

Nanofoams Creator

Nicolás Vazquez, Carlos Ruestes, Emmanuel Millán

27 de noviembre de 2022

Índice

1. Introducción	1
2. Fundamento teórico	2
2.1. El modelo de Soyarslan	2
2.2. Generación de estructuras periódicas	3
2.3. Caracterización morfológica	3
3. Implementación en código	4
4. Validación de la herramienta	6
5. Rendimiento	8
5.1. Comparando las distintas versiones	8
5.2. Corrida de varios cores contra GPU	8
5.3. Análisis de escala en cluster de cómputo	9
6. Instrucciones de uso	9
6.1. Versión script	9
6.2. Versión notebook	10

1. Introducción

Esta herramienta fue creada en el marco de una investigación que se dedicó al estudio de las propiedades mecánicas de los metales nanoporosos de tipo bcc. A saber nuestro, no se contaba hasta el momento con una herramienta que pudiera generar estas estructuras para las simulaciones de dinámica molecular. La misma se basa en una implementación de un trabajo previo de Soyarslan et al. [1].

Esta herramienta fue programada en Python, y tiene dos versiones, que en esencia hacen lo mismo pero están implementadas sobre diferente hardware:

- La primera versión corre por línea de comandos, mediante ejecución del script `build_nanofoams.py`, que requiere además un cierto script de entrada, cuyo template es el archivo `nanofoams_input.txt`
- La segunda versión está implementada dentro del framework TensorFlow, y está pensada para su uso con aceleración por GPU. La presentación fue hecha en formato Jupyter Notebook, y se puede utilizar dentro de la plataforma de cómputo en la nube Google Colab.

Sin más preámbulo, en las siguientes páginas comentaremos brevemente el fundamento teórico detrás del algoritmo, y también se añaden algunas pruebas de rendimiento que se realizaron en el marco de la validación de la herramienta.

2. Fundamento teórico

Los metales nanoporosos son un caso particular de un tipo de microestructura tipo *estocástica bicontinua*. Estas estructuras presentan dos fases contiguas interpenetradas, y son características de muchos materiales. Algunos ejemplos se encuentran en las aleaciones producidas mediante descomposición espinodal, o los metales nanoporosos producidos por corrosión selectiva [2].

En este contexto, la generación de modelos computacionales fidedignos resulta clave para la correcta evaluación de estos materiales. El esquema tradicional de preparación de muestras nanoporosas se ha basado principalmente en el método de campo de fase (*phase-field*) mediante la ecuación de Cahn-Hilliard [3, 4], y métodos Monte Carlo [5]. Usualmente, una dificultad en esta tarea es el costo computacional de preparar la topología requerida en las muestras, especialmente para sistemas de gran tamaño, factor que ha dificultado el avance de las investigaciones.

Recientemente, un trabajo de Soyarslan et al. [1], propone un nuevo método para la generación de estructuras bicontinuas periódicas, basado en la superposición de ondas sinusoidales con longitud de onda fija y un conjunto fijo de direcciones, en una celda cúbica. Este método ha sido aplicado exitosamente en la generación de muestras para simulaciones mediante elementos finitos [1] y MD [6]. Para la generación de las espumas nanoporosas sobre las que se realizaron posteriormente los ensayos de tracción-compresión, se escribió un código inspirado en este método, para la generación de estructuras bicontinuas de forma sencilla y eficiente.

Posteriormente, una segunda etapa fundamental en este proceso de generación de muestras representativas es el diseño de un esquema de relajación y minimización de la muestra previo a los ensayos de deformación. Este contenido se desarrollará a lo largo del siguiente capítulo.

2.1. El modelo de Soyarslan

A continuación se presenta un breve resumen adaptado del modelo teórico detrás del código con el que se generaron las espumas nanoporosas¹.

Un *elemento de volumen representativo* $\mathcal{V} \subset \mathbb{R}^3$ es una porción de material, que en virtud de sus dimensiones características, es capaz de reproducir de manera fidedigna una o varias de sus características macroscópicas de interés, de ese mismo material. En nuestras estructuras, este volumen está compuesto por dos fases: la fase sólida metálica, y la fase porosa, cuyos volúmenes son respectivamente \mathcal{B} y \mathcal{P} . Se tiene entonces, que $\mathcal{V} = \mathcal{B} + \mathcal{P}$. Por último, se denota la interfase de la microestructura por $\partial\mathcal{B}$.

