

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): Juan Miguel Castro Guerrero

Grupo de prácticas: B3

Fecha de entrega:

Fecha evaluación en clase:

---

**Versión de gcc utilizada: 4.8.2**

**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:
  - a. Modifique el código C para reducir el tiempo de ejecución del mismo. Justifique los tiempos obtenidos a partir de la modificación realizada.
  - b. Genere los programas en ensamblador para los programas modificados obtenidos en el punto anterior considerando las distintas opciones de optimización del compilador (-O1, -O2,...) e incorpórelos al cuaderno de prácticas. Compare los tiempos de ejecución de las versiones de código ejecutable obtenidas con las distintas opciones de optimización y explique las diferencias en tiempo a partir de las características de dichos códigos. Destaque las diferencias en el código ensamblador.
  - c. (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-modificado.c

**(ADJUNTAR CÓDIGO FUENTE AL .ZIP)**

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
    int i, j, k, N;
    struct timespec cgt1, cgt2;
    double ncgt;

    printf("Introduzca el numero de filas y columnas matrices:\n");
    scanf("%d", &N);
```

```

int mat1[N][N],mat2[N][N],mul[N][N],tam=N*N;

// Inicializar y Almacenar los valores de las matrices

for(i=0;i<N;i++)
    for(j=0;j<N;j++) {
        mul[i][j]=0;
    }

for(i=0;i<N;i++){
    for(j=0;j<N;j++){
        mat1[i][j] = rand() % tam;
        mat2[j][i] = rand() % tam;
    }
}

//Realiza la multiplicación e imprime el resultado

clock_gettime(CLOCK_REALTIME,&cgt1);
for(i=0;i<N;i++)
    for(j=0;j<N;j++)
        for(k=0;k<N;k++)
            mul[i][j]+=(mat1[i]
[k]*mat2[j][k]);
clock_gettime(CLOCK_REALTIME,&cgt2);

ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));

printf("Primera componente: %d, Última componente: %d",mul[0]
[0],mul[N-1][N-1]);
printf("\nTiempo(seg) Producto Matriz-Matriz:\n");
printf("%11.9f\t",ncgt);

printf("\n");
}

```

### MODIFICACIONES REALIZADAS: Tamaño utilizado matrices: 500

**Modificación a) –explicación-:** La primera modificación que he realizado consiste en agrupar en lo posible el contenido de los diferentes for del código.

**Modificación b) –explicación-:** Para poder multiplicar fila \* fila he decidido hacer la traspuesta a la segunda matriz.

Modificación	-O0	-O1	-O2	-O3	-Os
Sin modificar	0.763548085	0.188499689	0.177637695	0.151474117	0.263259938
Modificación a)	0.755673739	0.195655054	0.176280358	0.177787119	0.261475643
Modificación b)	0.610273401	0.089179022	0.107572498	0.058626455	0.258224697

**COMENTARIOS SOBRE LOS RESULTADOS:** Haciendo la traspuesta de la matriz(modificación b) se consiguen mejores resultados en las diferentes optimizaciones.

## CAPTURAS DE PANTALLA:

```

juanmi@juanmi-K53SD: ~/Escritorio/AC/Prácticas/P4$ gcc -O0 ej1.c -o ej1 -lrt
juanmi@juanmi-K53SD:~/Escritorio/AC/Prácticas/P4$ ./ej1 500
Primer componente: -833636401, Última componente: 149502353
Tiempo(seg) Producto Matriz-Matriz:
0.177310270
juanmi@juanmi-K53SD:~/Escritorio/AC/Prácticas/P4$ gcc -O1 ej1.c -o ej1 -lrt
juanmi@juanmi-K53SD:~/Escritorio/AC/Prácticas/P4$ ./ej1 500
Primer componente: -833636401, Última componente: 149502353
Tiempo(seg) Producto Matriz-Matriz:
0.190644211
juanmi@juanmi-K53SD:~/Escritorio/AC/Prácticas/P4$ gcc -O2 ej1.c -o ej1 -lrt
juanmi@juanmi-K53SD:~/Escritorio/AC/Prácticas/P4$ ./ej1 500
Primer componente: -833636401, Última componente: 149502353
Tiempo(seg) Producto Matriz-Matriz:
0.177310270
juanmi@juanmi-K53SD:~/Escritorio/AC/Prácticas/P4$ gcc -O3 ej1.c -o ej1 -lrt
juanmi@juanmi-K53SD:~/Escritorio/AC/Prácticas/P4$ ./ej1 500
Primer componente: -833636401, Última componente: 149502353
Tiempo(seg) Producto Matriz-Matriz:
0.140681105
juanmi@juanmi-K53SD:~/Escritorio/AC/Prácticas/P4$

```

