# Daisyworld
# Un acercamiento desde métodos montecarlo

David S. Duque-Castaño*
(Dated: Marzo 28, 2025)

Este trabajo analiza el impacto de distintas estrategias de paralelización sobre el rendimiento computacional del muestreo mediante cadenas de Markov Monte Carlo (MCMC) aplicado al modelo Daisyworld, comparando una implementación secuencial con dos enfoques paralelos: Joblib y MPI. Se realizaron experimentos variando el número de caminantes (50 a 700) en un sistema con 16 hilos físicos, evaluando métricas como tiempo de ejecución, speed-up, eficiencia y escalabilidad. Los resultados muestran que MPI ofrece mejoras sustanciales en rendimiento, reduciendo el tiempo de cómputo hasta en un ∼90% respecto al método secuencial y superando consistentemente a Joblib, especialmente en cargas elevadas. A partir de 200 caminantes, todas las estrategias alcanzan una precisión estadística comparable, pero sólo MPI mantiene una eficiencia creciente y una escalabilidad cercana a lo ideal según la Ley de Gustafson, con un rendimiento óptimo en torno a 10–12 procesos. En conjunto, el estudio demuestra que la paralelización con MPI permite acelerar significativamente simulaciones MCMC sin comprometer la calidad de los resultados, siempre que se mantenga un equilibrio adecuado entre carga de trabajo y número de procesos.

## I. INTRODUCCIÓN

La Tierra puede entenderse como un sistema compuesto por diferentes subsistemas interactuantes: la geosfera, la hidrosfera, la atmósfera y la biosfera. En particular, la biosfera —el conjunto total de ecosistemas vivos del planeta— interactúa de manera compleja con los otros componentes, influyendo en procesos como el ciclo del carbono, el balance energético y la regulación climática global. Esta perspectiva se formaliza en la hipótesis Gaia de Lovelock, quien sugiere que la Tierra podría comportarse como una entidad autorregulada, manteniendo condiciones favorables mediante retroalimentación negativa ante perturbaciones ambientales [1].

Para ejemplificar estos mecanismos, [2] introdujeron el modelo Daisyworld, un planeta imaginario cubierto por dos especies de margaritas: blancas (alto albedo) y negras (bajo albedo). La fracción de cobertura de cada especie, $a_w$ y $a_b$, evoluciona según:

$$\frac{da_w}{dt} = a_w \left[x \cdot \beta(T_w) - \gamma\right] \tag{1}$$

$$\frac{da_b}{dt} = a_b \left[x \cdot \beta(T_b) - \gamma\right] \tag{2}$$

con $\beta(T_i) = 1 - 0.003265, (22.5 - T_i)^2$, donde $T_i$ es la temperatura local y $\gamma$ la tasa de mortalidad vegetal.

En mi trabajo previo, apliqué MCMC para ajustar parámetros como la luminosidad solar y la mortalidad, buscando el equilibrio donde $a_w \approx a_b \approx 0.4$. Los métodos MCMC, iniciados con [3] y generalizados por [4], permiten muestrear distribuciones complejas mediante transiciones de Markov convergentes a la distribución objetivo. Para la implementación de estas MCMC se utilizó la librería de python `emcee` [5]

No obstante, en el trabajo anterior no se realizó ningún análisis relacionado con la eficiencia o tiempos de ejecución, ni tampoco se consideró la paralelización. Por ello, este proyecto explora específicamente dos estrategias de paralelización para mejorar el rendimiento computacional:

1. **Joblib**[6]: creación de un pool de procesos para ejecutar simultáneamente evaluaciones independientes de la función de verosimilitud en un entorno de memoria compartida.

2. **MPI (Message Passing Interface)** [7–11] : modelo distribuido tipo scatter/gather que permite distribuir los walkers entre diferentes procesos o nodos, optimizando la escalabilidad en clusters.

## II. MÉTRICAS DE RENDIMIENTO

Para comparar objetivamente las estrategias de paralelización, se definen y desarrollan las siguientes métricas:

### A. Speed-up

El **speed-up** ($S_p$) cuantifica la aceleración obtenida al paralelizar un algoritmo y se define como [12]:

$$S_p = \frac{T_1}{T_p}, \tag{3}$$

siendo $T_1$ el tiempo de ejecución secuencial y $T_p$ el tiempo con $p$ procesadores.

