

Partición De Modelos de Gran Escala Para Simulación En Paralelo De Sistemas De Eventos Discretos

Franco Sansone¹, Joaquin Fernandez¹ y Ernesto Kofman^{1,2}

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

Resumen Los algoritmos de simulación en paralelo de sistemas de eventos discretos utilizan un modelo dividido en tantas partes como procesadores requeridos de manera que cada procesador simula un sub-modelo más pequeño. Para que esta estrategia permita acelerar eficientemente las simulaciones se requieren dos condiciones: que el costo asociado a simular cada sub-modelo sea similar y que la comunicación necesaria entre los sub-modelos sea la mínima posible. Estas condiciones conducen a que la partición del modelo debe realizarse resolviendo un problema de balance de carga, lo que habitualmente se resuelve mediante un problema equivalente de teoría de grafos: como partir un grafo en un número dado de sub-grafos de manera que los sub-grafos tengan un número similar de vértices y que a la vez haya un número mínimo de aristas entre distintos subgrafos. De esta manera las simulaciones pueden avanzar en forma paralela y la sincronización sólo se hace necesaria cuando ocurre un cambio en un sub-modelo que afecta el comportamiento de otro sub-modelo. Una observación esencial es que habitualmente los modelos de gran escala son el resultado de utilizar estructuras repetitivas regulares (ecuaciones definidas de forma compacta en ciclos for loop que involucran arreglos de variables). Con el objetivo de explotar esta característica al representar un modelo mediante un grafo, se propuso recientemente el concepto de Grafos Basados en Conjuntos (SBG por Set Based Graphs). Los SBGs son grafos en los cuales los vértices y aristas están agrupados en distintos conjuntos que se pueden representar de manera compacta por comprensión. De esa forma, al cambiar el tamaño de los arreglos de un modelo sin cambiar la estructura del mismo la representación del SBG asociado se mantiene idéntica. En este trabajo presentamos una adaptación del algoritmo clásico de Kernighan-Lin que utilizando la representación SBG permite generar particiones con un costo computacional independiente del tamaño de los arreglos de variables definidos en el modelo original.

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

Large Scale Model Partitioning For Discrete Event Systems Parallel Simulation

Abstract. Parallel simulation algorithms for discrete-event systems use a model divided into as many parts as required processors, so that each processor simulates a smaller sub-model. For this strategy to efficiently accelerate simulations, two conditions are required: the cost associated with simulating each sub-model must be similar and the communication required between the sub-models should be as minimal as possible. These conditions lead to the model being partitioned by solving a load-balancing problem, which is usually solved using an equivalent graph-theoretic problem: how to split a graph into a given number of sub-graphs such that the sub-graphs have a similar number of vertices and at the same time there is a minimum number of edges between different sub-graphs. In this way, simulations can advance in parallel, and synchronization only becomes necessary when a change occurs in one sub-model that affects the behavior of another sub-model. An essential observation is that large-scale models typically result from the use of regular repetitive structures (equations compactly defined in for loops involving arrays of variables). To exploit this characteristic when representing a model using a graph, the concept of Set-Based Graphs (SBGs) was recently developed. SBGs are graphs in which vertices and edges are grouped into distinct sets that can be represented compactly by comprehension. Thus, when the size of the arrays in a model changes without changing its structure, the representation of the associated SBG remains identical. In this work, we present an adaptation of the classic Kernighan-Lin algorithm that, using the SBG representation, allows for the generation of partitions with a computational cost independent of the size of the arrays of variables defined in the original model.

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

1 Introducción

Casi todas las ramas de la ciencia y de la técnica utilizan modelos matemáticos, sobre los que se realizan simulaciones que permiten analizar el comportamiento de los sistemas ante distintos escenarios. En diferentes casos prácticos los modelos tienen gran tamaño y, para acelerar los cálculos que requiere la simulación, se suelen utilizar algoritmos que funcionan en paralelo sobre muchos procesadores, como se describe en Korch y Rauber, 2006; Rauber y Rüniger, 2013, 2021. Esta paralelización puede realizarse en las etapas de cálculo del propio algoritmo (lo que generalmente tiene sentido al utilizar pocos procesadores) o a nivel del modelo, partiendo al mismo en sub-modelos más pequeños. Esta última es la opción más conveniente para aprovechar la disponibilidad de muchos procesadores en las arquitecturas modernas de hardware.

