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): Rafael Nogales
Grupo de prácticas:
Fecha de entrega:
Fecha evaluación en clase:

Versión de gcc utilizada: gcc version 4.8.2 (Ubuntu 4.8.2-19ubuntu1)

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

```
processor   : 0
vendor_id   : GenuineIntel
cpu family  : 6
model                   : 58
model name  : Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz
stepping    : 9
microcode   : 0x19
cpu MHz                 : 2497.301
cache size  : 6144 KB
physical id : 0
siblings    : 4
core id                 : 0
cpu cores   : 4
apicid                  : 0
initial apicid          : 0
fpu                     : yes
fpu_exception           : yes
cpuid level : 5
wp                      : yes
flags                   : fpu vme de pse tsc msr pae mce cx8 apic sep
mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx
rdtscp lm constant_tsc rep_good nopl pni ssse3 lahf_lm
bogomips    : 4994.60
clflush size            : 64
cache_alignment         : 64
address sizes           : 36 bits physical, 48 bits virtual
power management:

processor   : 1
vendor_id   : GenuineIntel
cpu family  : 6
model                   : 58
model name  : Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz
stepping    : 9
microcode   : 0x19
cpu MHz                 : 2497.301
cache size  : 6144 KB
physical id : 0
```

```
siblings    : 4
core id                 : 1
cpu cores   : 4
apicid                  : 1
initial apicid          : 1
fpu                     : yes
fpu_exception           : yes
cpuid level : 5
wp                      : yes
flags                   : fpu vme de pse tsc msr pae mce cx8 apic sep
mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx
rdtscp lm constant_tsc rep_good nopl pni ssse3 lahf_lm
bogomips    : 4994.60
clflush size            : 64
cache_alignment         : 64
address sizes           : 36 bits physical, 48 bits virtual
power management:

processor   : 2
vendor_id   : GenuineIntel
cpu family  : 6
model                   : 58
model name  : Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz
stepping    : 9
microcode   : 0x19
cpu MHz                 : 2497.301
cache size  : 6144 KB
physical id : 0
siblings    : 4
core id                 : 2
cpu cores   : 4
apicid                  : 2
initial apicid          : 2
fpu                     : yes
fpu_exception           : yes
cpuid level : 5
wp                      : yes
flags                   : fpu vme de pse tsc msr pae mce cx8 apic sep
mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx
rdtscp lm constant_tsc rep_good nopl pni ssse3 lahf_lm
bogomips    : 4994.60
clflush size            : 64
cache_alignment         : 64
address sizes           : 36 bits physical, 48 bits virtual
power management:

processor   : 3
vendor_id   : GenuineIntel
cpu family  : 6
model                   : 58
model name  : Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz
stepping    : 9
microcode   : 0x19
cpu MHz                 : 2497.301
cache size  : 6144 KB
physical id : 0
```

```
siblings    : 4
core id                 : 3
cpu cores   : 4
apicid                  : 3
initial apicid          : 3
fpu                     : yes
fpu_exception           : yes
cpuid level : 5
wp                      : yes
flags                   : fpu vme de pse tsc msr pae mce cx8 apic sep
mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx
rdtscp lm constant_tsc rep_good nopl pni ssse3 lahf_lm
bogomips    : 4994.60
clflush size            : 64
cache_alignment         : 64
address sizes           : 36 bits physical, 48 bits virtual
power management:
```

1. Para el núcleo que se muestra en la Figura 1, 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,...). 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.

    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

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

main()
{
   …
   for (ii=1; ii<=40000;ii++) {
      for(i=0; i<5000;i++)  X1=2*s[i].a+ii;
      for(i=0; i<5000;i++)  X2=3*s[i].b-ii;

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

Figura 1: Núcleo de programa en C para el ejercicio 1.

Como hay dos bucles que hacen las mismas iteraciones lo he simplificado en un solo bucle, y esa es la mejora mas notable, ademas se podría usar alguna instrucción con predicados a nivel ensamblador para quitar el if, ya que produce bastantes saltos innecesarios (aunque esa mejora no la he hecho, así es como se resolvería el ejercicio extra).

La otra modificaciósn que le he puesto es realizar una precaptacion pero no mejora en nada los resultados, ya que aquí estamos realizando cargas de enteros y eso no requiere a penas esfuerzo, ademas los estamos cargando en variables distintas cada vez.

