2º curso / 2º cuatr.

Grado Ing. Inform.

Doble Grado Ing. Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 4. Optimización de código

Estudiante (nombre y apellidos): Francisco Carrillo Pérez
Grupo de prácticas: A2
Fecha de entrega:
Fecha evaluación en clase:

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo): *Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz*

Sistema operativo utilizado: *Kubuntu 16.04*

Versión de gcc utilizada: *gcc (Ubuntu 5.3.1-14ubuntu2) 5.3.1*

Adjunte el contenido del fichero /proc/cpuinfo de la máquina en la que ha tomado las medidas

1. Para el núcleo que se muestra en la Figura 1 (ver guion de prácticas), y para un programa que implemente la multiplicación de matrices (use variables dinámicas):

   1.1 Modifique el código C para reducir el tiempo de ejecución del mismo. Justifique los tiempos obtenidos (use –O2) a partir de la modificación realizada. Incorpore los códigos modificados en el cuaderno.

   1.2 Genere los códigos en ensamblador con -O2 para el original y dos códigos modificados obtenidos en el punto anterior (incluido el que supone menor tiempo de ejecución) e incorpórelos al cuaderno de prácticas. Destaque las diferencias entre ellos en el código ensamblador.

   1.3 (Ejercicio EXTRA) Intente mejorar los resultados obtenidos transformando el código ensamblador del programa para el que se han conseguido las mejores prestaciones de tiempo

A) MULTIPLICACIÓN DE MATRICES:
CÓDIGO FUENTE: `pmm-secuencial.c`
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

int main(int argc, char** argv){
            int i, j,k;
            double t1, t2, total;

            //Leer argumento de entrada (no de componentes del vector)
            if (argc<2){
                        printf("Falta tamaño de matriz y vector\n");
```

```
                        exit(-1);
            }

            unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
(sizeof(unsigned int) = 4 B)

            double **M1, **M2, **M3;
            M1 = (double**) malloc(N*sizeof(double *));
            M2 = (double**) malloc(N*sizeof(double *));
            M3 = (double**) malloc(N*sizeof(double *));


            for (i=0; i<N; i++){
                        M1[i] = (double*) malloc(N*sizeof(double));
                        M2[i] = (double*) malloc(N*sizeof(double));
                        M3[i] = (double*) malloc(N*sizeof(double));
                        if ( M1[i]==NULL ){
                                    printf("Error en la reserva de espacio
para los vectores\n");
                                    exit(-2);
                        }
            }
            //A partir de aqui se pueden acceder las componentes de la
matriz como M[i][j]

            //Inicializar matriz y vectores

            for(i = 0; i< N; i++)
            {
                        for(j = 0; j < N; j++ )
                        {
                                    M1[i][j] = 2;
                                    M2[i][j] = 2;
                                    M3[i][j] = 0;
                        }
            }


            //Medida de tiempo
            t1 = omp_get_wtime();

            //Calcular producto de matriz por vector v2 = M · v1

            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];
            }
        }
            }


            //Medida de tiempo
            t2 = omp_get_wtime();
            total = t2 - t1;

            //Imprimir el resultado y el tiempo de ejecución
            printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f V2[%d]=
%8.6f\n", total,N,M3[0][0],N-1,M3[N-1][N-1]);


            for (i=0; i<N; i++){
```

```
                        free(M1[i]);
                        free(M2[i]);
                        free(M3[i]);
            }
            free(M1);
            free(M2);
            free(M3);

            return 0;
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: Se realiza un desenrrollado de bucle de la variable i en el bucle for anidado para el cálculo.

Modificación b) –explicación-: Además de modificación del apartado anterior, se realiza otro desenrrollado de bucle de la variable j, en el mismo bucle anidado.

...