Para que esta estrategia permita acelerar eficientemente las simulaciones se requieren dos condiciones: que el costo asociado a simular cada sub-modelo sea similar y que la comunicación necesaria entre los sub-modelos sea la mínima posible. Estas condiciones conducen a que la partición del modelo debe realizarse resolviendo un problema de *balance de carga*, lo que habitualmente se resuelve

mediante un problema equivalente de teoría de grafos: como partir un grafo en un dado número de sub-grafos de manera que los sub-grafos tengan un número similar de vértices y que a la vez haya un número mínimo de aristas entre distintos subgrafos.

La resolución del problema de balance de carga en grafos de gran tamaño es un problema para el cual existen diversas herramientas y algoritmos disponibles Çatalyürek et al., 2023 entre los cuales podemos destacar algoritmos clásicos como el de Kernighan y Lin, 1970 o el de Fiduccia y Mattheyses, 1988, y herramientas tales como METIS (Karypis y Kumar, 1997), SCOTCH (Pellegrini, 2012) o KaHIP (Sanders y Schulz, 2013) entre otras. Estos algoritmos en general arrancan con una partición inicial balanceada del grafo y luego van usando heurísticas para conmutar vértices entre particiones tratando de reducir el número de aristas que unen vértices de distintos sub-grafos. Las implementaciones de los algoritmos son muy eficientes y alcanzan soluciones sub-óptimas (el problema óptimo es NP-completo) con costos computacionales hasta sub-lineales en algunos casos con el tamaño del grafo. De todas maneras, estos costos hacen que eventualmente se alcance un límite de tamaño para los modelos por encima del cual el tiempo de ejecución y/o el uso de memoria se vuelven inaceptables.

La forma usual con la que estos modelos son representados es mediante el uso de un lenguaje de alto nivel, por ejemplo, Modelica (ver Fritzson y Engelson, 1998) que es un lenguaje abierto y orientado a objetos para el modelado de sistemas continuos y discretos. Esta característica permite que los usuarios definan modelos complejos instanciando y conectando componentes más simples. Si bien el lenguaje es textual, las herramientas de modelado y simulación basadas en Modelica permiten componer los modelos de manera gráfica.

Como mencionamos anteriormente, los modelos de gran escala suelen ser el resultado de utilizar estructuras repetitivas regulares (ecuaciones definidas de forma compacta en ciclos `for loop` que involucran arreglos de variables). Con el objetivo de explotar esta característica al representar un modelo mediante un grafo, se desarrolló recientemente el concepto de *Grafos Basados en Conjuntos* (SBG por *Set Based Graphs*), presentado en Kofman et al., 2021; Marzorati et al., 2022; Zimmermann et al., 2019. Los SBGs son grafos en los cuales los vértices y aristas están agrupados en distintos conjuntos que se pueden representar de manera compacta por comprensión. De esa forma, al cambiar el tamaño de los arreglos de un modelo sin cambiar la estructura del mismo la representación del SBG asociado se mantiene idéntica (sólo cambian los parámetros que indican el tamaño de cada conjunto).

En este trabajo presentamos una adaptación del algoritmo clásico de Kernighan-Lin que, utilizando la representación SBG, permite generar particiones con un costo computacional independiente del tamaño de los arreglos de variables definidos en el modelo original. El algoritmo fue implementado en C++ en un repositorio de código abierto *SBG-Partitioner* Sansone et al., 2025 y presentamos los resultados obtenidos a partir de un modelo de Advección-Difusión-Reacción y su comparación con la herramienta METIS.

2 Balance de carga como problema de particionamiento de grafos

Podemos describir el problema de balance de carga formalmente de la siguiente manera:

Queremos elegir una partición de nodos \mathcal{C} en P particiones disjuntas $\mathcal{V} = (\mathcal{C}_1 \cup \dots \cup \mathcal{C}_p)$ tales que:

- La suma del peso de los nodos sea similar (particionamiento equilibrado).
- Se minimiza la suma del peso de todas las aristas que conectan las diferentes particiones.

Así, denotamos como costo la comunicación entre clusters (mejor conocido en la bibliografía como *edge-cut*).

$$\text{cost}(\mathcal{C}) = \text{edgeCut}(\mathcal{C}) = \sum_{\mathcal{C}_i, \mathcal{C}_j} \{w(u, v) | u \in \mathcal{C}_i, v \in \mathcal{C}_j, i \neq j\}, \quad (1)$$

El problema anterior de búsqueda de una partición balanceada de grafos se puede mostrar que es un problema NP-completo (ver Andreev y Räcke, 2004). Una solución es conmutar nodos iterativamente entre particiones para minimizar el *edgeCut*.

