QSS parallel simulation of large scale models with set-based graph partitioning algorithm

Franco Sansone[12], Joaquin Fernandez[1] y Ernesto Kofman[12]

[1] CIFASIS-CONICET, Bv. 27 de Febrero 210 bis, Rosario, Santa Fe, Argentina
sansone@cifasis-conicet.gov.ar
[2] FCEIA, Universidad Nacional de Rosario, Av. Pellegrini 250, Rosario, Santa Fe, Argentina

Abstract. Quantized State System (QSS) algorithms are a family of asynchronous integration methods for Ordinary Differential Equations (ODEs) where each state variable evolves independently. Unlike classical integration methods that rely on discrete-time approximations, QSS produces a Discrete Event System approximation of the original ODEs. The evolution of large-scale models has been driven by the continuous growth of available computational resources and parallel simulation is the primary method used to leverage these capabilities. A key observation is that large-scale models are typically the result of repetitive structures: equations defined compactly within `for-loops` involving `arrays` of variables. To exploit this characteristic, a Set-Based Graph (SBG) partitioning algorithm was designed, where arrays of variables are represented as sets of vertices defined by intension, and the connections between them are represented as a pair of piecewise linear functions whose domain are the edges and whose codomain are the vertices. In this work we present an integration between the SBG partitioning algorithm and the parallel simulation engine for QSS models, the former takes the input in a compact way and the result is sent to the latter.

Keywords: Large Scale Models, Set-Based Graphs, Graph Partitioning

Resumen Los algoritmos QSS (Quantized State System) son una familia de métodos de integración asíncronos para Ecuaciones Diferenciales Ordinarias (EDOs), en los cuales cada variable de estado evoluciona de manera independiente. A diferencia de los métodos de integración clásicos, que se basan en aproximaciones de tiempo discreto, QSS produce una aproximación de sistemas de eventos discretos de las ODEs originales. La evolución de los modelos a gran escala ha sido impulsada por el crecimiento continuo de los recursos computacionales disponibles, y la simulación en paralelo es el método principal utilizado para aprovechar estas capacidades. Una observación esencial es que habitualmente los modelos de gran escala son el resultado de utilizar estructuras repetitivas, ecuaciones definidas de forma compacta en ciclos *for loop* que involucran arreglos de variables. Con el objetivo de explotar esta característica un

algoritmo de particionamiento de Grafos Basados en Conjuntos (SBG) fue diseñado, donde los arreglos de variables son representados como conjuntos de vértices definidos por intensión, y las conexiones entre ellos como un par de funciones lineales por partes cuyo dominio son las aristas y cuyo codominio, los vértices. En este artículo se presenta una integración entre el algoritmo de particionamiento SBG y el motor de simulación en paralelo para modelos QSS, donde el algortimo de particionamiento recibe la entrada en forma compacta y transfiere el resultado al motor de simulación.

Keywords: Modelos De Gran Escala, Set-Based Graphs, Partición De Grafos

1 Introducción

Los modelos de tiempo continuo se expresan habitualmente como un conjunto de Ecuaciones Diferenciales Ordinarias (EDOs), o se transforman en uno, y la forma usual de obtener aproximaciones a las soluciones de este conjunto de ecuaciones es mediante métodos de integración numérica. Los métodos de integración numérica QSS (Quantized State System) son una familia de algoritmos donde cada variable de estado evoluciona de manera independiente y como consecuencia se obtiene una aproximación mediante un sistema de eventos discretos del modelo original a diferencia de la aproximación de tiempo discreto que realizan los métodos clásicos de integración numérica. Esta característica hace que sean eficientes para simular sistemas ralos y con discontinuidades.