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) `pmm-secuencial-modificado_a.c`

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

int main(int argc, char** argv){
            int i, j,k;
            double t1, t2, total;

            //Leer argumento de entrada (no de componentes del vector)
            if (argc<2){
                        printf("Falta tamaño de matriz y vector\n");
                        exit(-1);
            }

            unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
(sizeof(unsigned int) = 4 B)

            double **M1, **M2, **M3;
            M1 = (double**) malloc(N*sizeof(double *));
            M2 = (double**) malloc(N*sizeof(double *));
            M3 = (double**) malloc(N*sizeof(double *));


            for (i=0; i<N; i++){
                        M1[i] = (double*) malloc(N*sizeof(double));
                        M2[i] = (double*) malloc(N*sizeof(double));
                        M3[i] = (double*) malloc(N*sizeof(double));
                        if ( M1[i]==NULL ){
                                    printf("Error en la reserva de espacio
para los vectores\n");
                                    exit(-2);
                        }
            }
            //A partir de aqui se pueden acceder las componentes de la
matriz como M[i][j]

            //Inicializar matriz y vectores
```

```
              for(i = 0; i< N; i++)
              {
                            for(j = 0; j < N; j++ )
                            {
                                          M1[i][j] = 2;
                                          M2[i][j] = 2;
                                          M3[i][j] = 0;
                            }
              }


              //Medida de tiempo
              t1 = omp_get_wtime();

              //Calcular producto de matriz por vector v2 = M · v1

              for (i=0; i<N; i+=2){
        for (j=0; j<N; j++) {
            for (k=0; k<N; k++) {
               M3[i][j] += M1[i][k] * M2[k][j];
               M3[i+1][j] += M1[i+1][k] * M2[k][j];
            }
        }
              }


              //Medida de tiempo
              t2 = omp_get_wtime();
              total = t2 - t1;

              //Imprimir el resultado y el tiempo de ejecución
              printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f V2[%d]=
%8.6f\n", total,N,M3[0][0],N-1,M3[N-1][N-1]);


              for (i=0; i<N; i++){
                            free(M1[i]);
                            free(M2[i]);
                            free(M3[i]);
              }
              free(M1);
              free(M2);
              free(M3);

              return 0;
}
```

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

b)

```
// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

int main(int argc, char** argv){
              int i, j,k;
              double t1, t2, total;
```

```
            //Leer argumento de entrada (no de componentes del vector)
            if (argc<2){
                        printf("Falta tamaño de matriz y vector\n");
                        exit(-1);
            }

            unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
(sizeof(unsigned int) = 4 B)

            double **M1, **M2, **M3;
            M1 = (double**) malloc(N*sizeof(double *));
            M2 = (double**) malloc(N*sizeof(double *));
            M3 = (double**) malloc(N*sizeof(double *));


            for (i=0; i<N; i++){
                        M1[i] = (double*) malloc(N*sizeof(double));
                        M2[i] = (double*) malloc(N*sizeof(double));
                        M3[i] = (double*) malloc(N*sizeof(double));
                        if ( M1[i]==NULL ){
                                    printf("Error en la reserva de espacio
para los vectores\n");
                                    exit(-2);
                        }
            }
            //A partir de aqui se pueden acceder las componentes de la
matriz como M[i][j]

            //Inicializar matriz y vectores

            for(i = 0; i< N; i++)
            {
                        for(j = 0; j < N; j++ )
                        {
                                    M1[i][j] = 2;
                                    M2[i][j] = 2;
                                    M3[i][j] = 0;
                        }
            }


            //Medida de tiempo
            t1 = omp_get_wtime();

            //Calcular producto de matriz por vector v2 = M · v1

            for (i=0; i<N; i+=2){
        for (j=0; j<N; j+=2) {
            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];




            }
        }
            }


            //Medida de tiempo
```

```
            t2 = omp_get_wtime();
            total = t2 - t1;

