Análisis del desempeño de C versus C++ en la producción multihilo de cadenas L-System: un caso de estudio ABP

Performance analysis of C versus C++ in multi-threaded production of L-System strings: a PBL case study

J. Jesús Arellano Pimentel*, Guadalupe Toledo Toledo, Mario Andrés Basilo López
José Ricardo Salvador Nolasco, Guillermo Eduardo Reyes Rodríguez
José Alejandro Pérez Sibaja y Samuel Erasto López Díaz

Ingeniería en Computación, Universidad del Istmo, campus Tehuantepec.
Ciudad Universitaria S/N, Bo. Santa Cruz, 4a. Sección, 70760, Sto. Domingo Tehuantepec, Oax., México
*jjap@sandunga.unistmo.edu.mx

PALABRAS CLAVE:

ABP, Lenguajes de programación, L-Systems, Procesamiento multihilo

RESUMEN

La programación orientada a objetos en C++ facilita la codificación de algoritmos respecto al paradigma estructurado del lenguaje C, este echo suele provocar un cuestionamiento válido entre los estudiantes ¿por qué codificar cadenas con memoria dinámica en C cuando los objetos String en C++ evitan ese trabajo? Este tipo de inquietudes permiten gestar casos de estudio de Aprendizaje Basado en Problemas (ABP). En el presente artículo se reporta el contraste de los lenguajes C y C++ a través de la generación multihilo de cadenas L-System usando computadoras personales a disposición de estudiantes de ingeniería en computación. Se realizaron cien corridas para el cálculo de tiempos promedio de ejecución para dos tipos de L-System considerando el procesamiento con balanceo y sin balanceo de carga para dos, cuatro y ocho hilos. Los resultados muestran una mayor velocidad de ejecución para el lenguaje C, pero diferencias interesantes en el Speed Up respecto al lenguaje C++. Al final se concluye que la mejor eficiencia se logra paralelizando con multihilos, siempre y cuando el volumen de los datos sea considerable y esté balanceado, además, la cantidad de hilos no debe rebasar el número de núcleos. Bien vale la pena que los estudiantes lleguen a estas conclusiones mediante el aprendizaje por descubrimiento a través de un caso de estudio.

KEYWORDS:

PBL, Programming languages, L-Systems, Multi-threaded processing

ABSTRACT

Object-oriented programming in C++ facilitates the coding of algorithms with respect to the structured paradigm of the C language, this fact usually causes a valid question among students, why encode strings with dynamic memory in C when String objects in C++ avoid that work? These types of concerns allow the development of case studies of Problem-Based Learning (PBL). This paper reports the contrast of C and C++ languages through the multithreaded generation of L-System strings using personal computers available to computer engineering students. One hundred runs were made to calculate average execution times for two types of L-System considering balanced and unbalanced processing for two, four, and eight threads. The results show a higher execution speed for the C language, but interesting differences in the Speed Up compared to the C++ language. In the end, it is concluded that the best efficiency is achieved by parallelizing with multithreads, if the volume of data is considerable and balanced, and the number of threads should not exceed the number of cores. It is well worth it for students to reach these conclusions through discovery learning through a case study.

• **Recibido: 13 de septiembre de 2023** • **Aceptado: 10 de mayo de 2024** • **Publicado en línea: 1 de junio de 2024**

1. INTRODUCCIÓN

Muchos lenguajes de programación no han logrado superar el paso del tiempo, son pocos aquellos los que por su elegante estructura y poderosas características aún siguen vigentes [1], entre estos están: C, C++, Java y Python. De hecho, de acuerdo con el índice TIBOE para junio de 2023 estos son los cuatro lenguajes más populares [2], en orden del

DOI: 10.30973/progmat/2024.16.2/6

primer hasta el cuarto lugar están: Python, C, C++ y Java. Este dato es interesante considerando la longevidad de cada lenguaje: C fue creado en 1971, C++ en 1983, Python en 1991 y Java en 1995; una longevidad de 52, 40, 32 y 28 años, respectivamente. Prácticamente la edad cualquiera de estos lenguajes supera el promedio de edad de cualquier estudiante universitario que actualmente los esté usando en alguna de sus asignaturas.

En el presente artículo se tiene como objetivo lograr que estudiantes del séptimo semestre de la carrera de ingeniería en computación, que cursan la asignatura de sistemas de cómputo paralelo y distribuido, lleguen a sus propias conclusiones mediante el aprendizaje por descubrimiento a través de la estrategia didáctica de aprendizaje basado en problemas (ABP), respecto de si aún vale la pena invertir tiempo y esfuerzo en codificar cadenas de caracteres con manejo de memoria dinámica en C respecto a codificar objetos String en C++, así como hasta qué punto la ejecución multihilo puede mejorar el desempeño de las aplicaciones que trabajan con cadenas de caracteres. Para tal motivo, se diseñó como caso de estudio la generación multihilo de cadenas L-System [3] debido a su rápido crecimiento por cada iteración de reemplazos, además estas cadenas se pueden segmentar para producirlas en paralelo de forma simultánea, concatenando de forma ordenada, en un solo objeto String o memoria dinámica, lo producido por cada hilo al final de su ejecución.

Cabe mencionar que la elección particular de estos dos lenguajes, cada uno asociado con un paradigma de programación distinto, C del paradigma estructurado y C++ del paradigma orientado a objetos, estriba, por un lado, en que aún siguen siendo populares [2] sobre todo en la academia, y además dado el perfil de un ingeniero en computación ambos lenguajes deberían ser parte del cúmulo de habilidades y conocimientos de los estudiantes. Por otro lado, y no menos importante, también se parte de una inquietud válida entre los estudiantes que en ocasiones sienten que, después de haber conocido las bondades de los lenguajes de programación orientada a objetos, es pura maldad del docente solicitarles codificar algoritmos en lenguajes de programación antiguos sin otro propósito que hacer "músculo mental".

Así pues, a partir de una inquietud válida de los estudiantes respecto a algo que consideran deben aprender, se aprovecha el nicho de oportunidad para gestar un problema que ahora estén motivados a resolver a través de la estrategia didáctica ABP. De acuerdo con [4] el aprendizaje basado en problemas, también conocido como PBL, por sus siglas en inglés, es un enfoque instruccional de aprendizaje activo centrado en el alumno que permite realizar indagaciones, integrar teoría con práctica, y poner en marcha conocimientos y habilidades en el desarrollo de una solución válida a un problema definido.