Este trabajo se enfoca en el entorno de simulación *QSS Solver* (Fernández & Kofman, 2014) que implementa un motor de simulación por eventos discretos autónomo donde los modelos se definen mediante un subconjunto del lenguaje de modelado orientado a objetos `Modelica` (Fritzson & Bunus, 2002). Asimismo, este entorno de simulación implementa la familia completa de métodos QSS junto con algoritmos desarrollados específicamente para su ejecución en paralelo (Fernández et al., 2017). Estos algoritmos, a diferencia de los enfoques tradicionales para la simulación en paralelo de sistemas de eventos discretos (Fujimoto, 2016), no respetan la restricción causal de eventos con el costo de introducir un error numérico adicional acotado que depende del error perimtido por los métodos de simulación QSS.

Esto resulta fundamental, dado que el crecimiento continuo de los recursos computacionales disponibles impulsa el desarrollo de modelos cada vez más grandes y complejos, siendo la simulación en paralelo el método principal para aprovechar estas prestaciones.

Una condición fundamental para que la simulación en paralelo sea eficiente es que el modelo sea dividido en tantas partes como procesadores haya disponibles, de manera tal que cada procesador simule un submodelo más pequeño. Más aún, dada la naturaleza asíncrona de los métodos QSS, minimizar la comunicación entre submodelos es esencial para reducir los tiempos de simulación. Este es un problema clásico de balance de carga que puede resolverse mediante

técnicas de particionamiento de grafos con restricciones, cuyo objetivo es dividir los vértices del grafo en partes aproximadamente iguales para distribuir la carga computacional y minimizar el costo de comunicación (Çatalyürek et al., 2023). Herramientas como Metis, KaHiP o Scotch resuelven este problema mediante heurísticas o algoritmos de aproximación, puesto que el particionamiento de grafos es un problema NP-Hard, pero su eficiencia disminuye y el uso de memoria crece a medida que aumenta el número de vértices y aristas. Cuanto mayor es la cantidad de ecuaciones de un modelo, mayor es el tamaño del grafo que lo representa, lo que afecta la efectividad de las herramientas mencionadas.

Una observación esencial es que, habitualmente, los modelos de gran escala resultan de estructuras repetitivas y ecuaciones definidas de forma compacta en ciclos `for` que involucran arreglos de variables. Con el objetivo de explotar esta característica, se diseñó el algoritmo de particionamiento KL-SBG, basado en Grafos Basados en Conjuntos (SBG) (Sansone et al., 2025; Zimmermann et al., 2019). En este enfoque, los arreglos de variables se representan como conjuntos de vértices definidos por intensión, mientras que las conexiones entre ellos se modelan como pares de funciones afines por partes. Una vez generada la partición, esta debe ser procesada por el motor de simulación. En este trabajo se presenta un prototipo donde se integra el algoritmo KL-SBG con el motor de simulación en paralelo QSS. Dado que la evaluación de calidad de las particiones generadas por el algoritmo KL-SBG ya han sido evaluadas en (Sansone et al., 2025), el obejtivo de este trabajo es evaluar resultados de simulación y comparar el rendiemiento con diferentes herramientas para generar particiones utilizando el entorno de simulación QSS Solver. Aún la partición obtenida es transferida en forma manual y debe ser previamente expandida para poder ser consumida por el motor de simulación.

2 Motivación

La motivación principal de este trabajo reside en la integración entorno del simulación QSS Solver y los algoritmos de particionamiento KL-SBG.

Tal como se ha señalado, la etapa de particionamiento constituye una fase crítica del preprocesamiento para la ejecución de simulaciones en paralelo. Si bien el entorno de simulación actual admite diversos algoritmos de particionado, estos presentan las limitaciones anteriormente descritas. Adicionalmente, una vez obtenida la partición, la misma debe ser procesada por el motor de simulación para determinar la comunicación entre los distintos procesos y esto conlleva un costo computacional alto.