            //Imprimir el resultado y el tiempo de ejecución
            printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f V2[%d]=
%8.6f\n", total,N,M3[0][0],N-1,M3[N-1][N-1]);


            for (i=0; i<N; i++){
                        free(M1[i]);
                        free(M2[i]);
                        free(M3[i]);
            }
            free(M1);
            free(M2);
            free(M3);

            return 0;
}
```

1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	*7.370167454*
Modificación a)	3.627575052
Modificación b)	2.412372309

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

1.2. CÓDIGO EN ENSAMBLADOR DEL ORIGINAL Y DE DOS MODIFICACIONES (ADJUNTAR AL .ZIP):
(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR EVALUADA, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

pmm-secuencial.s	pmm-secuencial-modificado_a.s	pmm-secuencial-modificado_c.s

movl 20(%rsp), %eax
movsd %xmm0, (%rsp)
xorl %r11d, %r11d
leaq 8(, %rax,8), %r10
	movl 24(%rsp), %eax
	movsd %xmm0, 8(%rsp)
	leaq 8(%rbx), %r14
	leaq 8(%rbp), %r13
	xorl %r8d, %r8d
	leaq 8(,%rax,8), %r11
		movsd %xmm0, (%rsp)
		movl 24(%rsp), %r8d
		xorl %r13d, %r13d
		movq %rbp, 24(%rsp)
		.L10:
		movq -8(%r15), %rdi
		xorl

```
.L10:
            movq
(%r12,%r11,8), %r9
            movq
0(%r13,%r11,8), %rdi
            xorl
            %ecx, %ecx
            .p2align
4,,10
            .p2align 3
.L14:
            movsd
(%r9,%rcx), %xmm1
            xorl
            %eax, %eax
            .p2align
4,,10
            .p2align 3
.L11:
            movq
(%r15,%rax,8), %rdx
            movsd
            (%rdx,
%rcx), %xmm0
            mulsd
            (%rdi,
%rax,8), %xmm0
            addq
            $1, %rax
            cmpl
            %eax,
%r14d
            addsd
            %xmm0,
%xmm1
            ja
            .L11
            movsd
            %xmm1,
(%r9,%rcx)
            addq
            $8, %rcx
            cmpq
            %rcx, %r10
            jne
            .L14
            addq
            $1, %r11
            cmpl
            %r11d,
%r14d
```

```
.L10:
                movq
                -8(%r14),
%rsi
                movq
                -8(%r13),
%r10
                xorl
                %edx, %edx
                movq
                (%r14), %rcx
                movq
                0(%r13), %r9
                .p2align
4,,10
                .p2align 3
.L14:
                xorl
                %eax, %eax
                .p2align
4,,10
                .p2align 3
.L11:
                movq
(%r12,%rax,8), %rdi
                movsd
(%r10,%rax,8), %xmm1
                movsd
                (%rdi,%rdx),
%xmm0
                mulsd
                %xmm0, %xmm1
                mulsd
                (%r9,%rax,8),
%xmm0
                addq
                $1, %rax
                cmpl
                %eax, %r15d
                addsd
                (%rsi,%rdx),
%xmm1
                movsd
                %xmm1, (%rsi,
%rdx)
                addsd
                (%rcx,%rdx),
%xmm0
                movsd
                %xmm0, (%rcx,
%rdx)
                ja
                .L11
                addq
                $8, %rdx
                cmpq
                %r11, %rdx
                jne
                .L14
                addl
                $2, %r8d
                addq
                $16, %r14
                addq
                $16, %r13
                cmpl
                %r8d, %r15d
```

