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): Iván Ortega Alba
Grupo de prácticas: C2
Fecha de entrega: 07/06/2015
Fecha evaluación en clase: -

Versión de gcc utilizada: *(respuesta)*

```
cvi067161:Downloads ivanortegaalba$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/local/libexec/gcc/x86_64-apple-
darwin14.0.0/4.9.2/lto-wrapper
Target: x86_64-apple-darwin14.0.0
Configured with: ../gcc-4.9-20141029/configure --enable-
languages=c++,fortran
Thread model: posix
      gcc version 4.9.2 20141029 (prerelease) (GCC)
```

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

He usado mi pc para la multiplicación de matrices.

Para el Benchmark he usado otro pc (se adjunta cpu-info en el zip) porque mi pc saltaba el siguiente problema:

```
MacBook-Pro-de-Ivan:2-01 ivanortegaalba$ ./Daxpy 10000000
Killed: 9
```

Y no he podido solucionarlo.

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 <omp.h>

main(int argc, char **argv) {
    int N = atoi(argv[1]);
    int i,j,k;
    omp_sched_t kind;
    int modifier;
    double start=0,end=0,elapsed = 0;

    //Matriz 1
    double **a;
    a = (double **) malloc (N*sizeof(double *));
    for (i=0;i<N;i++)
        a[i] = (double *) malloc (N*sizeof(double));
    //Matriz 2
    double **b;
    b = (double **)malloc (N*sizeof(double *));
    for (i=0;i<N;i++)
        b[i] = (double *) malloc (N*sizeof(double));

    //Matriz tras
    double **tras;
    tras = (double **)malloc (N*sizeof(double *));
    for (i=0;i<N;i++)
        tras[i] = (double *) malloc (N*sizeof(double));

    //Matriz Resutado
    double **resultado;
    resultado = (double **)malloc (N*sizeof(double *));
    for (i=0;i<N;i++)
        resultado[i] = (double *) malloc (N*sizeof(double));

    if(argc < 2) {
        fprintf(stderr,"Faltan argumentos\n");
        exit(-1);
    }

    for (i=0; i<N; ++i)
        for (j=0; j<N; ++j){
            resultado[i][j] = 0;
            a[i][j] = j+1;
            b[i][j] = j+1;
        }

    start = omp_get_wtime();

    //Traspuesta
    for (i=0; i<N; ++i)
        for (j=0; j<N; ++j)
        tras[i][j] = b[j][i];

    int aux1=0.0,aux2=0.0,aux3=0.0,aux4=0.0;
    for (i=0; i<N; ++i){
        for (j=0; j<N; ++j){
```

```
            for (k=0; k<N; k+=4){
                aux1+=a[i][k]*b[j][k];  //desenrollado de bucle
                aux2+=a[i][k+1]*tras[j][k+1];
                aux3+=a[i][k+2]*tras[j][k+2];
                aux4+=a[i][k+3]*tras[j][k+3];
            }
                resultado[i][j]+=aux1+aux2+aux3+aux4;
        }
    }

    end = omp_get_wtime();
    elapsed = end - start;

    //Imprimimos
    printf("resultado[0][0] = %g\n",resultado[0][0]);
    printf("resultado[N-1][N-1] = %g\n",resultado[N-1][N-1]);
    printf("Tiempo(seg.):%11.9f\t / Tamaño Matrices:%u\n",elapsed,N);

}
```

MODIFICACIONES REALIZADAS:

Modificación a) –explicación-:

Para la modificación a hemos hecho uso del desenrollado de bucles en 4 elementos, reduciendo así el numero de iteraciones, es decir, el numero de saltos.

Modificación b)

Para la modificación b hemos aprovechado la localidad de acceso, haciendo la traspuesta a la segunda matriz, y poder acceder por filas y no por columnas a la matriz, evitando los saltos de posición en el acceso, y volviéndolos contiguos.

Modificación c)

Combinación de ambas.

Modificación	**-O0**	**-O1**	**-O2**	**-O3**	**-Os**
Sin modificar	26.615	9.537	9.648	9.678	13.253
Modificación a)	6.976	3.992	4.143	4.062	4.186
Modificación b)	9.266	1.508	1.515	1.490	4.066
Modificación c)	5.401	2.545	3.117	3.098	3.103

COMENTARIOS SOBRE LOS RESULTADOS:

En nuestro código ensamblador con la opción -O1, podemos encontrar las siguientes optimizaciones:

- Transformaciones de bucles: realiza modificaciones en los bucles para aumentar su velocidad de ejecución, como el desenrollado de bucles.
- Reducción de potencia: cambiar operaciones complejas por otra mas simples.

En nuestro código ensamblador con la opción -O2, podemos encontrar la siguiente optimización:

- Eliminación de código: eliminación de código inaccesible o que no afecta al resultado final. Esto lo podemos apreciar por la considerable reducción de código.

CAPTURAS DE PANTALLA:

```
rm salida-ModA.txt
ivan@ivan-Aspire5741G:~/Dropbox/UNIVERSIDAD_14-15/AC/CUADERNO/SCRIPTS/P4/1-ppm$
./ejecuta.sh ModA
ivan@ivan-Aspire5741G:~/Dropbox/UNIVERSIDAD_14-15/AC/CUADERNO/SCRIPTS/P4/1-ppm$
./ejecuta.sh ModB
ivan@ivan-Aspire5741G:~/Dropbox/UNIVERSIDAD_14-15/AC/CUADERNO/SCRIPTS/P4/1-ppm$
./ejecuta.sh ModC
ivan@ivan-Aspire5741G:~/Dropbox/UNIVERSIDAD_14-15/AC/CUADERNO/SCRIPTS/P4/1-ppm$
./ejecuta.sh SinMod
ivan@ivan-Aspire5741G:~/Dropbox/UNIVERSIDAD_14-15/AC/CUADERNO/SCRIPTS/P4/1-ppm$
cat salida-ModA.txt
resultado[0][0] = 8.3459e+07
resultado[N-1][N-1] = -1.74147e+09
Tiempo(seg.):6.976409340        / Tamaño Matrices:1000
resultado[0][0] = 8.3459e+07
resultado[N-1][N-1] = -1.74147e+09
Tiempo(seg.):3.992436322        / Tamaño Matrices:1000
resultado[0][0] = 8.3459e+07
resultado[N-1][N-1] = -1.74147e+09
Tiempo(seg.):4.143388875        / Tamaño Matrices:1000
resultado[0][0] = 8.3459e+07
resultado[N-1][N-1] = -1.74147e+09
Tiempo(seg.):4.062357507        / Tamaño Matrices:1000
resultado[0][0] = 8.3459e+07
resultado[N-1][N-1] = -1.74147e+09
Tiempo(seg.):4.186687014        / Tamaño Matrices:1000
ivan@ivan-Aspire5741G:~/Dropbox/UNIVERSIDAD_14-15/AC/CUADERNO/SCRIPTS/P4/1-ppm$
```

B) CÓDIGO FIGURA 1:

CÓDIGO FUENTE: figura1-modificado.c

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>
//#define PRINTF_ALL

main()
{
    struct estructura {
        int a;
        int b;
    } s[5000];

    int  R[40000];
    double start, end, elapsed;
    int ii,i;
    int X1,X2;
    //inicializacion
    for(i=0; i<5000; i++) {
        s[i].a=2;
        s[i].b=3;
    }

    start = omp_get_wtime();
    for (ii=0; ii<=40000; ii++) {
        for(i=0; i<5000; i=i+5) {//unificacion de los dos bubles
                                 //desenrrollado de buble
                                 //acceso ordenado a la estrucctura,
es decir promero la varibles a y despues la b
            X1+=2*s[i].a+ii;
            X2+=3*s[i].b-ii;
            X1+=2*s[i+1].a+ii;
            X2+=3*s[i+1].b-ii;
```

```
            X1+=2*s[i+2].a+ii;
            X2+=3*s[i+2].b-ii;
            X1+=2*s[i+3].a+ii;
            X2+=3*s[i+3].b-ii;
            X1+=2*s[i+4].a+ii;
            X2+=3*s[i+4].b-ii;
        }
        if (X1>X2) {
            R[ii]=X2;
        }
        else {
            R[ii]=X1;
        }
    }
    end = omp_get_wtime();
    elapsed = end - start;

    printf("Tiempo(seg.):%11.9f \n",elapsed);
#ifdef PRINTF_ALL
    for (ii=0; ii<=40000; ii++) {
        printf("%d ",R[ii]);
    }
    printf("\n");
#endif
}
```

MODIFICACIONES REALIZADAS:

Modificación a) –explicación-:

La primera modificación consiste en unir los dos bulces, evitando la mitad de iteraciones, por tanto, la mitad de saltos en ensamblador, aumentando así considerablemente el rendimiento.