Finalmente he cambiado el struct y en vez de tener N parejas tengo un solo struct gigante con dos arrays de N componentes, es esta ultima combinada con eliminar los saltos duplicados la que produce mejores efectos, se podría mejorar aun mas utilizando un desenrollado del bucle interno.

A) MULTIPLICACIÓN DE MATRICES:

CÓDIGO FUENTE: `pmm-secuencial-modificado.c`
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

#define MAX 4096

int main(int argc, char **argv){
             int i, j, k, h;
             int dimension_matrices;
             int suma = 0; int s1,s2,s3,s4,s5,s6,s7,s8;
        s1=s2=s3=s4=s5=s6=s7=s8=0;
        time_t t_inicio, t_final;
        double tiempo;

             int **matrizB;
             int **matrizC;
             int **matrizA;

             if (argc < 2){
                          printf("Falta el número de componentes\n");
                          return(1);
             }

    //El producto de matrices cuadradas requiere que ambas sean de la misma
    //dimension por tanto solo necesitamos un parametro para reservar la
memoria
             dimension_matrices = atoi(argv[1]);

             if (dimension_matrices > MAX){
                          printf("Tamaño demasiado grande. No superar
%d\n\n",MAX);
                          return(1);
             }

//Creamos las matrices
// matrizA = matrizB * matrizC
```

```
             matrizB = (int **)malloc(dimension_matrices * sizeof(int*));
             matrizC = (int **)malloc(dimension_matrices * sizeof(int*));
             matrizA = (int **)malloc(dimension_matrices * sizeof(int*));

             for (i=0; i<dimension_matrices; i++){
                          matrizB[i] = (int *)malloc(dimension_matrices *
sizeof(int));
                          matrizC[i] = (int *)malloc(dimension_matrices *
sizeof(int));
                          matrizA[i] = (int *)malloc(dimension_matrices *
sizeof(int));
             }

//Inicializamos las matrices
             for (j=0; j<dimension_matrices; ++j){
                          for (i=0; i<dimension_matrices; i++){
                                       matrizB[j][i] = j+i;
                                       matrizC[j][i] = j*i;
                          }
             }

        t_inicio=clock();
//Multiplicamos las matrices B y C
        int rondas_bucle_interno = dimension_matrices/8; // 8 =
cte_desenrollado
        //printf("Rondas internas %d\n",rondas_bucle_interno);
             for (i=0; i<dimension_matrices; i++){
                          for(j=0; j<dimension_matrices; j++){
                                       s1=0; s2=0; s3=0; s4=0;
                          s5=0; s6=0; s7=0; s8=0;
                                       for (h=0, k=0;h <
rondas_bucle_interno; ++h, k+=8){
                                                    s1 += (matrizB[i]
[k]*matrizC[k][j]);
                                  s2 += (matrizB[i][k+1]*matrizC[j][k+1]);
                                  s3 += (matrizB[i][k+2]*matrizC[j][k+2]);
                                  s4 += (matrizB[i][k+3]*matrizC[j][k+3]);
                                  s5 += (matrizB[i][k+4]*matrizC[j][k+4]);
                                  s6 += (matrizB[i][k+5]*matrizC[j][k+5]);
                                  s7 += (matrizB[i][k+6]*matrizC[j][k+6]);
                                  s8 += (matrizB[i][k+7]*matrizC[j][k+7]);
                                    }
                          suma= s1 + s2 + s3 + s4 + s5 + s6 + s7 + s8;
                                    matrizA[i][j]=suma;
                          for(k=rondas_bucle_interno*8; k<dimension_matrices; +
+k){
                                                    suma += (matrizB[i]
[k]*matrizC[j][k]);
                          }
                          matrizA[i][j]=suma;
                          }
             }

//Imprimimos resultados
        t_final=clock();
        tiempo = ((double)(t_final - t_inicio))/CLOCKS_PER_SEC;
        printf ("%f\n",tiempo);
             //printf ("Resultado[0][0] = %d\n",matrizA[0][0]);
             //printf ("Componente(N-1,N-1) del resultado de la
multiplicación de ambas matrices=%d\n",matrizA[dimension_matrices-1]
[dimension_matrices-1]);
```

