Simulación paralela multidispositivo de secuencias de imagen por resonancia magnética

I. Fernández-Arias[1], P. Villacorta-Aylagas[1], C. Castillo-Passi[2], P. Irarrazaval[2], F. Simmross-Wattenberg[1], M. Rodríguez-Cayetano[1], C. Alberola-López[1]

[1] Laboratorio de Procesado de Imagen. ETSI Telecomunicación. Universidad de Valladolid.
[2] Instituto de Ingeniería Biológica y Médica. Pontificia Universidad Católica de Chile.

Abstract

La simulación del fenómeno de resonancia magnética (RM) es de gran interés tanto en la investigación científica como en la formación de personal técnico. En la literatura hay documentados varios simuladores de RM, pero muchos de ellos no son suficientemente realistas, mientras que otros solamente funcionan en CPUs o en GPUs de un único fabricante. Uno de estos simuladores, KomaMRI.jl, simula de forma fidedigna el fenómeno de RM y funciona en CPU y GPU (estas últimas de un único fabricante), pero no se ha diseñado con especial énfasis en la rapidez de la simulación. La conjunción de velocidad y realismo es crucial para simuladores interactivos y para investigación en secuencias de RM. En este trabajo se presenta un simulador de RM que funciona sobre dispositivos de cualquier tipo y fabricante, y cuyo rendimiento supera al mencionado simulador KomaMRI.jl, a la vez que mantiene su grado de realismo.

1. Introducción

La imagen de resonancia magnética (MRI) es una de las modalidades de imagen médica más potentes entre las disponibles hoy en día. Su alto contraste en los llamados "tejidos blandos" y su gran versatilidad para la obtención de imagen anatómica y funcional la convierten en la técnica preferida en muchos ámbitos clínicos.

Desde una perspectiva educativa, en cambio, manejar un escáner de MRI no es tarea fácil, puesto que las adquisiciones dependen fuertemente de un gran número de parámetros que han de ser calibrados oportunamente para obtener una calidad de la imagen óptima y, en especial, para evitar la aparición de artefactos. Los parámetros a calibrar dependen de las secuencias de pulsos magnéticos que se programan en el escáner, existiendo una amplia variedad de secuencias y siendo, en la actualidad, un campo abierto de investigación que requiere de un uso intensivo de escáneres reales. Sin embargo, las demandas del ámbito clínico y los altos costes suponen que los escáneres sean utilizados constantemente, limitando las posibilidades de programar sesiones de aprendizaje para los técnicos en formación o para la investigación de nuevas secuencias de pulsos. Consecuentemente, el desarrollo de simuladores *in silico* constituye una alternativa realista. En este artículo nos centraremos en la paralelización de un simulador ya descrito, con el objetivo de maximizar sus prestaciones y posibilitar su uso universal, es decir, en cualquier dispositivo de cualquier fabricante.

2. Estado del Arte

La complejidad de la MRI ha hecho que la presencia de la simulación en este campo haya resultado indispensable desde sus primeros días [1]. Los objetivos perseguidos en el desarrollo de los simuladores de MRI difieren, pudiendo encontrar simuladores con fines educacionales o de investigación en materia de desarrollo de nuevas secuencias de pulsos. Centrando el análisis en los simuladores con fines no educativos y, por tanto, con motores de simulación basados en el principio físico subyacente, se pueden destacar los siguientes [2]: JEMRIS [3], que hace uso de las ecuaciones de Bloch y se centra en el desarrollo de secuencias; MRILab [4], que simula la evolución del vector de magnetización a través de la ecuación de Bloch y que fue creado para el desarrollo de nuevas técnicas de MRI; SpinBench [5], que permite simular secuencias de pulsos diseñadas a través de su interfaz gráfica, pero que no genera imágenes MRI; PSUdoMRI [6], que emplea la simulación realista de los campos magnéticos para la obtención de la señal y el ruido de la resonancia magnética, y POSSUM [7], diseñado para la investigación de artefactos que pueden aparecer en las imágenes MRI.

De forma adicional a los simuladores mencionados, destaca la existencia del simulador KomaMRI.jl [8], desarrollado en la Pontificia Universidad Católica de Chile, en el que se ha empleado el lenguaje de programación Julia [9] y que cuenta con un motor de simulación realista. Este simulador ofrece resultados muy competitivos con respecto a los simuladores que se acaban de mencionar, y, actualmente, es capaz de realizar operaciones paralelas. Sin embargo, dispone únicamente de soporte para dispositivos GPU de un único fabricante.