```
                %edx, %edx
                xorl
                %ebp, %ebp
                movq
                -8(%r14),
%r11
                movq
                (%r15), %rsi
                movq
                (%r14), %r10
                .p2align
4,,10
                .p2align 3
.L14:
                leaq
                8(%rdx), %r9
                xorl
                %eax, %eax
                .p2align
4,,10
                .p2align 3
.L11:
                movq
(%r12,%rax,8), %rcx
                movsd
                (%rcx,%rdx),
%xmm0
                mulsd
(%r11,%rax,8), %xmm0
                addsd
                (%rdi,%rdx),
%xmm0
                movsd
                %xmm0, (%rdi,
%rdx)
                movsd
                (%rcx,%r9),
%xmm0
                mulsd
(%r10,%rax,8), %xmm0
                addq
                $1, %rax
                cmpl
                %eax, %r8d
                addsd
                8(%rsi,%rdx),
%xmm0
                movsd
                %xmm0,
8(%rsi,%rdx)
                ja
                .L11
                addl
                $2, %ebp
                addq
                $16, %rdx
                cmpl
                %ebp, %r8d
                ja
                .L14
                addl
                $2, %r13d
                addq
                $16, %r15
                addq
                $16, %r14
```

ja .L10	ja .L10	cmpl %r13d, %r8d ja .L10 movq 24(%rsp), %rbp

B) CÓDIGO FIGURA 1:

CÓDIGO FUENTE: `figura1-original.c`

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

struct
{
            int a;
            int b;
} s[5000];

main()
{
            int ii,i;
            int X1,X2;
            int tama = 1000000;
            int R[tama];
            double t1, t2, total;


            //Medida de tiempo
            t1 = omp_get_wtime();

            for(i =0; i < 5000 ; i++)
            {
                        s[i].a = 1;
                        s[i].b = 1;
            }

            for(ii=0; ii<tama;ii++)
            {
                        X1=0;X2=0;
                        for(i=0; i <5000;i++) X1+=2*s[i].a+ii;
                        for(i=0; i <5000;i++) X2+=2*s[i].b-ii;
                        if(X1 < X2) R[ii]=X1; else R[ii]=X2;
            }

            //Medida de tiempo
            t2 = omp_get_wtime();
            total = t2 - t1;

            printf("El valor de R[%u] es: %u ",tama-1,R[tama-1]);


            printf("\n\n");

            //Imprimir el resultado y el tiempo de ejecución
```

```
            printf("Tiempo(seg.):%11.9f\t \n", total);
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: Se han unificado los dos bucles for en uno solo, además se ha realizado un desenrrollado de bucle y se ha utilizado un operador ternario en lugar del if else.

Modificación b) –explicación-: Se ha realizado todo lo anteriormente comentado además de realizar desplazamientos en lugar de las multiplicaciones.

...

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) figura1**-modificado_a.c**

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

struct
{
            int a;
            int b;
} s[5000];

main()
{
            int ii,i;
            int X1,X2,X1_1,X2_1,X1_2,X2_2;
            int tama = 1000000;
            int R[tama];
            double t1, t2, total;


            //Medida de tiempo
            t1 = omp_get_wtime();

            for(i =0; i < 5000 ; i++)
            {
                        s[i].a = 1;
                        s[i].b = 1;
            }

            for(ii=0; ii<tama;ii++)
            {
                        X1=0;X2=0;
                        X1_1 =0;
                        X1_2 = 0;
                        X2_2 =0;
                        X2_1 = 0;
                        for(i=0; i <5000;i+=2)
                        {
                                    //X1
                                    X1+=2*s[i].a+ii;
                                    X1_2+=2*s[i+1].a+ii;


                                    //X2
                                    X2 +=2*s[i].b-ii;
```

```
                                        X2_2+=2*s[i+1].b-ii;

                        }
                        X1 = X1 + X1_2;
                        X2 = X2 + X2_2;
                        R[ii]= (X1 < X2) ? X1 : X2;
        }
        //Medida de tiempo
        t2 = omp_get_wtime();
        total = t2 - t1;



        printf("El valor de R[%d] es: %d",tama-1,R[tama-1]);


        printf("\n\n");

        //Imprimir el resultado y el tiempo de ejecución
        printf("Tiempo(seg.):%11.9f\t \n", total);
}
```