En su artículo de 1965, Cahn mostró que el campo de composición de estas estructuras puede aproximarse como la superposición de infinitas ondas con dirección aleatoria y longitud de onda fija [7]. El trabajo de Soyarslan et al. explora una implementación moderna de este campo aleatorio f :

$$f(\mathbf{x}) = \sqrt{\frac{2}{N}} \sum_{i=1}^N \cos(\mathbf{q}_i \cdot \mathbf{x} + \varphi_i). \quad (1)$$

En (1), \mathbf{x} es el vector de posición, N es el número de ondas consideradas en la serie truncada, y \mathbf{q}_i y φ_i denotan la dirección de la onda y la fase de la onda i , respectivamente. El número de onda debe ser fijado a un valor constante, es decir $|\mathbf{q}_i| = q_0$, con direcciones distribuidas de manera uniforme sobre el el ángulo sólido 4π , y fases uniformemente distribuidas entre 0 y 2π . Bajo estas condiciones, $f(\mathbf{x})$ es un *campo gaussiano aleatorio*, con $\langle f \rangle = 0$, $\langle f^2 \rangle = 1$. La función de correlación de dos puntos viene dada por

$$C_2(\mathbf{x}_1, \mathbf{x}_2) = \langle f(\mathbf{x}_1)f(\mathbf{x}_2) \rangle = \frac{\sin(q_0 r)}{q_0 r}, \quad (2)$$

donde los brackets indican promedios de ensamble sobre diferentes realizaciones, y $r = |\mathbf{x}_1 - \mathbf{x}_2|$. Si N es suficientemente grande², el valor de la función $f(\mathbf{x})$ para una posición \mathbf{x} dada sigue una distribución gaussiana:

$$P(f) = \frac{1}{\sqrt{2\pi}} e^{-f^2/2}, \quad (3)$$

¹Para más detalles, el lector puede referirse al artículo original [1].

²En este trabajo, N está en el orden de 10^4 , siguiendo las recomendaciones en [1].

y la dirección de onda media

$$\nu = \frac{1}{N} \sum_{i=1}^N \mathbf{n}_i \quad (4)$$

tiende a cero como consecuencia del teorema central del límite. Se sigue de esto que f es una función isotrópica.

Dada la función aleatoria (1), las diferentes fases del sistema están dadas mediante un valor de corte ξ :

$$\begin{aligned} \mathbf{x} \in \mathcal{B} & \quad \text{si } f(\mathbf{x}) < \xi, \\ \mathbf{x} \in \partial\mathcal{B} & \quad \text{si } f(\mathbf{x}) = \xi, \\ \mathbf{x} \in \mathcal{P} & \quad \text{si } f(\mathbf{x}) > \xi. \end{aligned} \quad (5)$$

Este valor de corte ξ puede correlacionarse con el valor de fracción sólida, definido como $\phi_{\mathcal{B}} := |\mathcal{B}|/|\mathcal{V}|$, gracias a las propiedades del campo gaussiano aleatorio:

$$\xi(\phi_{\mathcal{B}}) = \sqrt{2} \operatorname{erf}^{-1}(2\phi_{\mathcal{B}} - 1), \quad (6)$$

donde $\operatorname{erf}^{-1}(x)$ denota la función error inversa.

2.2. Generación de estructuras periódicas

Las estructuras generadas mediante las ec. (1) y (5) son, en general, no periódicas. La periodicidad puede lograrse seleccionando un número finito de ondas con números de onda enteros en todas las direcciones, y módulo constante.

A partir de (1), tomamos $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ como vectores unitarios para formar una base ortonormal en el espacio real, y se pide que los \mathbf{q}_i tengan la forma

$$\mathbf{q} = \frac{2\pi}{a}(h, k, l), \quad (7)$$

donde los *índices de Miller* $h, k, l \in \mathbb{Z}$ y a es una constante. De esta forma, se verifica que $f(\mathbf{x}) = f(\mathbf{x} + n_1 a \mathbf{e}_1 + n_2 a \mathbf{e}_2 + n_3 a \mathbf{e}_3)$, con n_1, n_2 y n_3 enteros arbitrarios. Esto implica que f tiene periodicidad con respecto a los vectores de la base, y con magnitud a . Más aún, si en (1) se fijan los ϕ_i a un valor idéntico, entonces f es también invariante con respecto al intercambio de ejes, y f adquiere la simetría de una red cúbica con parámetro de red a .

De estas consideraciones, se tiene que las estructuras espinodales periódicas pueden ser construidas restringiendo la suma en la ec. (1) a un conjunto de \mathbf{q}_i con un valor dado y constante H , tal que

$$H = \sqrt{h^2 + k^2 + l^2}, \quad (8)$$

y donde $|\mathbf{q}| = q_0 = 2\pi H/a$. La función f entonces tiene condiciones periódicas en las 3 direcciones, formando celdas unitarias cúbicas e idénticas de lado a .