Dicha integración es el primer paso para la implementación de estrategias de particionado dinámico durante la ejecución de la simulación. Esta posibilidad se ve favorecida por el hecho de que tanto el algorimo KL-SBG como QSS Solver comparten la premisa de representar la información de manera compacta (Fernández & Kofman, 2014; Kofman et al., 2021)

El Modelo 1.1 muestra un ejemplo introductorio que motiva este trabajo, diseñado para representar facilmente definiciones de variables y sus dependencias.

En este contexto, dada una ecuación $x_i = f(\mathbf{x})$ decimos que una variable x_i depdende de otra x_j y x_j es parte de la expresión $f(\mathbf{x})$.

Modelo 1.1: Ejemplo muy sencillo de Modelica

```
Real a[N],b[N],c[N/2],d[N/2];
...
for i in 1:N loop
  der(a[i])=R*b[i];
end for;
for i in 1:N/2 loop
  der(c[i])=K*d[i];
end for;
```

Las dependencias para las variables c y d con $N = 10$ son representadas en el grafo representado en la Figura 1.

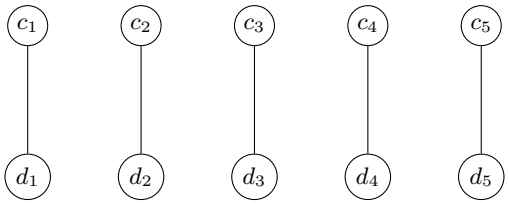

Figura 1: Grafo computacional para $N = 10$.

Se puede definir una representación compacta del grafo en la Figura 2 que toma en cuenta las los **bucles for** en la implementación.

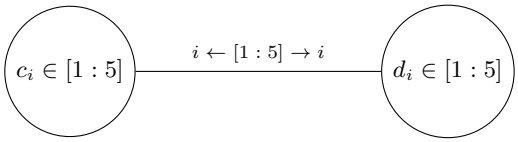

Figura 2: SBG computacional para $N = 10$.

En el ejemplo anterior podemos observar que el tamaño del grafo tradicional G depende del valor de N. Cuando más grande es N, mayor la cantidad de nodos y aristas. En la representación compacta, usamos un intervalo para cada **bucle for**. Si N aumenta, el grafo será practicamente el mismo, cambiando sólo los bordes de los intervalos y las aristas.

Buscamos explotar la naturaleza compacta de las implementaciones, dado que esta convergencia en la estructura de datos utilizada no solo facilita la integración, sino que optimiza significativamente el intercambio de información en

todas las fases: desde el preprocesamiento y la ejecución de la simulación hasta el post-procesamiento.

3 Desarrollo e implementación

En este trabajo pretendemos integrar el algoritmo KL-SBG (Sansone et al., 2025), un algoritmo que particiona Grafos basados en Conjuntos refinando una configuración inicial de vértices usando la heurísitica propuesta en (Kernighan & Lin, 1970), con el motor de simulación QSS.

Presentamos:

- la representación compacta de los cálculos y dependencias de un modelo, explotando estructuras repetitivas en la implementación del mismo,
- su óptimo particionamiento,
- la integración inicial entre el particionador y el motor de simulación.

En el directorio *algorithms/partitioner* del repositorio (Sansone et al., 2026), se encuentra la implementación del algoritmo **KL-SBG**. Si bien aún es un trabajo en proceso, es lo suficientemente maduro como para extraer algunos resultados sobre modelos reales, presentados en detalle en la sección siguiente. La función *sbg_partitioner::kl_sbg_imbalance_partitioner* es la encargada de optimizar una partición en k partes de un SBG.

Uno de los desafíos es optimizar la función *sbg_partitioner::update_diff* (implementación de su homónima en el algoritmo KL-SBG). La ganancia de intercambiar dos intervalos de vértices v_i y v_j, donde $|v_i| < |v_j|$, computaremos la ganancia de intercambiar v_i con los primeros $|v_i|$ vértices de v_j. En caso de que ese sea el intercambio elegido en una iteración, debemos computar la comunicación, tanto interna como externa, de los $|v_j| - |v_i|$ vértices restantes de $|v_j|$. Esto supone un costo alto y es lo que hoy no nos permite ser competitivos en eficiencia con respecto a otros algoritmos para grafos densos, puesto que nos vemos obligados a iterar sobre las partes de ambos mapas.

