Proyecto 0: Multiplicación a la Antigua

Jueves 1ro. de Agosto

I. Descripción General

A diferencia del algoritmo de multiplicación que nos enseñan en la escuela, hay un algoritmo diferente que fue usado por matemáticos egipcios en el año 1800 A.C. aunque también por campesinos rusos en el siglo XIX. En este proyecto evaluaremos la eficiencia práctica de este algoritmo. Se debe escribir un pequeño reporte que resuma los resultados obtenidos y sus conclusiones (obviamente en LaTeX). Toda la programación debe realizarse en C sobre Linux. No se pueden cambiar las especificaciones de este documento.

II. Algoritmo Antiguo

Sean A y B dos números que debemos multiplicar. A puede ser cualquier entero, pero B no puede ser negativo. Necesitamos obtener P que es su producto. El método es:

1. $P = 0$
2. Mientras que $B \neq 0$
   - *a*) Si B es **impar** entonces $P = P + A$
   - *b*) $A = A + A$
   - *c*) $B =$ parte entera de $\frac{B}{2}$

Al final $P = A \times B$. Tomado de "*A Fast, Ancient, Method for Multiplication*" de J. Nyberg (deben leer este artículo).

III. Datos de Entrada

Como paso inicial el programa generará 8 números enteros aleatorios escogidos entre 0 y 9999, los llamaremos x_0 a x_7. Los mismos datos serán usados para evaluar todas las variantes del algoritmo especificadas en este documento.

Estos 8 números etiquetarán las 4 filas y las 4 columnas de una tabla. Cada cruce define una pareja de números que queremos multiplicar. La **Tabla 1** muestra las 16 multiplicaciones que se deben calcular.

Tabla 1

	x_4	x_5	x_6	x_7
x_0	$x_0 \times x_4$	$x_0 \times x_5$	$x_0 \times x_6$	$x_0 \times x_7$
x_1	$x_1 \times x_4$	$x_1 \times x_5$	$x_1 \times x_6$	$x_1 \times x_7$
x_2	$x_2 \times x_4$	$x_2 \times x_5$	$x_2 \times x_6$	$x_2 \times x_7$
x_3	$x_3 \times x_4$	$x_3 \times x_5$	$x_3 \times x_6$	$x_3 \times x_7$

Aunque bien sabemos que la mutiplicación es conmutativa en términos del resultado final, no estamos seguros si son conmutativas con respecto al tiempo que toma calcular la operación con diversos algoritmos. Por tanto, también calcularemos los productos mostrados en la **Tabla 2**.

Tabla 2

	x_0	x_1	x_2	x_3
x_4	$x_4 \times x_0$	$x_4 \times x_1$	$x_4 \times x_2$	$x_4 \times x_3$
x_5	$x_5 \times x_0$	$x_5 \times x_1$	$x_5 \times x_2$	$x_5 \times x_3$
x_6	$x_6 \times x_0$	$x_6 \times x_1$	$x_6 \times x_2$	$x_6 \times x_3$
x_7	$x_7 \times x_0$	$x_7 \times x_1$	$x_7 \times x_2$	$x_7 \times x_3$

Para poder apreciar diferencias de tiempo entre las diversas versiones de las multiplicaciones, cada una se repetirá N veces. Cada grupo de proyecto debe escoger el valor apropiado para N y después, de así desearlo, lo pueden "alambrar" en su código.

Los criterios para escoger N son:

- El tiempo total de corrida del programa debe ser razonable para una "demo".
- Se deben apreciar diferencias en los tiempos acumulados de los algoritmos (usar la unidad más precisa posible de tiempo - no hagan redondeos groseros)
- La máquina particular en la que desarrollen su proyecto y hagan la "demo".

Así, por ejemplo, N podría ser 1 millón de veces, o 50 mil veces, o...

IV. Estructura General

La rutina principal de su programa debe generar los 8 números aleatorios x_0 a x_7. Como se explicó antes, esto va a implicar 32 multiplicaciones (16 en un orden y 16 en el orden contario).

