1=4294967295 (sizeof(int) = 4 B)
  double alpha = atof(argv[2]);
  #ifdef VECTOR_LOCAL
  double x[N], y[N], z[N];   // Tamaño variable local en tiempo de ejecución ...
                             // disponible en C a partir de C99
  #endif
  #ifdef VECTOR_GLOBAL
  if (N>MAX) N=MAX;
  #endif
  #ifdef VECTOR_DYNAMIC
  float *x, *y, *z;
  x = (float*) malloc(N*sizeof(float));// malloc necesita el tamaño en bytes
  y = (float*) malloc(N*sizeof(float));
  z = (float*) malloc(N*sizeof(float));
  #endif

  //Inicializar vectores
  if (N < 9)
    for (i = 0; i < N; i++)
    {
      x[i] = N * 0.1 + i * 0.1; y[i] = N * 0.1 - i * 0.1;
    }
  else
  {
    //srand(time(0));
    for (i = 0; i < N; i++)
    {
      x[i] = drand48();
      y[i] = drand48();
    }
  }

  clock_gettime(CLOCK_REALTIME,&cgt1);
  //Cálculos daxpyz
  for(i=0; i<N; i++)
    z[i] = alpha*x[i] + y[i];

  clock_gettime(CLOCK_REALTIME,&cgt2);
  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
       (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
```

Tiempos ejec. Longitud vectores N = 107374182	**-O0**	**-Os**	**-O2**	**-O3**
	0.157637456	0.113226048	0.094778493	0.088063316

CAPTURAS DE PANTALLA (que muestren la compilación y que el resultado es correcto)**:**

```
┌──(icaro㉿kali)-[~/…/Practica/Seminario4/CODIGO/bp4]
└─$ gcc -O0 daxpy.c -o daxpy -lrt

┌──(icaro㉿kali)-[~/…/Practica/Seminario4/CODIGO/bp4]
└─$ ./daxpy 107374182 1000
Tiempo:0.157637456       / Tamaño Vectores:33554432     / alpha*x[0]+y[0]=z[0](1000.000000*0.000000+0.000985=0.00098

┌──(icaro㉿kali)-[~/…/Practica/Seminario4/CODIGO/bp4]
└─$ gcc -Os daxpy.c -o daxpy -lrt

┌──(icaro㉿kali)-[~/…/Practica/Seminario4/CODIGO/bp4]
└─$ ./daxpy 107374182 1000
Tiempo:0.113226048       / Tamaño Vectores:33554432     / alpha*x[0]+y[0]=z[0](1000.000000*0.000000+0.000985=0.00098

┌──(icaro㉿kali)-[~/…/Practica/Seminario4/CODIGO/bp4]
└─$ gcc -O2 daxpy.c -o daxpy -lrt

┌──(icaro㉿kali)-[~/…/Practica/Seminario4/CODIGO/bp4]
└─$ ./daxpy 107374182 1000
Tiempo:0.094778493       / Tamaño Vectores:33554432     / alpha*x[0]+y[0]=z[0](1000.000000*0.000000+0.000985=0.00098
5) / / alpha*x[33554431]+y[33554431]=z[33554431](1000.000000*0.037781+0.434614=38.215512) /

┌──(icaro㉿kali)-[~/…/Practica/Seminario4/CODIGO/bp4]
└─$ gcc -O3 daxpy.c -o daxpy -lrt

