La programación funcional y las arquitecturas multicore: estado del arte

Functional programming and multicore architectures – review of prior research

A programação funcional e as arquiteturas multicore – estado da arte

Para citar este artículo / To reference this article / Para citar este artigo: Hoyos Pineda, J. G. y Puertas González, Á. (2015). La programación funcional y las arquitecturas multicore: estado del arte. *Ingenio Magno, 6*(2), 124-136.

Jorge Gabriel Hoyos-Pineda
Facultad de Ingeniería de Sistemas, Grupo de Investigación y Desarrollo de Ingeniería en Nuevas Tecnologías (GIDINT), Universidad Santo Tomás
jorge.hoyos@usantoto.edu.co

Álex Puertas-González
Facultad de Ingeniería de Sistemas, Grupo de Investigación y Desarrollo de Ingeniería en Nuevas Tecnologías (GIDINT), Universidad Santo Tomás
alex.puertas@usantoto.edu.co

Fecha de recepción: 9 de Septiembre de 2015
Fecha de Aprobación: 16 de Diciembre de 2015

Resumen

Este artículo presenta una revisión bibliográfica tendiente a identificar los conceptos fundamentales de la programación funcional, los diferentes enfoques y las aplicaciones que se han dado a este paradigma de programación. En una segunda parte, la revisión se centra en los trabajos orientados a la aplicación y el aprovechamiento de la programación funcional para el procesamiento paralelo sobre arquitecturas *multicore.* Se concluye que gracias a las características especiales que presenta, el paradigma funcional ha venido ganando terreno en diferentes campos de aplicación y paulatinamente se ha venido incorporando en los planes de estudio de programas universitarios en el área de la computación y afines.

Palabras clave: programación funcional, arquitecturas *multicore,* procesamiento paralelo

Abstract

A bibliographical review is presented, aimed at identifying the fundamental concepts of functional programming, the different focuses and applications which have been given to this programming paradigm. In the second part the review centers on works aimed at the application and utilization of functional programming for parallel processing on multicore architectures. Finally it is concluded that thanks to the special characteristics presented, the functional paradigm has been gaining ground in different fields of application, and has gradually been incorporated in plans of study in university programs and the area of computing and related areas.

Keywords: functional programming, multicore architectures, parallel processing

Resumo

Uma revisão da literatura é apresentada com o intuito de identificar os conceitos fundamentais da programação funcional, as diferentes abordagens e as aplicações que tem sido dadas a este paradigma da programação. Em uma segunda parte a revisão se concentra no trabalho orientado para a aplicação e utilização da programação funcional para o processamento paralelo em arquiteturas multicore.

No final, concluiu-se que devido às características especiais que apresenta, o paradigma funcional tem ganhado terreno nos diferentes campos de aplicação e tem sido gradualmente incorporado no currículo dos cursos universitários na área de computação e as áreas relacionadas.

Palavras Chave: programação funcional, arquiteturas multicore, procesamiento paralelo.

1. Introducción

En los últimos años han surgido nuevos lenguajes de programación en respuesta a los nuevos desarrollos tecnológicos, como los procesadores *multicore*, la computación en la nube o *cloud computing*, la computación de alto desempeño, los sistemas distribuidos y las tecnologías asociadas al desarrollo móvil (Ortiz, 2012). En lo que tiene que ver con las

arquitecturas *multicore*, se evidencia la subutilización de este tipo de *hardware*, ya que son muy pocas las aplicaciones capaces de aprovechar esta infraestructura para mejorar su desempeño. Este nuevo escenario propicia el resurgimiento de la programación funcional como el paradigma candidato a sustituir el de orientación a objetos, que ha estado vigente en los últimos quince años.

La programación funcional es un paradigma declarativo. Esta programación se caracteriza por el uso de expresiones y funciones; allí, un programa es una función, que a su vez está compuesta por funciones más simples. Además, una función puede llamar a otra, o bien, el resultado de una función puede ser utilizado como parámetro de otra. Una de las aplicaciones en que se ha advertido una gran potencialidad para la implementación de la programación funcional es la programación paralela a través de varios núcleos, con el propósito de mejorar el rendimiento en el *hardware* moderno. Esto, por supuesto, constituye un gran reto para los desarrolladores, que ahora tienen que pensar con la complejidad adicional de este enfoque de programación.

En este artículo se describen las bases de la programación funcional, los diferentes lenguajes que han surgido alrededor de este paradigma y sus aplicaciones. Se hace especial énfasis en la utilización de la programación funcional en la construcción de aplicaciones que explotan de mejor manera las arquitecturas *multicore* a través del paralelismo.

2. La programación funcional

La programación funcional (PF) es un paradigma de programación declarativo. Se caracteriza por el uso de expresiones y funciones, donde un programa es una función, la cual, a su vez, está compuesta por funciones más simples. Una función puede llamar a otra, o bien, el resultado de una función puede ser utilizado como parámetro de otra (González, 1998).