```
//Liberamos las matrices
    free(matrizA);
    free(matrizB);
    free(matrizC);

    return 0;
}
```

MODIFICACIONES REALIZADAS:

Modificación a) Desenrollado de bucle mas interno

–explicación-: Se reducen las instrucciones de salto pero sigue habiendo la misma cantidad de instrucciones de ejecucion.

Modificación b) Inicializamos una de las matrices como “su traspuesta”

–explicación-: Se mejora el acceso a los datos por el principio de localizacion espacial.

COMENTARIOS SOBRE LOS RESULTADOS: El acceso a los datos influye mucho mas de lo que cabría esperar.

B) CÓDIGO FIGURA 1:

CÓDIGO FUENTE: `figura1-modificado.c`

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
// struct.cc

#include <functional>
#include <numeric>
#include <random>
#include <time.h>
#include <iostream>

const int N = 5000, REP = 4000;

std::default_random_engine generator(N * REP);
std::uniform_int_distribution<int> distribution(0, 9);
auto rng = std::bind(distribution, generator);

struct S
{
             int a[N], b[N];
             S(){
        for(int i=0; i<N; ++i){
            a[i] = rng();
            b[i] = rng();
        }
    }
} s;

int main()
{
    time_t t_inicio, t_final;
    double tiempo;
             int R[REP];
    t_inicio=clock();
             for (int ii = 0; ii < REP; ++ii)
             {
                          int X1, X2;
                          for (int i = 0; i < N; ++i){
                                       X1 = 2 * s.a[i] + ii;
                                       X2 = 3 * s.b[i] - ii;
        }
                          if (X1 < X2)
```

```
                                        R[ii] = X1;
                            else
                                        R[ii] = X2;
                }
    t_final=clock();
    tiempo = ((double)(t_final - t_inicio))/CLOCKS_PER_SEC;
    std::cout << tiempo << "\n";

                return std::accumulate(R, R + REP, 0);
}
```

MODIFICACIONES REALIZADAS:

Modificación a) Desenrollado del bucle mas interno

Modificación b) Cambio de la estructura

CAPTURAS DE PANTALLA:

```
rafa@RNog-Ubuntu14:~/Escritorio/AC/p4$ ./todo1.sh
Compilando...
struct [OK]
modificacin A [OK]
modificacin B [OK]
modificacin C [OK]
modificacin D [OK]
COMPILACION OK!
Modificación  - - - -      O0             O1             O2             O3             Os
Sin Modificar - - - -   0.117957       0.014629       0.014338       0.014687       0.016372
Quitando saltos - - -   0.064342       0.010197       0.013943       0.003364       0.00876
Precaptacion  - - - -   0.122512       0.014903       0.015563       0.016158       0.015465
Ambas modificaciones    0.062126       0.013754       0.014113       0.013394       0.0198
Modificando el struct   0.055993       0.008266       0.007522       0.003866       0.01007
rafa@RNog-Ubuntu14:~/Escritorio/AC/p4$
```

```
rafa@RNog-Ubuntu14:~/Escritorio/AC/p4/ej2$ ./tabla1.sh 500
Modificación  - - - -      O0             O1             O2             O3             Os
Sin Modificar - - - -   0.771132       0.349009       0.312171       0.350557       0.347211
ModificadoA - - - -     0.686019       0.313649       0.339955       0.337118       0.318131
ModificadoB - - - -     0.691359       0.330028       0.299092       0.289323       0.338505
rafa@RNog-Ubuntu14:~/Escritorio/AC/p4/ej2$ ./tabla1.sh 1000
Modificación  - - - -      O0             O1             O2             O3             Os
Sin Modificar - - - -   18.954043      18.156732      17.450816      17.852804      17.914841
ModificadoA - - - -     18.324818      20.932910      18.165371      18.469696      18.077903
ModificadoB - - - -     19.180126      18.122933      18.141331      18.366599      17.259886
rafa@RNog-Ubuntu14:~/Escritorio/AC/p4/ej2$ ./tabla1.sh 50
Modificación  - - - -      O0             O1             O2             O3             Os
Sin Modificar - - - -   0.000476       0.000140       0.000183       0.000135       0.000146
ModificadoA - - - -     0.000698       0.000095       0.000132       0.000100       0.000153
ModificadoB - - - -     0.000404       0.000112       0.000287       0.000161       0.000167
```