┌──(icaro㉿kali)-[~/…/Practica/Seminario4/CODIGO/bp4]
└─$ ./daxpy 107374182 1000
Tiempo:0.088063316       / Tamaño Vectores:33554432     / alpha*x[0]+y[0]=z[0](1000.000000*0.000000+0.000985=0.00098
5) / / alpha*x[33554431]+y[33554431]=z[33554431](1000.000000*0.037781+0.434614=38.215512) /
```

https://www.rapidtables.com/code/linux/gcc/gcc-o.html

COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:
En comparación con la optimización O2 donde el compilador intenta aumentar el rendimiento del código sin comprometes el tamaño sin tomar demasiado tiempo para compilar, la bandera -O0 se caracteriza por no realizar ninguna optimización exhibiendo un código ensamblador bastante denso y con un alto tiempo de ejecución.

La optimización Os realiza mejora en lo referente a la longitud del código ensamblador obtenido, sin embargo no optimiza nada relacionado con el tiempo de ejecución.

Finalmente el flag -03 activa optimizaciones que implican una alta demanda en uso de memoria. No garantiza mejoras en el rendimiento, y de hecho, el sistema se puede resentir debido al uso abusivo de memoria.

El conclusión, el flag mas optimo para realizar una optimización de código es -O2.

CÓDIGO EN ENSAMBLADOR (no es necesario introducir aquí el código como captura de pantalla, ajustar el tamaño de la letra para que una instrucción no ocupe más de un renglón)**:**
(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR DONDE ESTÁ EL CÓDIGO EVALUADO, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

gcc -O0 -S daxpy.c -o daxpy0.s -lrt
gcc -Os -S daxpy.c -o daxpyOs.s -lrt
gcc -O2 -S daxpy.c -o daxpyO2.s -lrt
gcc -O3 -S daxpy.c -o daxpyO3.s -lrt

daxpyO0.s	daxpyOs.s	daxpyO2.s	daxpyO3.s

call clock_gettime@PLT
movl $0, -4(%rbp)
jmp .L10
 	
.L11:
movl -4(%rbp), %eax
cltq
leaq 0(,%rax,8), %rdx
leaq x(%rip), %rax
movsd (%rdx,%rax), %xmm0
movapd %xmm0, %xmm1
mulsd -16(%rbp), %xmm1
movl -4(%rbp), %eax
cltq
leaq 0(,%rax,8), %rdx
leaq y(%rip), %rax
movsd (%rdx,%rax), %xmm0
addsd %xmm1, %xmm0
movl -4(%rbp), %eax
cltq
leaq 0(,%rax,8), %rdx
leaq z(%rip), %rax
movsd %xmm0, (%rdx, %rax)
addl $1, -4(%rbp)
 	
.L10:
movl -4(%rbp), %eax
cmpl -8(%rbp), %eax
jl .L11
leaq -64(%rbp), %rax
movq %rax, %rsi
movl $0, %edi
call clock_gettime@PLT
	call clock_gettime@PLT
	xorl %eax, %eax
	leaq z(%rip), %rdx
	leaqx (%rip), %rcx
	leaqy (%rip), %rsi
	
	.L8:
	cmpl %eax, %ebp
	jle .L19
	movq %r14, %xmm0
	mulsd (%rcx,%rax,8), %xmm0
	addsd (%rsi,%rax,8), %xmm0
	movsd %xmm0, (%rdx, %rax,8)
	incq %rax
	jmp .L8
	
	.L19:
	leaq 16(%rsp), %rsi
	xorl %edi, %edi
	call clock_gettime@PLT
		call clock_gettime@PLT
		xorl %eax, %eax
		.p2align 4,,10
		.p2align 3
		
		.L8:
		movsd 8(%rsp), %xmm0
		mulsd (%rbx,%rax,8), %xmm0
		addsd (%r12,%rax,8), %xmm0
		movsd %xmm0, 0(%r13,%rax,8)
		addq $1, %rax
		cmpl %eax, %ebp
		jg .L8
		leaq 32(%rsp), %rsi
		xorl %edi, %edi
		call clock_gettime@PLT
			call clock_gettime@PLT
			movq 40(%rsp), %rax
			pxor %xmm0, %xmm0
			subq 24(%rsp), %rax
			cvtsi2sdq %rax, %xmm0
			pxor %xmm1, %xmm1
			movq 32(%rsp), %rax
			subq 16(%rsp), %rax
			cvtsi2sdq %rax, %xmm1
			divsd .LC8(%rip), %xmm0
			addsd %xmm1, %xmm0
			cmpl $10, %r12d
			jg .L34
			movl %r12d, %esi
			leaq .LC7(%rip), %rdi
			movl $1, %eax
			xorl %r13d, %r13d
			call printf@PLT
			leaq .LC5(%rip), %r14
			.p2align 4,,10
			.p2align 3
			.L14:
			movsd (%r15,%r13,8), %xmm3
			movl %r13d, %esi
			movl

			%r13d, %ecx movl %r13d, %edx movsd 0(%rbp,%r13,8), %xmm2 movsd (%rbx,%r13,8), %xmm1 movq %r14, %rdi movl $4, %eax movsd 8(%rsp), %xmm0 addq $1, %r13 call printf@PLT cmpl %r13d, %r12d jg .L14 .L29: addq $56, %rsp .cfi_remember_state .cfi_def_cfa_offset 56 xorl %eax, %eax popq %rbx .cfi_def_cfa_offset 48 popq %rbp .cfi_def_cfa_offset 40 popq %r12 .cfi_def_cfa_offset 32 popq %r13 .cfi_def_cfa_offset 24 popq %r14 .cfi_def_cfa_offset 16 popq %r15 .cfi_def_cfa_offset 8 ret .L34: .cfi_restore_state leal -1(%r12), %edx pushq

			%rcx
			.cfi_def_cfa_offset 120
			movsdz(%rip), %xmm4
			movl %r12d, %esi
			movslq %edx, %rax
			movsdy(%rip), %xmm3
			movl %edx, %r8d
			movl %edx, %ecx
			pushq (%r15,%rax,8)
			.cfi_def_cfa_offset 128
			movsd 24(%rsp), %xmm5
			leaq .LC6(%rip), %rdi
			movsd 0(%rbp,%rax,8), %xmm7
			movsd (%rbx,%rax,8), %xmm6
			movsdx(%rip), %xmm2
			movapd %xmm5, %xmm1
			movl $8, %eax
			call printf@PLT
			popq %rsi
			.cfi_def_cfa_offset 120
			popq %rdi
			.cfi_def_cfa_offset 112
			jmp .L29
			.L7:
			xorl %edi, %edi
			leaq 16(%rsp), %rsi
			leaqz(%rip), %r15
			call clock_gettime@PLT