En un sentido simple, se podría decir que a diferencia de la programación imperativa, la programación funcional no utiliza variables mutables, instrucciones de asignación, ciclos, ni las demás estructuras de control típicas de los lenguajes imperativos. Este tipo de programación permite inferir propiedades generales de la función, para lo cual utiliza principios de inducción matemática y razonamiento algebraico, lo que ofrece beneficios a los programadores para el desarrollo de *software* como la construcción rápida de prototipos, el enfoque modular, la minimización del tiempo de desarrollo, una menor cantidad de código y la facilidad para realizar modificaciones y reutilizar código.

Existen algunas características esenciales, que aunque no son exclusivas de la programación funcional, sí están presentes en los lenguajes funcionales (Glaser y Henderson, 1991):

— *La utilización de funciones de alto orden (high-order functions).* En palabras simples, es una función capaz de aceptar como argumento otras funciones (Perry, 2007) y producir otras más como resultado de su evaluación (Bailes, 2004).

— *Permitir la definición de tipos algebraicos.* Referido a la posibilidad de declarar nuevos tipos de datos (diferentes a los tipos primitivos) de forma recursiva, que permiten manejar estructuras complejas (González, 1998).

— *Manejo automático de la memoria.* La característica de recursividad presente en la definición de tipos algebraicos hace innecesaria la gestión de la memoria dinámica, que es fuente de muchos errores en programación.

— *Permitir el emparejamiento de patrones (pattern matching).* También conocido como *mapeo*. Se usa como un mecanismo que permite controlar el flujo

de ejecución, como si se tratara de la verificación de una serie de condiciones que determinará los cálculos por realizar o las instrucciones que habrán de ejecutarse. En el caso de una función, equivaldría a una especie de segmentación, para responder de forma diferente de acuerdo con un patrón que actúa de forma similar a un parámetro de la función. Aunque el emparejamiento de patrones no es un tema propio de la PF, constantemente se le asocia a ella por cuanto los lenguajes funcionales han hecho uso del concepto desde hace mucho tiempo, a diferencia de otros lenguajes como los imperativos.

— *Inferencia de tipos.* Constituye un punto intermedio entre los dos esquemas típicos utilizados por los lenguajes de programación para el manejo de tipos: el estricto y el no estricto. Aunque se conserva el carácter estricto del sistema de tipos, se dispone de mecanismos para la asignación automática, en tiempo de compilación, de los tipos de argumentos y valores de retorno de las funciones (González, 1998).

— *Evaluación perezosa (lazy evaluation).* Es una característica principal de la PF. Se refiere a que los argumentos de la función son evaluados "a lo sumo una vez". Así se evita el reprocesamiento en las implementaciones de las estrategias típicas de reducción de expresiones usadas por los lenguajes de programación y, por ende, se mejora su eficiencia (Hudak, 1989). La evaluación perezosa permite definir estructuras de datos infinitas, evita realizar cálculos innecesarios y hace posible definir estructuras de control como funciones simples (Hinsen, 2009). En particular, las características de funciones de alto orden y la evaluación perezosa, presentes en los lenguajes funcionales, contribuyen de forma importante a mejorar la modularidad de los programas, lo que a su vez conduce a facilitar el reúso y a disminuir los costos de mantenimiento del *software* (Hughes, 1990).

— *Transparencia referencial.* Dado que la programación funcional implementa funciones puras en su estructuración, es posible obviar componentes de forma referentes a un lenguaje de programación en específico. Al tener esta particularidad, es posible usar expresiones de programación de tipo genérico y transversales a los lenguajes de este tipo. De este modo se genera un comportamiento similar a la aplicación de fórmulas matemáticas dentro de un contexto problemático (Läufer y George, 2009).

— *Recursión.* Dentro de la estructura de la programación funcional, las rutinas que requieran del uso de ciclos son manejadas, exclusivamente, a partir de la aplicación de estructuras recursivas, es decir, se usan funciones que se llaman a sí mismas, teniendo en cuenta la variación de sus argumentos (Hinsen, 2009).

A. El cálculo lambda y la programación

Este cálculo sirve de fundamento a las principales ideas que soportan la programación funcional y ha sido utilizado para realizar análisis matemáticos de los lenguajes de programación (Vichare, 2013). Como lo menciona Machado (2013), esta clase de cálculo está construido a partir del concepto y la aplicación de funciones, desde un punto de vista minimalista y aplicable computacionalmente, dada la similitud entre, por un lado, los elementos de este tipo de cálculo en función de las operaciones booleanas, las expresiones condicionales, las listas y las recursividad, y, por otro, los elementos estructurales de la programación.