Habiendo formulado el problema, describiremos en la siguiente sección la construcción del grafo computacional a partir de un modelo dado.

2.1 Grafo Computacional

Los cálculos necesarios para la simulación de un modelo se pueden representar de forma natural mediante un grafo en donde los vértices representan unidades computacionales o tareas y las aristas representan dependencias entre las mismas. En líneas generales, un grafo computacional puede modelarse como un grafo, \mathcal{G} , con pesos definidos en sus nodos y en sus aristas.

Más específicamente, $\mathcal{G} = (\mathcal{V}, \mathcal{E}, l, w)$, donde:

- \mathcal{V} el conjunto de nodos/las unidades de computación más pequeñas,
- $\mathcal{E} : \mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ el conjunto de aristas/dependencias entre unidades,
- $l : \mathcal{V} \rightarrow \mathcal{R}^+$ asigna un peso computacional a cada nodo,
- $w : \mathcal{E} \rightarrow \mathcal{R}^+$ asigna un peso comunicacional a cada arista.

Las aristas del Grafo Computacional (en el conjunto \mathcal{E}), denotan dependencias entre variables que son calculadas en nodos y necesarias en una expresión contenida en otro nodo. Más específicamente, una arista $e = (i, j)$ que conecta los nodos u_i y u_j , significa que una variable (de estado o discreta) computada en el nodo u_i es contenida en una expresión (ecuación o condición de cruce cero) del nodo u_j o viceversa.

En cuanto a los pesos asignados a las aristas $w : \mathcal{E} \rightarrow \mathcal{R}^+$, relacionados al costo en la comunicación, en este trabajo son ingresados manualmente por el usuario, al igual que los pesos asignados a los nodos en la función $l : \mathcal{V} \rightarrow \mathcal{R}^+$.

A modo de ejemplo, en la Figura 1 podemos ver un modelo Modelica y su grafo computacional correspondiente de acuerdo a las reglas definidas anteriormente.

```

model Test
constant Integer N = 10;
Real u[N];
...
der(u[1])=-a*(u[1]-1);
for i in 2:N loop
  der(u[i])=-a*(u[i]-u[i-1]);
end for;
end Test;

```

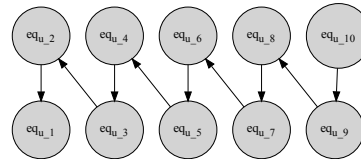


Figura 1: Grafo computacional obtenido a partir de un modelo Modelica

En la siguiente sección, veremos cómo aprovechar las estructuras cíclicas presentes en este tipo de modelos utilizando SBG para representar los grafos de manera compacta.

Para el desarrollo de este trabajo, hemos utilizado modelos matemáticos de tiempo continuo, donde las unidades de cálculo son las derivadas y las funciones de cruce cero. Los mismos son implementados en Modelica Fritzson y Engelson, 1998 por ser el lenguaje de modelado más ampliamente aceptado. Cabe aclarar que este algoritmo podría aplicarse al particionado de cualquier modelo siempre y cuando el grafo computacional lo abstraiga apropiadamente y, como se mencionó en la Sección 1, se sacaría provecho siempre y cuando se puedan capturar estructuras cíclicas en la implementación del mismo.

3 Grafos Basados en Conjuntos

Los *Grafos Basados en Conjuntos* (SBG por sus siglas en inglés) Zimmermann et al., 2019 se definen como sigue:

Definición 1 (Set-Based Graph) Dado un grafo $G = (V, E)$, un SBG asociado al mismo se compone de:

- Una partición $V = \{V^1, \dots, V^p\}$ de V a cuyos elementos llamaremos Vértices-conjunto.
- Una partición $E = \{E^{i_1, j_1}, \dots, E^{i_k, j_k}\}$ de E a cuyos elementos llamaremos Aristas-conjunto, donde $E^{i, j} = \{v_i, v_j\} : v_i \in V_i \wedge v_j \in V_j$.

Una manera de representar de forma compacta los SBGs dirigidos es la siguiente:

- Todos los vértices y aristas son etiquetados con una tupla n -dimensional de números naturales.