2.3. Caracterización morfológica

El método descrito en [1], tiene la ventaja de que a partir de su definición es posible deducir expresiones explícitas para propiedades morfológicas y topológicas de interés, tales como la relación superficie-volumen, el diámetro medio de ligamento, curvatura gaussiana, el genus (una forma de cuantificar la conectividad de la red), etc. Al medir estas cantidades en nuestras espumas generadas, se pudo corroborar el correcto funcionamiento del código implementado, contrastando resultados. A continuación, se dejan listadas algunas de las ecuaciones utilizadas en este trabajo para la caracterización.

Cada una de las componentes $\cos(\mathbf{q}_i \cdot \mathbf{x})$ que componen el campo gaussiano aleatorio (1) tienen una misma longitud de onda $\lambda = 2\pi/q_0$, que puede usarse como una primer estimación de un tamaño característico de la estructura. Sin embargo, un estudio más detallado requiere inspeccionar la función de autocorrelación (2). El primer máximo de esta ecuación está en $q_0 r \approx 1,23 \times 2\pi$, que

define una distancia característica \hat{L} , entre regiones de máximo valor de f . Al ser las espumas estructuras binarias, \hat{L} puede interpretarse como la distancia media entre centros locales de espacio sólido o poroso; más claramente, la distancia promedio entre los centros de dos ligamentos vecinos:

$$\hat{L} = 1,23 \frac{2\pi}{q_0} \quad (9)$$

Otra longitud característica importante de estas espumas nanoporosas es el diámetro medio de ligamento L , para la cual el modelo de Soyarslan predice una expresión en función de ϕ_B y de H :

$$L = \frac{a}{H}[0,53\phi_B + 0,41], \quad (10)$$

donde a es la misma constante introducida mediante la ec. (7). En el artículo original, Soyarslan propone esta ecuación tras hacer una ajuste de L en función de \hat{L} y ϕ_B .

Otra cantidad importante para caracterizar la espuma es la relación entre superficie y volumen o superficie específica $S = \frac{|\partial\mathcal{B}|}{|\mathcal{V}|}$. La expresión para calcularla es

$$S = \frac{2q_0}{\pi\sqrt{3}} e^{-\xi^2/2}, \quad (11)$$

con ξ dependiente de la fracción sólida por la ec. (6).

Por último, para caracterizar la topología de la red, se utiliza el número de genus G , que es un parámetro topológico que caracteriza la conectividad de la red [8]. Este número además puede interpretarse como el número de túneles o loops en la estructura; por ejemplo, el genus de una esfera es 0, el de un toroide es 1, y el de un pretzel es 2. Como este valor es una característica extensiva del volumen representativo, lo que suele hacerse es expresarlo de manera reducida por algún tamaño característico, dividiendo por ejemplo por el volumen sólido o total, a veces en conjunto con la relación superficie/volumen [9, 10]. En Soyarslan et al. [1], se presenta por ejemplo la cantidad g_V , que es el G escaleado por el volumen total-volumen sólido más volumen poroso:

$$g_V = \frac{1}{12\pi^2} \frac{q_0^3}{\sqrt{3}} [1 - \xi^2] e^{-\xi^2/2} \quad (12)$$

En el artículo de Guillotte et al. [9], se presenta una densidad de genus reducida $g_v \times S^{-3}$, que resulta de combinar las expresiones de las ec. (11) y (12):

$$g_V \times S^{-3} = \frac{3\pi}{32} [1 - \xi^2] e^{\xi^2/2} \quad (13)$$

Todos estos resultados analíticos son útiles para predecir las características de la espuma previamente a su generación. Por otra parte, si pueden obtenerse por algún otro método estas cantidades, es posible contrastarlos con la teoría para corroborar el correcto funcionamiento del software. Los resultados de este estudio son presentados en la sección ??.

3. Implementación en código

En total, se desarrollaron 4 versiones del código, cada una de ellas desarrollando mejoras de manera iterativa:

1. La primera versión fue escrita en lenguaje “estándar” de Python, es decir, sin el uso de bibliotecas externas. Esto implica el uso de grandes bucles anidados. Se hicieron algunas mejoras que fueron conservadas en todas las otras versiones, como es por ejemplo el añadido de un pequeño programa capaz de calcular el conjunto de direcciones $[hkl]$ posibles para un valor entero de H dado, para poder generar estructuras periódicas.
2. La segunda versión de este código reemplaza el uso de bucles anidados, para redefinir todo en términos de operaciones matriciales. Para ello se hizo uso de la librería de computación científica NumPy [11].

En esta nueva implementación (así como en todas las siguientes) se implementó un tensor de rango 3 definido

$$T_{ijk} := (\mathbf{q} \cdot \mathbf{x})_{ij} + \phi_k, \quad (14)$$

para todos los puntos \mathbf{x} del dominio y para todos los vectores de onda \mathbf{q} del conjunto de direcciones³.