El esquema de evaluación de expresiones utilizado por la programación funcional es conocido como *modelo de sustitución*, que esta formalizado a través del cálculo

lambda. Este modelo se traduce en que, al evaluar una expresión, esta puede ser reducida a un valor, lo que hace posible su utilización como parte de otra expresión más compleja, o bien, como parámetro de entrada de una función o valor de su retorno.

El cálculo lambda, según Slonneger (1995), ha hecho aportes importantes a las semánticas formales de los lenguajes de programación.

— La sencillez de su sintaxis y semántica facilita el estudio del significado de los conceptos de los lenguajes de programación.

— Los lenguajes de programación funcionales pueden ser considerados variaciones sintácticas del cálculo lambda, ya que la semántica e implementación de los primeros puede ser analizada en el contexto del segundo.

— Las funciones de alto orden propias del cálculo lambda son usadas para expresar las semánticas denotacionales.

Varios lenguajes de programación, a pesar de no definirse como funcionales, han incorporado las llamadas expresiones lambda, como Perl, Pthon, Ruby, C++ (Abel, 2011) y Java (Wampler, 2011a; Subramaniam, 2014).

B. Lenguajes funcionales

Son un tipo de lenguaje declarativo (expresan el qué, más que el cómo) cuyo modelo computacional es la función. Los lenguajes funcionales han evolucionado desde el cálculo lambda, hasta lenguajes funcionales modernos como Haskell y otros que incluyen características como funciones de orden superior, evaluación perezosa, correspondencia de ecuaciones y patrones, al igual que diferentes mecanismos de abstracción de datos. En gran medida, estas características son mantenidas con el estilo declarativo puro utilizado en este tipo de programación (Hudak, 1989).

Dentro de los distintos lenguajes funcionales puros están Haskell y Miranda, Pure Lisp, XSLT, XPath, XQuery, FP. Como lenguajes de programación funcional paralela se encuentran GpH como lenguaje implícito y Eden como lenguaje explícito, ambos derivados de Haskell (Vara, 2008). Se encuentra también el lenguaje Erlang como lenguaje funcional concurrente, y lenguajes funcionales híbridos como Scala, Lisp, Scheme, Racket, Clojure, Ocaml, SML, F# (Hansen, 2013). También se pueden encontrar APIs de programción funcional aplicables a soluciones existentes en el mercado Es el caso de Bricklayer (Winter, 2014), que funciona, basada en SML, sobre LEGO Digital Designer.

El lenguaje Scala, uno de los más utilizados en la actualidad, es un lenguaje multiparadigma que integra la programación funcional con la programación imperativa orientada a objetos. Scala compila en Bytecode de Java y promete hacer más fácil el desarrollo de *software* paralelo. Actualmente, empresas como Twitter hacen uso de este lenguaje (Ren y Nogiec, 2008).

También en el campo de las aplicaciones web se ha querido aprovechar las ventajas de la programación funcional. Thiry (2013) propone la forma en que el paradigma funcional puede ser usado para generar sitios web a partir de modelos de datos, o bien, para embeber lenguaje de consulta como parte de un motor de búsqueda especializada.

Por su parte, Vinoski (2012) describe un *framework* para aplicaciones web: Nitrogen Erlang, que adiciona características y mejora las ya existentes en el lenguaje Erlang orientadas al desarrollo web, que facilitan su utilización y aprovechamiento por parte de nuevos usuarios del paradigma.

De forma similar, Wampler (2011b) menciona diferentes alternativas de *frameworks* para aplicaciones web basadas en el lenguaje Scala. Allí se destaca Lift. También, en el caso del lenguaje Haskell, existen *frameworks* para aplicaciones web, como Snap, que es descrito en Collins y Doug (2011).

C. Enfoques

Meijer (2014) presenta una discusión sobre las tendencias a asignar diferentes grados de "pureza" a los desarrollos realizados utilizando programación funcional. En el trabajo mencionado se plantea que las soluciones obtenidas no son las mejores ni las más seguras, por cuanto el solo hecho de utilizar variables inmutables no garantiza por sí mismo la eliminación de los llamados *efectos laterales*, debido a que las aplicaciones deben utilizar operaciones comunes como las excepciones, los hilos de proceso y las operaciones de entrada y salida, que pueden generar más problemas que una variable mutable.

Desde este punto de vista, existen dos tipos de programación funcional. Por un lado, la pura, que es la aplicación de los preceptos del cálculo lambda para la resolución de problemas informáticos, teniendo en cuenta que los resultados de las expresiones usadas solo dependen de estas expresiones y su contenido, donde se observan lenguajes de programación como Haskell. Por otro lado, la impura, que permite la utilización de elementos imperativos, como conceptos de asignación a la par del cálculo lambda, lo cual denota la aparición de efectos sobre las rutinas de programación implementadas en un desarrollo. Estos aspectos son presentados en Scheme o ML estándar, que son lenguajes impuros funcionales (Wadler, 2001).