Cada multiplicación se calculará con un llamado a una función a la que se le pasan los argumentos A y B (el orden es muy importante). Esta invocación se repetirá N veces, y se medirá el tiempo **total** de las N ejecuciones. Este dato debe ser conservado para el despliegue final.

En la versión básica de este proyecto hay 3 versiones que deben ser consideradas. En la versión avanzada (**puntos extra**) hay 6 versiones que deben ser consideradas. En cualquiera de los casos, cada versión se repite N veces seguidas con la pareja actual de números y se recolecta el tiempo total de ejecución, con la mayor precisión posible.

V. Algoritmos

Hay 3 versiones de algoritmo que debe programar en Lenguaje C[1]:

1. **Versión Vacía**: la función invocada existe, pero no hace nada más que regresar un cero siempre (i.e., `return 0;`).

[1] se recomienda que deshabiliten la optimización del `gcc`

2. **Versión Estándar**: la función recibe los argumentos A y B y regresa $A \times B$ usando el operador normal de multiplicación de Lenguaje C (i.e., `return (A * B);`).
3. **Versión Antigua**: la función implementa el algoritmo descrito previamente en este documento en Lenguaje C. Deben usar los operadores de bits para *left shift* y *right shift*.

Trabajo extra opcional 1: Las 3 versiones mencionadas arriba deben ser programadas también en lenguaje ensamblador x86. Estas funciones serán invocadas desde el programa principal en C. Así habrá 6 versiones de algoritmo. Se deben recolectar datos de tiempos de ejecución para las 6 versiones.

VI. Salida

Su programa desplegará a consola (de la forma "más bonita posible") los 32 tiempos acumulados (2 tablas rotuladas de 4×4) para cada uno de los 3 ó 6 algoritmos. Etiquete todo apropiadamente. Podría ser recomendable una pausa entre un despliegue y el siguiente.

VII. Reporte Escrito

Cuando se sientan familiarizados con su programa y que hayan recolectado una cantidad representativa de datos, deben escribir un reporte que muestre "pantallazos" capturados de la ejecución de su programa para los 3 ó 6 algoritmos y sus conclusiones. Este reporte debe estar escrito en inglés usando LaTeX.

Entre otras, su reporte debe contestar cosas como:

- ¿Para qué sirve la versión vacía de los algoritmos?
- ¿Es bueno el algoritmo antiguo? ¿Qué tanto?
- ¿Hay ganancia en las versiones de ensamblador? (sólo si las hicieron)
- ¿El orden de los argumentos afecta? ¿Hay alguna regla?

VIII. Requisitos Indispensables

La ausencia de uno solo de los siguientes requisitos vuelve al proyecto "no revisable" y recibe un 0 de calificación inmediata:

- La colaboración entre grupos se considera fraude académico.
- Todo el código debe estar escrito en C (no C++).
- El proyecto debe compilar y ejecutar en Linux. Todo debe estar **integrado**, explicaciones del tipo "*todo está bien pero no pudimos pegarlo*"[2] provocan la cancelación automática de la revisión.
- La presentación debe ser de mucha calidad.
- No debe dar "Segmentation Fault" bajo ninguna circunstancia.
- Hacer la demostración en una máquina que levante Linux de manera real (puede ser dual), es decir no usar máquinas virtuales.

[2]esto incluye los supuestos casos cuando alguien del grupo de trabajo no hizo su parte – el profesor no está interesado en sus problemas de organización.

IX. Fecha de Entrega

Revisiones a las 11:30am el **Jueves 1ro. de Agosto** en la oficina del profesor. Mande además un `.tgz` con todo lo necesario (fuentes, makefile, readme, etc.) a `torresrojas.cursos.05@gmail.com`. Ponga como subject: `A.A. - Proyecto 0 - Fulano - Mengano`, donde Fulano y Mengano son los 2 miembros del grupo.