Por otro lado, también perdemos eficiencia cuando el número de partes crece. La clase *ICommunicationCost* guarda información sobre vértices y aristas para evitar repetir cómputos. Por el momento, muchos valores deben reiniciarse cuando una iteración de optimización es ejecutada. Estamos trabajando en optimizar este proceso, tanto desde el punto de vista teórico como en su implementación. La posibilidad de conocer el costo de manera global, similar al algoritmo de refinamiento *Global Kernighan-Lin Refinement* (Karypis & Kumar, 1998), pero siguiendo la misma idea de la modificación del cálculo de ganancia, utilizar conjuntos de aristas y operaciones de conjunto.

4 Resultados experimentales

En esta sección presentamos dos modelos con diferentes caraterísticas y mediremos los resultados obtenidos comparándolos con diferentes algoritmos de balance de carga (Metis, KaHiP y Scotch).

Para la comparación se utilizó una máquina con sistema operativo Linux (Ubuntu 24.04.4 LTS, desktop), procesador 12th Gen Intel(R) Core(TM) i5-12400 y 32 GB de RAM.

En este trabajo nos enfocamos en el rendimiento del entorno de simulación al utilizar las parciciones generadas por los diferentes algoritmos evaluados en términos de tiempo de cómputo necesario para cada una de las etapas, las métricas que utilizaremos son:

- **Tiempo de Particionado:** Es el tiempo que le toma al particionador en leer la entrada y construir el grafo de conexiones. El resultado que se exhibe es el resultado de 15 ejecuciones, descartando las primeras 5 por considerarlas *cold executions* y promediando las 10 restantes.
- **Tiempo de Simulación:** El tiempo de simulación del algoritmo QSS en paralelo.

4.1 Población de Aires Acondicionados

Este modelo, tomado de (Perfumo et al., 2012), estudia la dinámica de un conjunto de aires acondicionados (AAs) cuando siguen la misma temperatura de referencia. El i–ésimo AA es utilizado para controlar la temperatura de la i–ésima habitación, modelada por:

$$\dot{\theta}_i(t) = -\frac{1}{C_i \cdot R_i}[\theta_i(t) - \theta_a + R_i \cdot P_i \cdot m_i + w_i(t)] \tag{1}$$

donde R_i representa la resistencia térmica, C_i es la capacidad térmica, P_i representa el consumo de energía de la unidad de AA cuando está encendido, θ_a es la temperatura ambiente (común a todas las habitaciones), y $w_i(t)$ es una señal de ruido que representa perturbaciones térmicas.

La variable $m_i(t)$ representa el estado del i–ésimo AA que asume el valor 1 cuando está encendido y 0 cuando está apagado. Esta variable sigue la siguiente ley de control encendido–apagado con histéresis:

$$m_i(t^+) = \begin{cases} 0 & \text{if } \theta_i(t) \leq \theta_r^i(t) - 0{,}5 \text{ and } m_i(t) = 1 \\ 1 & \text{if } \theta_i(t) \leq \theta_r^i(t) + 0{,}5 \text{ and } m_i(t) = 0 \\ m_i(t) & \text{otherwise} \end{cases} \tag{2}$$

donde $\theta_r^i(t)$ es la temperatura de referencia, que tiene la siguiente forma:

$$\theta_r^i(t) = \begin{cases} 20 & \text{if } 0 \leq t < 1000 \\ 20{,}5 & \text{if } 1000 \leq t < 2000 \\ 20 & \text{if } 2000 \leq t \leq 3000 \end{cases} \tag{3}$$

Finalmente, las perturbaciones térmicas $w(t)$ son actualizadas cada minuto, asumiendo valores seudo-aleatorios en el rango $(-1, 1)$.