Por otra parte, la implementación de lenguajes funcionales en aplicaciones comerciales, teniendo en cuenta lo que hace Microsoft con F#, deriva en una evolución en la masificación de este tipo de programación. Además, la publicación, por parte de O'Reilly, del libro de libre distribución: *Real world Haskell* (O'Sullivan, Steward y Goerzen, 2008), representa un avance importante en la transmisión de los conceptos, el funcionamiento y el desarrollo de la programación hacia la comunidad de TI.

También es importante anotar que desde Henderson (1986) se mencionaba la relevancia de la programación funcional en términos de costos para las empresas desarrolladoras, puesto que, dado su enfoque matemático, el diseño y la corrección de aplicaciones es mucho más sencillo que si se usara el modelo imperativo.

D. Aplicaciones

Los lenguajes funcionales se caracterizan porque todo el procesamiento consiste en una evaluación de expresiones. Aunque estos lenguajes han sido utilizados más en ámbitos académicos que en comerciales o industriales, no se puede negar el impacto que ha tenido su desarrollo tanto en la teoría como en la práctica de los lenguajes de programación (Hudak, 1989).

Las áreas de aplicación de la programación funcional son muy variadas:

— *Big data (Tran et al., 2012).* En este trabajo se presenta una implementación de un *framework* de procesamiento paralelo distribuido que utiliza grafos acíclicos dirigidos, para la definición de tareas. El *software* está basado en el modelo Data Flow Graphs (DFG), en el cual cada nodo representa un trabajo o *job* (programa o rutina que debe ser ejecutado en paralelo). En este caso, el paradigma de programación funcional hace posible y facilita la definición de operadores genéricos y reusables.

— *Lenguajes de dominio específico.* Gibbons (2015) resalta como fortaleza de la programación funcional

el proporcionar herramientas poderosas para la definición, implementación y optimización de nuevos lenguajes, lo que la hace muy atractiva para la creación de lenguajes de dominio específico. Por su parte, Damjanovic y Dragan (2010) describen la forma en que la programación funcional puede ser usada en conjunto con un lenguaje de dominio específico y las lógicas descriptivas, para soportar la interacción con ataques de *software* y detección de vulnerabilidades de la capa de seguridad de un protocolo de aplicación *wireless.* Muranushi (2012) propone un lenguaje de dominio específico para el ajuste automático de solucionadores explícitos de ecuaciones diferenciales parciales utilizando GPUs y procesadores *multicore.*

— *Simulación de circuitos y dispositivos electrónicos (Cerný y Dobe, 2014).* En este trabajo se describe la forma de definir en un lenguaje funcional dos dispositivos básicos (diodo simple y fuente de voltaje) utilizados en un *software* de simulación de circuitos electrónicos.

— *En grandes empresas como Facebook, Bloomberg y Citrix.* Allí se ha utilizado el lenguaje OCAML en la construcción de herramientas de desarrollo (OCaml.org, 2015). Tal es el caso de Facebook, con Hack como compilador, Flow para el chequeo de tipos estáticos en Javacript y Pfff como un conjunto de herramientas para el análisis, la visualización y la preservación de estilo de código. En el caso de Blomberg, es usado en la aplicación avanzada de administración del riesgo financiero derivado. Por su parte, Citrix incorpora OCaml en su sistema de virtualización de servidores XenServer.

— *En empresas de telecomunicaciones.* Verizon, desde comienzos del 2014, adquirió un proyecto que había sido iniciado por Intel para desarrollar una nueva plataforma de servicio de televisión de última generación, con el uso de lenguajes de programación funcionales, en especial Scala (Commercial Users of Functional Programming [CUFP], 2014a). En esta misma área, la Swedish Space Corporation (SSC) ha utilizado el lenguaje Haskell en el desarrollo de diferentes herramientas para el entorno de operaciones de un satélite de telecomunicaciones (CUFP, 2014c).

— *En la industria aeroespacial.* Varias empresas como Siemens CVC Austria y DLR (Deutsche Zentrum für Luft- und Raumfahrt) han utilizado el lenguaje Haskell para desarrollar herramientas que evalúen el desarrollo, la mantenibilidad y el número de errores residuales del código usado en los sistemas de control de misiones espaciales (CUFP, 2014b).