Existen antecedentes en la literatura de cómo el ABP o PBL aporta al perfil de los egresados de ingeniería, por ejemplo, en [5] realizan una revisión de la literatura para sustentar que el ABP contribuye en el logro de competencias propias del perfil de egreso de estudiantes de ingeniería. En este sentido, identifican características importantes del ABP para la ingeniería, entre las que están: a) aprendizaje centrado en el estudiante abordando temas de su interés, y b) aprendizaje crítico entorno a un problema provocador que alienta a indagar, analizar y proponer soluciones. En particular estas dos características permiten constituir a partir de una inquietud, duda o cuestionamiento de los estudiantes un nicho de oportunidad para aplicar el ABP.

2. TRABAJOS RELACIONADOS

Dado que por muchos años los lenguajes de programación C, C++, Java y Python se han mantenido dentro de los más populares [2], es común que diversos análisis de desempeño

los contrasten entre ellos y con otros lenguajes. Por ejemplo, en [6] parten del hecho de que el manejo de la memoria es un componente esencial de cualquier aplicación, por ello estudian el desempeño en cuanto a la desasignación de memoria dinámica de los lenguajes de programación C#, Java y C++, a través del recolector de basura o el conteo de referencias a objetos. Las pruebas se realizaron en una computadora personal con procesador Intel® Core™ i7-2630QM @ 2.00 GHZ, de 4 núcleos y 8 procesadores lógicos. Sus resultados indican que el recolector de basura de C# es más eficiente en tiempo de ejecución que los procedimientos para el manejo de memoria C++, y este último a su vez es más eficiente que el recolector de basura de Java.

Otro estudio es el realizado en [7] donde reportan el análisis del desempeño de los lenguajes de programación Java, C++, Python y PHP, cada uno con una interfaz de desarrollo integrado propia. Como medida del desempeño utilizan el tiempo de ejecución de programas que involucran funciones recursivas e iterativas como el factorial de números grandes y la búsqueda binaria en grandes arreglos de datos, al final su idea es establecer los lineamientos para una selección apropiada del lenguaje de programación en función del dominio de la aplicación. Así, dentro de sus conclusiones establecen que Java es idóneo para sistemas altamente seguros, mientras que el C++ es mejor para la implementación de software que interacciona directamente con el hardware. No obstante, es de resaltar que en sus pruebas de desempeño Java resultó el mejor evaluado respecto al C++ en cuanto a tiempo de ejecución tanto para el cálculo del factorial como para la búsqueda binaria.

En [8] realizan un análisis comparativo de los lenguajes de programación C, C++, C# y Java. Los parámetros para determinar el rendimiento es el tiempo de ejecución y el Speed Up utilizando procesadores multinúcleo. Las pruebas se realizaron en equipos de cómputo con un total de 48 núcleos divididos en 4 sockets, cada socket tiene 2 nodos NUMA con 6 procesadores y 8GB de memoria RAM local. El número de pruebas consideró la ejecución para 1, 2, 4, 8, 16, 24, 36 y 48 hilos, abordando la factorización LU para matrices dispersas. En sus resultados queda en evidencia que la ganancia del Speed Up de C++ es superior a los otros lenguajes, sin embargo, en tiempo de ejecución el rendimiento del lenguaje C es dominante.

Por su parte, en [9] presentan los resultados de una comparativa del desempeño y complejidad de implementación de aplicaciones multihilo en los lenguajes Rust, Java y C++. El desempeño se midió a través del tiempo de ejecución con operaciones de lectura y escritura a una base de datos multiproceso, mientras que para la complejidad se consideró el número de líneas de código (LOC, acrónimo del inglés Lines of Code) necesarias en cada lenguaje para programar ambas operaciones. Sus resultados muestran que la complejidad de Rust es menor respecto a Java, y que C++ resulta el que más líneas de código requiere, es decir, C++ es el lenguaje más complejo de los tres. En cuanto al desempeño Java es el del menor rendimiento tanto en operaciones de lectura como de escritura, mientras que la diferencia entre C++ y Rust es mínima en ambas operaciones, pero a favor de Rust cuando se llega a un millón de lecturas o escrituras simultáneas. Sin embargo, al realizar las mismas pruebas con múltiples hilos, ahora C++ muestra ligeramente un mejor desempeño que Rust.

Más recientemente, en [10] realizaron una evaluación de desempeño de los lenguajes de programación C/C++, MicroPython, Rust y TinyGo para el microcontrolador ESP32. Sus resultados muestran que los códigos implementados en C/C++ fueron más rápidos en tiempo de ejecución, mientras que Python fue el más lento en comparación con los otros lenguajes. Dos datos de interés que pudieron haber influido en estos resultados son, en primer lugar, el soporte del ESP32 para cada

uno, ya que para C/C++ es completo, pero para los otros lenguajes es alto, limitado o medio. En segundo lugar, está el manejo de memoria, para C/C++ es manual y automático, para RUST completamente automático, y para TinyGo y MichroPython se utiliza el recolector de basura.

De los trabajos relacionados aquí brevemente descritos fue posible identificar tres elementos de contraste entre los lenguajes: 1) tiempo de ejecución, 2) Speed Up, y 3) complejidad en términos de líneas de código. Estos tres elementos conforman la base para determinar si tiene sentido o no codificar cadenas con memoria dinámica en C en lugar de utilizar objetos String en C++. Además, otra característica de los trabajos relacionados es avocarse a un caso de estudio, por ejemplo: a) desasignación de memoria dinámica, b) funciones recursivas o iterativas, así como grandes arreglos de datos, c) matrices dispersas y factorización LU, d) operaciones de lectura y escritura a una base de datos multiproceso, y e) transformada rápida de Fourier (FFT), Secure Hash Algorithm (SHA), entre otros. Para el caso de estudio en este artículo se han elegido dos tipos de cadenas L-System [3]: DOL-System y OL-System.