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

b) figura1-modificada_b.c

```
// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>
struct
{
        int a;
        int b;
} s[5000];

main()
{
        int ii,i;
        int X1,X2;
        int tama = 1000000;
        int R[tama];
        double t1, t2, total;
        //Medida de tiempo
        t1 = omp_get_wtime();

        for(i =0; i < 5000 ; i+=4)
        {
                        s[i].a = 1;
                        s[i].b = 1;
                        s[i+1].a = 1;
                        s[i+1].b = 1;
                        s[i+2].a = 1;
                        s[i+2].b = 1;
                        s[i+3].a = 1;
                        s[i+3].b = 1;
                        s[i+4].a = 1;
```

```
                        s[i+4].b = 1;
                }

                for(ii=0; ii<tama;ii++)
                {
                        X1=0;X2=0;
                        for(i=0; i <5000;i+=2)
                        {
                                //X1
                                X1+=(s[i].a << 1) +ii;
                                X1+=(s[i+1].a << 1) +ii;


                                //X2
                                X2+=(s[i].b << 1) -ii;
                                X2+=(s[i+1].b << 1)-ii;

                        }

                        R[ii]= (X1 < X2) ? X1 : X2;
                }
                //Medida de tiempo
                t2 = omp_get_wtime();

                total = t2 - t1;


                printf("El valor final de R[%d] es: %d",tama-1,R[tama-1]);


                printf("\n\n");

                //Imprimir el resultado y el tiempo de ejecución
                printf("Tiempo(seg.):%11.9f\t \n", total);
}
```

1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	*6.945207890*
Modificación a)	3.512716199
Modificación b)	3.843146893

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

1.2. CÓDIGO EN ENSAMBLADOR DEL ORIGINAL Y DE DOS MODIFICACIONES (ADJUNTAR AL .ZIP):
(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR EVALUADA, USE

COLORES PARA DESTACAR LAS DIFERENCIAS)

figura1_sinmodificar.s	figura1_modificada_a.s	figura1_modificada_b.s

movl $1, (%rax)
movl $1, 4(%rax)
addq $8, %rax
cmpq $s+40000, %rax
jne .L2
xorl %eax, %eax
ret
	movl $1, (%rax)
	movl $1, 4(%rax)
	addq $32, %rax
	movl $1, -24(%rax)
	movl $1, -20(%rax)
	movl $1, -16(%rax)
	movl $1, -12(%rax)
	movl $1, -8(%rax)
	movl $1, -4(%rax)
	movl $1, (%rax)
	movl $1, 4(%rax)
	cmpq $s+40000, %rax
	jne .L2
	xorl %eax, %eax
	ret
		movl $1, (%rax)
		movl $1, 4(%rax)
		addq $32, %rax
		movl $1, -24(%rax)
		movl $1, -20(%rax)
		movl $1, -16(%rax)
		movl $1, -12(%rax)
		movl $1, -8(%rax)
		movl $1, -4(%rax)
		movl $1, (%rax)
		movl $1, 4(%rax)
		cmpq $s+40000, %rax
		jne .L2
		xorl %eax, %eax
		ret

