Sistemas Paralelos - 2019

Trabajo Práctico N.º 2

Optimización de programas Serie

Facultad de Informática

Bibliografía:

- Introduction to High Performance Computing for Scientists and Engineers. CRC Press. Hager, Wellein.
- Temas de jerarquía de memoria:
  - Del libro de Grama: 2.2.1 y 2.2.2.
  - Lecturas alternativas en sección 2.1.2 del libro de Dongarra y 2.7 del libro de Thomas Rauber.
- Software Optimization Guide for AMD Family 15h Processors.
  http://support.amd.com/TechDocs/47414_15h_sw_opt_guide.pdf
- Intel 64 and IA-32 Architectures Optimization Reference Manual
  http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf

1) Describir los niveles de optimización 0, 2 y 3 de gcc.

2) Técnica: “Hacer menos trabajo y más liviano”.
a) Solamente aplicando el sentido común, mejorar el siguiente código de programa:

```
int flag = 0;
for (i = 0; i < 1000; i++) {
   if (complex_func(a[i]) < 55)
      flag = 1;
}
printf(“¿Elemento encontrado? %d\n”, flag);
```

b) ¿Cree posible reemplazar la siguiente expresión por una que requiera una instrucción más simple? Ayuda: utilizar operadores binarios.

```
a = a * 4
```

3) Técnica: “Reducción del almacenamiento para datos”.
El siguiente programa debe indicar la cantidad de células vecinas vivas para una célula del Juego de la Vida. Se debe optimizar la memoria utilizada teniendo en cuenta que la cantidad de vecinos siempre es 8 y los valores pueden ser 1 o 0.

```
#include <stdio.h>
void main(void)
{
     int vecinos[8] = {1,0,1,1,0,1,0,0};
     int suma_total = 0;
     int i;
     for(i=0;i<8;i++)
          suma_total += vecinos[i];
     printf("Cantidad de vecinos vivos: %d\n", suma_total);
}
```

4) Técnica: “Extracción de subexpresiones comunes”.
Optimizar el siguiente programa extrayendo subexpresiones comunes.
Compilar con: gcc ejercicio.c -lm -o ejercicio

```
#include <math.h>
#include <stdio.h>

void main(void) {
     int i, n=1000000;
     double x=90,s=500,r=200;
     double valores[1000000];

     for(i=0;i<n/2;i++) {
          valores[i]=i+s+r*sin(x);
          printf("\nValor %d: %lf",i,valores[i]);
     }

     for(i=n/2;i<n;i++) {
          valores[i]=i+s+r*cos(x);
          printf("\nValor %d: %lf",i,valores[i]);
     }
}
```

5) Técnica: “Código en Línea”.
El siguiente código realiza el cálculo de distintas potencias de 2 sin utilizar librerías matemáticas. Optimizar el código utilizando la técnica de Código en Línea.

```
#include<stdio.h>

int potencia (int base, int exponente) {
     int i;
     int res = 1;
     for(i=0;i<exponente;i++)
          res *= base;
     return res;
}

void main(void) {
     char i;
     int res;
     for(i=0;i<=30;i++) {
          res = potencia(2,i);
          printf("\n 2^%d = %d",i,res);
     }
}
```

6) Técnica: “Loop Unrolling”.
El siguiente código suma las filas de una matriz y almacena los resultados en un arreglo. Optimizar el bucle interior utilizando la técnica de Loop Unrolling.

```
#include<stdio.h>
void main(void)
{
     char matriz[4][4]= {{0,1,0,0},{1,0,1,1},{0,1,1,0},{0,0,1,0}};
     char suma[4] = {0,0,0,0};
```

```
     char i, j;
     for(i=0;i<4;i++) {
          for(j=0;j<4;j++)
               suma[i]+=matriz[i][j];
          printf("Fila %d = %d\n",i,suma[i]);
     }
}
```

7) Técnica: “Evitar saltos condicionales”
Optimizar el siguiente programa mediante evitación de saltos.

```
#include<stdio.h>
void main(void)
{
   int m[900][500];
   int i, j;

   for(i = 0; i < 900; i++)
      for(j = 0; j < 500; j++)
         switch (i) {
            case 0 ... 299: m[i][j]=0; break;
            case 300 ... 599: m[i][j]=1; break;
            default: m[i][j]=2; break;
         }

   for(i = 0; i < 900; i++) {
      for(j = 0; j < 500; j++)
         printf(" %d ", m[i][j]);
         printf("\n");
   }
}
```

8) Técnica: “Acceso a caché”.
Indicar el tamaño de las cachés de su procesador. ¿Cuántos datos *int* entran en cada nivel de caché de su procesador?
En el siguiente programa, si se modifica BS (*Block Size*), ¿se modifica el resultado del programa?.
Indicar dos tamaños de BS para los cuales usted está seguro de que el tiempo de ejecución sería muy diferente. Justificar.

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

#define N 100000000
#define BS 1000000

int main() {
   int *a, *b;
   int k, i, r;

   a = malloc(N * sizeof(int));
   memset(a, 0, N * sizeof(int));
```

```
    int f = BS;
    int *u = a + N - 1;
    for (k = 0; k < N; k+=BS) {
       b = a + k;
       if (k + BS > N)
          f = N % BS;
       for (r = 0; r < 50; r++)
          for (i = 0; i < f; i+=16)
             b[i] += 1;
    }
}
```

9) Implementar un programa en C que realice la suma horizontal de un vector de floats utilizando las intrínsecas de SSE2. El vector es muy grande (se requiere memoria dinámica) y el programa debería funcionar para diversos tamaños, simplemente modificando el valor de una constante.

Compilar en GCC con el flag correspondiente para el juego de instrucciones utilizado, en este caso, `-msse2`. Siempre es preferible indicar la microarquitectura del procesador para que el compilador incluya automáticamente todos los flags y además sintonice el código de máquina generado para el hardware específico. Esto último se realiza con la opción `-march=`*`cpu-type`*

En el cluster CDER el compilador GCC es más antiguo que la microarquitectura. Sin embargo, puede usar el compilador de intel que está actualizado. Primero debe cargar el módulo del compilador: `module load Compilers/intel_2018`
Luego compilar con ICC, usando la misma sintaxis (para este caso) que GCC.