El código completo del modelo Modelica se encuentra en (Sansone et al., 2026), en el directorio *test/partitioner/sim_ models/air_ conditioners/airconds.mo*.

Cabe mencionar, que en este modelo cada AA está modelado por una ecuación diferencial (líneas 3-5) y tres eventos: uno asociado a la ley de control con histéresis (líneas 8-14), otro correspondiente a la evolución de la temperatura de referencia (líneas 15-24) y el último asociado a la actualización de la señal de ruido (líneas 25-30).

Debido a esto, la dinámica de cada AA es independiente de la evolución de los restantes AAs y como consecuencia, se puede encontrar una partición ideal en la cual no existe comunicación.

En la Tabla 1 se muestran los resultados obtenidos para obtener particiones de 4, 8 y 12 partes variando la cantidad de AAs en el modelo.

Cabe mencionar que para KL_SBG, cuando se genera la partición inicial, la estrategia de particionamiento distributiva acomoda los nodos de modo tal que solo tengan comunicación interna y el algoritmo de optimización no itera.

En la Tabla 1 muestra el tiempo de simulación y de particionado correspondientes a simular el modelo para distintos valores de N y número de partes utilizando el motor de simulación *QSS Solver*.

A modo de ejemplo, la partición generada para $N = 10000$ es la siguiente:

```
{"partitions":[
    {"nodes":[[[0,2499]],[[10000,12499]],[[20000,
        22499]],[[30000,32499]]]},
    {"nodes":[[[2500,4999]],[[12500,14999]],[[22500,
        24999]],[[32500,34999]]]},
    {"nodes":[[[5000,7499]],[[15000,17499]],[[25000,
        27499]],[[35000,37499]]]},
    {"nodes":[[[7500,9999]],[[17500,19999]],[[27500,
        29999]],[[37500,39999]]]}
    ]
}
```

Donde cada elemento `nodes` indica una partición en forma de lista de intervalos. Los intervalos son representados con arreglos de dos elementos que indican principio y final del mismo. Teniendo en cuenta que:

- Los nodos $[0 : 9999]$ se corresponden con las líneas 3-5 del modelo.
- Los nodos $[10000 : 19999]$ se corresponden con las líneas 8-14 del modelo.
- Los nodos $[20000 : 29999]$ se corresponden con las líneas 15-24 del modelo.
- Los nodos $[30000 : 39999]$ se corresponden con las líneas 25-30 del modelo.

Podemos corroborar que la partición generada respeta la distribución ideal mencionada anteriormente.

Mas allá de que este caso en particular no hay comunicación entre los procesos, se pueden ver diferencias significativas entre los tiempos de particionado. Esto se debe a que la lectura de los grafos computacionales SBG es independiente del tamaño de N. Por limitaciones de implementación, el algoritmo escala mal con el número de particiones. Es esperable que Kahip y KL_SBG sean más eficientes al simular, ya que generan bloques contiguos en cada partición (la

Tabla 1: Resultados para el modelo AA (valores en milisegundos).

Número de partes	Valor de N	Model	Tiempo de Particionado	Tiempo de Simulación
4 Partes	10000	KL_SBG	1	8859
		Metis	103	10887
		Scotch	25	7513
		Kahip	117	7106
	100000	KL_SBG	1	106914
		Metis	1220	93605
		Scotch	137	63440
		Kahip	174	104203
8 Partes	10000	KL_SBG	1	10069
		Metis	12	10814
		Scotch	3	7114
		Kahip	1	7168
	100000	KL_SBG	1	193530
		Metis	1370	188250
		Scotch	134	186495
		Kahip	182	172195
12 Partes	10000	KL_SBG	137	17750
		Metis	12	12996
		Scotch	3	10944
		Kahip	12	10282
	100000	KL_SBG	133	110085
		Metis	1350	110384
		Scotch	16	116258
		Kahip	185	161872

contigüidad de las partes afecta el rendimiento de la simulación) y tanto Metis como Scotch generan particiones aleatoreas. Estamos estudiando por qué los resultados no se condicen a lo esperado.