```

juanmi@juanmi-K53SD: ~/Escritorio/AC/Prácticas/P4$ gcc -O0 ej1a.c -o ej1a -lrt
juanmi@juanmi-K53SD:~/Escritorio/AC/Prácticas/P4$ ./ej1a 500
Primer componente: -833636401, Última componente: 149502353
Tiempo(seg) Producto Matriz-Matriz:
0.59560819
juanmi@juanmi-K53SD:~/Escritorio/AC/Prácticas/P4$ gcc -O1 ej1a.c -o ej1a -lrt
juanmi@juanmi-K53SD:~/Escritorio/AC/Prácticas/P4$ ./ej1a 500
Primer componente: -833636401, Última componente: 149502353
Tiempo(seg) Producto Matriz-Matriz:
0.088511031
juanmi@juanmi-K53SD:~/Escritorio/AC/Prácticas/P4$ gcc -O2 ej1a.c -o ej1a -lrt
juanmi@juanmi-K53SD:~/Escritorio/AC/Prácticas/P4$ ./ej1a 500
Primer componente: -833636401, Última componente: 149502353
Tiempo(seg) Producto Matriz-Matriz:
0.107672035
juanmi@juanmi-K53SD:~/Escritorio/AC/Prácticas/P4$ gcc -O3 ej1a.c -o ej1a -lrt
juanmi@juanmi-K53SD:~/Escritorio/AC/Prácticas/P4$ ./ej1a 500
Primer componente: -833636401, Última componente: 149502353
Tiempo(seg) Producto Matriz-Matriz:
0.064718332
juanmi@juanmi-K53SD:~/Escritorio/AC/Prácticas/P4$ gcc -O5 ej1a.c -o ej1a -lrt
juanmi@juanmi-K53SD:~/Escritorio/AC/Prácticas/P4$ ./ej1a 500
Primer componente: -833636401, Última componente: 149502353
Tiempo(seg) Producto Matriz-Matriz:
0.255608225
juanmi@juanmi-K53SD:~/Escritorio/AC/Prácticas/P4$

```

## B) CÓDIGO FIGURA 1:

**CÓDIGO FUENTE:** figura1-modificado.c  
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

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

main()
{
    struct timespec cgt1,cgt2;
    double ncgt;
    int ii, R[40000];

```

```

        clock_gettime(CLOCK_REALTIME,&cgt1);
        for (ii=1; ii<=40000;ii++)
        {
            int X1=0,X2=0,X3=0,X4=0,X5=0,X6=0,X7=0,X8=0,i;

            for(i=0; i<5000;i+=4){
                X1+=2*s[i].a+ii;
                X3+=2*s[i+1].a+ii;
                X5+=2*s[i+2].a+ii;
                X7+=2*s[i+3].a+ii;

                X2+=3*s[i].b-ii;
                X4+=3*s[i+1].b-ii;
                X6+=3*s[i+2].b-ii;
                X8+=3*s[i+3].b-ii;
            }

            X1 += X3+X5+X7;
            X2 += X4+X6+X8;

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

        ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));

        printf("Tiempo(seg):%11.9f\t\n",ncgt);
        printf("Primera componente: %d, Última componente:
%d",R[0],R[39999]);
        printf("\n");
    }