En el campo de la educación, se han descrito simuladores que emulan el proceso llevado a cabo por un técnico en su labor diaria (ver [10] y sus referencias) y se ha descrito también la inclusión de un tutor inteligente [11]. Los si-

ISBN: 978-84-09-45972-8, pags. 85- 88

muladores educativos, sin embargo, suelen poner el énfasis en la interactividad, por lo que los motores de simulación suelen ser rápidos y simples, lo cual impide desarrollar con ellos nuevas secuencias de pulsos y, por lo tanto, limita su utilidad como herramienta de investigación.

Recientemente se ha desarrollado el *framework* OpenCLIPER [12], que facilita el desarrollo de aplicaciones en el lenguaje OpenCL [13]. El *framework* está diseñado como un conjunto de clases de C++ centradas en proporcionar al programador diversas funcionalidades como la gestión del dispositivo de computación, la carga y guardado de datos y la gestión de los algoritmos. El hecho de emplear OpenCL hace que el código desarrollado cumpla el paradigma WORA (*write once, run anywhere*), de forma que el código es multidispositivo (CPU, GPU, FPGA, etc.).

En este artículo describimos la metodología llevada a cabo para escribir el núcleo del simulador [8] en OpenCL, haciendo uso de OpenCLIPER, de forma que las ventajas de dicho simulador sean reforzadas con la potencia de OpenCLIPER y, así, los simuladores basados en principios físicos puedan ser empleados indistintamente, tanto con finalidades docentes como con finalidades investigadoras, y sobre cualquier dispositivo, con independencia del fabricante de que se trate. Para ello, describiremos los fundamentos del simulador (sección 3); a continuación describiremos cómo se ha llevado a cabo su desarrollo en OpenCL/OpenCLIPER (sección 4), y, finalmente, llevaremos a cabo un análisis comparativo de prestaciones de ambos simuladores sobre un fantoma que sirve como banco de pruebas (sección 5).

3. Diseño del Simulador

El simulador descrito en [8] hace uso de los siguientes conceptos:

- `Grad` es una estructura formada por una amplitud (`A`) y una duración (`T`), que permite, a partir de la concatenación de bloques, construir las formas de ondas de los gradientes.
- `RF` es una estructura formada por la amplitud (`A`) del pulso de RF ($B_{1,x} + jB_{1,y}$) y una duración (`T`).
- `ADC` es una estructura formada por un número de muestras (`N`) y una duración (`T`), que permite determinar los instantes temporales de lectura de la señal de MRI.
- `Sequence` es una estructura formada por las formas de onda de los gradientes, los pulsos de RF y los tiempos de adquisición de datos. Por tanto, integra tres *arrays* de bloques `Grad`, `RF` y `ADC`.
- `Phantom` representa el objeto virtual sobre el que se aplica la simulación del fenómeno MRI. Se trata de una estructura que incluye las posiciones espaciales iniciales de los *spins* (x, y, z), la densidad de los protones ρ, las constantes de tiempo $T1$ y $T2$, entre otros, mayoritariamente definidos como vectores.
- `Mag` corresponde con el vector de magnetización y, como tal, es una estructura formada por su componente transversal (M_{xy}) y longitudinal (M_z).
- `Spinor` es una estructura empleada durante la simulación para aplicar una rotación sobre el vector de magnetización y evitar el uso de matrices para tal fin.

Estos conceptos son los elementos que se emplean para resolver las ecuaciones de Bloch mediante una aproximación por eventos discretos. No entraremos en los detalles en este artículo, sino en cómo se ha llevado a cabo esta operativa en OpenCL, haciendo uso del *framework* OpenCLIPER.

4. Desarrollo del Simulador en OpenCL/OpenCLIPER

El simulador KomaMRI.jl [8] está implementado en Julia, un lenguaje de alto nivel que abstrae la disposición de los datos en memoria, lo cual permite desarrollar código rápidamente, pero puede influir negativamente en su rendimiento. Nuestro simulador, en cambio, se ha diseñado con el objetivo de alcanzar el máximo rendimiento posible, minimizando las transferencias *host*-dispositivo y los movimientos de datos durante las operaciones, y maximizando la cohesión de los datos para favorecer el uso eficiente de las cachés. El simulador se incorporará al *framework* OpenCLIPER y se publicará con licencia GNU, igual que el resto del *framework*.

