JIDeTEV
JORNADAS DE INVESTIGACIÓN, DESARROLLO TECNOLÓGICO, EXTENSIÓN, VINCULACIÓN Y MUESTRA DE LA PRODUCCIÓN

Vol1-Año XXXX-**ISSN 2591-4219**

Estrategias de Implementación Multihilo del Algoritmo MEMD Basada en el Paradigma de Datos Disponibles

Roberto N. Schuster[a,c], Javier E. Kolodziej[a,b]

[a] *Facultad de Ingeniería, Universidad Nacional de Misiones (UNaM), Oberá, Misiones, Argentina.*
[b] *Grupo de Investigación y Desarrollo en Ingeniería Electrónica (GID-IE), IMAM, UNaM-CONICET, Facultad de Ingeniería, Oberá, Argentina.*
[c] *LABSE, FI-UNaM, Juan Manuel de Rosas 325, Oberá, Misiones, Argentina*
e-mails: schusterrobertonicolas@gmail.com, koloj@fio.unam.edu.ar

Resumen

El siguiente artículo, presenta algunas estrategias de programación para la implementación del algoritmo de descomposición modal empírica multivariable (*Multivariate Empirical Mode Decomposition* - MEMD), teniendo como objetivo principal hacer uso de la capacidad multihilo que presentan los sistemas operativos actuales y el aprovechamiento de los nuevos procesadores multinúcleo. Se utiliza para la implementación el software LabVIEW.

Palabras Clave *– MEMD, Multihilo, LabVIEW*.

*Roberto N. Schuster

1. Introducción

El avance de la tecnología de los procesadores hacia chips que contienen más de un núcleo llamados multinúcleo, ha llevado a popularizar el paradigma de multihilo debido a su capacidad de ejecutar múltiples procesos de forma más eficiente y hasta de forma paralela en algunos casos.

Sin embargo, la técnica de programación multihilo es un tópico avanzado que precisa un conocimiento fundamental sobre el mismo de parte del programador. Es aquí donde el modelo de "flujo de datos" o código G, que utiliza LabVIEW®, presenta ventajas cuando se trabaja con multitarea y multihilo, ya que el programador observa a simple vista las tareas que pueden ser ejecutadas de forma paralelas. Otra ventaja se da en la completa abstracción de los hilos, ya que los programadores que utilizan este concepto no deben generar, destruir o sincronizar los mismos [1] lo cual permite implementar esta tecnología sin incrementar el tiempo de desarrollo o la complejidad del programa.

2. ¿Qué es la Multitarea, multihilo y multiprocesamiento?

2.1. Multitarea

Que un sistema sea multitarea, o *multitask*s en inglés, significa que el mismo es capaz de coordinar múltiples tareas y ejecutarlas de tal manera que parezca que las mismas se realizan de forma simultánea. Para ello, el sistema operativo tiene la capacidad de cambiar entre tareas de forma rápida.

Existen dos tipos básicos de multitarea, la multitarea cooperativa y la multitarea preventiva. En la primera, el sistema operativo nunca inicia un cambio de contexto, sino que, los procesos entregan el control de forma voluntaria periódicamente [2], este tipo de multitarea funciona debido a que todas las tareas deben tener más o menos las mismas prioridades, por lo tanto, no hay necesidad de interrumpir una para pasar a otra [3].

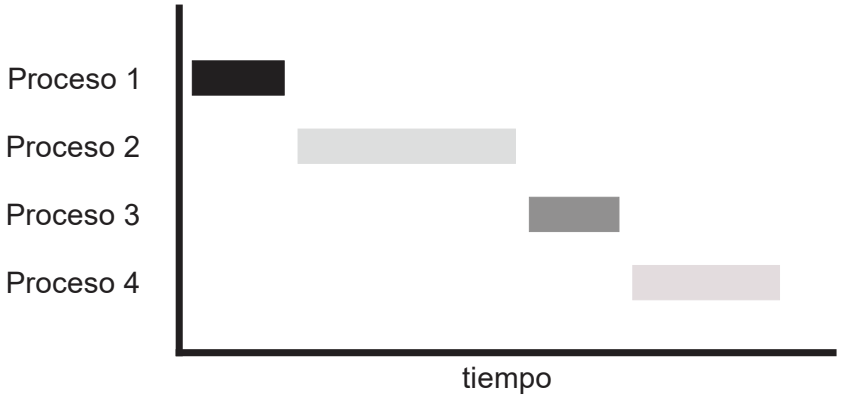


Fig. 1. Ejemplo multitarea cooperativo [2].