— *En la enseñanza de las matemáticas.* Choppella, Manjula y Viswanth (2012) desarrollan la idea de aplicar la programación funcional para la implementación de algoritmos de álgebra en secundaria. Se basan en las ventajas de uso de funciones de orden superior y en la facilidad en el aprendizaje de la codificación, dado que no hay que enseñar la estructura del lenguaje de programación, sino el proceso algebraico específico. De la misma forma, Karczmarczuk (1999) ya mencionaba el uso de programación funcional en el área de la física, más específicamente en el campo de la programación de tipo científico, dado que la limpieza del código hace que se resuelva la dificultad en la escritura de programas, presente en las estructuras imperativas. Además, el aprovechamiento de la funcionalidad de los lenguajes funcionales como Haskell permite un desarrollo más robusto de programas de tipo experimental.

— *En aplicaciones en bioinformática.* En concreto, ello ha sido posible desde la simulación de procesos

biológicos (resonancias magnéticas nucleares) para crear repositorios de funciones que usan este tipo de programación e involucren a la comunidad científica en el uso de este paradigma (Fenwick, 2012).

— *En la implementación de Easel.* Finalmente, Nelson, Archer y Rushton (2014) encuentran la implementación de Easel, un *framework* basado en SequenceL, que funciona como motor para el desarrollo de videojuegos en tiempo real desde funciones puras.

3. Programación funcional y arquitecturas *multicore*

En la producción continua de procesadores *multicore* se ha creado la necesidad de mejorar la programación paralela. Los equipos de cómputo son máquinas de 4 núcleos y 8 hilos; los servidores tienen 12, 32 y más; los dispositivos integrados y teléfonos móviles también se están convirtiendo en máquinas paralelas. Por ello, los desarrolladores ahora tienen que pensar con la complejidad adicional de la programación paralela para mejorar el rendimiento en el *hardware* moderno (Pankratius, Schmidt y Garreton, 2012). Por lo tanto, la exploración de la programación paralela a través de varios núcleos debe ser abordada (Ren y Nogoiec, 2008).

En este punto es necesario mencionar que cuando se habla de *sistemas multicore,* se hace referencia a la integración de varios *cores* en un solo procesador (usualmente entre dos y diez), y que existe un segundo enfoque conocido como *manycore*, donde se usa una gran cantidad de *cores* (incluso decenas o centenas) alojados en varios procesadores (Rivoire, 2010).

De forma similar se hace una distinción en la forma como los sistemas paralelos utilizan la memoria. En este sentido, se habla de *memoria compartida* y *memoria distribuida.* En el primer caso, los diferentes procesadores comparten un mismo espacio de memoria; en el segundo, cada procesador dispone de su propio espacio de memoria (Díaz, Muñoz-Caro y Niño, 2012).

Se han explorado diferentes alternativas para el procesamiento paralelo utilizando programación funcional. Varios de los lenguajes funcionales incluyen constructos paralelos implícitos y primitivas paralelas; es el caso de NESL (Blelloch, 1996), ML (Bergstrom *et al.*, 2010), BSML (Bousdira *et al.*, 2010) y Haskell (Chakravarty *et al.*, 2007).

La naturaleza declarativa de los lenguajes funcionales los hacen ideales para la programación paralela (Phan y Hansen, 2014). Existen varios ejemplos de aplicación de paralelismo con PF en temas como búsqueda paralela (Reck y Fischer, 2009); también es el caso de Dung ty Hansen (2011), que estudian la paralelización de programas funcionales a partir de la implementación del algoritmo de Cooper y el test Omega, teniendo en cuenta la potencia del uso de la tecnología *multicore.*

Por otra parte se plantean también soluciones híbridas, es decir, que contengan elementos de programación imperativa y funcional. Por ejemplo, en Veldema y Philippsen (2010) se aprecia un desarrollo de este estilo: se usa programación funcional para optimizar la operación *multicore*, teniendo en cuenta los problemas de paralelización que presenta la programación orientada a objetos, en función del *lack* entre procesos y la falta de sincronización.

También es posible observar el desarrollo de ambientes de ejecución que involucran la utilización de tecnología *multicore* a través de programación funcional. Tal es el caso de MultiMLton (Sivaramakrishnan, Ziarek y Jagannathan, 2014), una optimización del compilador MIton, basado a su vez en SML. Análogamente,

en Nelson y Ruschton (2013) puede observarse la implementación de programas en paralelo a través de lenguajes funcionales, en este caso usando SequenceL y particularmente analizando la velocidad de ejecución de estos.

En el caso del lenguaje Haskell, se conocen por lo menos dos extensiones para procesamiento paralelo, que utilizan diferentes enfoques para aprovechar una arquitectura *multicore.* Berthold *et al.* (2009) realizan una comparación de dos implementaciones diferentes: la primera, conocida como GpH, usa un mecanismo de memoria *heap* compartida físicamente, mientras la segunda, conocida como Eden, está diseñada para el uso en máquinas paralelas de memoria distribuida, donde existe una memoria *heap* independiente por cada núcleo y la comunicación entre núcleos se realiza mediante el intercambio de mensajes.