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];
```

   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.

   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) y compárela con el valor obtenido para Rmax.

CÓDIGO FUENTE: `daxpy.cc`
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
#include <functional>
#include <numeric>
#include <random>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <iostream>

using namespace std;

const int N = 5000, REP = 4000;
default_random_engine generator(N * REP);
uniform_int_distribution<int> distribution(0, 9);
auto rng = bind(distribution, generator);

void daxpy(int* A, int* B, int N, double alpha){
    int i=0;
    for(i=0; i<N; ++i){
        A[i]= alpha*B[i] + A[i];
    }
}

int main(int argc,char **argv) {
        double tiempo;
        time_t t_inicio, t_final;
        if (argc<2) {
                cout << "Faltan no componentes del vector" <<endl;
                return(-1);
        }
        const int N=atoi(argv[1]);
```

```
        //Inicializacion de los datos:
        int *A = new int[N];
        int *B = new int[N];
        int i;
        for(i=0; i<N; ++i){
                A[i] = rng();
                B[i] = rng();
        }

        const double alpha = 10.5;

        t_inicio=clock();
        daxpy(A, B, N, alpha);
        t_final=clock();
        tiempo = ((double)(t_final - t_inicio))/CLOCKS_PER_SEC;

        //Imprimir resultado de la suma y el tiempo de ejecución

        if(N<10){
                cout <<"Vector resultado:"<<endl;
                for(i=0; i<N-1; i++) {
                        cout << A[i] << ", ";
                }
                cout <<  A[i] << endl;

        cout<<"Tiempo(seg.): "<<tiempo<<" / Tamaño Vectores: "<<N<<endl;
        cout<<"Primer componente del vector resultado ["<<A[0]<<"] y el ultimo
componente del vector resultado ["<<A[N-1]<<"]\n";
        }
        else
                cout << tiempo << endl;

        delete [] A;
        delete [] B;
        //Eliminamos la memoria dinamica.
}
```

CAPTURAS DE PANTALLA:

```
rafa@RNog-Ubuntu14:~/Escritorio/AC/p4/ej3$ ./tabla3.sh 1000
Tamaño del vector       1000      10000          100000          1000000
Optimizacion -O0  -     2e-05    5.6e-05         0.000741        0.004359
Optimizacion -O1  -     2e-05    5.9e-05         0.00044         0.004593
Optimizacion -O2  -     1.8e-05  6.6e-05         0.000419        0.004158
Optimizacion -O3  -     1.1e-05  2.8e-05         0.000108        0.000891
Optimizacion -Os  -     1.8e-05  5.3e-05         0.000429        0.003809
```

COMENTARIOS SOBRE LAS DIFERENCIAS EN ENSAMBLADOR:

Claramente el codigo que se saca de la opcion -O3 es el más eficiente, pero tambien es bastante incomprensible, de hecho en el main no hay ninguna llamada a la funcion que calcula el mínimo

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

```
0000000000400a0d <daxpy(int*, int*, int, double)>:
400a0d:    55                       push   %rbp
400a0e:    48 89 e5                 mov    %rsp,%rbp
400a11:    48 89 7d e8              mov    %rdi,-0x18(%rbp)
400a15:    48 89 75 e0              mov    %rsi,-0x20(%rbp)
400a19:    89 55 dc                 mov    %edx,-0x24(%rbp)
400a1c:    f2 0f 11 45 d0           movsd  %xmm0,-0x30(%rbp)
400a21:    c7 45 fc 00 00 00 00     movl   $0x0,-0x4(%rbp)
400a28:    c7 45 fc 00 00 00 00     movl   $0x0,-0x4(%rbp)
400a2f:    eb 5f                    jmp    400a90 <daxpy(int*, int*, int, double)+0x83>
400a31:    8b 45 fc                 mov    -0x4(%rbp),%eax
400a34:    48 98                    cltq
400a36:    48 8d 14 85 00 00 00     lea    0x0(,%rax,4),%rdx
400a3d:    00
400a3e:    48 8b 45 e8              mov    -0x18(%rbp),%rax
400a42:    48 01 c2                 add    %rax,%rdx
400a45:    8b 45 fc                 mov    -0x4(%rbp),%eax
400a48:    48 98                    cltq
400a4a:    48 8d 0c 85 00 00 00     lea    0x0(,%rax,4),%rcx
400a51:    00
400a52:    48 8b 45 e0              mov    -0x20(%rbp),%rax
400a56:    48 01 c8                 add    %rcx,%rax
400a59:    8b 00                    mov    (%rax),%eax
400a5b:    f2 0f 2a c0              cvtsi2sd %eax,%xmm0
400a5f:    66 0f 28 c8              movapd %xmm0,%xmm1
400a63:    f2 0f 59 4d d0           mulsd  -0x30(%rbp),%xmm1
400a68:    8b 45 fc                 mov    -0x4(%rbp),%eax
400a6b:    48 98                    cltq
400a6d:    48 8d 0c 85 00 00 00     lea    0x0(,%rax,4),%rcx
400a74:    00
400a75:    48 8b 45 e8              mov    -0x18(%rbp),%rax
400a79:    48 01 c8                 add    %rcx,%rax
400a7c:    8b 00                    mov    (%rax),%eax
400a7e:    f2 0f 2a c0              cvtsi2sd %eax,%xmm0
400a82:    f2 0f 58 c1              addsd  %xmm1,%xmm0
400a86:    f2 0f 2c c0              cvttsd2si %xmm0,%eax
400a8a:    89 02                    mov    %eax,(%rdx)
400a8c:    83 45 fc 01              addl   $0x1,-0x4(%rbp)
400a90:    8b 45 fc                 mov    -0x4(%rbp),%eax
400a93:    3b 45 dc                 cmp    -0x24(%rbp),%eax
400a96:    7c 99                    jl     400a31 <daxpy(int*, int*, int, double)+0x24>
400a98:    5d                       pop    %rbp
400a99:    c3                       retq
```