2. 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 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=1;i<=N,i++) y[i]= a*x[i] + y[i];
```

2.1. Genere los programas en ensamblador para cada una de las opciones de optimización del compilador (-O0, -O2, -O3) y explique las diferencias que se observan en el código justificando las mejoras en velocidad que acarrean. Incorpore los códigos al cuaderno de prácticas y destaque las diferencias entre ellos.

2.2. (Ejercicio EXTRA) Para la mejor de las opciones, obtenga los tiempos de ejecución con distintos valores de N y determine para su sistema los valores de Rmax (valor máximo del número de operaciones en coma flotante por unidad de tiempo), Nmax (valor de N para el que se consigue Rmax), y N1/2 (valor de N para el que se obtiene Rmax/2). Estime el valor de la velocidad pico (Rpico) del procesador (consulte en [4] el número de ciclos por instrucción punto flotante para la familia y modelo de procesador que está utilizando) y compárela con el valor obtenido para Rmax. -Consulte la Lección 3 del Tema 1.

CÓDIGO FUENTE: daxpy.c
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
// Compilar con -O2 y -fopenmp
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

main()
{
            const float a = 1000000000;
            int tama = 4000000;
            int *X,*Y;
            Y = (int*) malloc(tama*sizeof(int));
    X = (int*) malloc(tama*sizeof(int));
            int i;
            double t1, t2, total;

            for(i=0; i< tama; i++)
            {
                        X[i] = 2;
                        Y[i] = 2;
            }
            t1 = omp_get_wtime();

            for(int i=0; i < tama; i++)
            {
                        Y[i] = a*X[i]+Y[i];
            }

            //Medida de tiempo
            t2 = omp_get_wtime();

            total = t2 - t1;
            //Imprimir el resultado y el tiempo de ejecución
            printf("Tiempo(seg.):%11.9f\t \n", total);
            free(X);
            free(Y);

}
```

	-O0	-O2	-O3
Tiempos ejec.	*0.021178915*	0.006747928	0.002932128

CAPTURAS DE PANTALLA:

COMENTARIOS SOBRE LAS DIFERENCIAS EN ENSAMBLADOR:

CÓDIGO EN ENSAMBLADOR (ADJUNTAR AL .ZIP):
(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR DONDE ESTÁ EL CÓDIGO EVALUADO, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

daxpyO0.s

```
call            omp_get_wtime
                movq
                %xmm0, %rax
                movq
                %rax,-
24(%rbp)
                movl
                $0, -52(%rbp)
                jmp .L4
.L5:
                movl
                -52(%rbp),
%eax
                cltq
                leaq
                0(,%rax,4),
%rdx
                movq
                -40(%rbp),
%rax
                addq
                %rax, %rdx
                movl
                -52(%rbp),
%eax
                cltq
                leaq
                0(,%rax,4),
%rcx
                movq
                -32(%rbp),
%rax
                addq
                %rcx, %rax
                movl
                (%rax), %eax
                pxor
                %xmm0, %xmm0
                cvtsi2ss
                %eax, %xmm0
                movaps
                %xmm0, %xmm1
                mulss
                -48(%rbp),
%xmm1
                movl
                -52(%rbp),
%eax
                cltq
                leaq
                0(,%rax,4),
%rcx
                movq
                -40(%rbp),
%rax
                addq
                %rcx, %rax
                movl
                (%rax), %eax
                pxor
                %xmm0, %xmm0
                cvtsi2ss
                %eax, %xmm0
                addss
                %xmm1, %xmm0
                cvttss2si
                %xmm0, %eax
                movl
```

daxpyO2.s

```
call            omp_get_wtime
                movsd
                %xmm0,
8(%rsp)
                xorl
                %eax, %eax
                movss
                .LC0(%rip),
%xmm2
                .p2align
4,,10
                .p2align 3
.L3:
                pxor
                %xmm0, %xmm0
                pxor
                %xmm1, %xmm1
                cvtsi2ss
                0(%rbp,%rax),
%xmm0
                cvtsi2ss
                (%rbx,%rax),
%xmm1
                mulss
                %xmm2, %xmm0
                addss
                %xmm1, %xmm0
                cvttss2si
                %xmm0, %edx
                movl
                %edx, (%rbx,
%rax)
                addq
                $4, %rax
                cmpq
                $16000000,
%rax
                jne
                .L3
                call
                omp_get_wtime
```