- Dos mapas que representan conexiones, $\text{map}_B, \text{map}_F : \mathbb{N}^n \mapsto \mathbb{N}^n$ tales que para cada arista $e = (u, v)$, $\text{map}_B(e) = u$ y $\text{map}_D(e) = v$.
- Un mapa que represente vértices-conjunto, $V_{\text{map}} : \mathbb{N}^n \mapsto \mathbb{N}$, $V_{\text{map}}(v) = i$ si $v \in V^i$.
- Un mapa que represente aristas-conjunto, $E_{\text{map}} : \mathbb{N}^n \mapsto \mathbb{N}^2$, $E_{\text{map}}(e) = (i, j)$ si $e \in E^{i,j}$.

Cuando los conjuntos y mapas se expresan de manera compacta, la representación completa de un SBG se torna independiente del número de elementos que tiene cada conjunto.

Esta propiedad se puede ver en la Figura 2 que muestra la representación SBG del modelo Modelica definido en la Figura 1.

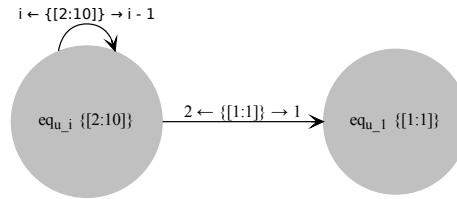


Figura 2: Grafo computacional SBG correspondiente a la Figura 1

Veremos ahora, como utilizando esta representación se puede adaptar el algoritmo KL conservando la misma propiedad.

4 Algoritmo de Partición de Grafos KL-SBG

La idea de este trabajo es adaptar el algoritmo KL utilizando la representación SBG para aprovechar la presencia de estructuras cíclicas. Comenzaremos por describir brevemente la idea del algoritmo clásico de KL.

4.1 Algoritmo Clásico de Kernighan–Lin

Sea S un conjunto de $2n$ puntos, con una matriz de costo asociada $C = (c_{ij})$, $i, j = 1, \dots, 2n$. Queremos partir S en dos conjuntos A y B , cada una con n puntos, tales que la comunicación entre ellos se minimice.

Para esto, suponiendo que (A^*, B^*) es una partición con costo mínimo y dadas A y B dos particiones iniciales arbitrarias. Se buscan subconjuntos $X \subset A$, $Y \subset B$ con $|X| = |Y| \leq n/2$ tales que intercambiando X e Y se obtiene A^* y B^* . Cuando no se encuentra una mejora posible, la partición resultante A', B'

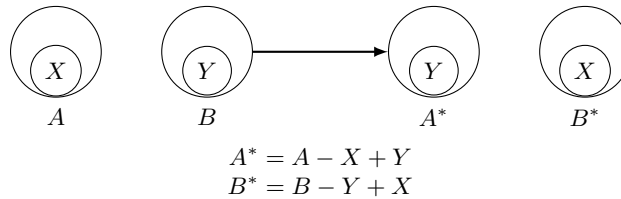


Figura 3: Descripción gráfica de la fase de optimización

luego de realizar el intercambio es localmente mínima, como podemos ver en la Figura 3.

Para identificar X e Y a partir de A y B sin considerar todas las posibles opciones, podemos definir para cada $a \in A$, un *costo externo* $E_a = \sum_{y \in B} c_{ay}$ y un *costo interno* $I_a = \sum_{x \in A} c_{ax}$.

Análogamente, definimos E_b, I_b para cada $b \in B$. Luego podemos definir la diferencia D como: $D_x = E_x - I_x$ para todo $x \in S$.

Considerando cualquier $a \in A, b \in B$. Si a y b son intercambiados, la *ganancia* (esto es, la reducción del costo) es precisamente $D_a + D_b - 2c_{ab}$.

Luego, elegiremos $a_i \in A, b_j \in B$ tal que la ganancia $g_{a_i b_j} = D_{a_i} + D_{b_j} - 2c_{a_i b_j}$ es máxima. Los nodos a_i y b_j se descartan y se reinicia el proceso para los conjuntos $A' = A \setminus \{a_i\}$ y $B' = B \setminus \{b_j\}$. Se repite hasta que los nodos se hayan agotado.

