# Programación paralela y concurrente en C++

**William De la Cruz De los Santos** [*]
*Universidad "Juárez" Autónoma de Tabasco, DACB*
*Carr. Cunduacán-Jalpa Km 1, Cunduacán Tabasco, México*
*A.P. 24 C.P. 86690. Tel.(+52)914 336-0928*

**Leopoldo Altamirano Robles** [**]
*Instituto Nacional de Astrofísica, Óptica y Electrónica,*
*Coordinación de Ciencias Computacionales,*

**Rubén A. González García** [***]
**Alejandro González Sánchez** [****]
*Universidad "Juárez" Autónoma de Tabasco, DACB*
*Carr. Cunduacán-Jalpa Km 1, Cunduacán Tabasco, México*
*A.P. 24 C.P. 86690. Tel.(+52)914 336-0928*

Existen muchos lenguajes de programación para especificar programas concurrentes y en paralelo, y frecuentemente no se acoplan con las aplicaciones existentes. Como resultado, han surgido grupos de investigación que buscan la implementación de lenguajes capaces de especificar programas en paralelo y concurrentes, en lenguajes ya existentes, que aprovechen las características actuales de las computadoras. Por otro lado, la Programación Orientada a Objetos (POO) proporciona abstracciones de más alto nivel. Sin embargo, los modelos y lenguajes orientados a objetos, no proporcionan las facilidades para soportar la programación paralela y concurrente. Aquí se revisan las aproximaciones actuales de esquemas y extensiones del lenguaje de programación C++, que soportan la programación paralela y concurrente, así como sus alcances.

Many programming languages exist to specify concurrent and parallel programs and these programming languages frequently do not adapt themselves to existing applications. As a consequence, research teams have risen which seek the implementation of parallel and concurrent programming languages, based on existing languages which take advantage of the present computer's characteristics. On the other hand, the Object Oriented Programming (OOP) provides higher level abstractions. Nevertheless, the Object Oriented models and languages do not provide the facilities to support parallel and concurrent programming. This paper reviews recent approaches of schemes and extensions to C++ programming language that support parallel and concurrent programming.

*Palabras clave: Programación paralela, Programación concurrente.*
*Keywords: Parallel programming, Concurrent programming*

## 1. Introducción

En la actualidad existen muchos lenguajes de programación para especificar programas concurrentes y en paralelo, que muy frecuentemente no se acoplan a las aplica-

[*] william_bosco@hotmail.com
[**] robles@inaoep.mx
[***] ruben.gonzalez@basicas.ujat.mx
[****] alegs@basicas.ujat.mx

ciones ya existentes desarrolladas en lenguajes de programación de uso más común, como es el C++. Como resultado, han surgido grupos de investigación que buscan la implementación de lenguajes capaces de especificar programas en paralelo y concurrentes, en lenguajes ya existentes, que aprovechen las características actuales de las computadoras.

Por otro lado, la programación orientada a objetos hace más fácil el diseño y desarrollo de software, ya que proporciona abstracciones de más alto nivel. Sin embargo, los modelos y lenguajes orientados a objetos, no proporcionan todas las facilidades necesarias para soportar la programación paralela y concurrente.

Durante las últimas décadas, se ha puesto atención en la programación concurrente entre la comunidad de ciencias de la computación, donde un programa concurrente especifica dos o más procesos que cooperan para llevar a cabo una tarea. Cada proceso es un programa secuencial que ejecuta una secuencia de instrucciones; los procesos están sincronizados y cooperan entre si comunicándose. Mientras que un programa paralelo ejecuta estos procesos simultáneamente sobre múltiples procesadores, un programa distribuido es un programa concurrente en el cual los procesos se ejecutan sobre computadoras diferentes y se comunican entre sí a través de una red. Así, la programación concurrente engloba a la programación paralela y distribuida.

Existen varias motivaciones para la programación concurrente: reducir el tiempo de ejecución, incrementar la tolerancia a fallas y explotar el paralelismo explícito e inherente de ciertas aplicaciones, entre otras.

La organización de este documento es de la siguiente manera: en la sección 2 se revisa la historia del lenguaje C++, los intentos para llevarlo a ser un lenguaje de programación concurrente, y los lenguajes actuales para la programación concurrente; la sección 3 discute algunos esquemas para especificar programas concurrentes y en paralelo sobre C++, y en la sección 4 se presenta las conclusiones.