daxpyO3.s

```
.L6:
                call
                omp_get_wtime
                movq
                %rbx, %rax
                movsd
                %xmm0,
8(%rsp)
                andl
                $15, %eax
                shrq
                $2, %rax
                negq
                %rax
                andl
                $3, %eax
                je
                .L17
                pxor
                %xmm1, %xmm1
                movss
                .LC1(%rip),
%xmm0
                cmpl
                $1, %eax
                cvtsi2ss
                0(%rbp),
%xmm1
                movaps
                %xmm1, %xmm2
                pxor
                %xmm1, %xmm1
                mulss
                %xmm0, %xmm2
                cvtsi2ss
                (%rbx), %xmm1
                addss
                %xmm2, %xmm1
                cvttss2si
                %xmm1, %edx
                movl
                %edx, (%rbx)
                je
                .L18
                pxor
                %xmm1, %xmm1
                cmpl
                $3, %eax
                cvtsi2ss
                4(%rbp),
%xmm1
                movaps
                %xmm1, %xmm2
                pxor
                %xmm1, %xmm1
                mulss
                %xmm0, %xmm2
                cvtsi2ss
                4(%rbx),
%xmm1
                addss
                %xmm2, %xmm1
                cvttss2si
                %xmm1, %edx
                movl
                %edx, 4(%rbx)
                jne
                .L19
```

```
                %eax, (%rdx)
                addl
                $1, -52(%rbp)
.L4:
                movl
                -52(%rbp),
%eax
                cmpl
                -44(%rbp),
%eax
                jl
                .L5
                call
                omp_get_wtime
```

```
                pxor
                %xmm1, %xmm1
                movl
                $3999997,
%r9d
                movl
                $3, %r8d
                cvtsi2ss
                8(%rbp),
%xmm1
                mulss
                %xmm0, %xmm1
                pxor
                %xmm0, %xmm0
                cvtsi2ss
                8(%rbx),
%xmm0
                addss
                %xmm1, %xmm0
                cvttss2si
                %xmm0, %edx
                movl
                %edx, 8(%rbx)
.L9:
                movl
                $4000000,
%r10d
                movl
                %eax, %ecx
                movl
                $3999996,
%r11d
                subl
                %eax, %r10d
                movl
                $999999, %esi
.L8:
                salq
                $2, %rcx
                movaps
                .LC2(%rip),
%xmm2
                leaq
                0(%rbp,%rcx),
%rdi
                xorl
                %eax, %eax
                addq
                %rbx, %rcx
                xorl
                %edx, %edx
                .p2align
4,,10
                .p2align 3
.L10:
                movdqu
                (%rdi,%rax),
%xmm0
                cvtdq2ps
                (%rcx,%rax),
%xmm1
                addl
                $1, %edx
                cvtdq2ps
                %xmm0, %xmm0
                mulps
                %xmm2, %xmm0
                addps
                %xmm1, %xmm0
```

		cvttps2dq %xmm0, %xmm0
		movaps %xmm0, (%rcx, %rax)
		addq $16, %rax
		cmpl %esi, %edx
		jb .L10
		subl %r11d, %r9d
		cmpl %r11d, %r10d
		leal (%r11,%r8), %eax
		je .L12
		pxor %xmm1, %xmm1
		movslq %eax, %rdx
		addl $1, %eax
		leaq (%rbx, %rdx,4), %rcx
		movss .LC1(%rip), %xmm0
		subl $1, %r9d
		cvtsi2ss 0(%rbp, %rdx,4), %xmm1
		movaps %xmm1, %xmm2
		pxor %xmm1, %xmm1
		mulss %xmm0, %xmm2
		cvtsi2ss (%rcx), %xmm1
		addss %xmm2, %xmm1
		cvttss2si %xmm1, %edx
		movl %edx, (%rcx)
		je .L12
		pxor %xmm1, %xmm1
		cltq
		leaq (%rbx, %rax,4), %rdx
		cvtsi2ss 0(%rbp, %rax,4), %xmm1
		mulss %xmm0, %xmm1
		pxor %xmm0, %xmm0
		cvtsi2ss (%rdx), %xmm0
		addss

		%xmm1, %xmm0 cvttss2si %xmm0, %eax movl %eax, (%rdx) .L12: call omp_get_wtime