Para el caso de Jones *et al.* (2008), se enuncia el aprovechamiento de las ventajas de Haskell como lenguaje aplicable a soluciones *multicore*. En este caso, se tiene en cuenta el uso del compilador GHC para implementar la transformación por vectorización, mediante la cual se cambia el paralelismo de datos anidado en un paralelismo de datos estándar.

Por otro lado, Marlow (2012) menciona dos características fundamentales para la implementación de la programación paralela en Haskell: por una parte, la granuralidad, que busca dar un tamaño lógico a las subdivisiones de código, para así aprovechar las capacidades de los procesadores; por otra, las dependencias de datos en la secuencialización de los programas en tiempo de ejecución, bien sea de forma explícita o implícita.

Aswad, Trinder y Loidl (2012) proponen cuatro constructos orientados a la arquitectura, como una extensión orientada al procesamiento paralelo y creada para el lenguaje Haskell. Esta aprovecha la información acerca del tamaño de las tareas para controlar los datos de forma local y distribuir el trabajo.

En Al Zain *et al.* (2009) se hace una implementación de programación *multicore* usando Haskell, como "lenguaje coordinador" de código escrito en otros lenguajes de programación, con el fin de crear secuencias computacionales en los núcleos físicos de un procesador. Análogamente, en Voellmy, Wang y Hudak (2014) se puede encontrar una implementación de hilos desarrollados en Haskell, en función de optimizar el desarrollo de aplicaciones de comunicaciones, en un ambiente *multicore.* Se genera así un administrador de entrada/salida denominado *Mio*, que optimiza el rendimiento de un servidor web, dada una red de datos de software definido, o Software Defined Network (SDN).

4. Conclusiones

Como conclusión de este trabajo cabe mencionar, en primera instancia, la importancia de la programación funcional como paradigma de implementación en el campo del desarrollo de sistemas, dada la ausencia, dentro de este tipo de aplicación, de una escritura basada en declaración de variables, ciclos y sentencias de control, ya que su estructura depende casi que en exclusivo del modelamiento matemático derivado del cálculo lambda, dado su enfoque funcional y minimalista.

Esto le da a este tipo de programación características específicas y particulares; así, por ejemplo, es posible definir tanto tipos de datos algebraicos diferentes a los tipos primitivos como funciones de alto nivel, que son capaces de aceptar otras funciones como argumento, teniendo en cuenta para esto la implementación de métricas de recursividad.

Otro aspecto importante es la posibilidad de definir, para este tipo de programación, revisiones de código más eficientes que las usadas para el desarrollo imperativo. A

este tipo de implementación se le denomina *evaluación perezosa*, y es posible usarla por el carácter reduccionista del código funcional y la transversalidad de este en los diferentes lenguajes de programación que han sido creados en este enfoque.

También se han podido mostrar diferentes aplicaciones prácticas de la programación funcional: se demuestra su utilidad en escenarios de producción como *big data*, electrónica y redes de comunicación, aunque es evidente su aplicabilidad e influencia dentro del ámbito educativo, dada su robustez en procesos de simulación y en algoritmos iterativos.

Por otra parte, se encuentra relación del desarrollo funcional en ambientes *multicore*, dado que las características antes mencionadas dejan ver sus ventajas en esta clase de arquitecturas. Cabe anotar que lenguajes de desarrollo funcionalmente puros, como Haskell, resultan particularmente útiles para procesos de programación paralela. De hecho se pueden encontrar experiencias en las que los lenguajes funcionales se usan como plataforma de operación para aplicaciones que pueden estar implementadas en lenguajes de programación imperativos o que funcionan de manera híbrida con elementos de desarrollo orientado a objetos, con el fin de reducir el *lack* entre procesos y optimizar las dependencias de los datos. Ello deja ver la versatilidad de los lenguajes funcionales en este tipo de escenarios y su aplicabilidad en función de resolver los problemas clásicos del procesamiento en paralelo.

Referencias

Abel, S. (2011). Nested Lambda Expressions with Let Expressions in C++ Template Metaprograms. *Electronic Notes in Theoretical Computer Science*, *279*(3), 27-41. doi: 10.1016/j.entcs.2011.11.036

Al Zain, A., Trinder, P., Aswad, M., Michaelson, G., Hammond, K. y Berthold, J. (2009). Low-pain, high-gain multicore programming in Haskell. Paper presented at the DAMP'09, Savannah, Georgia, USA.

Aswad, M., Thinder, P. W. y Loidl, H. (2012). Architecture aware parallel programming in glasgow parallel Haskell (GPH). *Procedia Computer Science*, *9*, 1807-1816. Doi: 10.1016/j.procs.2012.04.199

Bailes, P. y Kemp, C. (2004). Obstacles to a totally functional programming style. Paper presented at the 2004 Australian Software Engineering Conference.