## 2. Revisión histórica

El lenguaje C fue diseñado e implementado en 1972 por Dennis Ritchie. Originalmente fue desarrollado por informáticos para escribir sistemas operativos. Aunque se trata de un lenguaje de alto nivel, incorpora características para trabajar en bajo nivel, permitiendo al programador escribir código extremadamente eficiente. En 1983 un grupo de investigación de los laboratorios Bell, de AT&T, integrado por Bjarne Stroustrup comenzaron a desarrollar el nuevo lenguaje de programación C++ [9]. La meta fue facilitar el desarrollo de programas en C a través de un soporte mejorado, especialmente por una fuerte verificación de tipos y por permitir implementar métodos orientados a objetos [8]. Adicionalmente la eficiencia del C no fue deteriorada y la compatibilidad se mantuvo. Sin embargo, varias características fueron agregadas después al corazón del lenguaje. De estas nuevas característica, probablemente las más importantes son: el concepto de *templates* y *namespaces*, y principalmente el rediseño completo de la biblioteca estándar de C++, conteniendo la conocida Standard Template Library (STL). Los nuevos conceptos fundamentales en C++ son: el concepto de objeto, clases, herencia, un fuerte concepto de tipos, y sobrecarga de operadores y funciones.

La integración de los paradigmas de concurrencia y POO han sido motivo de mu-

chas investigaciones. Dos aproximaciones pueden ser usadas en la integración de los dos paradigmas de concurrencia y POO [1]. La primera aproximación consiste en la introducción de nuevos lenguajes (o la extensión de los lenguajes existentes), que integren las facilidades para la concurrencia dentro del mismo lenguaje. La segunda aproximación usa POO para desarrollar una biblioteca de abstracciones que encapsulen los detalles de bajo nivel de concurrencia. En particular, clases de `C++` pueden ser diseñadas para agregar concurrencia a objetos mediante el mecanismo de herencia.

Existen diversas aproximaciones para la programación concurrente [11, 14]. En el nivel de abstracción más bajo se encuentran los tradicionales modelos de memoria compartida y de paso de mensajes. Los lenguajes basados en el paso de mensajes incluyen a Occam, Ada, SR, entre otros, además de las bibliotecas de funciones como MPI y PVM. Por otra parte se ha diseñado varios modelos para la programación paralela, los cuales proporcionan abstracciones de más alto nivel que ocultan, a los programadores, los detalles del hardware de comunicación y sincronización.

## 3. Esquemas para la programación concurrente en `C++`

Existen muchos debates acerca de la cantidad de lenguajes diseñados y la forma que usan para la integración de la concurrencia y POO. La aproximación basada en bibliotecas, por un lado, mantiene al lenguaje pequeño; permitiendo al programador trabajar con compiladores y herramientas familiares, proporcionando la opción de soportar muchos modelos actuales, y código exportable a otras arquitecturas. Por otro lado, lenguajes nuevos o extendidos pueden usar el compilador para mejorar explotar el paralelismo implícito, para realizar la verificación de tipos y mejorar el desempeño. La mayoría de las aproximaciones para incorporar concurrencia se basan en el uso de bibliotecas. Las características que están presentes en un lenguaje de Programación Orientado a Objetos Concurrentes (POOC) son como se establece en [1]: modelo de memoria, interacción entre objetos, objetos pasivos/activos, concurrencia intra-objetos, y herencia.

### 3.1 Programación concurrente empleando la STL

Las primitivas empleadas para construir programas concurrentes en `C++`, son subclases de la clase base abstracta `Thread` [1] que puede ser usada para construir abstracciones de más alto nivel. Los hilos (*threads*) son unidades de actividades independientes programadas en el sistema. Cada hilo se ejecuta secuencialmente hasta que termina su ejecución, permite ser bloqueado por el sistema o permite ser interrumpido por otro hilo. La clase `Thread` es usada puramente para especificar el comportamiento. El usuario especifica su propia clase `Thread`, derivando de esta clase.