Al momento del desarrollo, se vio que la cantidad de memoria necesaria para alojar este tensor en memoria era para todos los casos de interés muy superior al terabyte(Tb), con lo cual se hicieron dos modificaciones importantes, que se conservaron en el resto de las versiones implementadas:

- Convertir todos los flotantes de doble precisión (estándar para flotante en Python) a flotante de simple precisión. Por sí sola, esta modificación conlleva una reducción del 50 % de costos de memoria.
- Partitionar el tensor en subconjuntos para llevar a cabo el cálculo por partes. Para evitar almacenar todo el tensor en memoria de una sola vez, el programa toma un pequeño lote de átomos por vez (del orden de 32) y calcula el tensor, evaluando en ellos la función.

Por último, vale la pena mencionar que gracias a ciertas funcionalidades de la creación de grillas en NumPy, el uso de *arrays* permite poder crear dominios con celdas unitarias tipo cúbica, bcc y fcc. Esto también se conservó en versiones posteriores.

3. La segmentación del tensor en partes más pequeñas e independientes hizo posible repartir el trabajo en los distintos cores del procesador, permitiendo la paralelización del trabajo. Para asignar las distintas tareas a los hilos y procesadores se recurrió al uso de las librerías *joblib* y *loky*⁴.
4. Una vez reescrito el código en términos de operaciones matriciales, se consideró desarrollar una versión alternativa, capaz de utilizar la potencia de cálculo de las GPUs. Esto se pudo llevar a cabo, reescribiendo el código de la versión (2) para funcionar bajo el framework *Tensorflow 2* [12], conocido por su uso extendido en el ámbito del *Machine Learning*.

Una versión resumida del desarrollo de las distintas versiones del código y las mejoras introducidas está contenido en la tabla 1.

Versión del código	Plataforma	Mejoras introducidas
1ra (baseline)	CPU 1 core/1 thread	Versión inicial del código. Esquema de bucles anidados para el cálculo.
2da (NumPy - Single thread)	CPU 1core/1 thread	Implementación de los cálculos en matrices. Cálculo por batches.
3ra (Numpy - Multi-thread)	CPU Multicore/Multi thread	Implementación de los cálculos en matrices con división de cálculo por batches en los distintos threads. Optimización para uso en clústers gracias a la paralelización.
4ta (Tensorflow GPU)	GPU	Implementación de los cálculos mediante grafos y tensores de Tensorflow. Cálculo masivo de batches por medio de distribución en GPU.

Cuadro 1: Resumen de las características básicas de las cuatro versiones implementadas para el algoritmo de generación de las espumas basado en el método de Soyarsan et al. [1]

La comparación de rendimiento de las diferentes versiones se puede ver en la sección 5.

El programa corre en su totalidad mediante la línea de comandos. El esquema básico de funcionamiento es el siguiente:

³Es decir, para un número N de átomos, un conjunto de Q direcciones y M fases, este tensor tiene dimensiones $N \times Q \times M$

⁴<https://github.com/joblib/loky>

- Como entrada, es necesario que el usuario pase un archivo de texto con algunos parámetros de entrada para la configuración de la espuma: dimensiones, diámetro de ligamento, fracción sólida, número de ondas N , constante a y semilla de números aleatorios. Estos parámetros definen las características de la muestra generada.
- A partir del diámetro de ligamento ingresado por el usuario, el programa calcula el valor de H necesario mediante la ec. (10), y el conjunto de direcciones asociado según (7). Luego de eso, se generan las fases aleatorias y la grilla de puntos, que luego son evaluados en la función (1). Esta última etapa es la más costosa en términos computacionales.
- Por último, el programa clasifica los átomos en dos fases de acuerdo con la ec. (5), para luego eliminar la fase porosa. Los átomos remanentes son exportados a un archivo de texto en un formato compatible con LAMMPS, que puede ser tomado como archivo de entrada para una simulación atomística.

4. Validación de la herramienta

La figura 1 muestra algunas de las muestras típicas generadas por la herramienta. Para una fracción sólida ϕ_b pequeña, o bien para diámetros de ligamentos cercanos al orden del parámetro de red, es posible que la estructura se encuentre por debajo del límite de percolación, y se vean regiones sólidas inconexas, en forma de clusters aislados [1]. Para fracciones sólidas por encima de 0.20 y para L/L_{box} mayores a 0.04 (siendo L_{box} el lado de la caja de simulación), las pruebas muestran que las espumas resultan totalmente conexas.