Las simulaciones puede recrearse corriendo el comando *bash run_ air_ conditioners.bash* en el directorio *test/partitioner/sim_ models/air_ conditioners* del repositiorio (Sansone et al., 2026). Por la salida estándar se exhiben los tiempos de simulación total y por LP.

4.2 Modelo Advección–Difusión–Reacción 2D

El siguiente modelo, presentado en (Bergero et al., 2016) representa una ecuación de Advección-Difusión–Reacción (ADR) en dos dimensiones. Esta ecuación puede describir, por ejemplo, un río transportando una substancia que experimenta una reacción química.

Las Ecuaciones 4 se obtienen luego de aplicar el Método de Líneas a las ecuaciones en Derivadas Parciales (PDE) originales.

El código completo del modelo Modelica se encuentra en (Sansone et al., 2026), en el directorio *test/partitioner/sim_ models/advection_ 2D/advection2D.mo*.

$$
\begin{aligned}
\frac{\partial u_{1,1}}{\partial t} &= -a_x \cdot \frac{u_{1,1}}{\Delta x} + a_y \cdot \frac{u_{1,1}}{\Delta y} + r \cdot (u_{1,1}^2 - u_{1,1}^3) \\
\frac{\partial u_{i,1}}{\partial t} &= -a_x \cdot \frac{u_{i,1}}{\Delta x} + a_y \cdot \frac{(u_{i,1} - u_{i-1,1})}{\Delta y} + r \cdot (u_{i,1}^2 - u_{i,1}^3) \quad i = 2 \ldots, N \\
\frac{\partial u_{1,j}}{\partial t} &= -a_x \cdot \frac{(u_{1,j} - u_{1,j-1})}{\Delta x} + a_y \cdot \frac{u_{1,j}}{\Delta y} + r \cdot (u_{1,j}^2 - u_{1,j}^3) \quad j = 2, \ldots, N \\
\frac{\partial u_{i,j}}{\partial t} &= -a_x \cdot \frac{(u_{i,j} - u_{i,j-1})}{\Delta x} + a_y \cdot \frac{(u_{i,j} - u_{i-1,})}{\Delta y} + r \cdot (u_{i,j}^2 - u_{i,j}^3) \\
& \quad i = 2, \ldots, N \quad j = 2, \ldots, N
\end{aligned}
\tag{4}
$$

donde N es el número de puntos de la grilla, consideramos que los parámetros a_x, a_y, r, Δx y Δy tienen como valor inicial 1 y condiciones iniciales $u_{1,1} = 1$ y $u_{i,j} = 0$ para $i = 2, \ldots, N \quad j = 2, \ldots, N$.

En la Tabla 2 se muestran los resultados modelos de diferente tamaño generando particiones de 4, 8 y 12 partes. Podemos observar que el tiempo de particionado para el algoritmo KL-SBG se mantiene en el mismo orden al cambiar el tamaño del problema mientra que el resto de las herrmientas evaluadas crecen al menos de manera lineal. Por otro lado los tiempos de simulación para modelos pequeños se mantienen en el mismo orden para todos los algoritmos mientras que para modelos de tamaño 1000 los tiempos de ejecución dependen de la dinámica del sistema que afecta la actividad de los procesos, i.e. existen procesos que están inactivos durante toda la simulación.

La actidavidad de los procesos se puede ver en las Figuras 3 y 4 donde podemos ver como la dinámica del sistema afecta el balance de carga estático generado por los diferentes algoritmos. Cabe mencionar que la actividad en este modelo se vuelve estable entre los diferentes procesos al extender el tiempo final de simulación. Sin embargo, este ejemplo ilustra un modelo donde recalcular el balance de carga de los procesos de manera dinámica puede incrementar los tiempos de simulación.