En la mayoría de los casos es suficiente definir una nueva clase, con sólo especificar la función `main()` para ser ejecutada por alguna instancia de esta clase; la clase constructora y destructora. La ejecución del hilo comienza por medio del método `start()`. Algunas otras aproximaciones que emplean bibliotecas para incorporar concurrencia se pueden ver en [5, 13].

```
class Thread
{
protected:
 Thread(const int size, const int pri=0);
 Thread(const Region &stack,const int pri=0);
 virtual Thread( );
 virtual void start(void);
 virtual void main(void);
public :
 virtual void terminate(void);
 static void wait(Resource*);
 static void yield(void);
 static void self terminate(void);
}
```

```
Thread::start(void)
{
 try {
 main();
 }
 catch(...){
 }
 terminate();
}
```

### 3.2 Esquemas para la programación paralela

El primer problema que debe ser tratado con un lenguaje para la programación distribuida es la ejecución en paralelo. Ya que un sistema distribuido tiene por definición más de un procesador, es posible tener más de una parte de un programa corriendo al mismo tiempo. Existen dos clases de paralelismo [4]; el paralelismo puro y el pseudo-paralelismo, donde en el paralelismo puro, cada tarea es ejecutada en un procesador independiente; mientras que en el pseudo-paralelismo dos o más tareas podrían compartir un mismo procesador. Los detalles del paralelismo a veces son ocultados al programador y en otras se conocen. El paralelismo puede ser expresado de varias maneras; un factor importante en estos lenguajes, es la unidad de paralelismo. En un lenguaje secuencial la unidad de paralelismo es el programa completo. Mientras que en un lenguaje para la programación distribuida, la unidad de paralelismo puede ser un proceso, un objeto, una declaración, o una expresión.

En el esquema en el que se extiende al lenguaje para soportar la concurrencia, el paralelismo puede ser expresado como (en donde en ambos casos cada operación es ejecutada en paralelo según la unidad de paralelismo soportada por la arquitectura):

```
PAR
 S1
 S2
```

```
PAR i=0 FOR n
   Process
```

En este esquema es necesario cambiar la sintaxis y semántica del lenguaje de programación, lo que requiere el rediseño del compilador para soportar instrucciones en paralelo. Por otro lado, como en la programación concurrente, en el esquema de POO y programación paralela a través de bibliotecas, también se han hecho algunas implementaciones para soportar algoritmos paralelos. Esta aproximación proporciona un poderoso mecanismo para la extensión del lenguaje por medio de clases, herencia y templates. Dos ejemplos de este enfoque son: el presentado por Matthew Austern, Ross Towle y Alexander Stepanov de Silicon Graphics, Inc. en [2], donde describen un nuevo tipo de adaptador para la biblioteca STL de C++, que puede ser usada para expresar paralelismo; y el presentado por Scott Baden de la Universidad de San Diego California en [3], en el cual propone dos bibliotecas en C++, LPARX y KeLP

principalmente para soportar cálculos no uniformes como los que surgen al emplear métodos adaptivos para resolver ecuaciones diferenciales parciales.

Particularmente existe la PSTL (Parallel STL), una extensión de la STL creada por el consorcio HPC++ (High Performance C++) [10], en donde la arquitectura estándar del modelo soportado por HPC++ es un sistema compuesto por un conjunto de nodos interconectados, donde cada nodo es un multiprocesador con memoria compartida (SMP) que puede tener varios contextos y espacios de direcciones virtuales. Los iteradores paralelos en HPC++ son generalizaciones de los punteros globales [12] en HPC++, y que extienden su funcionalidad.

Así, cada contenedor distribuido (clase), tendrá una subclase para la forma más fuerte de iterador paralelo que este soporte y funciones iteradoras "begin" y "end". Por ejemplo, cada clase contenedora tendrá funciones de la forma:

```
Template <class T>
    class Container {
          ...
          class ParallelIterator{...}
          ParallelIterator parbegin();
          ParallelIterator parend();
};
```

## 4. El lenguaje C++

C++ (DC++) es un lenguaje para escribir aplicaciones paralelas distribuidas. La idea fundamental se apoya 3 tipos de clases de C++: clases normales de C++, clases que actúen como *gateways* entre procesadores abstractos, y clases cuyas instancias puedan ser enviadas por valor entre procesadores abstractos a través de *gateways* [7]. Las 3 categorías de clases están implementadas como clases normales de C++. Las declaraciones del compilador de DC++ añaden clases *gateway* y de "*valor*" con operaciones que soporten su utilización. Esta propuesta permite que los *gateways* puedan ser redistribuidos (dinámica o estáticamente) sin la necesidad de modificar el programa. En este esquema, la concurrencia se logra creando múltiples procesadores abstractos e iniciando múltiples hilos de control dentro de estos procesadores. DC++ soporta transparentemente la multitarea, de manera que el número actual de procesadores puede cambiar sin hacer modificaciones al programa.