Bergstrom, L., Rainey, M., Reppy, J., Shaw,A. y Fluet, M. (2010). Lazy tree splitting. Paper presented at the International Conference on Functional Programming.

Berthold, J., Marlow, S., Hammond, K. y Al Zain, A. (2009). Comparing and optimising parallel Haskell implementations for multicore machines. Paper presented at the 2009 International Conference on Parallel Processing Workshops.

Blelloch, G. E. (1996). Programming parallel algorithms. *Communications of the ACM*, *39*(3), 13. Doi: 10.1145/227234.227246

Bousdira, W., Gava, F., Gesbert, L., Loulergue, F. y Petiot, G. (2010). Functional parallel programming with revised bulk synchronous parallel ML. Paper presented at the Networking and Computing (ICNC), 2010 First International Conference on.

Cerný, D. y Josef, D. (2014). Functional programming languages in computer simulation of electronics circuits. 2014 *International Conference on Computational Science and Computational Intelligence*, 229-235. Doi: 10.1109/CSCI.2014.46

Collins, G. y Doug, B. (2011). The snap framework: a web toolkit for Haskell. Recuperado de http://static.

googleusercontent.com/media/research.google.com/en//pubs/archive/37267.pdf

Commercial Users of Functional Programming (CUFP) (2014a). Functional Programming at Verizon OnCue. Recuperado de http://cufp.org./2014/timothy-perrett-functional-programming-at-verizon-oncue.html

Commercial Users of Functional Programming (CUFP) (2014b). Haskell in the Misson Control Domain. Recuperado de http://cufp.org./2014/michael-oswald-haskell-in-the-misson-control-domain.html

Commercial Users of Functional Programming (CUFP) (2014c). Haskell tools for satellite operations. Recuperado de http://cufp.org./2014/bjrn-buckwalter-haskell-tools-for-satellite-operations.html

Chakravarty, M., Leshchinskiy. R., Peyton, S., Keller, G. y Marlow, S. (2007). Data parallel Haskell: a status report. Paper presented at the Workshop on Declarative Aspects of Multicore Programming.

Choppella, V., Kumar, H., Manjula, P. y Viswanath, K. (2012). From high-school algebra to computing through functional programming. Paper presented at the Technology for Education (T4E), 2012 IEEE Fourth International Conference.

Damjanovic, V. y Dragan, D. (2010). Functional programming way to interact with software attacks and vulnerabilities. *Third International Conference of Software Testing, Verification, and Validation Workshops*, 388-393. doi: 10.1109/ICSTW.2010.53

Diaz, J., Muñoz-Caro, C. y Niño, A. (2012). A survey of parallel programming models and tools in the multi and many-core era. *IEEE Transactions on Parallel and Distributed Systems*, *23*(8).

Dung, P. y Hansen, M. (2011). From functional programming to multicore parallelism: A case study based on Presburger Arithmetic. Paper presented at the Nordic Workshop on Programming Theory.

Fenwick, M., S,esanker, C., Schiller, M. R., Ellis, H. J., Vyas, J. y Grik, M. R. (2012). An open-source sandbox for increasing the accessibility of functional programming to the bioinformatics and scientific communities. Paper presented at the Ninth International Conference on Information Technology- New Generations. IEEE computer Society.

Gibbons, J. (2015). Functional programming for domain-specific languages. Recuperado de http://www.cs.ox.ac.uk/jeremy.gibbons/publications/fp4dsls.pdf

Glaser, H. y Henderson, P. (1991) *Functional programming. Software Engineer's Reference Handbook.* Butterworth, 35/1-35/6.

González, F. (1998). Programación funcional: conceptos y perspectivas. *Ingeniería e Investigación*, *40*, 65-71.

Hansen, M. y Rischel, H. (2013). *Functional programming using F#.* Cambridge: Cambridge University Press.

Henderson, P. (1986). Functional programming, formal specification, and rapid prototyping. *IEEE Transactions on Software Engineering*, *SE-I2*(2), 10.

Hinsen, K. (2009). The promises of Functional Programming. *Computing in Science & Engineering*, 86-90. Doi: 10.1109/MCSE.2009.129

Hudak, P. (1989). Conception, evolution, and application of functional programming languages. ACM Computing Surveys, *21*(3), 359-411.

Hughes, J. (1990). Why functional programming matters.

Recuperado de https://www.cs.kent.ac.uk/people/staff/dat/miranda/whyfp90.pdf

Jones, S., Leshchinskiy, R., Keller, G. y Chakravarty, M. M. (2008). Harnessing the multicores: nested data parallelism in Haskell. Paper presented at the FSTTCS.

Karczmarczuk, L. (1999). Scientific computation and functional programming. *Computing in Science and Engineering Magazine*, *9*.