Las simulaciones puede recrearse corriendo el comando *bash run_ advection.bash* en el directorio *test/partitioner/sim_ models/advection_ 2D* del repositiorio (Sansone et al., 2026). Por la salida estándar se exhiben los tiempos de simulación total y por LP.

5 Conclusiones

En este trabajo presentamos resultados obtenidos en dos modelos de gran escala para el algoritmo KL-SBG comparando su eficiencia con respecto con herramientas de particionamiento de grafos de vanguardia, evaluando tanto los tiempos ejecución para obtener las particiones como así también los los tiempos de simulación del algoritmo de en paralelo QSS.

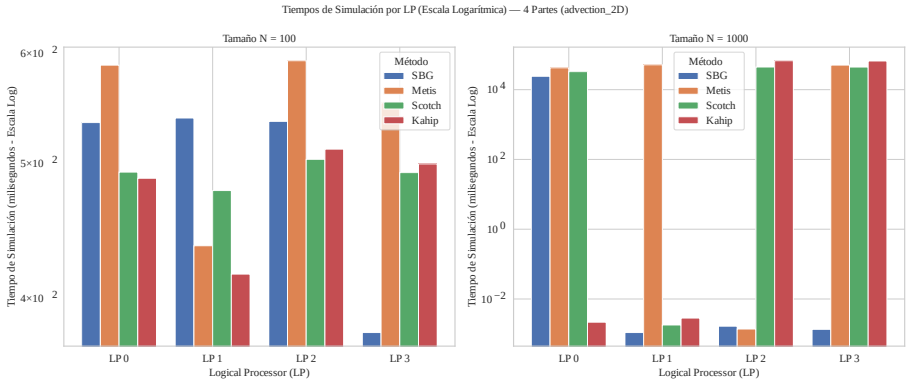

Figura 3: Tiempo de simulación por LP para el modelo *ADR 2D* con 4 procesadores lógicos.

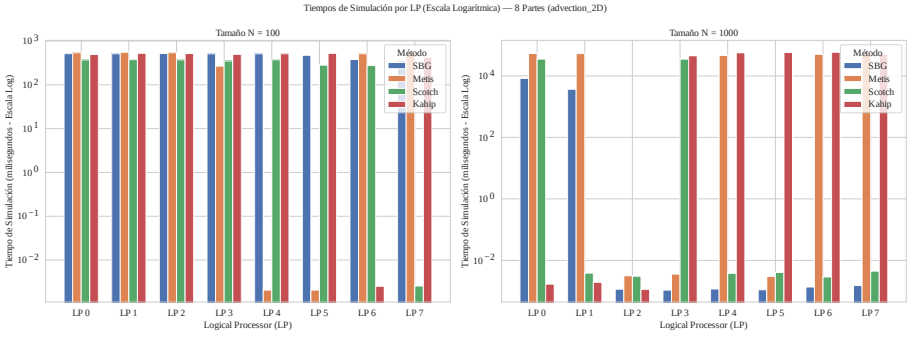

Figura 4: Tiempo de simulación por LP para el modelo *ADR 2D* con 8 procesadores lógicos.

Tabla 2: Resultados consolidados unificados para el modelo *ADR 2D* agrupados por número de particiones (valores en milisegundos)

Número de partes	Valor de N	Método	Tiempo de particionado	Tiempo de simulación
4 Partes	100	KL_SBG	3	544
		Metis	4	600
		Scotch	11	508
		Kahip	184	518
	1000	KL_SBG	4	24193
		Metis	285	51676
		Scotch	139	4464
		Kahip	7857	67568
8 Partes	100	KL_SBG	22	525
		Metis	4	560
		Scotch	25	380
		Kahip	262	529
	1000	KL_SBG	10	8409
		Metis	332	53725
		Scotch	166	35254
		Kahip	9430	58755
12 Partes	100	KL_SBG	94	5255
		Metis	6	10625
		Scotch	27	4626
		Kahip	365	904
	1000	KL_SBG	89	43584
		Metis	313	29487
		Scotch	204	37350
		Kahip	12570	67203

Los resultados de simulación obtenidos para las particiones generadas con el algoritmo KL-SBG comparables con las de herramientas de particionamiento evaluadas. Estos resultados permiten avanzar en la integración del algoritmo KL-SBG con el entorno de simulación QSS Solver y aprovechar la representación compacta que ofrecen ambas implemtaciones.