* Maestría en Física, Facultad de Ciencias Exactas y Naturales, Universidad de Antioquia.

### B. Eficiencia

La **eficiencia** ($E_p$) mide la utilización efectiva de los recursos y se calcula como [13]:

$$E_p = \frac{S_p}{p}. \tag{4}$$

Valores de $E_p$ cercanos a 1 indican aprovechamiento casi óptimo de cada procesador; una caída sustancial al aumentar $p$ revela sobrecarga de comunicación o secciones seriales dominantes [13].

### C. Escalabilidad

La Ley de Gustafson reformula la **escalabilidad** considerando que el tamaño del problema puede aumentar con los recursos disponibles. Según Gustafson [14]:

$$S_p = p - \alpha\,(p-1), \tag{5}$$

donde $\alpha$ es la fracción serial en un problema cuyo tamaño crece proporcionalmente a $p$. Al incrementar la carga de trabajo (por ejemplo, más *walkers* en MCMC), la parte paralelizable predomina, permitiendo un escalado cercano a lineal.

## III. MÉTODOS

Este estudio tiene como propósito explorar y evaluar cómo distintas estrategias de paralelización pueden mejorar la eficiencia computacional del muestreo mediante MCMC aplicado al modelo Daisyworld. Para ello, se comparó una implementación secuencial tradicional con dos enfoques paralelos específicos: el primero, basado en la librería `Joblib` para aprovechar múltiples núcleos en una arquitectura de memoria compartida; y el segundo, utilizando la interfaz MPI (*Message Passing Interface*) para distribuir la carga computacional en múltiples procesos. Todos los experimentos fueron realizados en un computador equipado con un procesador AMD Ryzen 7 5700X3D, de 8 núcleos físicos y 16 hilos operando a 3.00GHz, junto con una memoria RAM de 64GB.

El modelo Daisyworld fue definido en un espacio paramétrico de seis dimensiones, que incluyó variables críticas como la luminosidad solar y la tasa de mortalidad de las margaritas. El proceso de muestreo contempló una fase inicial de estabilización (*burn-in*) de 200 pasos, seguida por 700 pasos adicionales de muestreo efectivo por cada caminante. Se realizó una evaluación sistemática variando la cantidad total de caminantes en el intervalo: 50, 100, 200, 300, 400, 500 y 700. Las posiciones iniciales de los caminantes fueron generadas mediante distribuciones uniformes dentro de rangos previamente definidos. La función de pérdida empleada en la optimización del modelo se definió como la suma del cuadrado de las desviaciones entre las coberturas simuladas y las coberturas objetivo, establecidas ambas en un 40% para margaritas blancas y negras (Idéntico al ejercicio anterior).

En la implementación secuencial se utilizó directamente el objeto `EnsembleSampler` de la biblioteca `emcee`, sin implementar ningún mecanismo adicional de paralelización. Se midió el tiempo total requerido para cada configuración experimental mediante la función `time.time()`. Posteriormente, las cadenas obtenidas fueron analizadas estadísticamente para determinar la mediana de los parámetros y evaluar la precisión alcanzada respecto a la cobertura objetivo.

La primera estrategia paralela, basada en `joblib`, permitió distribuir las evaluaciones de la función de verosimilitud entre múltiples núcleos mediante un conjunto o "pool" de procesos. Esta arquitectura se benefició de mecanismos de memoria compartida, lo que evitó la duplicación de grandes estructuras de datos y minimizó la sobrecarga computacional. Los resultados obtenidos, incluyendo el tiempo total y las coberturas logradas, fueron analizados de manera coherente con el enfoque secuencial.

La segunda estrategia se fundamentó en MPI, aprovechando todos los 16 hilos disponibles del procesador para maximizar la eficiencia del cómputo distribuido. En este enfoque, el proceso raíz generó y distribuyó las condiciones iniciales mediante la función `comm.scatter`. Cada proceso realizó independientemente la evaluación MCMC sobre un subconjunto asignado de caminantes, midiendo el tiempo localmente con la función `MPI.Wtime()`. Al finalizar, los resultados fueron recopilados y concatenados por el proceso raíz usando `comm.gather`, que luego efectuó el análisis estadístico final.

Cabe señalar que esta metodología presentó dificultades específicas cuando la cantidad de caminantes era baja, pues era necesario limitar la cantidad de procesos para asegurar una distribución efectiva del trabajo computacional. Así, para 50 caminantes se emplearon únicamente 4 procesos, y para 100 caminantes, 6 procesos.