En realidad DC++ es una extensión de C++. Dichas extensiones son muy simples y directas:

1. dos nuevas clases pre-construidas: DcDomain y DcThread.
2. dos nuevos tipos de clases: *gateways* (pasarelas) entre dominios e instancias de "valor" que pueden ser enviadas entre dominios a través de la invocación y el retorno de funciones miembros.

Estas instancias de valor son similares a los objetos serializados en JAVA 2.

Un domino (*domain*) es similar a un monitor (un mecanismo IPC); solamente un hilo de control puede estar activo en un momento dado. Un dominio es creado dándole

a su constructor el número de procesador físico en el cual se va alojar. En este sentido, un dominio puede ser considerado como un procesador abstracto. Para ejecutar una función o una función miembro en un dominio específico, es necesario crear un hilo (*thread*). Una vez iniciado, un hilo puede ser enviado a través de múltiples dominios. La única restricción es que solamente puede haber un hilo activo a la vez en un dominio dado.

Los hilos en DC++ se comunican a través de *gateways*. Un *gateway* es un objeto creado dentro de un dominio. Las referencias a un *gateway* pueden ser enviadas a otros dominios. Cualquier método invocado en un gateway referenciado remotamente, será por definición una llamada a un procedimiento remoto (*Remote Procedure Call*, RPC).

DC++ no soporta variables globales, ya que estas representan un problema en los sistemas de memoria distribuida. Un programa típico en DC++ tiene una función *main()* que crea dominios y gateways para esos dominios. Después crea hilos en uno o más de esos dominios invocando una función miembro del *gateway*. Todo el acceso a los datos es a través de las funciones miembro.

Si es necesario manejar variables globales, se puede crear un domino "global" que implemente y manipule dichas variables. Dicho dominio "global" tendrá asociado un *gateway* cuya referencia será enviada a otros dominos, y a través de la cual se podrá tener acceso a los métodos para manipular las variables "globales" [6].

Finalmente, un objeto de "valor" es un objeto que hereda de la clase DcValue. Esto le dice al compilador que cuando uno de estos objetos es pasado por valor en un argumento de invocación a un *gateway* o retornado como resultado de la ejecución de un *gateway* (por razones obvias, DC++ no permite el paso de parámetros por referencia al invocar un *gateway*), dicho objeto es copiado totalmente. Esto significa que cualquier objeto, punteros a objetos y tipos de datos inter-construidos, contenidos en el objeto enviado, deben ser copiados recursivamente, y cualquier estructura de compartición, presente en forma de punteros, debe ser preservada. A este tipo de copias también se le conoce como copia en profundidad y es necesaria debido a la naturaleza de la memoria distribuida del sistema.

DC++ ha inspirado la creación de otras herramientas y modelos de programación concurrente, tal como CxC.

## 5. Conclusiones

Los intentos por extender al lenguaje C++ a diferentes paradigmas de programación a través de la programación orientada a objetos, contradice lo que muchos pensaban acerca de las áreas de aplicación del C++. Particularmente en la programación concurrente y paralela, donde los lenguajes dominantes son JAVA y JFORTRAN, respectivamente. Sin embargo, el esquema empleado, de extender la biblioteca STL de C++, en donde los niveles jerárquicos y de abstracción juegan un papel importante, demuestra que el alcance del lenguaje C++, no está limitado.

## Referencias

[1] Arjomandi, E., Kalas, I., O'Farrell, W., *Concurrency Abstractions in a C++ Class Library*. IBM Press. 1993.

[2] Austern, M. H., Towle, R. A. and Stepanov, A. A., *Range Partition Adaptors: A Mechanism for Parallelizing STL*, Silicon Graphics, Inc., (http://www.stepanovpapers.com/p5-austern.pdf)

[3] Baden, S. B., *Software Infrastructure for Non-Uniform Scientific Computations on Parallel Processors*, San Diego supercomputer Center CA. 1996.

[4] Bal, H. E., Steiner, J. G., Tenenbaum, A. S.,*Programming Languages for Distributed Computing Systems*, ACM Computing Surveys **21**(3), September 1989.

[5] Briot, J. P., Guerraoui, R., Lohr, K. P., *Concurrency and Distribution in Objetc-Oriented Programming*, ACM Computing Surveys, **30**(3), September 1998.

[6] Carr, H., *Distributed C++*, PhD Thesis, University of Utah, 1994.