```

**MODIFICACIONES REALIZADAS:**

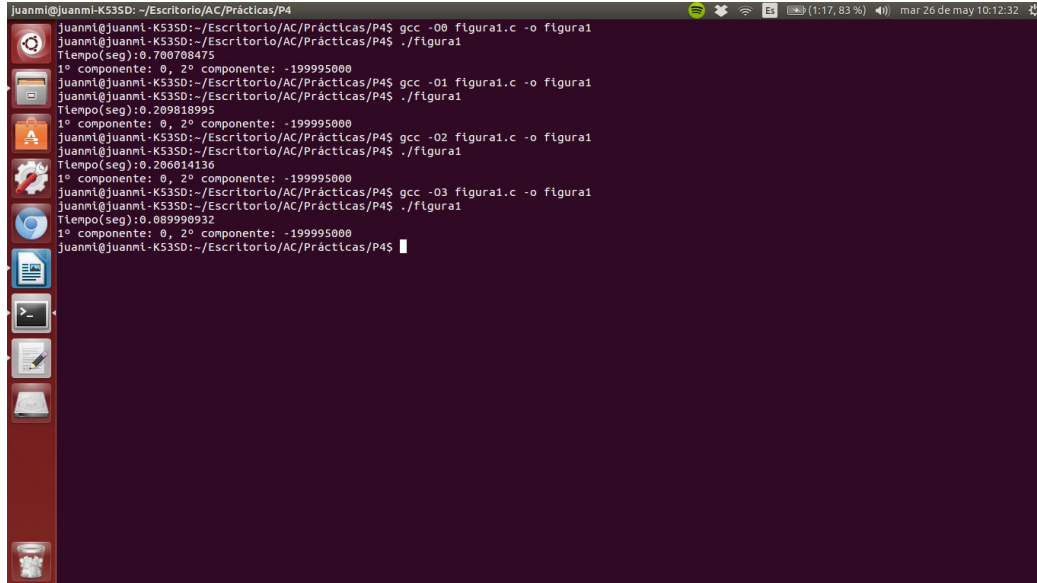
**Modificación a) –explicación-:** He eliminado el segundo for del código de la figura 1 y he incluido el código que había dentro de ese for en el for anterior.

**Modificación b) –explicación-:** Desenrollado de bucles.

Modificación	-O0	-O1	-O2	-O3	-Os
Sin modificar	0.992717773	0.284429039	0.281609708	0.121730404	0.271823287
Modificación a)	0.685848846	0.200049017	0.201895317	0.088642039	0.173601949
Modificación b)	0.524325507	0.157767652	0.154575201	0.071185975	0.152671358

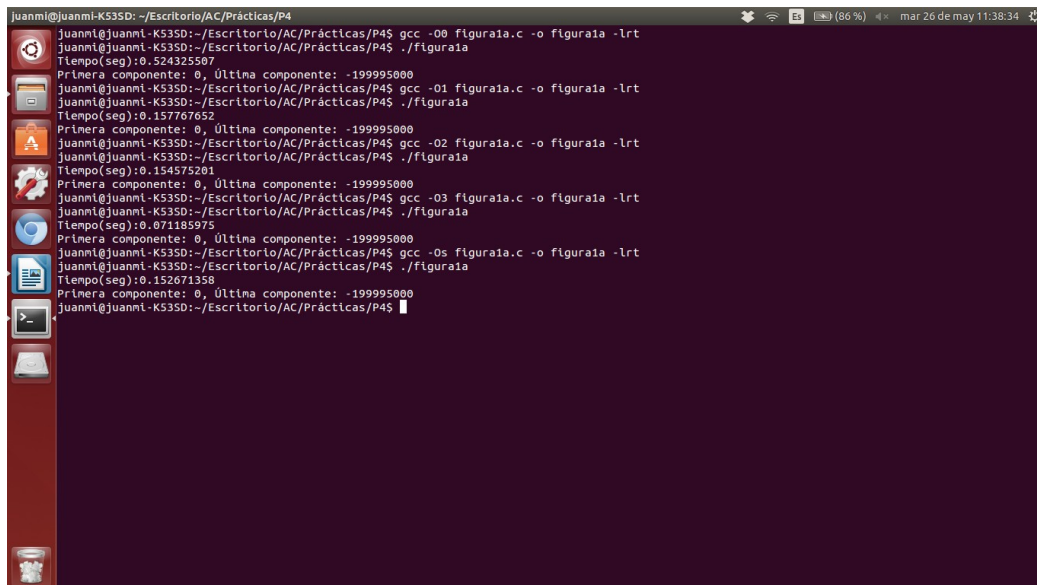
**COMENTARIOS SOBRE LOS RESULTADOS:** Con el desenrollado de bucles(modificación b) se obtienen mejores tiempos respecto a la modificación a(agrupación de bucles for).