Para $L/L_{\text{box}} > 0.08$, el diámetro de ligamento es tan grande en relación al dominio de la simulación que puede no resultar en un elemento de volumen representativo del material estudiado. Es por ello que los diámetros de ligamento de las muestras generadas en este trabajo de tesis se mantienen en el rango que exhibe la figura 1. Como regla general, los elementos de volumen representativos se diseñan de forma tal que la relación $L/L_{\text{box}} < 0.1$.

Para validar la herramienta, se contrastaron algunas cantidades de interés predichas por el método de Soyarslan, con algunas mediciones sobre espumas de diferente fracción sólida entre 0.20 y 0.50. Estos resultados se encuentran graficados en la fig. 2. En todos los casos, los puntos son los valores medidos, y las curvas los modelos analíticos.

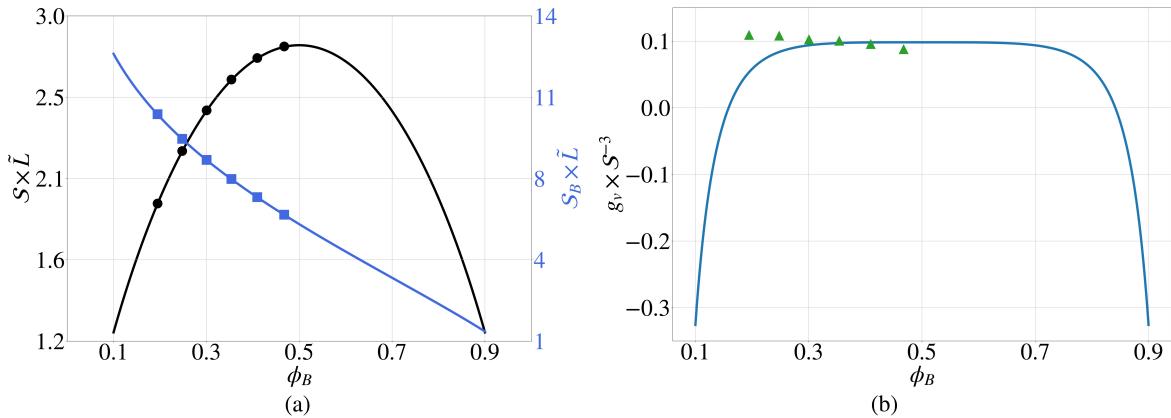


Figura 2: Resultados de la comparación entre algunos resultados analíticos del modelo de Soyarslan (líneas sólidas), y las mediciones realizadas sobre espumas generados con el software (puntos): **(a)** En color negro, comparación de longitudes características reducidas, $S \times \hat{L}$ es la superficie específica (11) multiplicada por la magnitud \hat{L} definida en (9). La curva y puntos azules muestran la cantidad S_B , que es la superficie específica, tomada en relación al volumen sólido. **(b)** Mediciones para la caracterización de la conectividad de la red, mediante el parámetro $g_V \times S^{-3}$ (13), en función de la fracción sólida.