Una ventaja fundamental de esta representación es que el consumo de recursos es independiente del tamaño del problema lo cuál permite ejecutar el algoritmo de particionado durante la simulación y de esta manera actualizar el balance de carga de manera dinámica, lo representa una ventaja en problemas como el descripto en 4.2 donde la actividad de los procesos involucrados en la simulación no se puede detectar de manera estática.

Por lo tanto, como trabajos futuros nos planteamos:

– la transferencia de información entre el particionador y el motor de simulación en forma compacta, sin necesidad de expandir,

– la simulación del modelo, leyendo la información mencionada en forma compacta,
– el particionamiento dinámico del mismo, para acelerar simulaciones en paralelo.

Referencias

Bergero, F., Fernández, J., Kofman, E., & Portapila, M. (2016). Time Discretization versus State Quantization in the Simulation of a 1D Advection-Diffusion-Reaction Equation. *Simulation: Transactions of the Society for Modeling and Simulation International*, *92*(1), 47-61.

Çatalyürek, Ü., Devine, K., Faraj, M., Gottesbüren, L., Heuer, T., Meyerhenke, H., Sanders, P., Schlag, S., Schulz, C., Seemaier, D., et al. (2023). More recent advances in (hyper) graph partitioning. *ACM Computing Surveys*, *55*(12), 1-38.

Fernández, J., & Kofman, E. (2014). A stand-alone quantized state system solver for continuous system simulation. *Simulation*, *90*(7), 782-799.

Fernández, J., Kofman, E., & Bergero, F. (2017). A parallel quantized state system solver for ODEs. *Journal of Parallel and Distributed Computing*, *106*, 14-30.

Fritzson, P., & Bunus, P. (2002). Modelica-a general object-oriented language for continuous and discrete-event system modeling and simulation. *Proceedings 35th Annual Simulation Symposium. SS 2002*, 365-380.

Fujimoto, R. M. (2016). Research challenges in parallel and distributed simulation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, *26*(4), 1-29.

Karypis, G., & Kumar, V. (1998). Multilevelk-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed computing*, *48*(1), 96-129.

Kernighan, B. W., & Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, *49*(2), 291-307.

Kofman, E., Fernández, J., & Marzorati, D. (2021). Compact sparse symbolic Jacobian computation in large systems of ODEs. *Applied Mathematics and Computation*, *403*, 126181.

Perfumo, C., Kofman, E., Braslavsky, J., & Ward, J. (2012). Load Management: Model-Based Control of Aggregate Power for Populations of Thermostatically Controlled Loads. *Energy Conversion and Management*, *55*, 36-48.

Sansone, F., Fernandez, J., & Kofman, E. (2025). Partición de modelos de gran escala para simulación en paralelo de sistemas de eventos discretos. *JAIIO, Jornadas Argentinas de Informática*, *11*(15), 169-182.

Sansone, F., Fernández, J., & Kofman, E. (2026). *SBG Partitioner: a load balancing algorithm to simulate discrete event systems in parallel using set-based graphs.* (Ver. 55JAIIO). CIFASIS. https://github.com/CIFASIS/sb-graph/releases/tag/55JAIIO

Zimmermann, P., Fernández, J., & Kofman, E. (2019). Set-based graph methods for fast equation sorting in large dae systems. *Proceedings of the 9th International Workshop on Equation-based Object-oriented Modeling Languages and Tools*, 45-54.