3. MÉTODO ABP Y DISEÑO DE PRUEBAS

3.1. MÉTODO DE LAS CINCO FASES DEL ABP

Una de las posibles secuencias didácticas para aplicar el ABP es el método de las cinco fases [11]: 1) abordaje inicial del problema, 2) tormenta de ideas y generación de hipótesis, 3) Identificación de los objetivos de aprendizaje, 4) lectura, investigación y trabajo individual como preparación de la plenaria final, y 5) discusión final en grupo para el descarte o aceptación de las hipótesis. Estas cinco fases fueron adaptadas para lograr el objetivo propuesto en el presente artículo. No obstante, como una fase preliminar, al inicio de la asignatura de sistemas de cómputo distribuido y paralelo se planteó a los estudiantes la programación en C con el propósito de abordar conceptos propios de dicha asignatura, por ejemplo: procesamiento multihilo, balanceo de carga, Speed Up, entre otros. Es entonces cuando surge la inquietud de los estudiantes por la posibilidad de sustituir el lenguaje C por Java o C++, después de todo es un hecho que la codificación se facilita y de igual forma es posible corroborar en la práctica los conceptos a tratar.

La secuencia didáctica del ABP se comienza a gestar al proponer como problema medir el desempeño de los lenguajes C y C++ en la generación multihilo de cadenas DOL-System y OL-System, tomando como punto de partida el trabajo de [12] donde se expone la implementación de un generador de L-Systems desde la perspectiva de la teoría de compiladores. Cabe mencionar, que también de forma preliminar se realizaron pruebas para la generación de L-Systems de forma secuencial con los lenguajes Java y JavaScript. Al final se optó por acotar el problema a los lenguajes C y C++ bajo la plataforma Windows utilizando el Visual Studio 2022 como ambiente de desarrollo integrado. Esta decisión se tomó porque en pruebas preliminares los tiempos de ejecución de Java y JavaScript fueron demasiado altos, además JavaScript es monohilo, mientras que la elección de Windows obedece su disponibilidad en todos los equipos de cómputo personal propios o a disposición de los estudiantes, algo que no sucedía con Linux.

Durante la segunda fase del método ABP, después de revisar los referentes de [12], así como algunos de los trabajos relacionados [7], [8] y [9] en la tormenta de ideas surgieron las siguientes hipótesis:

1. La complejidad del código en C será de aproximadamente el doble que el código en C++, esto en términos de la cantidad de líneas de código necesarias.

2. El tiempo de ejecución del código en C no siempre será significativamente mayor que la ejecución del código en C++.
3. El Speed Up básicamente será el mismo para ambos lenguajes al incrementar el número de hilos y la carga de trabajo.
4. El balanceo de carga mejora el rendimiento en cuanto a tiempo de ejecución y Speed Up.

En la tercera fase del método ABP los objetivos de aprendizaje establecidos fueron: 1) conocer y aplicar la forma más precisa posible de obtener el tiempo de ejecución de una tarea en el sistema operativo Windows, 2) contrastar de forma práctica la computación secuencial contra la computación paralela a través de la programación multihilo a nivel de procesadores multinúcleo usando el tiempo de ejecución como parámetro de contraste, 3) comprender y analizar de forma práctica el concepto de Speed Up al resolver un mismo problema con dos, cuatro y ocho hilos de ejecución, es decir, con una paralelización de granularidad media, y 4) evaluar de forma práctica el tiempo de ejecución y Speed Up de la descomposición del dominio de una tarea que crece en tamaño rápidamente balanceando la carga de la forma más equitativa posible entre el número de hilos de ejecución.

Como se puede notar, los cuatro objetivos de aprendizaje establecidos tienen una relación directa con las tres últimas hipótesis, y a su vez dichos objetivos son el punto de partida para la cuarta fase del método ABP. Así, cada estudiante realizó lectura, investigación y trabajo individual como preparación de la plenaria final. Se leyó e investigó un poco más de trabajos relacionados que contrastaran distintos lenguajes de programación, también se indagó y probó de forma práctica respecto a cómo medir el tiempo de ejecución y cómo lanzar múltiples hilos en C y C++ utilizando Windows. Los detalles de codificación y dudas fueron compartidas y debatidas en diferentes sesiones de clase bajo la guía del docente, para posteriormente establecer el diseño de las pruebas a realizar, las cuales se abordan en la siguiente sección. Finalmente, la quinta fase del método ABP se plasma en el apartado de los resultados y discusión de este artículo.

3.2. DISEÑO DE LAS PRUEBAS DE DESEMPEÑO

Como ya se ha mencionado, el caso de estudio es la generación multihilo de cadenas L-System [3] debido a su rápido crecimiento por cada iteración de reemplazos, los cuales se basan en la reescritura sucesiva de ciertas partes de una cadena de inicio o axioma a través de un conjunto de reglas o producciones. Los DOL-Systems se consideran como los sistemas deterministas más simples, como ejemplo característico se tiene el denominado curva del dragón, con un axioma inicial definido por la cadena FX y dos producciones de reescritura: $p_1 \rightarrow$ X = X+YF+, y $p_2 \rightarrow$ Y = -FX-Y; así por cada iteración todos los elementos X deberán ser reescritos por la producción p_1, y todos los elementos Y por la producción p_2, el número de veces que se ejecutan los reemplazos es denominado número de iteraciones. Para la curva del dragón si se ejecuta una iteración el axioma inicial FX se convierte en FX+YF+, si se ejecuta una segunda iteración entonces la cadena se transforma en FX+YF++-FX-YF+, para una tercera iteración la cadena se transforma en FX+YF++-FX-YF++-FX+YF+--FX-YF+, así sucesivamente.

La interpretación de estas cadenas es bastante simple, cada elemento de la cadena se corresponde con un comando gráfico al estilo de la programación logo [12], para los casos de estudio de este artículo los comandos básicos son los siguientes: las letras mayúsculas como F, X o Y indican avanzar trazando, las letras minúsculas indican avanzar sin trazar, y los símbolos + y - indican girar a la derecha y girar a la izquierda respectivamente un ángulo determinado, para la curva del dragón este ángulo es de 90°. La imagen generada después de aplicar nueve iteraciones de

reemplazo para la curva del dragón con una longitud de línea de 5 pixeles se muestra en la Fig. 1.