## CAPTURAS DE PANTALLA:



```

juanmi@juanmi-K53SD: ~/Escritorio/AC/Prácticas/P4$ gcc -O0 figura1.c -o figura1
juanmi@juanmi-K53SD: ~/Escritorio/AC/Prácticas/P4$ ./figura1
Tiempo(seg):0.700708475
1º componente: 0, 2º componente: -199995000
juanmi@juanmi-K53SD: ~/Escritorio/AC/Prácticas/P4$ gcc -O1 figura1.c -o figura1
juanmi@juanmi-K53SD: ~/Escritorio/AC/Prácticas/P4$ ./figura1
Tiempo(seg):0.209818995
1º componente: 0, 2º componente: -199995000
juanmi@juanmi-K53SD: ~/Escritorio/AC/Prácticas/P4$ gcc -O2 figura1.c -o figura1
juanmi@juanmi-K53SD: ~/Escritorio/AC/Prácticas/P4$ ./figura1
Tiempo(seg):0.206014136
1º componente: 0, 2º componente: -199995000
juanmi@juanmi-K53SD: ~/Escritorio/AC/Prácticas/P4$ gcc -O3 figura1.c -o figura1
juanmi@juanmi-K53SD: ~/Escritorio/AC/Prácticas/P4$ ./figura1
Tiempo(seg):0.089990932
1º componente: 0, 2º componente: -199995000
juanmi@juanmi-K53SD: ~/Escritorio/AC/Prácticas/P4$
  
```



```

juanmi@juanmi-K53SD: ~/Escritorio/AC/Prácticas/P4$ gcc -O0 figura1a.c -o figura1a -lrt
juanmi@juanmi-K53SD: ~/Escritorio/AC/Prácticas/P4$ ./figura1a
Tiempo(seg):0.524325307
Primer componente: 0, Última componente: -199995000
juanmi@juanmi-K53SD: ~/Escritorio/AC/Prácticas/P4$ gcc -O1 figura1a.c -o figura1a -lrt
juanmi@juanmi-K53SD: ~/Escritorio/AC/Prácticas/P4$ ./figura1a
Tiempo(seg):0.157767652
Primer componente: 0, Última componente: -199995000
juanmi@juanmi-K53SD: ~/Escritorio/AC/Prácticas/P4$ gcc -O2 figura1a.c -o figura1a -lrt
juanmi@juanmi-K53SD: ~/Escritorio/AC/Prácticas/P4$ ./figura1a
Tiempo(seg):0.154575201
Primer componente: 0, Última componente: -199995000
juanmi@juanmi-K53SD: ~/Escritorio/AC/Prácticas/P4$ gcc -O3 figura1a.c -o figura1a -lrt
juanmi@juanmi-K53SD: ~/Escritorio/AC/Prácticas/P4$ ./figura1a
Tiempo(seg):0.071185975
Primer componente: 0, Última componente: -199995000
juanmi@juanmi-K53SD: ~/Escritorio/AC/Prácticas/P4$ gcc -O5 figura1a.c -o figura1a -lrt
juanmi@juanmi-K53SD: ~/Escritorio/AC/Prácticas/P4$ ./figura1a
Tiempo(seg):0.152671358
Primer componente: 0, Última componente: -199995000
juanmi@juanmi-K53SD: ~/Escritorio/AC/Prácticas/P4$
  
```

- 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 (*Double precision- real Alpha X Plus 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];

- a. Genere los programas en ensamblador para cada una de las opciones de optimización del compilador (-O1, -O2,..) y explique las diferencias que se observan en el código justificando las mejoras en velocidad que acarrearán. Incorpore los códigos al cuaderno de prácticas y destaque las diferencias entre ellos.
- b. (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)**