En el caso de la multitarea preventiva el sistema operativo es el encargado de administrar el tiempo de uso entre los distintos procesos, donde cada uno de ellos utilizará el procesador durante lapsos de tiempo cortos. El cambio entre procesos se da mediante una interrupción la cual suspende el proceso que se está ejecutando e invoca al planificador para determinar que proceso se debe

ejecutar a continuación. Este tipo de multitarea se utiliza cuando existe una gran variedad de prioridades entre las tareas que se encuentran ejecutando [3].

A diferencia del método cooperativo, el preventivo permite al sistema garantizar a cada proceso una porción regular de tiempo operativo, además, de atender de forma rápida eventos externos como ingreso de datos que pueden requerir la atención de uno u otro proceso.

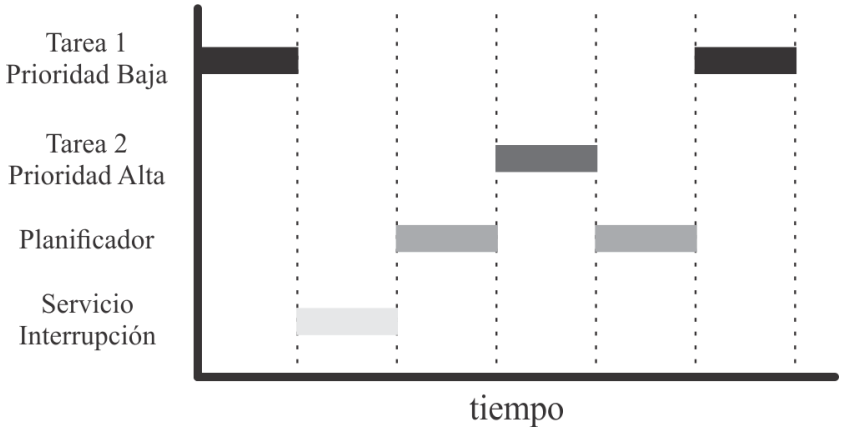


Fig. 2. Ejemplo multitarea cooperativo [3].

2.2. Multihilo

El paradigma del multihilo, extiende la idea de multitarea a las aplicaciones, de tal forma que operaciones específicas dentro de una aplicación pueden ser divididas en hilos individuales [4]. Estos hilos, en teoría, podrían llegar a ejecutarse de forma paralela, especialmente en sistemas con multiprocesadores donde cada procesador puede ejecutar un hilo en forma simultánea. Por ello, una aplicación se ejecuta de forma más eficiente cuando lo hace utilizando multihilos.

La programación multihilo es un tópico avanzado, el cual requiere que los programadores posean un conocimiento fundamental de la tecnología. Esto se ve en lenguajes de programación basados en texto, cuya ejecución de las aplicaciones en dicho lenguaje se realiza de forma secuencial, donde la creación, destrucción y sincronización debe ser realizada por el programador [1]. Entre los beneficios que presenta el uso de multihilo se pueden enumerar [4]:

- Uso más eficiente del CPU: en un sistema multihilo se evita que una tarea bloquee a las siguientes mientras se espera que se termine dado que las otras tareas se pueden ejecutar en otro hilo de forma independiente.
- Mejor confiabilidad del sistema: el uso de diferentes hilos de ejecución evita que ciertas operaciones afectan a otras más importantes.
- Mejor rendimiento en sistemas multiprocesador: esto se debe a que cada procesador puede ejecutar un hilo diferente.