En ese sentido, el simulador desarrollado va a aprovechar las facilidades ofrecidas por el *framework* OpenCLIPER en cuanto a estructuras de datos y gestión de algoritmos, definiendo, a partir de las clases predefinidas por el mismo [12], nuevas clases destinadas a estructuras de datos y al proceso de simulación, las cuales aparecen representadas en la *Figura 1*.

Las principales clases son:

- `Grad`, `RF` y `DAC` son clases que se definen de forma inmediata a partir de sus correspondientes estructuras en el simulador KomaMRI.jl (`Grad`, `RF` y `ADC`, respectivamente).
- `Sequence` es la clase correspondiente a las secuencias del simulador. Un objeto `Sequence` se constituye a partir de tres punteros a vectores de objetos `Grad`, cada uno en una dirección (x, y, z), un puntero a un vector de objetos `RF` y un puntero a un vector de objetos `DAC`. Una representación aproximada de un objeto de la clase `Sequence` correspondería con la matriz mostrada en la *Figura 2*.

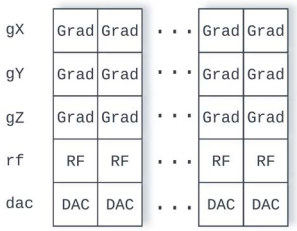

Figura 2. *Representación objeto* `Sequence`.

- `Phantom` es la clase que representa el objeto escaneado cuya imagen se desea reconstruir. Es una clase derivada de la clase `Data` de OpenCLIPER y, como tal, consta de un único `NDArray` con 11 filas, que co-

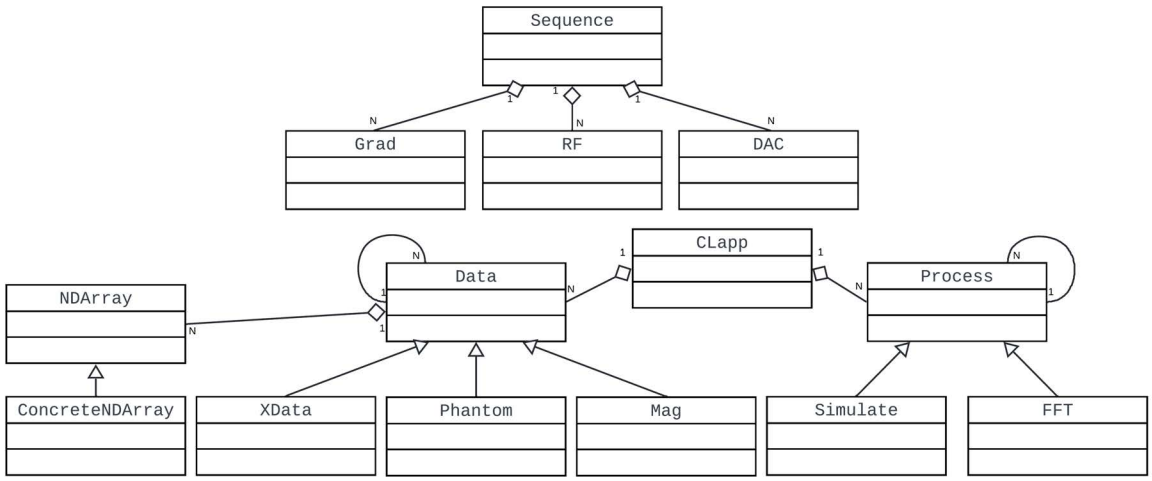

Figura 1. *Diagrama de clases del simulador en OpenCLIPER.*

rresponden a cada una de las componentes del objeto `Phantom`, tal como se muestra en la *Figura 3*.

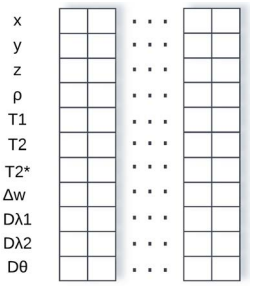

Figura 3. *Representación objeto* `Phantom`.

- `Mag` es la clase que corresponde al vector de magnetización. Deriva también de la clase `Data` y consta de un único `NDArray` de 2 filas correspondientes a la componente transversal (`Mxy`) y longitudinal (`Mz`) del vector de magnetización.