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(int argc, char **argv)
{
    if(argc < 3) {
        fprintf(stderr, "Falta valor N y/o valor a\n");
        exit(-1);
    }

    int i, N, a;
    struct timespec cgt1, cgt2;
    double ncgt;

    N = atoi(argv[1]);
    a = atoi(argv[2]);

    int x[N], y[N];

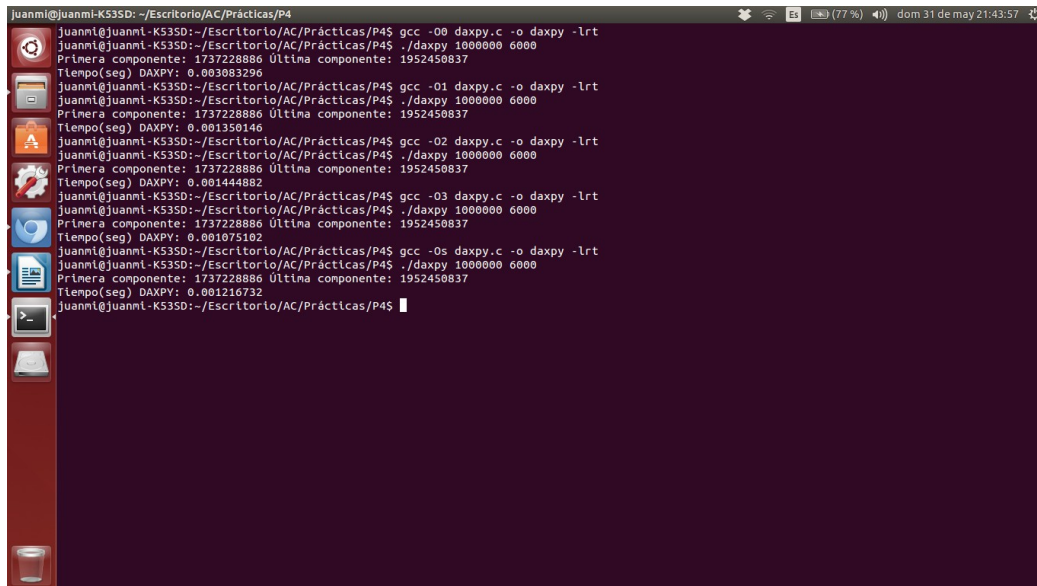
    for(i=0; i<N; i++){
        x[i] = rand() % N;
        y[i] = rand() % N;
    }

    clock_gettime(CLOCK_REALTIME, &cgt1);
    for (i=0; i<N; i++) y[i]= a*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));

    printf("Primera componente: %d Última componente: %d", y[0], y[N-
1]);
    printf("\nTiempo(seg) DAXPY: %11.9f\t\n", ncgt);
}
```

Tiempos ejec.	-O0	-O1	-O2	-O3	-Os
	0.003083296	0.001350146	0.001444882	0.001075102	0.001216732

**CAPTURAS DE PANTALLA:**

**COMENTARIOS SOBRE LAS DIFERENCIAS EN ENSAMBLADOR:** Si utilizamos la optimización -O0 el código no se optimizará, si utilizamos la optimización -O1 el compilador intenta obtener un código rápido y pequeño de nuestro programa, si utilizamos la optimización -O2 el compilador intentará aumentar el rendimiento del código sin comprometer el tamaño y sin tomar mucho más tiempo de compilación, la optimización -O3 activa optimizaciones que son caras en términos de tiempo de compilación y uso de memoria y la optimización -Os optimizará el tamaño del código.

**CÓDIGO EN ENSAMBLADOR: (ADJUNTAR AL .ZIP)**

daxpyO0.s