3. Multihilo con LabVIEW

LabVIEW presenta un lenguaje de programación gráfico basado en flujo de datos, el cual a su vez se basa en el paradigma de programación de “datos disponibles”, en el cual el flujo de ejecución

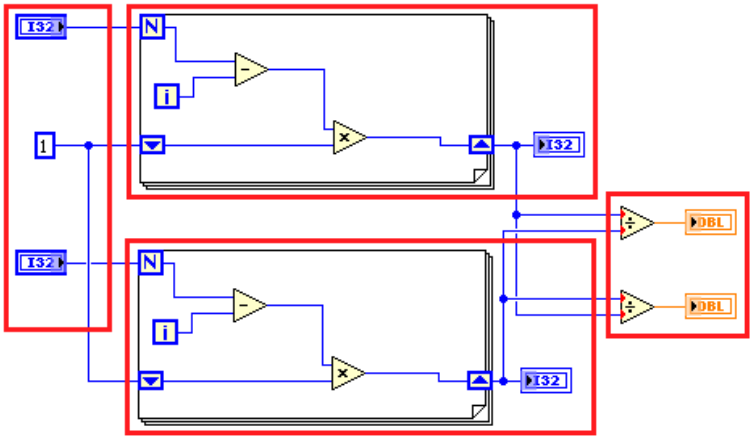
se encuentra determinado por la estructura del diagrama de bloques, y la ejecución de las funciones que componen la aplicación se da a medida que los datos de entrada se encuentran disponibles.

Al ser gráfico el lenguaje de programación utilizado, facilita el desarrollo de código que se ejecute en paralelo, dado que el mismo evidencia las porciones de código que con estas características. Sin embargo, para aprovechar la tecnología multihilo y el procesamiento en paralelo es necesario tener ciertos recaudos, que eviten una ejecución secuencial del código.

En la Fig. 3 se puede observar una comparación entre un script de Matlab y el mismo código desarrollado en código G en LabVIEW. Mientras el primero se ejecutaría solamente en un hilo debido a que es secuencial, el que se encuentra desarrollado en LabVIEW tendría tres hilos, uno correspondiente a la entrada y salida de datos, y los otros dos relacionados a cada ciclo *for*.

```
A = 5;
C = 1;
for i = 1 : A
C = A - (1 + 1)*C;
end

B = 4;
D = 1;

for i = 1 : B
D = B - (1 + 1)*D;
end

X = C / D;

Y = D / C;
```

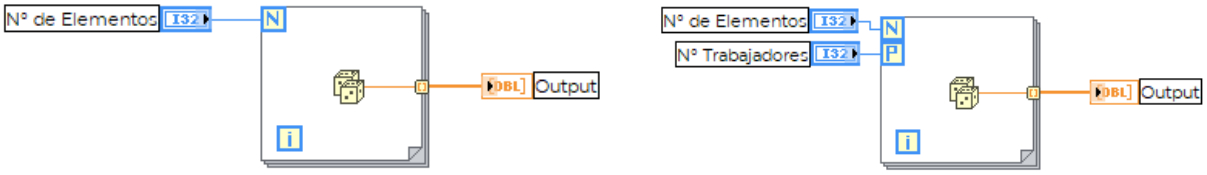
(a)



(b)

Fig. 3. (a) Programa realizado en Matlab. (b) Programa realizado en LabVIEW.

Además, LabVIEW presenta la capacidad de procesar ciclos *for* de forma paralela lo cual permite que los mismos se ejecuten de manera más rápida en sistemas multiprocesador, para ello es necesario que los ciclos sean independientes entre si. Esta herramienta debe utilizarse en casos de operaciones intensivas, sino no se observará mejoría en el rendimiento del código.



(a) (b)

Fig. 4. (a) Bucle *for* sin ejecución en paralelo. (b) Bucle *for* configurado para ejecutarse de forma paralela con *N* trabajadores.

Los patrones de programación en paralelo más usados de la industria [5] son el paralelismo de tareas (Fig. 5) que representa a dos o más tareas donde las operaciones que se realizan no dependen una de otra por lo tanto pueden ser ejecutada de forma paralela.

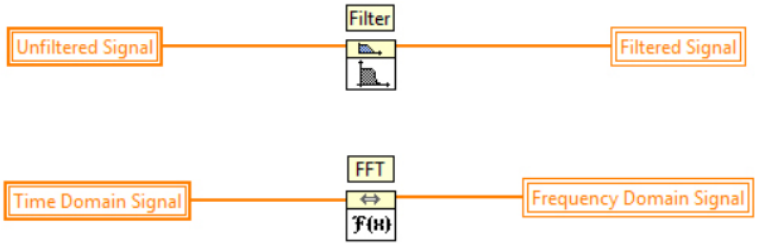


Fig. 5. Ejemplo de ejecución de tareas en forma paralela.