daxpyO1.s

```
0000000000400a6d <daxpy(int*, int*, int, double)>:
  400a6d:    85 d2                    test   %edx,%edx
  400a6f:    7e 26                    jle    400a97 <daxpy(int*, int*, int, double)+0x2a>
  400a71:    b8 00 00 00 00           mov    $0x0,%eax
  400a76:    f2 0f 2a 0c 86           cvtsi2sdl (%rsi,%rax,4),%xmm1
  400a7b:    f2 0f 59 c8              mulsd  %xmm0,%xmm1
  400a7f:    f2 0f 2a 14 87           cvtsi2sdl (%rdi,%rax,4),%xmm2
  400a84:    f2 0f 58 ca              addsd  %xmm2,%xmm1
  400a88:    f2 0f 2c c9              cvttsd2si %xmm1,%ecx
  400a8c:    89 0c 87                 mov    %ecx,(%rdi,%rax,4)
  400a8f:    48 83 c0 01              add    $0x1,%rax
  400a93:    39 c2                    cmp    %eax,%edx
  400a95:    7f df                    jg     400a76 <daxpy(int*, int*, int, double)+0x9>
  400a97:    f3 c3                    repz retq
```

daxpyO2.s

```
  400a3e:    e8 fd fe ff ff           callq  400940 <clock@plt>
  400a43:    f2 0f 10 15 85 06 00     movsd  0x685(%rip),%xmm2        # 4010d0
<_IO_stdin_used+0xd0>
  400a4a:    00
  400a4b:    49 89 c5                 mov    %rax,%r13
  400a4e:    31 c0                    xor    %eax,%eax
  400a50:    f2 0f 2a 44 85 00        cvtsi2sdl 0x0(%rbp,%rax,4),%xmm0
  400a56:    f2 0f 59 c2              mulsd  %xmm2,%xmm0
  400a5a:    f2 0f 2a 0c 83           cvtsi2sdl (%rbx,%rax,4),%xmm1
  400a5f:    f2 0f 58 c1              addsd  %xmm1,%xmm0
  400a63:    f2 0f 2c d0              cvttsd2si %xmm0,%edx
  400a67:    89 14 83                 mov    %edx,(%rbx,%rax,4)
  400a6a:    48 83 c0 01              add    $0x1,%rax
  400a6e:    41 39 c4                 cmp    %eax,%r12d
  400a71:    7f dd                    jg     400a50 <main+0xd0>
  400a73:    e8 c8 fe ff ff           callq  400940 <clock@plt>
```