Si $X = a_1, a_2, \dots, a_k, Y = b_1, b_2, \dots, b_k$, entonces la disminución en la comunicación si los conjuntos X y Y son intercambiados es $g_1 + g_2 + \dots + g_k$, donde g_i es la ganancia de intercambiar a_i y b_i con $i \in \{1, \dots, k\}$. Se elige el k tal que $\sum_{i=1}^k g_i = G$ es máxima. Si $G > 0$, entonces existe una reducción en la comunicación. Se realiza el intercambio y el proceso se repite con las nuevas particiones. Si $G = 0$, hemos alcanzado un mínimo local y la fase de optimización se da por terminada. Que este mínimo local sea mínimo global depende de las particiones iniciales elegidas.

Basados en esta idea, presentaremos presentaremos las modificaciones introducidas para utilizar grafos SBG.

4.2 Algoritmo KL-SBG

El Algoritmo 1 calcula la comunicación del conjunto de nodos A con respecto a B , donde A es un intervalo de nodos y B es la partición. Las aristas que comunican nodos de A con nodos de A serán consideradas comunicación interna (IC), mientras que las que comunican nodos de A con nodos de B , comunicación externa (EC).

Algorithm 1 Cómputo del Costo Externo e Interno

- 1: **function** $EC_IC(A, B, map_B, map_D)$
- 2: $D \leftarrow map_B.preImage(A) \cup map_D.preImage(A)$ ▷ En este paso obtenemos las aristas que en uno de sus extremos tiene elementos de A

```

3:  $I \leftarrow \text{map}_D.\text{image}(D) \cup \text{map}_B.\text{image}(D)$  ▷ Obtenemos los nodos con los cuales se conectan
dichas aristas.
4:  $IC' \leftarrow (A \cap I)$  ▷ Los nodos que se encuentran en  $A$  son comunicación interna.
5:  $IC \leftarrow \text{map}_D.\text{preImage}(IC') \cup \text{map}_B.\text{preImage}(IC')$  ▷ Nos quedamos con las aristas de la
comunicación interna.
6:  $EC' \leftarrow (I/IC') \cap B$  ▷ Los nodos que no son comunicación interna, son comunicación
externa.
7:  $EC \leftarrow \text{map}_D.\text{preImage}(EC') \cup \text{map}_B.\text{preImage}(EC')$  ▷ Finalmente, computamos las
aristas que comunican  $A$  con  $B$ .
8: return  $\{EC, IC\}$ 
9: end function

```

Con este resultado, podemos computar $D_a = \#EC_a - \#IC_a, \forall a \in A$.

Luego, al igual que el algoritmo KL, necesitamos una forma de almacenar los costos de comunicación para calcular la ganancia de intercambiar conjuntos de nodos entre particiones. Para eso, usamos el Algoritmo 1 para calcular la comunicación interna y externa, y computamos una *Matriz de Ganancia*.

Algorithm 2 Cómputo de la Matriz de Ganancia

```

1: function COMPUTE_GAIN_MATRIX( $A, B, G, C$ )
2:  $D_A \leftarrow \text{compute.D}(A, B, G.\text{map}_B, G.\text{map}_D)$  ▷ Obtenemos la diferencia entre costo
externo e interno para los elementos de  $A$ .
3:  $D_B \leftarrow \text{compute.D}(B, A, G.\text{map}_B, G.\text{map}_D)$  ▷ Análogo para los elementos de  $B$ .
4:  $G = \emptyset$ 
5: for  $d_i \in D_A$  do
6:    $\text{gain}_{d_i} = \langle \rangle$ 
7:   for  $d_j \in D_B$  do
8:      $s \leftarrow \min(\#d_i, \#d_j)$  ▷ Tomamos conjunto con cardinalidad más pequeña para no
desbalancear en caso de intercambio.
9:      $g \leftarrow \text{diff}(d_i, s) + \text{diff}(d_j, s) - 2 \times c_{i,j}$  ▷ Computamos la ganancia de intercambiarlos.
10:     $\text{gain}_{d_i}.\text{insert}(i, j, g, s)$ 
11:   end for
12:    $G.\text{insert}(\text{gain}_{d_i})$ 
13: end for
14: return
15: end function

```

En este trabajo, los nodos son representados como conjuntos de intervalos. La principal diferencia con el algoritmo KL es que, en lugar de intercambiar nodos individuales, se intercambian intervalos de nodos (es decir, muchos nodos a la vez). Las operaciones realizadas por estos algoritmos operan sobre conjuntos, y su costo computacional es independiente del tamaño de los mismos.