Además, se llevó a cabo un análisis complementario en el que se evaluó el efecto del número de procesos MPI sobre el tiempo de cómputo, manteniendo fijas tres cantidades representativas de caminantes (200, 500 y 700). Este experimento permitió un estudio más detallado de la escalabilidad y el rendimiento del enfoque basado en MPI, proporcionando información valiosa sobre el comportamiento del sistema ante diferentes configuraciones de paralelización.

Finalmente, los resultados obtenidos fueron recopilados y analizados en términos del tiempo de ejecución, errores respecto a la cobertura objetivo, y mediante métricas específicas como el speed-up y la eficiencia, proporcionando así una evaluación cuantitativa robusta del rendimiento y fiabilidad de las estrategias estudiadas.

## IV. RESULTADOS

En esta sección se presentan los resultados obtenidos del análisis comparativo entre diferentes estrategias de paralelización para implementar métodos Monte Carlo de cadenas de Markov (MCMC). Se evaluaron específicamente tres estrategias: ejecución secuencial, paralelización mediante Joblib y paralelización mediante MPI. Las comparaciones realizadas consideran tanto métricas de rendimiento (tiempo de ejecución, speed-up y eficiencia) como métricas de calidad estadística (error promedio), permitiendo identificar claramente las ventajas y limitaciones de cada enfoque bajo distintos escenarios de carga computacional.

La FIG. 1 presenta la comparación del tiempo de ejecución y el error promedio de tres estrategias diferentes para implementar métodos MCMC (Secuencial, Joblib y MPI), en función del número de walkers utilizados (desde 50 hasta 700). La estrategia secuencial muestra un crecimiento casi lineal en el tiempo requerido para completar la tarea (aproximadamente desde 28s hasta 385s), lo cual evidencia claramente la dependencia directa del tiempo de ejecución con la cantidad de trabajo sin aprovechar paralelización alguna. Por otro lado, Joblib mejora el tiempo frente al método secuencial, aunque su rendimiento resulta inestable especialmente en rangos medios (entre 200 y 500 walkers), lo que podría atribuirse a sobrecargas internas relacionadas con el paralelismo gestionado por el pool en emcee y la gestión de tareas de alto nivel en Python. Finalmente, MPI alcanza tiempos consistentemente menores (de aproximadamente 8s hasta 50s), debido a su eficiente división de tareas en procesos independientes mediante mpi4py.

En cuanto al error promedio, MPI muestra inicialmente valores altos (alrededor de 0.11 a 0.048) para cantidades pequeñas de walkers (entre 50 y 100), consecuencia de un particionamiento poco eficiente del espacio de búsqueda en pocos procesos, afectando negativamente la exploración inicial. Sin embargo, a partir de 200 walkers, todas las estrategias convergen hacia errores reducidos cercanos a 0.01, siendo MPI la estrategia que alcanza el mínimo absoluto (aproximadamente 0.002 en 400 walkers). Más allá de este punto, el error presenta un leve incremento, alcanzando alrededor de 0.013 a 700 walkers, sugiriendo fluctuaciones estadísticas naturales y posibles desequilibrios puntuales en la distribución de muestras.

En la FIG. 2 se analiza específicamente el comportamiento del speed-up y la eficiencia en función del número de walkers con una cantidad fija de 16 procesos. Para MPI, el speed-up muestra un crecimiento progresivo desde aproximadamente 3.0 (50 walkers) hasta 8.5 (700 walkers). En contraste, Joblib alcanza valores significativamente menores (alrededor de 2.2 como máximo). Por su parte, la eficiencia asociada a MPI mejora considerablemente al aumentar la carga, incrementándose de 0.18 hasta 0.53, lo cual indica que al aumentar el número de walkers, cada proceso realiza más trabajo antes de requerir sincronización, amortizando así la sobrecarga inicial de la comunicación entre procesos. La baja eficiencia sostenida de Joblib (entre 0.06 y 0.13) refleja nuevamente sus limitaciones en la gestión del paralelismo interno.