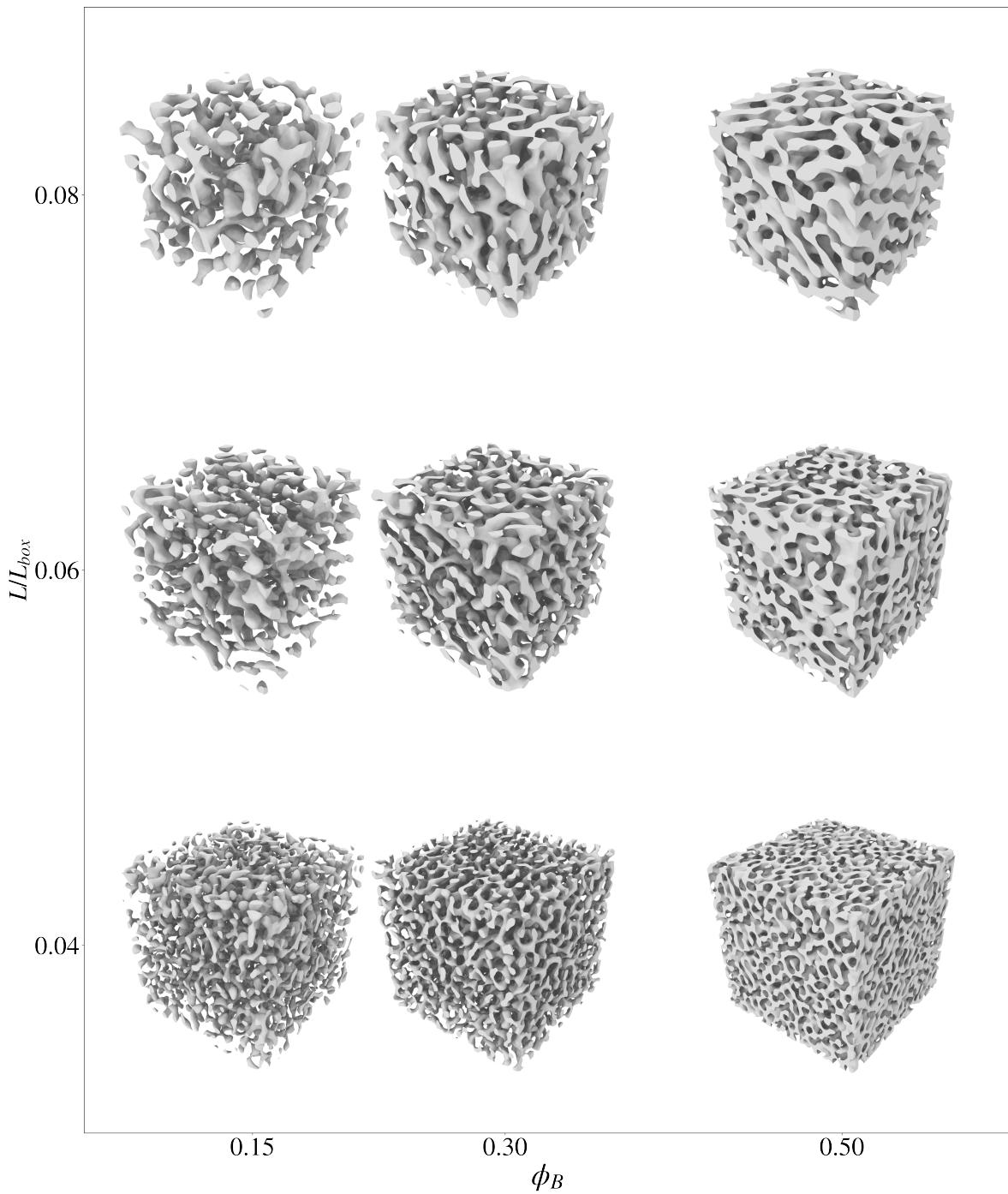


Figura 1: Este panel ilustra cómo los diferentes parámetros de entrada en el código afectan características morfológicas de la estructura generada. En el eje de ordenadas, se grafica el diámetro L de ligamento promedio solicitado, en relación al volumen del dominio, y en el eje de abscisas, la fracción sólida ϕ_B . Para $\phi_B = 0,15$, pueden notarse regiones aisladas de sólido, debido a la alta porosidad. Los rangos mostrados aquí de estas cantidades son típicos de las espumas que serán trabajadas en las simulaciones. La naturaleza simétrica de la construcción teórica del modelo de Soyarslan hace que para $\phi_b > 0,5$, se pueda encontrar algo muy similar a lo mostrado en la figura, en forma “espejada”.

5. Rendimiento

En la siguiente sección, se comparan las velocidades de las distintas implementaciones del código de generación de las nanoespumas.

5.1. Comparando las distintas versiones

Para evaluar la eficiencia del programa, se realizaron ejecuciones de prueba sobre las 4 implementaciones distintas.

En la fig. 3 puede verse el resultado de las pruebas de velocidad realizadas sobre una misma espuma de 100^3 celdas cúbicas simples, sumando sobre 48 direcciones y con 10000 ondas. Las barras azules en la figura indican el tiempo de pared “*Walltime*” medido en horas, y las barras naranjas muestran el *speedup*, una métrica utilizada para medir la escalabilidad de un código computacional paralelizable, definido como

$$\text{speedup} = \frac{t_1}{t_N}, \quad (15)$$

donde t_1 es el tiempo requerido al correr el código en un único procesador, y t_N el tiempo para N procesadores. En un escenario ideal, el speedup debería tener una relación lineal con el número de procesadores utilizados. Esto significaría que cada procesador está otorgando el 100 % de su potencia computacional, lo que en la práctica representa un gran desafío [13].

Los resultados demuestran una clara aceleración a partir de la implementación de los cálculos mediante matrices, como era de esperarse. Llama la atención también la gran velocidad que tiene la versión implementada en GPU, incluso respecto de las versiones corridas en CPU (casi 10 veces más rápida que corriendo en 4 cores).