Modificación b) –explicación-:

Hemos aplicado el desenrollado de bucles en cada uno de los for aumentando el rendimiento más significativamente que la anterior.

Modificación c) –explicación-:

Hemos combinado las dos modificaciones anteriores. Podemos ver como sorprendentemente da menos rendimiento que la solución b por si sola, y no encuentro el motivo.

Modificación	-O0	-O1	-O2	-O3	-Os
Sin modificar	1.295173822	0.306065421	0.000000200	0.000000283	0.000000218
Modificación a)	1.033086251	0.153870086	0.000000264	0.000000195	0.000000190
Modificación b)	0.287625647	0.072924900	0.000000203	0.000000210	0.000000225
Modificación c)	0.820158737	0.031945159	0.000000216	0.000000331	0.000000214

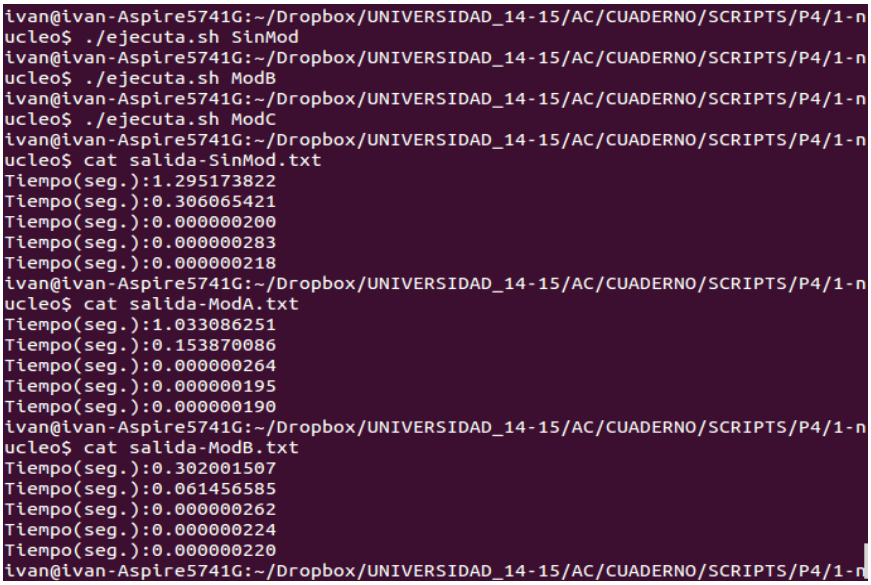
En nuestro código ensamblador con la opción -O1, podemos encontrar las siguientes optimizaciones:

- Transformaciones de bucles: realiza modificaciones en los bucles para aumentar su velocidad de ejecución, como el desenrollado de bucles.
- Reducción de potencia: cambiar operaciones complejas por otra mas simples.
- Eliminación de código: eliminación de código inaccesible o que no afecta al resultado final. Esto lo podemos apreciar por la considerable reducción de código.

En nuestro código ensamblador con la opción -O2, podemos encontrar la siguiente optimización:

- Movimiento de código: sacar del bucle un cálculo que siempre da el mismo valor y se repite en todas las iteraciones.

CAPTURAS DE PANTALLA:



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):

$$\text{for (i=1;i<=N,i++) y[i]= a*x[i] + y[i];}$$

   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 acarrean. 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 <omp.h>

void daxpy(double *x,double *y,int N,int a){
    int i;

    for(i=1;i<=N;i++)
        y[i] = a * x[i] + y[i];
}

int main(int argc, char ** argv)
{
    unsigned int N=500,i;
    double start,end,elapsed;

    if(argc>=2) N=atoi(argv[1]);

    double *x,*y;
    x = (double *) malloc(N*sizeof(double));
    y = (double *) malloc(N*sizeof(double));

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

    start = omp_get_wtime();

    daxpy(x,y,N,20);

    end = omp_get_wtime();
    elapsed = end - start;

    printf("Tiempo(seg.):%11.9f\t / Tam:%u\n",elapsed,N);
    printf("Primer componente del resultado [%f]\nultimo componente
del resultado [%f]\n",y[i],y[N-1]);

    free(x);
    free(y);
}
```

Tiempos ejec.	-O0	-O1	-O2	-O3	-Os
	0.585	0.285	0.283	0.276	0.452

CAPTURAS DE PANTALLA:

```
Tiempo(seg.):0.002742202         / Tam:1000000
Primer componente del resultado [0.000000]
ultimo componente del resultado [20999979.000000]
Tiempo(seg.):0.004578976         / Tam:1000000
Primer componente del resultado [0.000000]
ultimo componente del resultado [20999979.000000]
ivan@ivan-Aspire5741G:~/Dropbox/UNIVERSIDAD_14-15/AC/CUADERNO/SCRIPTS/P4/2$
alida-Daxpy.txt
ivan@ivan-Aspire5741G:~/Dropbox/UNIVERSIDAD_14-15/AC/CUADERNO/SCRIPTS/P4/2$
ecuta.sh Daxpy
ivan@ivan-Aspire5741G:~/Dropbox/UNIVERSIDAD_14-15/AC/CUADERNO/SCRIPTS/P4/2$
salida-Daxpy.txt
Tiempo(seg.):0.585960972         / Tam:100000000
Primer componente del resultado [0.000000]
ultimo componente del resultado [2099999979.000000]
Tiempo(seg.):0.285173949         / Tam:100000000
Primer componente del resultado [0.000000]
ultimo componente del resultado [2099999979.000000]
Tiempo(seg.):0.283114625         / Tam:100000000
Primer componente del resultado [0.000000]
ultimo componente del resultado [2099999979.000000]
Tiempo(seg.):0.276141020         / Tam:100000000
Primer componente del resultado [0.000000]
ultimo componente del resultado [2099999979.000000]
Tiempo(seg.):0.452932986         / Tam:100000000
Primer componente del resultado [0.000000]
ultimo componente del resultado [2099999979.000000]
ivan@ivan-Aspire5741G:~/Dropbox/UNIVERSIDAD_14-15/AC/CUADERNO/SCRIPTS/P4/2$
```

COMENTARIOS SOBRE LAS DIFERENCIAS EN ENSAMBLADOR:

- Con la opción -O1:
  - Reducción de potencia: cambiar operaciones complejas por otra mas simples. Por lo que vemos más operaciones.

- Con la opción -O2:
  - Movimiento de código: sacar del bucle un cálculo que siempre da el mismo valor y se repite en todas las iteraciones.
  - Eliminación de código: eliminación de código inaccesible o que no afecta al resultado final. Esto lo podemos apreciar por la considerable reducción de código.

- Con la opción -O3:
  - Transformaciones de bucles: realiza modificaciones en los bucles para aumentar su velocidad de ejecución, como el desenrollado de bucles.

CÓDIGO EN ENSAMBLADOR: (ADJUNTAR AL .ZIP)
(LIMITAR AQUÍ EL CÓDIGO INCLUÍDO A LA ZONA DEL CÓDIGO ENSAMBLADOR DONDE SE REALIZA LA OPERACIÓN CON VECTORES)

daxpyO0.s

```
daxpy:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movq    %rdi, -24(%rbp)
        movq    %rsi, -32(%rbp)
        movl    %edx, -36(%rbp)
        movl    %ecx, -40(%rbp)
        movl    $1, -4(%rbp)
        jmp     .L2
.L3:
        movl    -4(%rbp), %eax
        cltq
        leaq    0(,%rax,8), %rdx
        movq    -32(%rbp), %rax
        addq    %rdx, %rax
        cvtsi2sd    -40(%rbp), %xmm0
        movl    -4(%rbp), %edx
        movslq  %edx, %rdx
        leaq    0(,%rdx,8), %rcx
        movq    -24(%rbp), %rdx
        addq    %rcx, %rdx
        movsd   (%rdx), %xmm1
        mulsd   %xmm1, %xmm0
        movl    -4(%rbp), %edx
        movslq  %edx, %rdx
        leaq    0(,%rdx,8), %rcx
```

```
            movq        -32(%rbp), %rdx
            addq        %rcx, %rdx
            movsd       (%rdx), %xmm1
            addsd       %xmm1, %xmm0
            movsd       %xmm0, (%rax)
            addl        $1, -4(%rbp)
.L2:
            movl        -4(%rbp), %eax
            cmpl        -36(%rbp), %eax
            jle         .L3
            popq        %rbp
            .cfi_def_cfa 7, 8
            ret
            .cfi_endproc
.LFE0:
            .size       daxpy, .-daxpy
            .ident      "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
            .section    .note.GNU-stack,"",@progbits
```

daxpyO1.s