Finalmente, en el Algoritmo 3 se muestran las modificaciones generales realizadas algoritmo KL.

Algorithm 3 Algoritmo de optimización Kernighan-Lin para SBG

```

1: function KL-SBG( $G, P = (A, B), C$ )
2:  $A_c = A$  ▷ Hacer una copia de las particiones originales
3:  $B_c = B$ 
4:  $\text{max\_par\_sum} = 0$  ▷ Inicializar variables
5:  $\text{max\_par\_sum\_set} = \emptyset$ 
6:  $\text{par\_sum} = 0$ 
7:  $A_v = \emptyset$ 
8:  $B_v = \emptyset$ 
9:  $GM = \text{compute\_gain\_matrix}(A_c, B_c, G, C)$  ▷ Computar matriz de ganancia
10: while  $A_c \neq \emptyset \wedge B_c \neq \emptyset$  do ▷ Mientras nos queden elementos en alguna de las particiones,
seguir iterando
11:    $\langle i, j, g, s \rangle = \text{max\_diff}(GM)$  ▷ Obtener intercambio con mayor ganancia

```

```

12:    $(A', B') = \text{update\_sets}(A_c, B_c, A_v, B_v, i, j, s)$       ▷ Actualizar conjuntos acorde a los
    cambios propuestos
13:    $\text{update\_sum}(par\_sum, g, max\_par\_sum, max\_par\_sum\_set, A', B')$  ▷ Actualizar suma
    parcial
14:    $\text{update\_diff}(A_c, B_c, i, j, s, GM, G)$                     ▷ Actualizar las copias  $A_c$  y  $B_c$ 
15:   end while
16:   if  $max\_par\_sum\_set \neq \emptyset$  then      ▷ Si existió alguna ganancia positiva, efectuar el cambio
17:      $(X, Y) = max\_par\_sum\_set$ 
18:      $A = (A/X) \cup Y$ 
19:      $B = (B/Y) \cup X$ 
20:   end if
21:   return
22: end function

```

4.3 Ejemplo de Optimización de Particiones de un Grafo SBG

En la Figura 4 tenemos la representación gráfica de un grafo SBG dividido en dos particiones **A** y **B** y el intercambio de dos intervalos $\{[11 : 15]\}$ y $\{[21 : 25]\}$. El resultado podemos visualizarlo en el grafo particionado del lado derecho.

La ganancia de intercambiar los nodos **a** y **b** se puede obtener de la siguiente manera:

Sean $a = \{[11 : 15]\}$ y $b = \{[21 : 25]\}$:

$$D_a = \#EC_a - \#IC_a = 5$$

$$D_b = \#EC_b - \#IC_b = 5$$

$$gain_{ab} = D_a + D_b - 2c_{ab} = 10$$

c_{ab} es la comunicación entre los nodos a y b.

La ganancia de intercambiarlos es igual a 10. Tras el intercambio, el *edgeCut* pasó de 15 a 5, como puede verse del lado derecho de la Figura 4. Lo que este ejemplo intenta demostrar es que, en una sola operación intercambiamos diez nodos (cinco de cada partición). En un grafo tradicional esto se llevaría a cabo en cinco operaciones individuales

5 Ejemplo

En esta sección, presentamos los resultados obtenidos con la implementación en C++ del algoritmo *SBG-Partitioner* Sansone et al., 2025 y su comparación con la herramienta de partición de grafos METIS. Elegimos METIS porque es una herramienta ampliamente aceptada por la comunidad. Nuestra expectativa es mejorar el rendimiento del particionado a medida que el grafo computacional aumente, sin que las particiones pierdan calidad. Para la comparación se utilizó una máquina con sistema operativo Linux (Ubuntu 20.04.6 LTS, desktop), procesador Intel® Core™ i98950HK CPU @ 2.90GHz × 12 y 32 GB de RAM. A continuación se detallan las métricas utilizadas para comparar la calidad de los resultados.