Läufer, K. y George, K. (2009). The promises of typed, pure, and lazy functional programming: Part II. Recuperado de http://ecommons.luc.edu/cgi/viewcontent.cgi?article=1022&context=cs_facpubs

Machado, R. (2013). An introduction to lambda calculus and functional programming. *2013 2nd Workshop-School on Theoretical Computer Science*, 26-33. Doi: 10.1109/WEIT.2013.40

Marlow, S. (2012). *Parallel and concurrent programming in Haskell.* Cambridge: Microsoft Research Ltd.

Meijer, E. (2014). The curse of the excluded middle. *Communications of de ACM*, *12*(4), 50-55.

Muranushi, T. (2012). Paraiso: an automated tuning framework for explicit solvers of partial differential equations. *Computational Science & Discovery*, *5*, 1-40. Doi: 10.1088/1749-4699/5/1/015003

Nelson, B., Archer, J. y Rushton, N. (2014). Easel: purely functional game programming. Paper presented at the SERP 2014.

Nelson, B. y Rushton, N. (2013). Fully automatic parallel programming. Paper presented at the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA).

OCaml.org. (2015). Companies using OCaml. Recuperado de http://ocaml.org/learn/companies.html

Ortiz, S. (2012). Computing trends lead to new programming languages. *Computer, 45*(7).

Pankratius, V., Schmidt, F. y Garreton, G. (2012). Combining functional and imperative programming for multicore software: An empirical study evaluating Scala and Java. Paper presented at the 34th International Conference on Software Engineering (ICSE).

Perry, A. (2007). *System level design with Rosetta.* San Francisco: Morgan Kaufmann.

Phan, A.-D. y Hansen, M. (2014). An approach to multicore parallelism using functional programming: A case study based on Presburger Arithmetic. *Journal of Logical and Algebraic Methods in Programming*, *84*(1), 1-17. Doi: 10.1016/j.jlamp.2014.07.002

Reck, F. y Fischer, S. (2009). Towards a parallel search for solutions of non-deterministic computations. Paper presented at the 39th Annual Meeting of the German Informatics Society.

Ren, S. y Nogiec, J. (2008). Developing concurrent applications on emerging multicore platforms. Paper presented at the International Conference on Complex, Intelligent and Software Intensive Systems (CISIS).

Rivoire, S. (2010). A breadth-first course in multicore and manycore programming. Proceedings of the 41st ACM technical symposium on Computer Science education, 214-218. Doi: 10.1145/1734263.1734339

Sivaramakrishnan, K. C. Z., L ; Jagannathan, S. (2014). MultiMLton: Amulticore-aware runtime for standard ML. *Journal of Functional Programming, 24*(6), 1-62. Doi: 10.1017/S0956796814000161

Slonneger, K. (1995). *Formal syntax and semantics of programming languages: a loboratory-based approach.* Reading, Massachusetts: Addison-Wesley.

Subramaniam, V. (2014). *Functional Programming in Java - Harnessing the Power of Java 8 Lambda Expressions.* Dallas, Texas: The Pragmatic Programmers.

Thiry, L. y Hassenforder, M. (2013). Surfing with fun. Paper presented at the 2013 International Symposium of Theoretical Aspects of Software Engineering.

Tran, N.-L., Skhiri, S., Lesuisse, A. y Zimányi, E. (2012). AROM: processing big data with data flow graphs and functional pogramming. IEEE 4th International Conference on Cloud Computing Technology and Science, 875-882. Doi: 10.1109/CloudCom.2012.6427487

Vara, A. (2008). *Formalizando el proceso de depuración en programación funcional paralela y perezosa* (tesis de doctorado). Madrid: Universidad Complutense de Madrid.

Veldema, R. y Philippsen, M. (2010). Safe and familiar multi-core programming by means of a hybrid functional and imperative language. Recuperado de http://link.springer.com/chapter/10.1007%2F978-3-642-13374-9_11#page-2

Vichare, A. (2013). Algorithms, the lambda calculus and programming - An intuitive approach. *Resonance*, *18*(4), 345-367.

Vinoski, S. (2012). The Nitrogen Erlang Web Framework. *The Functional Web, 4.*

Voellmy, A., Wang, J., Hudak, P. y Yamamoto, K. (2014). Mio: a high-performance multicore IO manager for GHC. Recuperado de http://haskell.cs.yale.edu/wp-content/uploads/2013/08/hask035-voellmy.pdf

Wadler, P. (2001). *Monads for functional programming. Computing science*. Glasgow: Universidad de Glasgow.

Wampler, D. (2011a). *Functional programming for Java developers*. Estados Unidos: O'Reilly.

Wampler, D. (2011b). Scala web frameworks: looking beyond lift. *The Functional Web*, *8*.

Winter, V. (2014). Bricklayer: an authentic introduction to the functional programming language SML. Paper presented at the TFPIE 2014.