Paralelismo de datos, se da en los casos donde es posible partir una gran secuencia de datos y procesar cada pieza de forma simultánea antes de combinar los resultados.

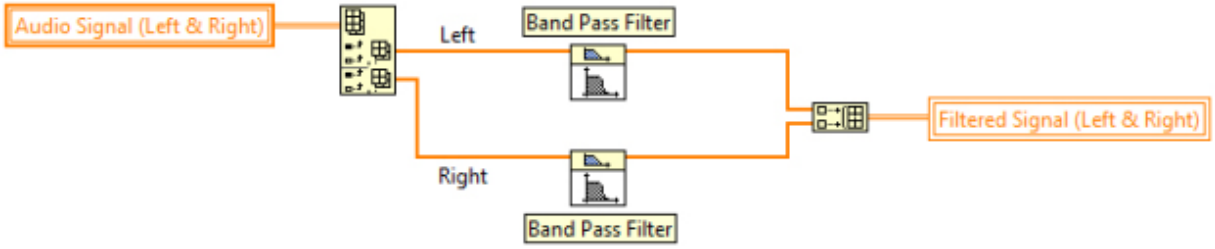


Fig. 6. Ejemplo de paralelismo de datos.

Por último, se tiene el *pipelining*, el cual en cada iteración ejecuta una operación. En este caso, se busca que todas las operaciones tengan un tiempo de ejecución parecido, para que no exista tiempo muerto porque una operación toma más tiempo en terminar que el resto.

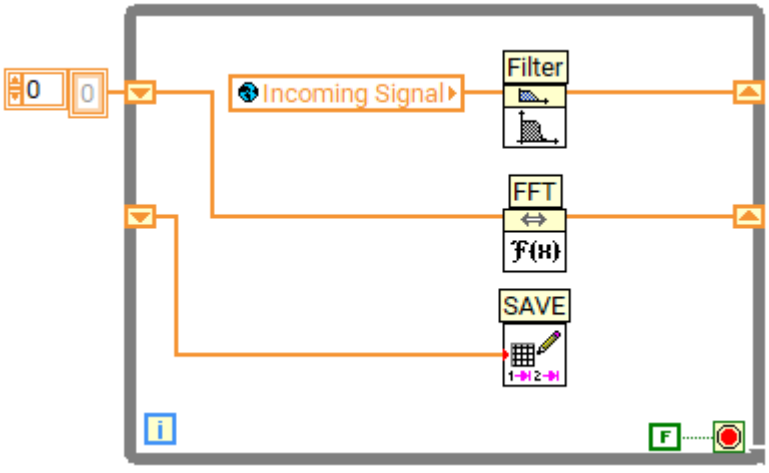


Fig. 7. Ejemplo de *pipelining* en LabVIEW utilizando *shift register*.

4. Descomposición Modal Empírica Multivariable (*Multivariate Empirical Mode Decomposition* – MEMD)

El algoritmo MEMD [6] es una extensión del algoritmo EMD que es un método totalmente basado en datos, diseñado para la descomposición multiescala y el análisis en tiempo y frecuencia de señales reales. La versión multivariable aparece como una solución a aquellos casos que se tiene más de una dimensión, donde hasta el momento se había aplicado el algoritmo EMD, en su versión

unidimensional, de forma individual a cada variable. El uso del EMD multivariable ayuda a evitar el problema de alineación de modos que parece al utilizar el algoritmo EMD estándar.