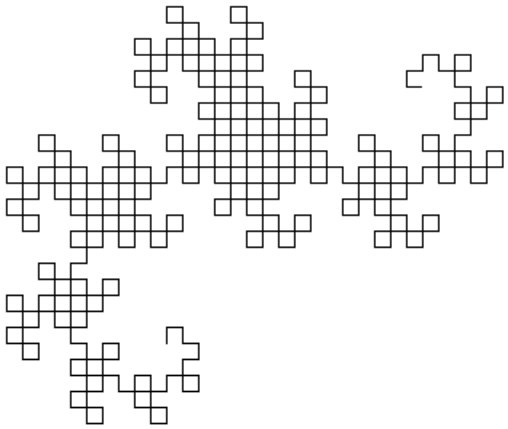


Figura 1. DOL-System Curva del dragón con 9 iteraciones.

En lo que respecta a los OL-Systems se utilizan para modelar cómo se desarrollan ciertas estructuras ramificadas, por ejemplo, árboles simétricos. Básicamente tienen las mismas bases que los DOL-Systems, pero con dos operadores más para señalar una ramificación: el operador [para señalar el inicio de una rama, y el operador] para señalar el fin de la rama. La interpretación de las cadenas generadas es también igual a los DOL-System, pero con la adición de estos dos nuevos operadores que permiten guardar en una pila la posición y orientación actual al iniciar una rama y cuando llega el fin de una rama recuperar de la pila dicha posición. Un ejemplo clásico de los OL-Systems es el denominado árbol binario, el cual parte del axioma inicial X, con una producción $p_1 \rightarrow$ X = X=F[-X][+X]. Para tres iteraciones sobre el axioma inicial la cadena resultante es: F[-F[-F[-X][+X]][+F[-X][+X]]][+F[-F[-X][+X]][+F[-X][+X]]]; y la imagen que se produce, con un ángulo de 25°, las tres iteraciones y una longitud de línea de 30 pixeles es la mostrada en la Fig. 2.

El diseño de la prueba secuencial para estos dos casos genera, en un solo hilo de ejecución, la cadena L-System correspondiente a un determinado número de iteraciones. El diseño de las pruebas para 2, 4 y 8 hilos sin balanceo dividen el tamaño de la cadena entre el número hilos de ejecución considerando un axioma inicial que permita que al menos un elemento reemplazable sea procesado por alguno de los hilos, pero no se garantiza un reparto equitativo de elementos reemplazables entre los hilos. Para el diseño de las pruebas con balanceo sí se considera, de forma predeterminada y sin importar el tamaño del axioma inicial, que cada segmento a procesar por un hilo tenga el mismo número de elementos reemplazables. La importancia de lo anterior se puede ejemplificar fácilmente con el caso del árbol binario, por ejemplo, si el axioma inicial para 2 y 4 hilos fuera el mismo F[-X][+X], entonces, para dos hilos no habría problema si se divide dicho axioma por la mitad de su tamaño, pues cada segmento tiene elementos reemplazables, pero a partir de 4 hilos, nuevamente, solo dos segmentos tienen elementos reemplazables y los otros dos no, por lo tanto no tendría sentido lanzar 4 hilos de ejecución si solo dos ejecutan reemplazos.

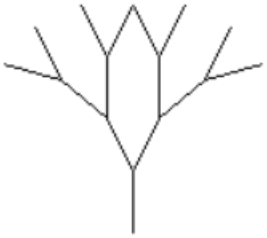


Figura 2. OL-System árbol binario con 3 iteraciones.

Dada la sencillez de código en C++, en la Fig. 3 se presenta la implementación del algoritmo para producir un L-System tipo árbol binario de forma secuencial.

En cuanto al código en C con manejo de memoria dinámica, por razones de espacio, no se coloca en este artículo. Sin embargo, es necesario precisar que al trabajar con arreglos de caracteres, en lugar de objetos string, no está sobrecargado el operador de asignación =, ni mucho menos el operador += para concatenar, por lo tanto, se requiere por cada iteración calcular cuánta memoria

dinámica se requiere y solicitarla al sistema operativo, para tal caso se están invocando funciones propias de Windows [13]: HeapAlloc(), HeapReAlloc() y HeapFree(). En lo que respecta a la toma de tiempo de ejecución, tanto para el caso del código en C y del código en C++, también se hace uso de dos funciones propias de Windows que trabajan en conjunto para obtener mayor precisión en la medición, estas son [14]: QueryPerformanceFrequency() y QueryPerformanceCounter(). De igual forma, para el procesamiento multihilo y sincronización en ambos casos se utilizan las funciones de Windows [15]: CreateThread() y WaitForMultipleObjects(). Cabe mencionar que el ambiente de desarrollo integrado fue el Visual Studio Community 2019 con Microsoft Visual C++ 2019.

```
string w = "X";
string pX = "F[-X][+X]";
string lsys = "", ltemp = "";
unsigned int nr = 15, i, j;
lsys = w;
for (i = 0; i < nr; i++)
{
    ltemp = "";
    for (j = 0; j < lsys.length(); j++) {
        if (lsys[j] == 'X') {
            ltemp += pX;
        }
        else {
            ltemp += lsys[j];
        }
    }
    lsys = ltemp;
}
```

Figura 3. Algoritmo codificado en C++ que produce una cadena L-System tipo árbol binario.

Con los códigos en C y C++ probados y depurados, se planeó medir el tiempo de ejecución de 100 corridas por cada rango de iteraciones de la Tabla 1 en dos equipos de cómputo a disposición de los estudiantes de la asignatura, una máquina de escritorio disponible en el laboratorio de cómputo de la carrera de ingeniería en computación, y un equipo portátil a resguardo del profesor de la asignatura. Las características técnicas de estos dos equipos son: a) Dell Vostro 3471 con procesador Intel® Core™ i5 @ 2.90 Ghz de 6 núcleos y 6 procesadores lógicos, 8 GB RAM, y Sistema Operativo Windows 10 de 64 bits, y b) Lenovo Legion Y740-15IRHg con procesador Intel® Core™ i7 @ 2.60GHz de 4 núcleos y 8 procesadores lógicos, 16 GB de RAM, y Sistema Operativo Windows 11 Home de 64 bits.

Tabla 1. Diseño de pruebas a ejecutar en los equipos disponibles.