En cuanto a la simulación, se define la clase `Simulate`, que deriva de la clase abstracta `Process` de OpenCLIPER. El proceso `Simulate` requiere esencialmente, para que la simulación pueda tener lugar, de un objeto `Phantom`, que se establece como objeto `Data` de entrada, y de un objeto `Sequence`.

Para favorecer el rendimiento, en OpenCLIPER la inicialización y ejecución son fases separadas, de forma que se puedan lanzar múltiples ejecuciones con una única inicialización allá donde sea posible. En la inicialización se efectúan las operaciones relativas al cálculo de los instantes de tiempo de la simulación. El número de instantes de tiempo resulta imprescindible para la obtención en esta misma fase, de todos aquellos objetos de la clase `XData` de OpenCLIPER imprescindibles para el posterior procesado a través de *kernels*.

En la fase de ejecución, el vector correspondiente a los instantes temporales se divide en un número de particiones, de forma que el procesado se realiza por intervalos de tiempo. Las operaciones a realizar en cada intervalo dependen de si el pulso de RF se encuentra activo, en cuyo caso se está ante *forced precession*, o, en caso contrario, ante *free precession*. En ambos casos, las operaciones se efectúan a través de distintos *kernels*, de forma análoga a las operaciones realizadas por el simulador KomaMRI.jl [8]. Los *kernels* implementados en nuestro simulador efectúan un procesamiento en paralelo de todas las estructuras de datos anteriormente descritas, sin necesidad de reorganización de los datos ni transferencias *host*-dispositivo, maximizando así el rendimiento.

La señal de MRI únicamente se lee en los intervalos de *free precession*. Posteriormente, la señal es interpolada en los instantes de tiempo determinados por los bloques `DAC`, obteniendo así las posiciones correctas en el espacio K. Finalmente, se efectúa una Transformada Inversa de Fourier para la obtención de la imagen final reconstruida, para lo cual se hace uso del proceso `FFT`, que deriva de la clase `Process` de OpenCLIPER.

5. Resultados y discusión

Los resultados se evalúan a través de la comparación del rendimiento del simulador desarrollado en OpenCL/OpenCLIPER y del simulador KomaMRI.jl [8]. Para ello, se ha efectuado el proceso de simulación en ambos simuladores mediante una secuencia común y un mismo fantoma, cuya imagen reconstruida se muestra en la *Figura 4*. El fantoma está constituido por 12714 *spins*, que dan lugar a una primera señal que es posteriormente interpolada en 10201 instantes de tiempo. Esta señal se estructura en una matriz de dimensiones 101×101, que en el simulador desarrollado en OpenCLIPER se redimensiona mediante la introducción de ceros a una matriz de 128×128, para facilitar el empleo del proceso `FFT`.

Las medidas de rendimiento se efectúan en términos de tiempos de ejecución, a partir de 100 ejecuciones, utilizando como parámetros la media y la varianza. Se emplean tres dispositivos distintos, ejecutando en los dos primeros ambos simuladores y en el tercero únicamente el desarrollado en OpenCLIPER (puesto que KomaMRI.jl no está actualmente preparado para funcionar en una GPU AMD).

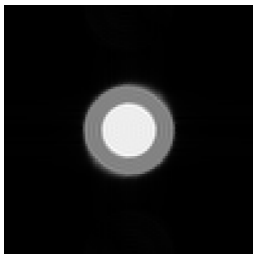

Figura 4. *Fantoma reconstruido.*

En la *Tabla 1* se muestran los tiempos, medidos en segundos, para cada dispositivo.

		Media	Varianza
Nvidia Quadro RTX 6000	OpenCLIPER	0.789284	0.159327
	KomaMRI.jl	4.634991	0.059775
Intel Core i7-4790	OpenCLIPER	2,545896	0,080647163
	KomaMRI.jl	7,024032	0,086608
AMD Radeon RX 5700 XT	OpenCLIPER	2,701813	0,000027

Tabla 1. *Tiempos de ejecución de los simuladores en distintos dispositivos (en segundos).*

De acuerdo con la *Tabla 1*, se aprecia la importante ganancia en términos de tiempos de ejecución que ha supuesto la implementación del simulador en OpenCL/OpenCLIPER, así como la posibilidad de extender los dispositivos en que puede ejecutarse. Los valores reducidos de la varianza en todos los casos permiten determinar la fiabilidad de los resultados obtenidos.