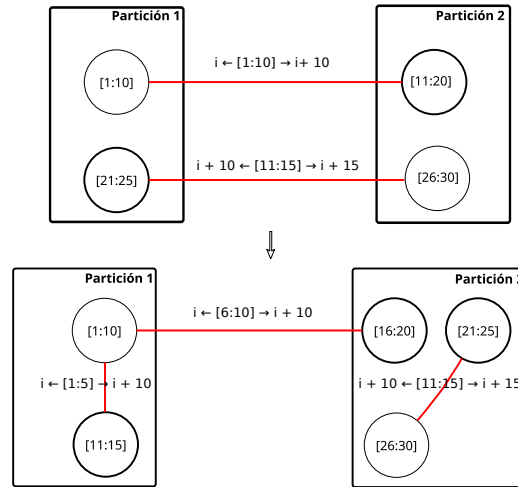


Figura 4: Representación gráfica del intercambio de dos intervalos de nodos en un Grafo SBG, dividido en dos particiones **A** y **B**.

5.1 Métricas de particionado

Las métricas de particionado evalúan la calidad de las particiones generadas. Las utilizadas para estos ejemplos son:

- **Edge-Cut:** La suma del peso de las aristas que conectan nodos entre particiones. Puede representarse como la siguiente función:

$$edgeCut(C, G) = \sum_{C_i, C_j} \{w(u, v) | u \in C_i, v \in C_j, i \neq j\}$$

- **Volumen de comunicación total:**

$$totCommVol(C, G) = \sum_{i=1}^P \sum_{u \in C_i} w(u, \cdot) (\lambda(u, C) - 1)$$

donde $w(u, \cdot)$ es el peso de una de las aristas de salida del nodo u (puesto que en estos ejemplos, todas tienen el mismo peso), mientras que $(\lambda(u, C) - 1)$ denota el número de particiones únicas a las que u o alguno de sus vecinos pertenece. Esta métrica también es conocida en la literatura como $(\lambda - 1)$.

- **Volumen de comunicación máximo:** Calcula el volumen de comunicación máximo entre particiones:

$$maxCommVol(C, G) = \max_{C_i} \sum_{u \in C_i} w(u, \cdot) (\lambda(u, C) - 1).$$

- **Desbalance máximo:** Calcula el desbalance máximo entre particiones.

$$\text{maxImbal}(C, G) = \max_{C_i} \left| \frac{\sum_{u \in C_i} [l(u)] - (\sum_{v \in V} l(v))/P}{(\sum_{v \in V} l(v))/P} \right|$$

donde C es una partición obtenida, $G = (V, E, l, w)$ es un grafo de conexiones donde V son sus nodos, E sus aristas, l la función que asigna un peso computacional a cada nodo y w la función que asigna un peso computacional a cada arista.

5.2 Métricas de rendimiento

Las métricas de rendimiento buscan comparar tiempo de ejecución entre distintos algoritmos en distintas etapas.

- **Tiempo de Inicialización:** Es el tiempo que le toma al particionador en leer la entrada y construir el grafo de conexiones³.
- **Tiempo de Particionado:** Es el tiempo que le toma al algoritmo generar las particiones iniciales y optimizarlas³.

5.3 Modelo Advección–Difusión–Reacción

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

Las Ecuaciones 2, 3 se obtienen luego de aplicar el Método de Líneas a las ecuaciones en Derivadas Parciales (PDE) originales.

$$\dot{u}_1 = -a \frac{(u_N - 1)}{\Delta x} + d \frac{-2u_1}{\Delta x^2} + r(u_1^2 - u_1^3) \quad (2)$$

y

$$\dot{u}_i = -a \frac{(u_i - u_{i-1})}{\Delta x} + d \frac{(u_{i+1} - 2u_i + u_{i-1})}{\Delta x^2} + r(u_i^2 - u_i^3) \quad (3)$$

para $i = 2, \dots, N$

A partir de estas ecuaciones se puede definir el siguiente modelo Modelica:

```

model adv_dif_reac
  constant Integer N=1000;
  parameter Real a=1;
  parameter Real d=1e-4;
  parameter Real r=10;
  parameter Real L=10;
  parameter Real dx=L/N;
  Real u[N];

  initial algorithm
    for i in 1:N/3 loop

```

³ El resultado que se exhibe es el promedio de 5 ejecuciones.

```

    u[i]:=1;
end for;

equation
der(u[1])=-a*(u[1]-1)/dx+d*(-2*u[1]+1)/(dx^2)+r*(u[1]^2)*(1-u[1]);
for i in 2:N loop
der(u[i])=-a*(u[i]-u[i-1])/dx+ d*(-2*u[i]+u[i-1])/(dx^2)+ r*(u[i]^2)
*(1-u[i]);
end for;
end adv_dif_reac;

```

En las Tablas 1, 2 y 3 se muestran los resultados para modelos de diferente tamaño generando 4 particiones.