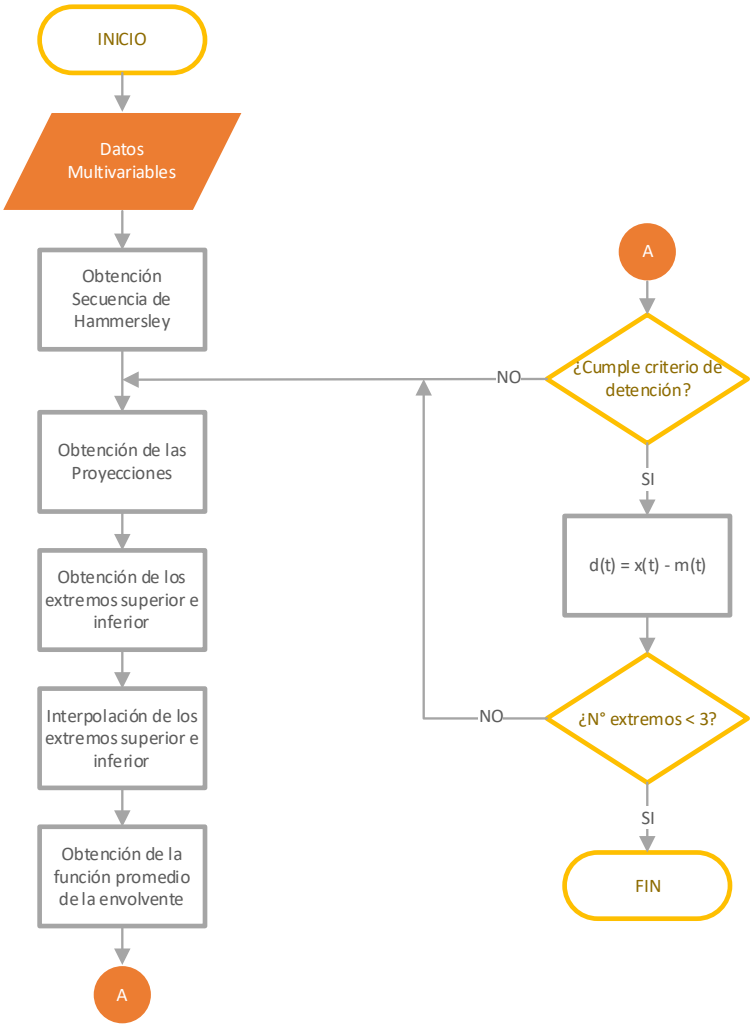


Fig. 8. Diagrama en bloques básico del algoritmo MEMD.

5. Algoritmo MEMD con capacidad multihilo desarrollado en LabVIEW

En la siguiente sección se presentan algunas partes de un algoritmo MEMD desarrollado en LabVIEW que presenta capacidad multihilo. El algoritmo implementado es el propuesto en [6] para Matlab. Cabe resaltar que este algoritmo no presenta capacidad de operación en tiempo real.

En la Fig. 9 se presenta el algoritmo MEMD desarrollado, el mismo está compuesto por tres subVIs dentro de los cuales se ejecutan las diferentes operaciones necesarias para el cálculo de las funciones de modo intrínseco (*intrinsec mode functions* - IMF).