Equipos de Cómputo	Lenguaje de programación	L-System	Con/Sin balanceo	Número de Hilos	Rango de iteraciones
Dell Vostro 700 (4 procesadores lógicos) y Lenovo Legion Y740-15IRHg (8 procesadores lógicos)	C	DOL-System Curva de dragón	No aplica	1	1-25
			Sin balanceo	2	2-25
				4	4-25
				8	5-25
			Con balanceo	2	2-25
				4	3-25
				8	4-25
		OL-System Árbol binario	No aplica	1	1-25
			Sin balanceo	2	2-25
				4	4-25
				8	5-25
			Con balanceo	2	2-25
				4	3-25
				8	4-25
	C++	DOL-System Curva de dragón	No aplica	1	1-25
			Sin balanceo	2	2-25
				4	4-25
				8	5-25
			Con balanceo	2	2-25
				4	3-25
				8	4-25
		OL-System Árbol binario	No aplica	1	1-25
			Sin balanceo	2	2-25
				4	4-25
				8	5-25
			Con balanceo	2	2-25
				4	3-25
				8	4-25

4. RESULTADOS Y DISCUSIÓN

El análisis de la complejidad entre ambos lenguajes en términos de líneas de código se presenta en la Tabla 2. Como se puede observar, siempre es mayor el número de líneas de código en el lenguaje C respecto al lenguaje C++, en dos ocasiones con una proporción de poco más del doble de líneas de código para C que para C++, y mínimo con un 42% más de código. Cabe mencionar que las líneas de código contabilizadas incluyen lo necesario para medir el tiempo de ejecución, así como para lanzar el número de hilos correspondiente.

Recuperando la primera hipótesis planteada en la segunda fase del ABP, se puede afirmar que cuando se requiere balancear la carga de trabajo entre múltiples hilos la hipótesis se confirma, es decir, la complejidad del código en C sí es de aproximadamente el doble que el código en C++, sin embargo, esto no

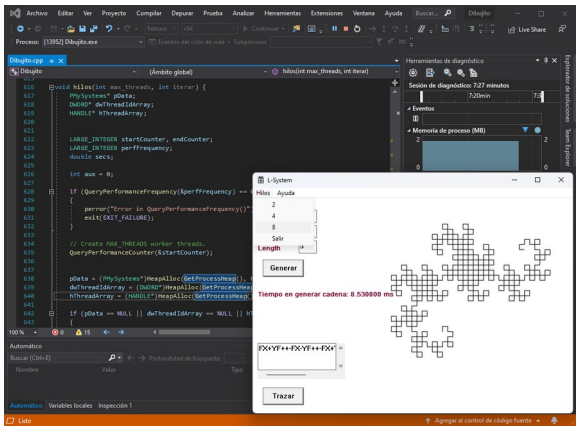
siempre resulta así, ya que si no se requiere paralelizar y la diferencia entre códigos solo estriba en reservar memoria de forma explícita, entonces la diferencia entre códigos no llega a un 50% más de líneas requeridas. A partir de este primer hallazgo resulta de mayor interés para resolver la inquietud que dio origen al problema, observar los contrastes de la eficiencia en términos de tiempo de ejecución y Speed Up entre la generación del DOL-System curva de dragón con ocho hilos para ambos lenguajes, así como la generación del OL-System árbol binario con ocho hilos también.

Tabla 2. Diseño de pruebas a ejecutar en los equipos disponibles.

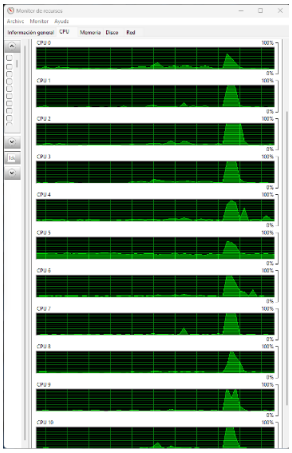
Lenguaje de programación	L-System	Con/Sin balanceo	Número de Hilos	Rango de iteraciones
C	DOL-System Curva de dragón	No aplica	1	61
		Sin balanceo	2, 4 y 8	193
		Con balanceo	2, 4 y 8	249
	OL-System Árbol binario	No aplica	1	58
		Sin balanceo	2, 4 y 8	193
		Con balanceo	2, 4 y 8	245
C++	DOL-System Curva de dragón	No aplica	1	43
		Sin balanceo	2, 4 y 8	122
		Con balanceo	2, 4 y 8	122
	OL-System Árbol binario	No aplica	1	39
		Sin balanceo	2, 4 y 8	121
		Con balanceo	2, 4 y 8	121

La Fig. 4(a) muestra la ejecución y depuración de la aplicación desde el ambiente de desarrollo de Visual Studio Community 2019. En la aplicación es posible apreciar un menú para elegir la cantidad de hilos a ejecutar, así como la visualización de la curva del dragón del caso descrito en el apartado 3.2. En la Fig. 4(b), se muestra la visualización de la carga de trabajo por núcleo en el monitor del sistema de Windows 11, se trata del árbol binario con balanceo de carga y 8 hilos de ejecución.

En el inciso a de la Fig. 5 es posible apreciar como a medida que crece el tamaño de la cadena a generar de la curva del dragón el tiempo de ejecución se eleva, pero siempre el código en C con balanceo reporta el mejor desempeño a medida que el tamaño final de la cadena generada crece. En el inciso b de la Fig. 5 también se muestra el cálculo del Speed Up, tomando en consideración que el equipo de cómputo Dell tiene solo cuatro hilos, y que además se están ejecutando tareas en segundo plano que también demandan tiempo de procesamiento, tiene un poco más de sentido observar la caída en el desempeño a partir de las cadenas producidas de aproximadamente 2 MB de tamaño. Además, también resultó de interés notar que, si bien el C++ muestra un mejor Speed Up siempre, esto no quiere decir que sea más rápido que el código en C de forma generalizada, pues solo refleja que es más rápido hasta por arriba de 5 veces, que la ejecución del algoritmo secuencial escrito también en C++, este comportamiento del Speed Up es similar al observado en [8].



(a) Ambiente de desarrollo y prueba.



(b) Carga de trabajo de los núcleos.