6. Conclusiones y Líneas Futuras

Se ha presentado la implementación de un simulador MRI sobre el *framework* OpenCLIPER, empleando el simulador KomaMRI.jl como base respecto al proceso de simulación.

El trabajo desarrollado ha dado lugar a unos resultados que superan al simulador de partida en tiempos de ejecución, que mantiene su mismo grado de realismo y que funciona sobre dispositivos de cualquier fabricante, tanto CPU como GPU. Adicionalmente, se pone de manifiesto la facilidad que proporciona OpenCLIPER respecto a la ejecución del simulador en cualquier dispositivo de cualquier vendedor.

Los resultados obtenidos son favorables e introducen, además, la posibilidad de futuras mejoras. En particular, la implementación actual no hace uso de memoria local, cuyo uso junto con otras técnicas de optimización podría resultar en una reducción de los tiempos de ejecución aún mayor.

7. Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia e Innovación a través del proyecto PID2020-115339RB-I00 y por el Ministerio de Educación y Formación Profesional a través de la Beca SIA 998142.

Referencias

[1] J. Bittoun, J. Taquin, y M. Sauzade, "A computer algorithm for the simulation of any nuclear magnetic resonance (NMR) imaging method," *Magnetic Resonance Imaging*, vol. 2, pp. 113–120, 1984.

[2] R. de Luis-García, D. Treceño-Fernández, J. Calabia-del Campo, F. J. de Paz-Fernández, y C. Alberola-López, "Magnetic resonance imaging simulator: From a proof of concept to a useful tool in the education of radiographers," in *Technological Adoption and Trends in Health Sciences Teaching, Learning, and Practice*. IGI Global, 2022, pp. 66–93.

[3] P. D. Stoecker T, Vahedipour K y S. N, "High-performance computing MRI simulations," *Magnetic Resonance in Medicine*, vol. 64, no. 1, pp. 186–193, 2010.

[4] F. Liu, J. V. Velikina, W. R. Block, R. Kijowski, y A. A. Samsonov, "Fast realistic MRI simulations based on generalized multi-pool exchange tissue model," *IEEE Transactions on Medical Imaging*, vol. 36, no. 2, pp. 527–537, 2017.

[5] W. Overal y J. Pauly, "An extensible, graphical environment for pulse sequence design and simulation," *Int. Soc. Magn. Reson. Med.*, 2007.

[6] Z. Cao, C. Sica, S. Oh, J. McGarrity, T. Horan, B. Park, y C. Collins, "An MRI simulator for effects of realistic field distributions and pulse sequences, including SAR and noise correlation array coils," *Proceedings of ISMRM*, p. 1456, 2010.

[7] I. Drobnjak, G. S. Pell, y M. Jenkinson, "Simulating the effects of time-varying magnetic fields with a realistic simulated scanner," *Magnetic Resonance Imaging*, vol. 28, no. 7, pp. 1014–1021, 2010.

[8] C. Castillo-Passi y P. Irarrazaval. (2021, Sep.) cncastillo/komamri.jl. Disponible en: https://doi.org/10.5281/zenodo.6627503

[9] J. Bezanson, A. Edelman, S. Karpinski, y V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017.

[10] D. Treceño-Fernández, J. Calabia-del Campo, M. L. Bote-Lorenzo, E. Gómez-Sánchez, R. Luis-García, y C. Alberola-López, "A web-based educational magnetic resonance simulator: Design, implementation and testing," *Journal of Medical Systems*, vol. 44, 2019.

[11] D. Treceño-Fernández, J. Calabia-del Campo, M. L. Bote-Lorenzo, E. Gómez-Sánchez, R. Luis-García, y C. Alberola-López, "Integration of an intelligent tutoring system in a magnetic resonance simulator for education: Technical feasibility and user experience," *Computer Methods and Programs in Biomedicine*, vol. 195, 2020.

[12] F. Simmross-Wattenberg, M. Rodríguez-Cayetano, E. Moya-Sáez, M. Martín-Fernández, y C. Alberola-López, "OpenCLIPER: An OpenCL-based C++ framework for overhead-reduced medical image processing and reconstruction on heterogeneous devices," *IEEE journal of biomedical and health informatics*, vol. 23, pp. 1702–1709, 2019.

[13] J. E. Stone, D. Gohara, y G. Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems," *Computing in science & engineering*, vol. 12, no. 3, p. 66, 2010.