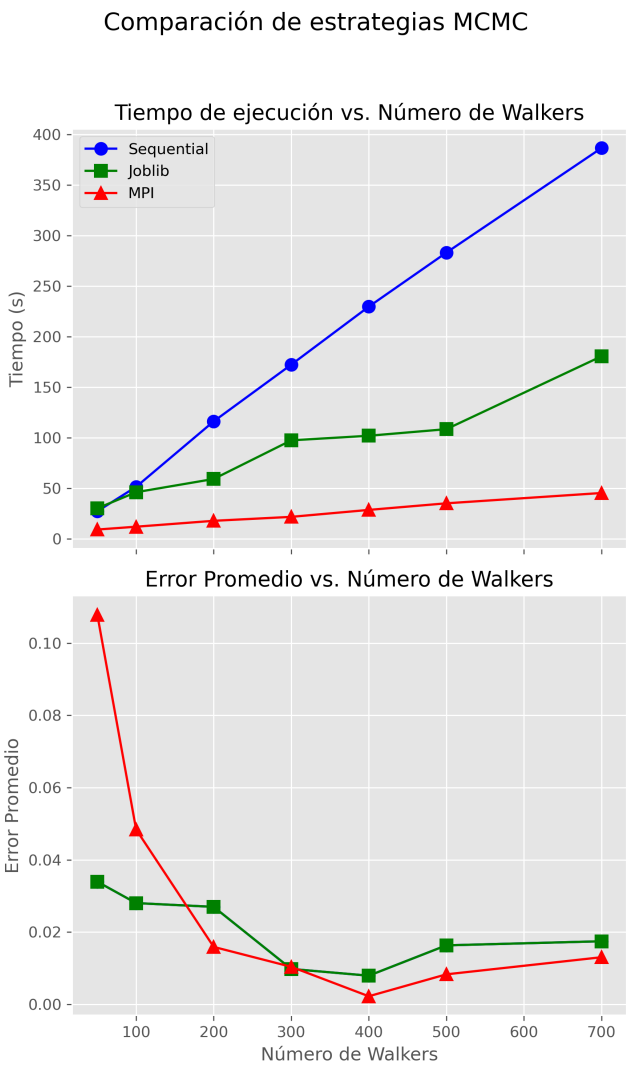


FIG. 1. Comparación entre las estrategias secuencial, Joblib y MPI. El panel superior muestra el tiempo de ejecución en función del número de walkers. El panel inferior muestra el error promedio, destacando cómo todas las estrategias convergen a errores bajos conforme se incrementa el número de walkers, aunque MPI presenta errores elevados en bajas cargas debido a la estrategia de particionamiento inicial.

Estos resultados indican que incrementar la carga de trabajo por proceso (más walkers con un número fijo de procesos) mejora claramente el rendimiento general hasta un cierto punto (aproximadamente entre 300 y 500 walkers), donde se alcanza una estabilización del rendimiento adicional. A partir de allí, aunque el tiempo sigue reduciéndose ligeramente, el beneficio marginal comienza a disminuir notablemente porque la proporción de tiempo dedicado exclusivamente al cómputo se estabiliza, habiendo amortizado ya gran parte del costo de coordinación

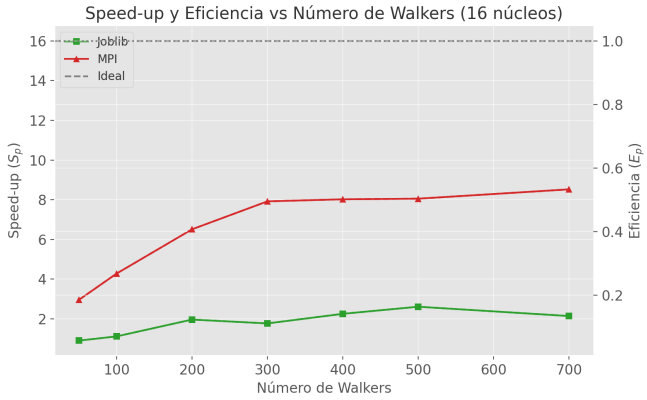


FIG. 2. Speed-up y eficiencia en función del número de walkers, manteniendo fijos 16 procesos.