En este caso, dado que las variables que comunican datos entre las distintas particiones son $u[i]$ con $u[i - 1]$ para un determinado rango, el *edgeCut* es igual a 3, pues es la arista que comunica los dos nodos adyacentes entre particiones es independiente del tamaño. Tanto METIS como SBG-Partitioner obtienen soluciones de la misma calidad puesto el agrupamiento de nodos que representan valores contiguos de u proveen la solución óptima. Dadas las condiciones del modelo, podemos observar que el tiempo de inicialización y particionado se mantiene constante para SBG-Partitioner, mientras que crece de manera lineal para METIS.

Tabla 1: Modelo con tamaño 10000

Particionador	Inicialización (ms)	Particionado (ms)	Edge cut	Communication volume	Maximum volume	Maximum imbalance
SBG-Partitioner	0,4	5,31	3	6	2	0
METIS	2,9	1,2	3	6	2	1,28e2

Tabla 2: Modelo con tamaño 100000

Particionador	Inicialización (ms)	Particionado (ms)	Edge cut	Communication volume	Maximum volume	Maximum imbalance
SBG-Partitioner	0,5	5,51	3	6	2	0
METIS	26	12	3	6	2	1,76e-2

Tabla 3: Modelo con tamaño 1000000

Particionador	Inicialización (ms)	Particionado (ms)	Edge cut	Communication volume	Maximum volume	Maximum imbalance
SBG-Partitioner	0,7	5,42	3	6	2	0
METIS	272	190	3	6	2	1,76e-4

6 Conclusiones

Presentamos un algoritmo para resolver el problema de balance de carga para simulación en paralelo de grandes modelos. Cuando estos modelos presentan estructuras regulares, el costo computacional asociado al mismo no depende del tamaño de los arreglos. Una ventaja adicional de esta representación es que los resultados obtenidos también son definidos como conjuntos por intersección, lo que permite que en la etapa posterior de simulación los mismos puedan ser tratados de manera eficiente.

Agradecimientos

Este trabajo fue financiado parcialmente por el proyecto PICT-2021-I-A-00826 (ANPCYT) y PIP-2022/2024 11220210100093CO (CONICET).

Referencias

- Andreev, K., & Räcke, H. (2004). Balanced graph partitioning. *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, 120-124.
- 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.
- Fiduccia, C. M., & Mattheyses, R. M. (1988). A linear-time heuristic for improving network partitions. En *Papers on Twenty-five years of electronic design automation* (pp. 241-247).
- Fritzson, P., & Engelson, V. (1998). Modelica—A unified object-oriented language for system modeling and simulation. *ECOOP'98—Object-Oriented Programming: 12th European Conference Brussels, Belgium, July 20-24, 1998 Proceedings 12*, 67-90.

- Karypis, G., & Kumar, V. (1997). *METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices* (inf. téc.). University of Minnesota, Department of Computer Science y Engineering, Army HPC Research Center, Minneapolis, MN.
- 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.
- Korch, M., & Rauber, T. (2006). Optimizing locality and scalability of embedded runge-kutta solvers using block-based pipelining. *Journal of Parallel and Distributed Computing*, 66(3), 444-468.
- Marzorati, D., Fernández, J., & Kofman, E. (2022). Efficient connection processing in equation-based object-oriented models. *Applied Mathematics and Computation*, 418, 126842.
- Pellegrini, F. (2012). Scotch and PT-scotch graph partitioning software: an overview. *Combinatorial Scientific Computing*, 373-406.
- Rauber, T., & Rünger, G. (2013). *Parallel programming*. Springer.
- Rauber, T., & Rünger, G. (2021). Modeling the effect of application-specific program transformations on energy and performance improvements of parallel ODE solvers. *Journal of Computational Science*, 51, 101356.
- Sanders, P., & Schulz, C. (2013). Think locally, act globally: Highly balanced graph partitioning. *International Symposium on Experimental Algorithms*, 164-175.
- Sansone, F., Fernández, J., & Kofman, E. (2025). *SBG Partitioner: a load balancing algorithm to simulate discrete event systems in parallel using set-based graphs*. <https://github.com/CIFASIS/sbg-partitioner>
- 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.