Figura 4. Ejecución y monitoreo de la aplicación L-System.

Si se observa la misma prueba, pero ahora utilizando la computadora Lenovo con más procesadores lógicos, prácticamente un

procesador lógico por hilo, el comportamiento general de los tiempos de ejecución se mantiene, aunque en general también el tiempo disminuye casi a la mitad (ver Fig. 6-a). En lo que respecta al Speed Up, a partir de las cadenas que se producen con un tamaño por arriba de 2 MB, el Speed Up del código en C con balanceo supera al resto de las configuraciones (ver Fig. 6-b). Dado esto, también resulta de interés conocer qué pasa cuando las cadenas finales alcanzan un tamaño de 256 MB, con el caso del OL-System árbol binario dicho tamaño ocurre en la iteración 25. En la Fig. 7 se muestra el desempeño en tiempo de ejecución y Speed Up de la producción de cadenas del OL-System árbol binario, se puede observar que en términos generales el comportamiento se mantiene colocando al caso del código en C con balanceo como el más dominante a medida que crece el tamaño de la cadena final.

Las gráficas presentadas hasta ahora muestran claramente el desempeño de los lenguajes de programación C y C++ cuando el volumen de la carga de trabajo se incrementa. Sin embargo, también es interesante observar qué es lo que ocurre cuando para el lenguaje con mejor desempeño en tiempo se lanzan dos, cuatro y ocho hilos de ejecución para generar una cadena OL-System. La Fig. 8 presenta las gráficas de desempeño para el caso del lenguaje de programación C generando con balanceo el OL-System del árbol binario ejecutándose en la máquina Dell (inciso a) y la máquina Lenovo (inciso b). Es de notarse, por un lado, como el desempeño no mejora incrementando el número de hilos cuando se supera el número máximo de procesadores lógicos, sin embargo, el tiempo sigue siendo menor que con solo dos hilos. Por otro lado, cuando el número de hilos no supera la cantidad de procesadores lógicos el desempeño siempre se incrementa, aunque no al doble.

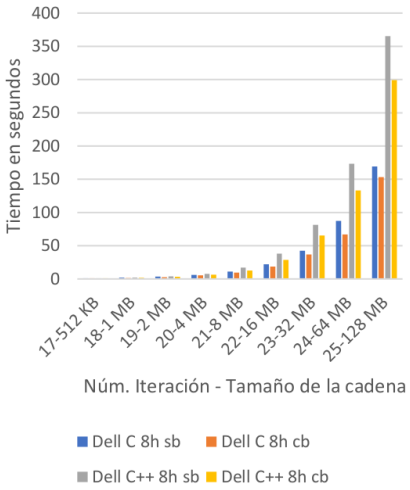
Es importante recalcar que los códigos en C y C++ se diferencian uno del otro solo por el uso de memoria dinámica o administrada, y no por uso de interfaces de desarrollo integrado diferentes o librerías para la programación multihilo, por lo tanto, se tiene una comparativa más equitativa entre ambos lenguajes, esto es algo que no ocurrió en el estudio hecho en [7]. Además, como se menciona en [6], desde la perspectiva de un programador de aplicaciones, es importante tener noción de cuáles son las prestaciones en rendimiento de un sistema de administración de memoria "listo para usar", como el que representa el uso de objetos string en C++, pues, aunque puede resultar más fácil su codificación, no necesariamente podría resultar ideal para ciertos casos, en particular para cuando el tamaño de lo que se va a generar crece con rapidez y la cantidad de procesadores lógicos es limitada.

Con los resultados obtenidos ahora es posible aceptar o rechazar total o parcialmente el resto de las hipótesis formuladas en la segunda fase del método ABP, y a la vez concluir la quinta fase de dicho método:

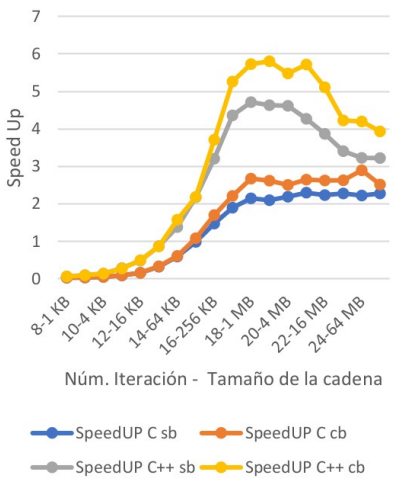
1. El tiempo de ejecución del código en C no siempre será significativamente mayor que la ejecución del código en C++.
   Se acepta parcialmente cuando el tamaño de la cadena final no sobrepasa los 2 MB, pero se debe rechazar si dicho tamaño corre a partir de los 2 MB, pues el desempeño del código en C es mayor.

2. El Speed Up básicamente será el mismo para ambos lenguajes al incrementar el número de hilos y la carga de trabajo.
   Se rechaza esta hipótesis, ya que en general el C++ tiene un mejor rendimiento en Speed Up, excepto para cuando existen suficientes procesadores lógicos y la cadena generada sobrepasa un tamaño de 4 MB.

3. El balanceo de carga mejora el rendimiento en cuanto a tiempo de ejecución y Speed Up.

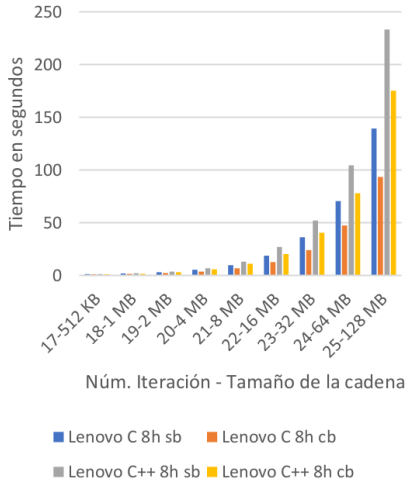


(a) Tiempo de ejecución con 8 hilos

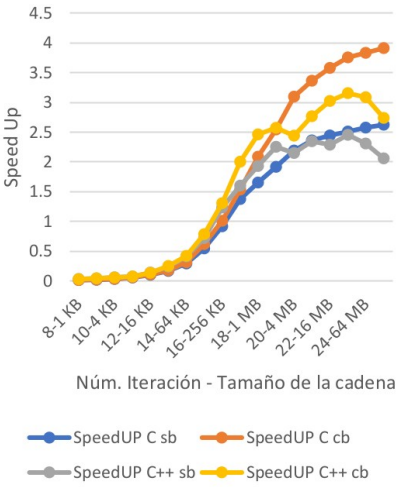


(b) Speed Up con 8 hilos

Figura 5. Desempeño C versus C++ con DOL-System. Curva de dragón en la máquina Dell.