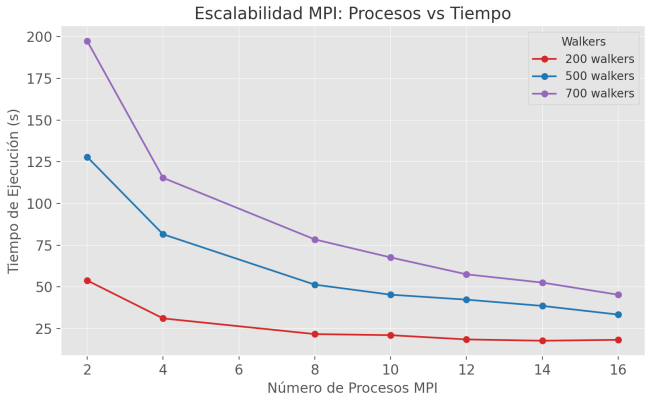


FIG. 3. Tiempo de ejecución de la estrategia MPI para tres diferentes cargas de trabajo (200, 500 y 700 walkers) en función del número de procesos utilizados.

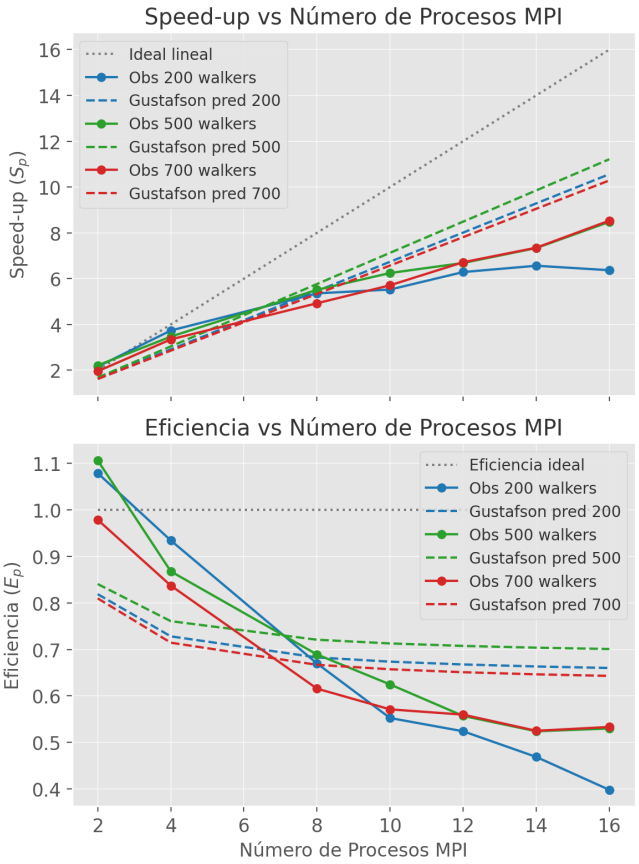


FIG. 4. Comparación entre los valores observados y los predichos por la Ley de Gustafson para speed-up (arriba) y eficiencia (abajo), en función del número de procesos MPI.

inicial. Esto indica que seguramente pueden lograrse tiempos similares con una menor cantidad de procesos antes de alcanzar esta estabilización.

En la FIG. 3, se evalúa el impacto del número de procesos MPI sobre el tiempo total de ejecución para tres cantidades diferentes de walkers (200, 500 y 700). En general, el tiempo se reduce rápidamente al aumentar de 2 a aproximadamente 12 procesos, indicando un aprovechamiento eficiente del paralelismo. Sin embargo, desde este punto en adelante, el tiempo de ejecución se estabiliza considerablemente, reflejando que la sobrecarga asociada a la comunicación y sincronización comienza a superar los beneficios del paralelismo. Así, existe un punto óptimo cercano a los 10–12 procesos, tras el cual añadir más procesos deja de ser eficiente, aunque este punto se desplaza ligeramente hacia arriba conforme aumenta el número de walkers debido a que la mayor carga de trabajo amortigua parcialmente los costos del paralelismo.

Finalmente, en la FIG. 4 se realiza una validación empírica de la Ley de Gustafson comparando los valores observados de speed-up y eficiencia con los valores predichos teóricamente. Los ajustes realizados muestran un excelente acuerdo con los datos experimentales para cantidades bajas de procesos (de 2 a 8 procesos), mientras que para valores mayores se observa una divergencia significativa entre la predicción y los resultados experimentales. Este desacople se debe principalmente a factores como latencias en comunicación, sobrecargas de sincronización y granularidad cada vez menor de las tareas individuales, que aumentan el costo relativo de gestión del paralelismo. Además, la fracción serial estimada ($\alpha$) es menor cuando se utilizan más walkers (por ejemplo, $\alpha$=0.20 para 500 walkers comparado con $\alpha$=0.30 para 200 walkers), reflejando una mejor escalabilidad del problema cuando la carga total es mayor. En conjunto, estos resultados permiten identificar claramente las condiciones en las que el paralelismo aporta beneficios significativos y aquellas en las que estos beneficios comienzan a disminuir, ofreciendo información valiosa para optimizar el balance entre número de walkers y procesos utilizados.