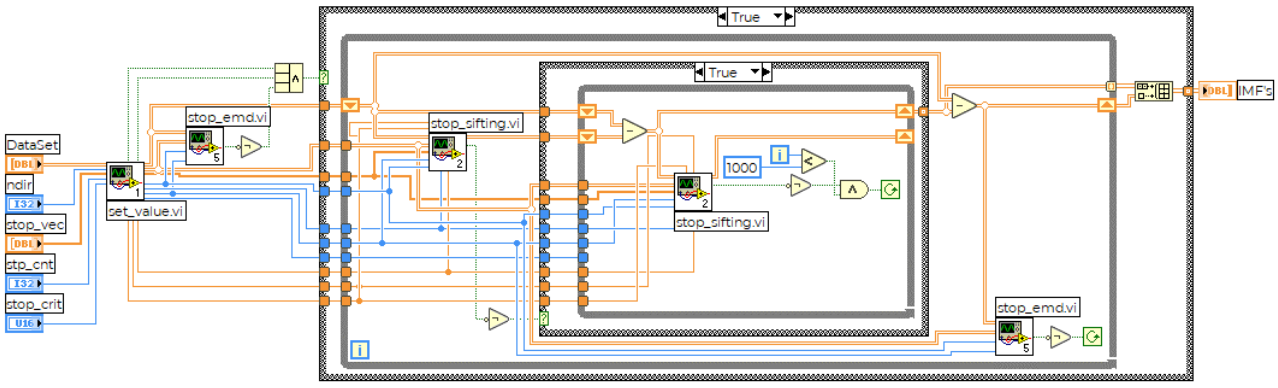


Fig. 9. Algoritmo MEMD en LabVIEW

En la Fig. 10 se presenta el subVI denominado stop_sifting, en el mismo se puede observar que la programación esta realizada de una forma no secuencial a excepción de los casos en que los bloques precisan datos provenientes de otros bloques. Además, en el recuadro rojo se observa un bucle *for* el cual computa la suma de cada fila, por lo tanto, el mismo puede ser paralelizado.

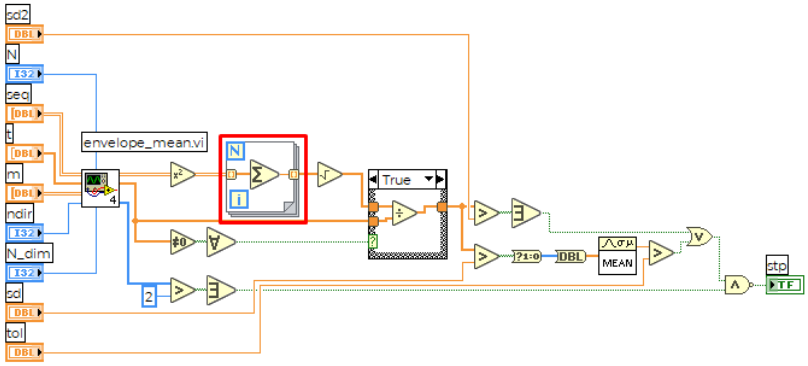


Fig. 10. stop_sifting

En la Fig. 11 se observa parte del código con el cual se obtiene las envolventes superior e inferior, las cuales se encuentran separadas en dos ciclos *for*, de tal forma que el compilador de LabVIEW puede colocar cada ciclo en un hilo diferente. Lo mismo se observa en el código para el cálculo de los extremos superior e inferior de la Fig. 12, en la misma se puede observar que se utiliza un bloque denominado “peaks.vi” tanto para el cálculo de los extremos superiores como los inferiores.

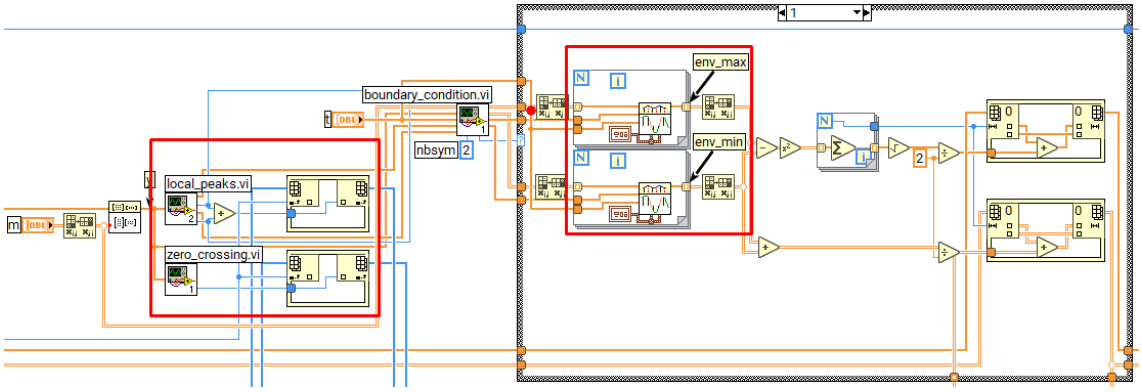


Fig. 11. Algoritmo con el cual se obtiene el vector dirección a partir de la secuencia de Hammersley.