```
/daxpy:
.LFB0:
            .cfi_startproc
            testl       %edx, %edx
            jle         .L1
            movl        $1, %eax
            cvtsi2sd    %ecx, %xmm1
.L3:
            movslq      %eax, %r8
            leaq        (%rsi,%r8,8), %rcx
            movapd      %xmm1, %xmm0
            mulsd       (%rdi,%r8,8), %xmm0
            addsd       (%rcx), %xmm0
            movsd       %xmm0, (%rcx)
```

```
            addl        $1, %eax
            cmpl        %eax, %edx
            jge         .L3
.L1:
            rep ret
            .cfi_endproc
.LFE0:
            .size       daxpy, .-daxpy
            .ident      "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
            .section    .note.GNU-stack,"",@progbits
```

daxpyO2.s

```
daxpy:
.LFB0:
            .cfi_startproc
            testl       %edx, %edx
            jle         .L1
            cvtsi2sd    %ecx, %xmm1
            subl        $1, %edx
            leaq        8(,%rdx,8), %rdx
            xorl        %eax, %eax
            .p2align 4,,10
            .p2align 3
.L4:
            movsd       8(%rdi,%rax), %xmm0
            mulsd       %xmm1, %xmm0
            addsd       8(%rsi,%rax), %xmm0
            movsd       %xmm0, 8(%rsi,%rax)
            addq        $8, %rax
```

daxpyO3.s

```
daxpy:
.LFB0:
            .cfi_startproc
            testl       %edx, %edx
```

```
            jle         .L1
            leaq        8(%rdi), %rax
            cvtsi2sd    %ecx, %xmm2
            leaq        24(%rsi), %rcx
            leaq        24(%rdi), %r8
            cmpq        %rax, %rcx
            leaq        8(%rsi), %rax
            setbe       %cl
            cmpq        %rax, %r8
            setbe       %al
            orb         %al, %cl
            je          .L3
            cmpl        $11, %edx
            jbe         .L3
            movapd      %xmm2, %xmm3
            movl        %edx, %r8d
            shrl        %r8d
            xorl        %eax, %eax
            xorl        %ecx, %ecx
            unpcklpd    %xmm3, %xmm3
            leal        (%r8,%r8), %r9d
.L7:
            movsd       8(%rdi,%rax), %xmm0
            addl        $1, %ecx
            movsd       8(%rsi,%rax), %xmm1
            movhpd      16(%rdi,%rax), %xmm0
            movhpd      16(%rsi,%rax), %xmm1
            mulpd       %xmm3, %xmm0
            addpd       %xmm1, %xmm0
            movlpd      %xmm0, 8(%rsi,%rax)
            movhpd      %xmm0, 16(%rsi,%rax)
            addq        $16, %rax
            cmpl        %r8d, %ecx
            jb          .L7
```

```
                cmpl            %r9d, %edx
                leal            1(%r9), %eax
                je              .L1
                movslq          %eax, %rdx
                mulsd           (%rdi,%rdx,8), %xmm2
                leaq            (%rsi,%rdx,8), %rax
                addsd           (%rax), %xmm2
                movsd           %xmm2, (%rax)
                ret
                .p2align 4,,10
                .p2align 3
.L3:
                subl            $1, %edx
                xorl            %eax, %eax
                leaq            8(,%rdx,8), %rdx
                .p2align 4,,10
                .p2align 3
.L9:
                movsd           8(%rdi,%rax), %xmm0
                mulsd           %xmm2, %xmm0
                addsd           8(%rsi,%rax), %xmm0
                movsd           %xmm0, 8(%rsi,%rax)
                addq            $8, %rax
                cmpq            %rdx, %rax
                jne             .L9
.L1:
                rep ret
                .cfi_endproc
.LFE0:
                .size           daxpy, .-daxpy
                .ident          "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
                .section        .note.GNU-stack,"",@progbits
```

daxpyOs.s

```
daxpy:
```

```
.LFB0:
            .cfi_startproc
            xorl        %eax, %eax
.L2:
            incq        %rax
            cmpl        %eax, %edx
            jl          .L5
            cvtsi2sd    %ecx, %xmm0
            mulsd       (%rdi,%rax,8), %xmm0
            addsd       (%rsi,%rax,8), %xmm0
            movsd       %xmm0, (%rsi,%rax,8)
            jmp         .L2
.L5:
            ret
            .cfi_endproc
.LFE0:
            .size       daxpy, .-daxpy
            .ident      "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
            .section    .note.GNU-stack,"",@progbits
```