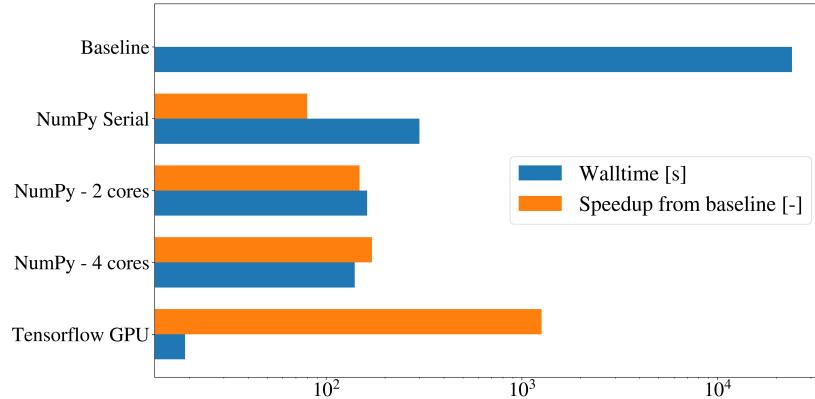


Figura 3: Resultados en una prueba de tiempos realizada sobre las 4 versiones implementadas del código. En barras azules, el tiempo total de ejecución en segundos (Walltime); en barras naranjas , el speedup respecto del código inicial (baseline). Estas pruebas se ejecutaron en un CPU de laptop, modelo AMD Ryzen3 4000 series de 4 cores, en conjunto con una GPU NVidia K80 de 4.1 TFlops.

5.2. Corrida de varios cores contra GPU

Finalmente, para hacer un recorrido integral, resulta interesante comparar los tiempos de ejecución de las dos versiones finales del código. Para ello se realizó una ejecución en 1 y 16 cores de la versión CPU en el cluster Toko, contra una corrida en GPU de la misma espuma bcc con 300^3 celdas unitarias y 20000 ondas. Los resultados de esta evaluación de rendimiento están comparados en la fig. 4.a. Nuevamente, es notorio aquí que el rendimiento de la versión GPU ha alcanzado es muy alto comparado con la versión diseñada para CPU en paralelo.

5.3. Análisis de escala en cluster de cómputo

La computación en paralelo, donde varios procesadores trabajan simultáneamente para generar potencia computacional y reducir el tiempo de cómputo, es una técnica usual en el desarrollo e implementación de códigos de alto desempeño. Una métrica utilizada para caracterizar cómo cambia el tiempo de cómputo al aumentar los recursos disponibles para la tarea es la **escalabilidad**. Para esta última prueba, se eligió una muestra del mismo tamaño que en la prueba anterior (300^3 celdas bcc cúbicas y 20000 ondas, que son alrededor de 54 millones de átomos), y se hizo una prueba de *escalamiento fuerte*. Es decir, fijando el tamaño de muestra, se aumenta el número de procesadores.

Las pruebas que corresponden a este análisis se hicieron sobre los nodos del cluster Toko (<http://toko.uncu.edu.ar>), en particular los nodos 05, 07, y mini. Los resultados de las pruebas se presentan en la fig. 4.b .

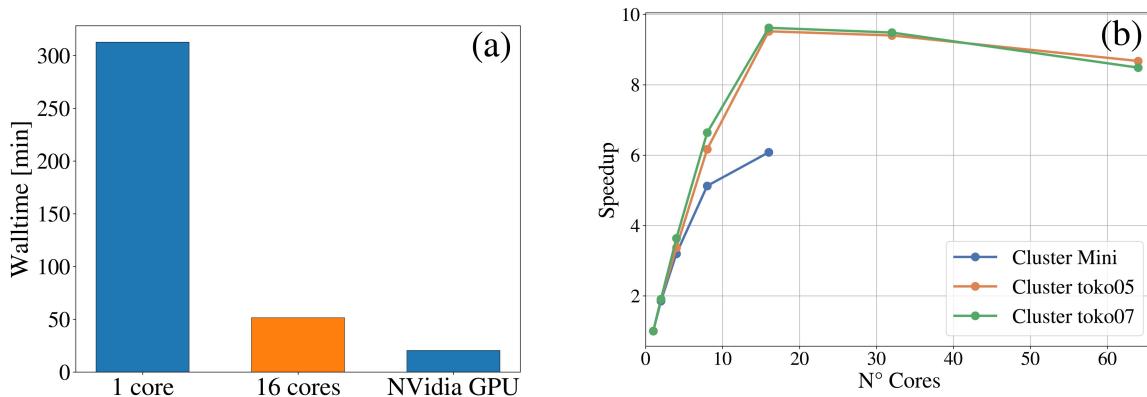


Figura 4: (a) Comparación entre tres corridas de la misma espuma, ejecutando en serie y paralelo sobre la versión CPU, contra la versión GPU. La corrida CPU se realizó sobre uno de los nodos *mini* del cluster Toko (16 cores, procesador AMD Ryzen 7, 64 Gb RAM), mientras que la ejecución GPU se realizó en Google Colab, utilizando una tarjeta NVidia K80. (b) Speedup vs. nro de cores para la implementación paralela del código, ejecutado sobre 3 nodos distintos del cluster Toko (FCEN-UNCuyo).