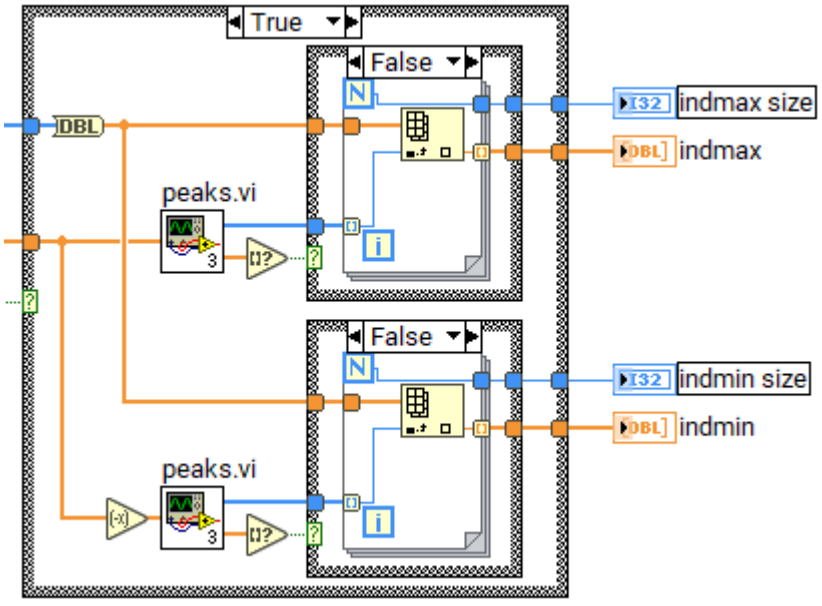


Fig. 12. Cálculo de los extremos superior e inferior.

En este caso el bloque “peaks.vi” se encuentra configurado como un subVI del tipo reentrante para que múltiples instancias del mismo puedan ser ejecutadas de forma paralela y por ende calcular los picos superiores e inferiores de forma paralela.

6. Conclusiones

En el presente trabajo se muestra la herramienta de análisis MEMD y una propuesta de implementación multihilo en LabVIEW, enfatizando las principales partes del código cuya programación favorece la ejecución en múltiples hilos, al igual que aquellas partes del código donde se puede mejorar aún más esto.

Además, se presentan definiciones de multitarea y multihilo en los sistemas de cómputos, referenciando las ventajas que presenta el lenguaje de programación basado en flujo de datos para la visualización de partes del código que pueden correr de forma paralela, algo que en los lenguajes de programación basado en texto se dificulta.

Se evidencian las ventajas que presenta LabVIEW respecto a la programación multihilo, siendo el mismo transparente para el programador dado que LabVIEW se encarga de la creación, destrucción y sincronización de los hilos. Sin embargo, se hace hincapié que para poder aprovechar la capacidad multihilo es necesario seguir ciertas pautas de codificación.

Finalmente, cabe aclarar que, si bien la propuesta presentada es para procesamiento *off line* de datos, si se requiere un análisis en tiempo real, el código presentado podría ser modificado para tal caso, llevando el mismo a un esquema productor/consumidor, donde el productor sería la adquisición y el consumidor la parte de procesamiento. Utilizando este esquema lo que se realiza es la separación del bucle de adquisición del de procesamiento, por lo tanto, los mismos se ejecutan en hilos diferentes.

Referencias

[1] R. Bitter, T. Mohiuddin, M. Nawrocki, "Multithreading in LabVIEW", in *LabVIEW Advanced Programming Techniques* 2nd ed., CRC Press, 2001, chapter 9.

[2] J. Bartel, "Non-Preemptive Multitasking", *The Computer Journal*, n°30, pp. 37 – 38, 1987, ISSN 0748-9331

[3] D. M. Auslander, "Control Implementation: Real Time Multitasking/Multiprocessing", *Advanced in Control Education*, pp. 181 – 187, Massachusetts, 1991

[4] N. Dorst, *AN 114: Using LabVIEW to Create Multithreaded VI for Maximum Performance and Reliability*, National Instruments, 2000.

[5] "Multicore Programming with LabVIEW," Accessed on: Jul, 7, 2020, https://www.ni.com/es-cr/innovations/white-papers/13/multicore-programming-with-ni-labview.html, [Online].

[6] N. Rehman and D. P. Mandic, "Multivariate Empirical Mode Decomposition," *Proc. R. Soc. A*, no. 466, pp. 1291-*1302*, Dec., 2009. Accessed on: Jul, 10, 2020, DOI: 10.1098/rspa.2009.0502, [Online].