## V. CONCLUSIONES

El análisis comparativo llevado a cabo entre las estrategias secuencial, Joblib y MPI ha puesto de manifiesto importantes diferencias en cuanto al rendimiento computacional, aunque todas las implementaciones logran resultados estadísticamente similares a partir de un número suficiente de walkers (alrededor de 200). De estas, MPI resulta claramente superior, reduciendo el tiempo de ejecución hasta un ∼90% respecto a la estrategia secuencial y aventajando a Joblib en más del 50% en situaciones de alta carga computacional. Este beneficio es sostenible únicamente cuando cada proceso tiene una carga suficientemente alta como para compensar el tiempo adicional que implica la coordinación y comunicación entre procesos.

Al examinar detalladamente el comportamiento del speed-up y la eficiencia mediante las ecuaciones y , se observa un punto de retorno decreciente alrededor de 300–500 walkers. Por encima de este umbral, el incremento en rendimiento al aumentar la carga por proceso tiende a estabilizarse debido al balance entre el tiempo de cálculo efectivo y el esfuerzo adicional necesario para coordinar múltiples procesos.

El análisis de escalabilidad basado en la Ley de Gustafson refuerza estos hallazgos al mostrar que la fracción serial del algoritmo se reduce con cargas mayores, alcanzando así rendimientos cercanos al ideal para cantidades moderadas de procesos. No obstante, cuando se emplea un gran número de procesos, factores prácticos como la latencia en las comunicaciones y la necesidad de sincronización limitan severamente el speed-up efectivo.

En conclusión, para problemas del tipo Daisyworld en simulaciones MCMC, MPI ofrece una notable ventaja en tiempos de ejecución sin detrimento en la calidad estadística del muestreo. Es esencial, sin embargo, realizar un cuidadoso balance entre el número de walkers y procesos utilizados para maximizar los beneficios del paralelismo frente a sus costos asociados.


[1] J. E. Lovelock, *Gaia: A New Look at Life on Earth*, second edition ed., Oxford Landmark Science (Oxford University Press, Oxford, United Kingdom, 2016).

[2] A. J. Watson and J. E. Lovelock, Biological homeostasis of the global environment: The parable of Daisyworld, Tellus B **35B**, 284 (1983).

[3] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, Equation of State Calculations by Fast Computing Machines, The Journal of Chemical Physics **21**, 1087 (1953).

[4] W. K. Hastings, Monte Carlo sampling methods using Markov chains and their applications, Biometrika **57**, 97 (1970).

[5] D. Foreman-Mackey, D. W. Hogg, D. Lang, and J. Goodman, emcee: The mcmc hammer, PASP **125**, 306 (2013), 1202.3665.

[6] The joblib developers, joblib.

[7] L. Dalcin, R. Paz, and M. Storti, MPI for Python, Journal of Parallel and Distributed Computing **65**, 1108 (2005).

[8] L. Dalcin, R. Paz, M. Storti, and J. D'Elia, MPI for Python: performance improvements and MPI-2 extensions, Journal of Parallel and Distributed Computing **68**, 655 (2008).

[9] L. Dalcin, P. Kler, R. Paz, and A. Cosimo, Parallel Distributed Computing using Python, Advances in Water Resources **34**, 1124 (2011).

[10] L. Dalcin and Y.-L. L. Fang, mpi4py: Status Update After 12 Years of Development, Computing in Science & Engineering **23**, 47 (2021).

[11] M. Rogowski, S. Aseeri, D. Keyes, and L. Dalcin, mpi4py.futures: MPI-Based Asynchronous Task Execution for Python, IEEE Transactions on Parallel and Distributed Systems **34**, 611 (2023).

[12] G. M. Amdahl, Validity of the single processor approach to achieving large scale computing capabilities, AFIPS conference proceedings **30**, 483 (1967).

[13] M. D. Hill and M. R. Marty, Amdahl's law in the multicore era, Computer **41**, 33 (2008).

[14] J. L. Gustafson, Reevaluating amdahl's law, Communications of the ACM **31**, 532 (1988).