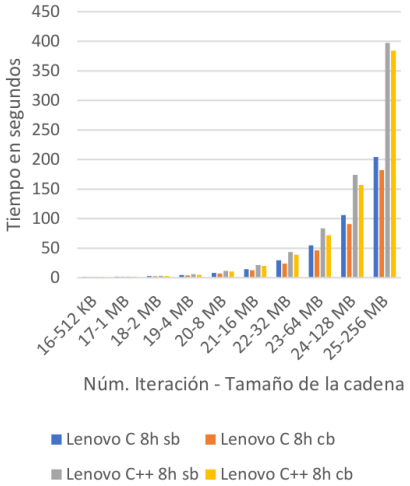


(a) Tiempo de ejecución con 8 hilos

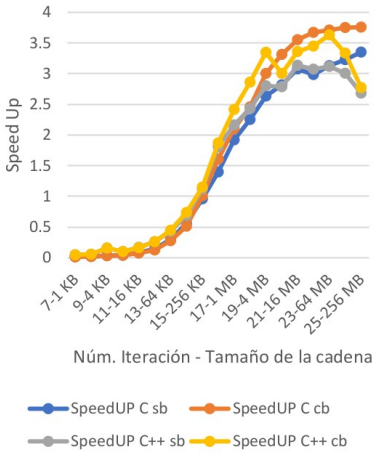


(b) Speed Up con 8 hilos

Figura 6. Desempeño C versus C++ con DOL-System. Curva de dragón en la máquina Lenovo.

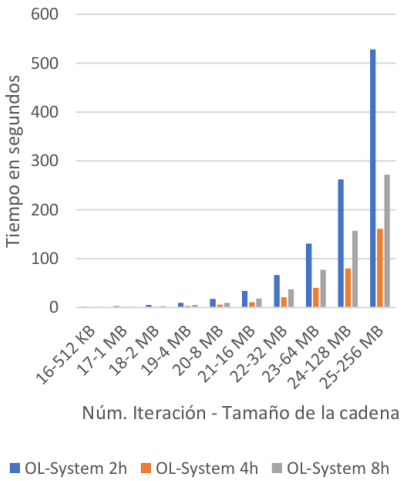


(a) Tiempo de ejecución con 8 hilos

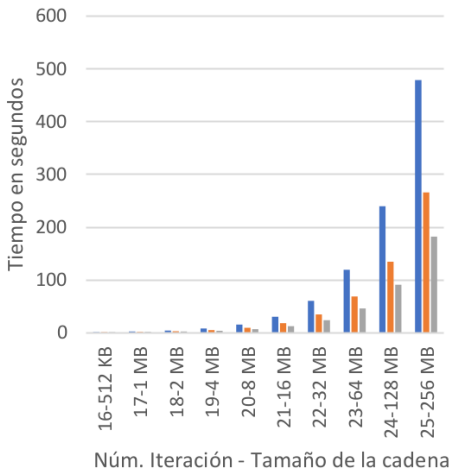


(b) Speed Up con 8 hilos

Figura 7. Desempeño C versus C++ con OL-System. Árbol binario en la máquina Lenovo.



(a) Máquina Dell con 4 procesadores lógicos.



(b) Máquina Lenovo con 8 procesadores lógicos.

Figura 8. Desempeño para el OL-System árbol binario en diferentes procesadores multinúcleo.

Se acepta parcialmente, ya que depende sustancialmente del volumen de los datos, que para las pruebas realizadas es significativo en tiempo de ejecución para un tamaño igual o mayor de 32 MB, mientras que para el Speed Up a partir de un1 MB es suficiente para mejorar el rendimiento.

Una vez que los estudiantes han concluido la quinta fase del método de las cinco fases del ABP [11], han hecho conjeturas, acopio, lectura, investigación, desarrollo de códigos, pruebas y generado suficiente información para llegar a sus propias conclusiones de forma activa, además, en este proceso también se han cubierto los objetivos de aprendizaje planteados en la tercera fase de dicho método.

5. CONCLUSIONES

En el presente artículo se desarrolló un caso de estudio enmarcado en la estrategia didáctica del aprendizaje basado en problemas (ABP o PBL) con el método de las cinco fases [11], para lo cual se tomó como caso de estudio la generación multihilo de cadenas L-System [3] con memoria dinámica en C en contraste con el uso de objetos String de C++. Esto permitió dar respuesta a una inquietud de los estudiantes ¿por qué codificar cadenas en memoria dinámica en C cuando los objetos stirng de C++ evitan ese trabajo?, en este sentido ahora está claro que el desempeño de la implementación de memoria dinámica con C para el manejo de cadenas es superior al uso de los mecanismos de administración de memoria de C++ cuando se trabaja con tamaños de cadenas considerables, para este caso de estudio a partir de los 8 MB la diferencia es muy notoria. Además, también se concluye que si lo que se busca es un mejor desempeño en tiempo de ejecución y Speed Up, lo ideal es la paralelización multihilo con balanceo de carga, pero siempre y cuando no se supere el número máximo de procesadores lógicos disponibles.

Finalmente, es de resaltar que la estrategia didáctica del ABP [4], instrumentada a través del método de las cinco fases [11] resultó idónea para motivar a los estudiantes a querer resolver un problema, más que tener que resolver un problema, utilizando para ello equipos de cómputo a su alcance con la posibilidad de contrastar sus resultados con los resultados de otros trabajos relacionados donde se utilizan equipos de cómputo más especializados, como por ejemplo los del estudio hecho en [8]. Además, lo aquí expuesto a través de la estrategia didáctica del ABP puede ser un punto de partida para otros trabajos a futuro similares a éste, o para

otros que no se limiten a un solo caso de estudio e incursionen en problemas de paralelización más complejos y que no son tan fáciles de asimilar para los estudiantes

REFERENCIAS

[1] Shoaib, M., Sumail, M., Sanjrani, A. A., Ahmed, A. A. Comparative Study of Contemporary Programming Languages in Implementation of Classical Algorithms. *Journal of Information & Communication Technology*. 2021, 14(1), 23-32.