```

call    clock_gettime
        movl    $0, -132(%rbp)
        jmp     .L5
.L6:
        movq    -112(%rbp), %rax
        movl    -132(%rbp), %edx
        movslq   %edx, %rdx
        movl    (%rax,%rdx,4), %eax
        imull    -124(%rbp), %eax
        movl    %eax, %ecx
        movq    -96(%rbp), %rax
        movl    -132(%rbp), %edx
        movslq   %edx, %rdx
        movl    (%rax,%rdx,4), %eax
        addl    %eax, %ecx
        movq    -96(%rbp), %rax
        movl    -132(%rbp), %edx
        movslq   %edx, %rdx
        movl    %ecx, (%rax,%rdx,4)
        addl    $1, -132(%rbp)
.L5:

```

```

movl    -132(%rbp), %eax
cmpl    -128(%rbp), %eax
jl      .L6
leaq    -64(%rbp), %rax
movq    %rax, %rsi
movl    $0, %edi
call    clock_gettime

```

## daxpyO1.s

```

call    clock_gettime
movl    $0, %eax
jmp     .L7
.L3:
leaq    -80(%rbp), %rsi
movl    $0, %edi
call    clock_gettime
.L8:
leaq    -64(%rbp), %rsi
movl    $0, %edi
call    clock_gettime

```

## daxpyO2.s

```

call    clock_gettime
xorl    %eax, %eax
.p2align 4,,10
.p2align 3
.L7:
movl    (%r12,%rax,4), %edx
imull   %r13d, %edx
addl    %edx, (%r15,%rax,4)
addq    $1, %rax
cmpl    %eax, %ebx
jg      .L7
.L8:
leaq    -64(%rbp), %rsi
xorl    %edi, %edi
call    clock_gettime
movq    -64(%rbp), %rax
subq    -80(%rbp), %rax
movl    $.LC2, %esi
movl    -96(%rbp), %r14d
movl    $1, %edi
cvtsi2sdq %rax, %xmm0
movq    -56(%rbp), %rax
subq    -72(%rbp), %rax
subl    $1, %r14d
movslq   %r14d, %r14
movl    (%r15,%r14,4), %ecx
cvtsi2sdq %rax, %xmm1
movq    -88(%rbp), %rax
movl    0(%rax,4), %edx
xorl    %eax, %eax
divsd   .LC1(%rip), %xmm1
addsd   %xmm1, %xmm0
movsd   %xmm0, -104(%rbp)
call    __printf_chk
movsd   -104(%rbp), %xmm0
movl    $.LC3, %esi
movl    $1, %edi
movl    $1, %eax
call    __printf_chk

```



```

        leaq    -40(%rbp), %rsp
        xorl    %eax, %eax
        popq    %rbx
        popq    %r12
        popq    %r13
        popq    %r14
        popq    %r15
        popq    %rbp
        .cfi_restore_state
        .cfi_def_cfa 7, 8
        ret
.L3:
        .cfi_restore_state
        leaq    -80(%rbp), %rsi
        xorl    %edi, %edi
        call    clock_gettime

```

## daxpyO3.s

```

call    clock_gettime
        movq    %r12, %rax
        andl    $15, %eax
        shrq    $2, %rax
        negq    %rax
        andl    $3, %eax
        cmpl    %r14d, %eax
        cmova   %r14d, %eax
        cmpl    $4, %r14d
        cmovbe  %r14d, %eax
        testl   %eax, %eax
        je      .L18
        movq    -96(%rbp), %rcx
        movq    -104(%rbp), %rsi
        movl    0(%rcx,4), %edx
        imull   %esi, %edx
        addl    %edx, (%r12)
        cmpl    $1, %eax
        jbe     .L19
        movl    4(%rcx,4), %edx
        imull   %esi, %edx
        addl    %edx, 4(%r12)
        cmpl    $2, %eax
        jbe     .L20
        movl    8(%rcx,4), %edx
        imull   %esi, %edx
        addl    %edx, 8(%r12)
        cmpl    $3, %eax
        jbe     .L21
        movl    12(%rcx,4), %edx
        imull   %esi, %edx
        addl    %edx, 12(%r12)

```

.L12:	movl	\$4, %edx
	cmpl	%eax, %r14d
	je	.L16
.L11:	movl	%r14d, %r9d
	movl	%eax, %esi
	subl	%eax, %r9d
	movl	%r9d, %edi
	shrl	\$2, %edi
	leal	0(,%rdi,4), %r8d
	testl	%r8d, %r8d
	je	.L14
	movd	-108(%rbp), %xmm5
	salq	\$2, %rsi
	xorl	%eax, %eax
	leaq	0(%r13,%rsi), %r10
	xorl	%ecx, %ecx
	addq	%r12, %rsi
	pshufd	\$0, %xmm5, %xmm2
	movdqa	%xmm2, %xmm3
	psrlq	\$32, %xmm3
.L15:	movdqu	(%r10,%rax), %xmm1
	addl	\$1, %ecx
	movdqa	%xmm1, %xmm4
	psrlq	\$32, %xmm1
	pmuludq	%xmm3, %xmm1
	pshufd	\$8, %xmm1, %xmm1
	pmuludq	%xmm2, %xmm4
	pshufd	\$8, %xmm4, %xmm0
	punpckldq	%xmm1, %xmm0
	paddb	(%rsi,%rax), %xmm0
	movdqa	%xmm0, (%rsi,%rax)
	addq	\$16, %rax
	cmpl	%ecx, %edi
	ja	.L15
	addl	%r8d, %edx
	cmpl	%r8d, %r9d
	je	.L16
.L14:	movslq	%edx, %rax
	movq	-104(%rbp), %rdi
	movl	0(%r13,%rax,4), %ecx
	imull	%edi, %ecx
	addl	%ecx, (%r12,%rax,4)