daxpyO3.s

```
0000000000400eb0 <daxpy(int*, int*, int, double)>:
  400eb0:    85 d2                    test   %edx,%edx
  400eb2:    0f 8e 31 01 00 00        jle    400fe9 <daxpy(int*, int*, int, double)
+0x139>
  400eb8:    48 8d 47 10              lea    0x10(%rdi),%rax
  400ebc:    48 39 c6                 cmp    %rax,%rsi
  400ebf:    48 8d 46 10              lea    0x10(%rsi),%rax
  400ec3:    0f 93 c1                 setae  %cl
  400ec6:    48 39 c7                 cmp    %rax,%rdi
  400ec9:    0f 93 c0                 setae  %al
  400ecc:    08 c1                    or     %al,%cl
  400ece:    0f 84 ec 00 00 00        je     400fc0 <daxpy(int*, int*, int, double)
+0x110>
  400ed4:    83 fa 04                 cmp    $0x4,%edx
  400ed7:    0f 86 e3 00 00 00        jbe    400fc0 <daxpy(int*, int*, int, double)
+0x110>
  400edd:    66 0f 28 e0              movapd %xmm0,%xmm4
  400ee1:    41 89 d1                 mov    %edx,%r9d
  400ee4:    41 c1 e9 02              shr    $0x2,%r9d
  400ee8:    31 c0                    xor    %eax,%eax
  400eea:    31 c9                    xor    %ecx,%ecx
  400eec:    66 0f 14 e4              unpcklpd %xmm4,%xmm4
  400ef0:    46 8d 04 8d 00 00 00     lea    0x0(,%r9,4),%r8d
  400ef7:    00
  400ef8:    f3 0f 6f 0c 06           movdqu (%rsi,%rax,1),%xmm1
```

```
  400efd:     83 c1 01                    add    $0x1,%ecx
  400f00:     f3 0f e6 d1                 cvtdq2pd %xmm1,%xmm2
  400f04:     66 0f 70 c9 ee              pshufd $0xee,%xmm1,%xmm1
  400f09:     f3 0f 6f 1c 07              movdqu (%rdi,%rax,1),%xmm3
  400f0e:     66 0f 59 d4                 mulpd  %xmm4,%xmm2
  400f12:     f3 0f e6 c9                 cvtdq2pd %xmm1,%xmm1
  400f16:     66 0f 59 cc                 mulpd  %xmm4,%xmm1
  400f1a:     f3 0f e6 eb                 cvtdq2pd %xmm3,%xmm5
  400f1e:     66 0f 70 db ee              pshufd $0xee,%xmm3,%xmm3
  400f23:     66 0f 58 d5                 addpd  %xmm5,%xmm2
  400f27:     f3 0f e6 db                 cvtdq2pd %xmm3,%xmm3
  400f2b:     66 0f 58 cb                 addpd  %xmm3,%xmm1
  400f2f:     66 0f e6 d2                 cvttpd2dq %xmm2,%xmm2
  400f33:     66 0f e6 c9                 cvttpd2dq %xmm1,%xmm1
  400f37:     66 0f 6c d1                 punpcklqdq %xmm1,%xmm2
  400f3b:     f3 0f 7f 14 07              movdqu %xmm2,(%rdi,%rax,1)
  400f40:     48 83 c0 10                 add    $0x10,%rax
  400f44:     44 39 c9                    cmp    %r9d,%ecx
  400f47:     72 af                       jb     400ef8 <daxpy(int*, int*, int, double)+0x48>
  400f49:     44 39 c2                    cmp    %r8d,%edx
  400f4c:     0f 84 97 00 00 00           je     400fe9 <daxpy(int*, int*, int, double)
+0x139>
  400f52:     49 63 c8                    movslq %r8d,%rcx
  400f55:     f2 0f 2a 0c 8e              cvtsi2sdl (%rsi,%rcx,4),%xmm1
  400f5a:     f2 0f 59 c8                 mulsd  %xmm0,%xmm1
  400f5e:     48 8d 04 8f                 lea    (%rdi,%rcx,4),%rax
  400f62:     f2 0f 2a 10                 cvtsi2sdl (%rax),%xmm2
  400f66:     f2 0f 58 ca                 addsd  %xmm2,%xmm1
  400f6a:     f2 0f 2c c9                 cvttsd2si %xmm1,%ecx
  400f6e:     89 08                       mov    %ecx,(%rax)
  400f70:     41 8d 48 01                 lea    0x1(%r8),%ecx
  400f74:     39 ca                       cmp    %ecx,%edx