6. Instrucciones de uso

6.1. Versión script

Esta versión del código usa la librería NumPy, en conjunto con *joblib* para la división del cómputo entre los diferentes cpus. Para ejecutar, sólo se debe correr por línea de comandos algo como

```
python3 build_nanofoams.py inputFile.txt 4
```

donde *inputFile.txt* es un archivo de entrada que el programa necesita para la ejecución. En el mismo se insertan parámetros de entrada como las dimensiones del dominio, tamaño esperado de ligamento, densidad relativa, etc. El formato de este archivo debería ser algo similar al siguiente

```

#####
# Input script nanofoams_input.txt
# Datos de entrada para el armado de las espumas
# Para usar con script build_nanofoams.py
# Nota: Las unidades de longitud son a elección del usuario ,
# siempre que sean consistentes en todos los parametros.
#####

# Tipo de red cristalina (Cubica:0 ,BCC:1 ,FCC:2)
# (ante un valor distinto de 1 o 2, la red es cubica)
1

# Tamanio de caja: x_size y_size z_size (Unidad: AA)
330.3
330.3
330.3

# Parametro de red
3.303

# Fraccion solida de la espuma
0.30

# Datos para el armado de la funcion f
# Tamanio medio de ligamento deseado
# (este valor puede variar al generar la espuma ,
# pero se le informa al usuario dicho cambio antes de continuar)
50.2
# Numero de ondas N
10000

# Constante a
330.3

# Semilla para inicializar numeros aleatorios
42

```

El segundo parámetro que se debe ingresar por línea de comandos es el número de procesadores que se desean utilizar. **Advertencia:** Por defecto, si el usuario no ingresa ningún número, el programa inicia utilizando todos los procesadores disponibles.

6.2. Versión notebook

El cuaderno que fue preparado para este trabajo está explicado en modo autocontenido en el notebook, referirse allí para instrucciones de uso.

Referencias

- ¹C. Soyarslan, S. Bargmann, M. Pradas y J. Weissmüller, «3D stochastic bicontinuous microstructures: Generation, topology and elasticity», *Acta Materialia* **149**, 326-340 (2018).
- ²J. Weissmüller, R. C. Newman, H.-J. Jin, A. M. Hodge y J. W. Kysar, «Nanoporous Metals by Alloy Corrosion: Formation and Mechanical Properties», *MRS Bulletin* **34**, 577-586 (2009).
- ³D. CROWSON, D. FARKAS y S. CORCORAN, «Geometric relaxation of nanoporous metals: The role of surface relaxation», *Scripta Materialia* **56**, 919-922 (2007).
- ⁴D. Farkas, A. Caro, E. Bringa y D. Crowson, «Mechanical response of nanoporous gold», *Acta Materialia* **61**, 3249-3256 (2013).

- ⁵B.-N. D. Ngô, A. Stukowski, N. Mameka, J. Markmann, K. Albe y J. Weissmüller, «Anomalous compliance and early yielding of nanoporous gold», *Acta Materialia* **93**, 144-155 (2015).
- ⁶C. Liu y P. S. Branicio, «Efficient generation of non-cubic stochastic periodic bicontinuous nanoporous structures», *Computational Materials Science* **169**, 109101 (2019).
- ⁷J. W. Cahn, «Phase Separation by Spinodal Decomposition in Isotropic Systems», *The Journal of Chemical Physics* **42**, 93-99 (1965).
- ⁸J. R. Munkres, *Topology*, Featured Titles for Topology (Prentice Hall, Incorporated, 2000).
- ⁹M. Guillotte, J. Godet y L. Pizzagalli, «A fully molecular dynamics-based method for modeling nanoporous gold», *Computational Materials Science* **161**, 135-142 (2019).
- ¹⁰Y. Li, B.-N. Dinh Ngô, J. Markmann y J. Weissmüller, «Topology evolution during coarsening of nanoscale metal network structures», *Physical Review Materials* **3**, 076001 (2019).
- ¹¹C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke y T. E. Oliphant, «Array programming with NumPy», *Nature* **585**, 10 . 1038 / s41586-020-2649-2 (2020).
- ¹²Y. Tang, «TF. Learn: TensorFlow's high-level module for distributed machine learning», arXiv preprint arXiv:1612.04251 (2016).
- ¹³G. M. Amdahl, «Validity of the single processor approach to achieving large scale computing capabilities», en Proceedings of the April 18-20, 1967, spring joint computer conference on - AFIPS '67 (Spring) (1967), pág. 483.