```

        leal    1(%rdx), %eax
        cmpl    %r14d, %eax
        jge     .L16
        cltq
        addl    $2, %edx
        movl    0(%r13,%rax,4), %ecx
        imull    %edi, %ecx
        addl    %ecx, (%r12,%rax,4)
        cmpl    %edx, %r14d
        jle     .L16
        movslq   %edx, %rdx
        movl    -104(%rbp), %eax
        imull    0(%r13,%rdx,4), %eax
        addl    %eax, (%r12,%rdx,4)
.L16:
        leaq    -64(%rbp), %rsi
        xorl    %edi, %edi
        subl    $1, %r14d
        movslq   %r14d, %r14
        call    clock_gettime
        movq    -64(%rbp), %rax
        subq    -80(%rbp), %rax
        movl    $.LC2, %esi
        movl    (%r12,%r14,4), %ecx
        movl    $1, %edi
        cvtsi2sdq %rax, %xmm0
        movq    -56(%rbp), %rax
        subq    -72(%rbp), %rax
        cvtsi2sdq %rax, %xmm1
        movq    -88(%rbp), %rax
        movl    0(%rax,4), %edx
        xorl    %eax, %eax
        divsd    .LC1(%rip), %xmm1
        addsd    %xmm1, %xmm0
        movsd    %xmm0, -96(%rbp)
        call    __printf_chk
        movsd    -96(%rbp), %xmm0
        movl    $.LC3, %esi
        movl    $1, %edi
        movl    $1, %eax
        call    __printf_chk
        leaq    -40(%rbp), %rsp
        xorl    %eax, %eax
        popq    %rbx
        popq    %r12
        popq    %r13

```

```

        popq        %r14
        popq        %r15
        popq        %rbp
        .cfi_remember_state
        .cfi_def_cfa 7, 8
        ret

.L18:
        .cfi_restore_state
        xorl        %edx, %edx
        jmp         .L11

.L21:
        movl        $3, %edx
        jmp         .L12

.L19:
        movl        $1, %edx
        jmp         .L12

.L20:
        movl        $2, %edx
        jmp         .L12

.L3:
        leaq        -80(%rbp), %rsi
        xorl        %edi, %edi
        call        clock_gettime

```

## daxpyOs.s

```

call    clock_gettime
xorl    %eax, %eax

.L5:
        cmpl        %eax, %ebx
        jle         .L10
        movl        -88(%rbp), %edx
        imull       (%r14,%rax,4), %edx
        addl        %edx, 0(%r13,%rax,4)
        incq        %rax
        jmp         .L5

.L10:
        leaq        -64(%rbp), %rsi
        xorl        %edi, %edi
        decl        %ebx
        movslq       %ebx, %rbx
        call        clock_gettime

```