  400f76:     7e 71                       jle    400fe9 <daxpy(int*, int*, int, double)
+0x139>
  400f78:     48 63 c9                    movslq %ecx,%rcx
  400f7b:     41 83 c0 02                 add    $0x2,%r8d
  400f7f:     f2 0f 2a 0c 8e              cvtsi2sdl (%rsi,%rcx,4),%xmm1
  400f84:     f2 0f 59 c8                 mulsd  %xmm0,%xmm1
  400f88:     48 8d 04 8f                 lea    (%rdi,%rcx,4),%rax
  400f8c:     44 39 c2                    cmp    %r8d,%edx
  400f8f:     f2 0f 2a 10                 cvtsi2sdl (%rax),%xmm2
  400f93:     f2 0f 58 ca                 addsd  %xmm2,%xmm1
  400f97:     f2 0f 2c c9                 cvttsd2si %xmm1,%ecx
  400f9b:     89 08                       mov    %ecx,(%rax)
  400f9d:     7e 51                       jle    400ff0 <daxpy(int*, int*, int, double)
+0x140>
  400f9f:     4d 63 c0                    movslq %r8d,%r8
  400fa2:     f2 42 0f 2a 0c 86           cvtsi2sdl (%rsi,%r8,4),%xmm1
  400fa8:     f2 0f 59 c8                 mulsd  %xmm0,%xmm1
  400fac:     4a 8d 04 87                 lea    (%rdi,%r8,4),%rax
  400fb0:     f2 0f 2a 00                 cvtsi2sdl (%rax),%xmm0
  400fb4:     f2 0f 58 c8                 addsd  %xmm0,%xmm1
  400fb8:     f2 0f 2c d1                 cvttsd2si %xmm1,%edx
  400fbc:     89 10                       mov    %edx,(%rax)
  400fbe:     c3                          retq
  400fbf:     90                          nop
  400fc0:     31 c0                       xor    %eax,%eax
  400fc2:     66 0f 1f 44 00 00           nopw   0x0(%rax,%rax,1)
  400fc8:     f2 0f 2a 0c 86              cvtsi2sdl (%rsi,%rax,4),%xmm1
  400fcd:     f2 0f 59 c8                 mulsd  %xmm0,%xmm1
  400fd1:     f2 0f 2a 14 87              cvtsi2sdl (%rdi,%rax,4),%xmm2
  400fd6:     f2 0f 58 ca                 addsd  %xmm2,%xmm1
```

```
  400fda:     f2 0f 2c c9                cvttsd2si %xmm1,%ecx
  400fde:     89 0c 87                   mov    %ecx,(%rdi,%rax,4)
  400fe1:     48 83 c0 01                add    $0x1,%rax
  400fe5:     39 c2                      cmp    %eax,%edx
  400fe7:     7f df                      jg     400fc8 <daxpy(int*, int*, int, double)
+0x118>
  400fe9:     f3 c3                      repz retq
  400feb:     0f 1f 44 00 00             nopl   0x0(%rax,%rax,1)
  400ff0:     f3 c3                      repz retq
  400ff2:     66 2e 0f 1f 84 00 00       nopw   %cs:0x0(%rax,%rax,1)
  400ff9:     00 00 00
  400ffc:     0f 1f 40 00                nopl   0x0(%rax)
```

daxpyOs.s

```
0000000000400c4d <daxpy(int*, int*, int, double)>:
  400c4d:     31 c0                      xor    %eax,%eax
  400c4f:     39 c2                      cmp    %eax,%edx
  400c51:     7e 1e                      jle    400c71 <daxpy(int*, int*, int, double)+0x24>
  400c53:     f2 0f 2a 0c 86             cvtsi2sdl (%rsi,%rax,4),%xmm1
  400c58:     f2 0f 59 c8                mulsd  %xmm0,%xmm1
  400c5c:     f2 0f 2a 14 87             cvtsi2sdl (%rdi,%rax,4),%xmm2
  400c61:     f2 0f 58 ca                addsd  %xmm2,%xmm1
  400c65:     f2 0f 2c c9                cvttsd2si %xmm1,%ecx
  400c69:     89 0c 87                   mov    %ecx,(%rdi,%rax,4)
  400c6c:     48 ff c0                   inc    %rax
  400c6f:     eb de                      jmp    400c4f <daxpy(int*, int*, int, double)+0x2>
  400c71:     c3                         retq
```