[2] TIBOE. TIOBE Index for June 2023 [en línea]. [recuperado el 20 de junio de 2023] https://www.tiobe.com/tiobe-index.

[3] Prusinkiewicz, P., Lindenmayer, A. *The Algoritmic Beauty of Plants*. New York: Springer-Verlag, 1990.

[4] Savery, J. R. Overview of Problem-based Learning: Definitions and Distinctions. *Interdisciplinary Journal of Problem-Based Learning*. 2006, 1(1), 9-20. doi: https://doi.org/10.7771/1541-5015.1002.

[5] Diestra, S. N., Apolaya, J. P. Perfil de egreso en los estudiantes de ingeniería: aportes significativos de la metodología del Aprendizaje Basado en Problemas. SCIÉNDO. 2021, 24(1), 35-43. doi: 10.17268/sciendo.2021.004.

[6] Henriques, L., Bernardino, J. Performance of Memory Deallocation in C++, C# and Java. In: 18.ª *Conferência da Associação Portuguesa de Sistemas de Informação* (CAPSI'2018). Santarém, Portugal, 2018, 1-18.

[7] Bukie, P. T., Udeze, C. L., Obono, I. O., Edim, B. E. *Comparative Analysis of Compiler Performances and Program Efficiency*. Recuperado el 6 de Junio de 2022 de https://www.preprints.org/manuscript/201909.0322/v1.

[8] Ogala, J. O., Ojie, D. V. Comparative analysis of C, C++, C# and Java programming languages. *Global Scientific Journals*. (2020), 8(5), 1899-1913.

[9] Brandefelt, L., Heyman, H. A Comparison of Performance & Implementation Complexity of Multithreaded Applications in *Rust, Java and C++*, *Thesis, KTH Royal Institute of Technology*, Swedish, 2020.

[10] Plauska, I., Liutkevičius, A., Janavičiūtė, A. Performance Evaluation of C/C++, MicroPython, Rust and TinyGo Programming Languages on ESP32 Microcontroller. *Electronics*. 2023, 12(1), 1-19. doi: 10.3390/electronics12010143.

[11] Restrepo, B. Aprendizaje basado en problemas (ABP): una innovación didáctica para la enseñanza universitaria. *Educación y Educadores*. 2005, 8, 9-19.

[12] Arellano, J. J., Nieva, O., Algredo, I. Aprendizaje Basado en Proyectos Utilizando L-Systems en un Curso de Compiladores. *Programación Matemática y Software*. 2013, 5(1), 82-96. doi: 10.30973/progmat/2013.5.1/7.

[13] Microsoft. Heap Functions [en línea]. *Windows app development documentation* [recuperado el 15 de junio de 2022] de: https://learn.microsoft.com/en-us/windows/win32/memory/heap-functions.

[14] Microsoft. Acquiring high-resolution time stamps [en línea]. *Windows app development documentation* [recuperado el 15 de junio de 2022] de: https://learn.microsoft.com/en-us/windows/win32/sysinfo/acquiring-high-resolution-time-stamps.

[15] Microsoft. Creating Threads [en línea]. *Windows app development documentation* [recuperado el 15 de junio de 2022]: https://learn.microsoft.com/en-us/windows/win32/procthread/creating-threads.

ACERCA DE LOS AUTORES

J. Jesús Arellano Pimentel es Ingeniero en Sistemas Computacionales por el Instituto Tecnológico de Morelia y obtuvo el grado de Maestro en Ciencias en Ingeniería Eléctrica con opción en Sistemas Computacionales por la Universidad Michoacana de San Nicolás de Hidalgo. Además, es candidato a Doctor en Educación con Tecnologías del Aprendizaje y el Conocimiento por la Universidad Virtual del Estado de Michoacán. Se desempeña como Profesor-Investigador de Tiempo Completo adscrito a la carrera de Ingeniería en Computación de la Universidad del Istmo, campus Tehuantepec. También es miembro del Cuerpo Académico de Realidad Virtual y Aplicaciones Didácticas. Sus áreas de interés incluyen: desarrollo de software educativo, sistemas de realidad virtual y prototipos didácticos.

Guadalupe Toledo Toledo es Ingeniera en Sistemas Computacionales por el Instituto Tecnológico de Tuxtepec, y Maestra en Computación Aplicada por parte del Laboratorio Nacional de Informática Avanzada. Actualmente es Profesor-Investigador de Tiempo Completo adscrito a la carrera de Ingeniería en Computación de la Universidad del Istmo. También es miembro del Cuerpo Académico de Realidad Virtual y Aplicaciones

Didácticas. Sus áreas de interés se centran en el desarrollo de software y prototipos didácticos con aplicación en ingeniería y computación, aplicación de evaluaciones de usabilidad a productos de software centrados en el usuario y la integración del cómputo aplicado en la solución de problemas multidisciplinares.

Mario Andrés Basilio López es estudiante de la carrera de Ingeniería en Computación de la Universidad del Istmo, campus Tehuantepec. Sus áreas de interés se centran en el desarrollo de software y bases de datos.

José Ricardo Salvador Nolasco es estudiante de la carrera de Ingeniería en Computación de la Universidad del Istmo, campus Tehuantepec. Sus áreas de interés se centran en el desarrollo de software y bases de datos.

Guillermo Eduardo Reyes Rodríguez es estudiante de la carrera de Ingeniería en Computación de la Universidad del Istmo, campus Tehuantepec. Sus áreas de interés se centran en el desarrollo de software y bases de datos.

José Alejandro Pérez Sibaja es estudiante de la carrera de Ingeniería en Computación de la Universidad del Istmo, campus Tehuantepec. Sus áreas de interés se centran en el desarrollo de software y bases de datos.

Samuel Erasto López Díaz es estudiante de la carrera de Ingeniería en Computación de la Universidad del Istmo, campus Tehuantepec. Sus áreas de interés se centran en el desarrollo de software y bases de datos.