PARTICIÓN DE MODELOS DE GRAN ESCALA PARA SIMULACIÓN EN PARALELO POR MÉTODOS QSS

Franco Sansone[♭], Joaquin Fernandez[♭] y Ernesto Kofman[♭,†]

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

Resumen: Este artículo describe un algoritmo para resolver el problema de balance de carga para simulación en paralelo de grandes modelos que presentan estructuras regurales. Esta característica permite la utilización de *Set-Based Graphs* (SBG) y como consecuencia el costo computacional del algoritmo es independiente de la cantidad de vétices y aristas del grafo.

Palabras clave: *modelos de gran escala, set–based graphs, partición de grafos*
2000 AMS Subject Classification: 68W99

1. INTRODUCCIÓN

En general, los algoritmos de simulación en paralelo usan un modelo dividido en tantas partes como procesadores de manera que cada procesador simula un sub-modelo más pequeño[6]. 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 de resolviendo un problema de *balance de carga*[2], 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.

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 [3].

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 entre los cuales podemos destacar algoritmos clásicos como el de Kernighan–Lin [5](KL) y herramientas tales como METIS [4] que utiliza *Métodos Multinivel* debido a su eficiencia y a la calidad de las particiones generadas.

Una observación esencial es que habitualmente los modelos muy grandes 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*) [7]. 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 SBG permite generar particiones con un costo computacional independiente del tamaño de los arreglos de variables definidos en el modelo original.

2. SET-BASED GRAPHS

Los *Set-Based Graphs* (SBG) [7] 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 $\mathcal{V} = \{V^1, \ldots, V^p\}$ de V a cuyos elementos llamaremos* Set–Vértices.
- *Una partición $\mathcal{E} = \{E^{i_1,j_1}, ..., E^{i_k,j_k}\}$ de E a cuyos elementos llamaremos* Set-Aristas*, 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 naturales.
- Dos mapas que representan conexiones, $\mathrm{map}_B, \mathrm{map}_F : \mathbb{N}^n \mapsto \mathbb{N}^n$ tales que para cada arista $e = (u, v)$, $\mathrm{map}_B(e) = u, \mathrm{map}_D(e) = v$.
- Un mapa que represente vértices–conjunto, $V_{\mathrm{map}} : \mathbb{N}^n \mapsto \mathbb{N}, V_{\mathrm{map}}(v) = i$ si $v \in V^i$.
- Un mapa que represente aristas–conjunto, $E_{\mathrm{map}} : \mathbb{N}^n \mapsto \mathbb{N}^2, E_{\mathrm{map}}(e) = (i, j)$ 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.

3. Algoritmo de Partición de Grafos KL-SBG

La idea de este trabajo es adaptar el algortimo de KL utilizando la representación SBG para aprovechar la presencia de estructuras cíclicas. La idea fundamental del algoritmo KL [5] es la siguiente:

Sea S un conjunto de $2n$ puntos, con una matriz de costo asociada $C = (c_{ij})$, $i,\ j = 1, \ldots, 2n$. Queremos partir S en dos conjuntos A y B, cada una con n puntos, tales que el *costo externo* $T = \sum_{A \times B} c_{ab}$ 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' luego de realizar el intercambio es localmente mínima.

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

$$E_a = \sum_{y \in B} c_{ay} \tag{1}$$

y un *costo interno* I_a

$$I_a = \sum_{x \in A} c_{ax} \tag{2}$$

Análogamente, definimos E_b, I_b para cada $b \in B$. Luego podemos definit la diferencia D como:

$$D_x = E_x - I_x \text{ para todo } x \in S; \tag{3}$$

Considerando cualquier $a \in A$, $b \in B$. Si a y b son intercambiados, la *ganancia* (esto es, la reducción del costo) es precisamente

$$D_a + D_b - 2c_{ab} \tag{4}$$

El Algortimo 1 calcula la diferencia dadas dos particiones y los mapas del grafo SBG correspondiente donde las operaciones involucradas actúan sobre conjuntos.

Algorithm 1 EC_IC

1: **function** EC_IC(A, B, map_B, map_D)
2: $D \leftarrow map_B.preImage(A) \cup map_D.preImage(A)$
3: $I \leftarrow map_D.image(D) \cup map_B.image(D)$
4: $IC' \leftarrow (A \cap I)$
5: $IC \leftarrow map_D.preImage(IC') \cup map_B.preImage(IC')$
6: $EC' \leftarrow (I/IC') \cap B$
7: $EC \leftarrow map_D.preImage(EC') \cup map_B.preImage(EC')$
8: **return** $\{EC, IC\}$

9: **end function**

Luego podemos utilizar el Algoritmo 1 para calcular la matriz de ganancia:

Algorithm 2 compute_gain_matrix

1: **function** COMPUTE_GAIN_MATRIX(A, B, G, C)
2: $D_A \leftarrow$ **EC_IC**$(A, B, G.map_B, G.map_D)$
3: $D_B \leftarrow$ **EC_IC**$(B, A, G.map_B, G.map_D)$
4: $G = \emptyset$
5: **for** $d_i \in D_A$ **do**
6: $gain_{d_i} =<>$
7: **for** $d_j \in D_B$ **do**
8: $s \leftarrow \mathbf{min}(\#d_i, \#d_j)$
9: $g \leftarrow \mathbf{diff}(d_i, s) + \mathbf{diff}(d_j, s) - 2 \times c_{i,j}$
10: $gain_{d_i}.\mathbf{insert}(i, j, g, s)$
11: **end for**
12: $G.\mathbf{insert}(gain_{d_i})$
13: **end for**
14: **return**
15: **end function**

Nuevamente, las operaciones realizas por este algoritmo actúan sobre conjuntos Finalmente, en el Algoritmo 3 se muestran las moificaciones 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: $max\_par\_sum = 0$ ▷ Inicializar variables
5: $max\_par\_sum\_set = \emptyset$
6: $par\_sum = 0$
7: $A_v = \emptyset$
8: $B_v = \emptyset$
9: $GM = \mathbf{compute\_gain\_matrix}(A_c, B_c, G, C)$ ▷ 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: $< i, j, g, s >= \mathbf{max\_diff}(GM)$ ▷ Obtener intercambio con mayor ganancia
12: $(A', B') = \mathbf{update\_sets}(A_c, B_c, A_v, B_v, i, j, s)$ ▷ Actualizar conjuntos acorde a los cambios propuestos
13: $\mathbf{update\_sum}(par\_sum, g, max\_par\_sum, max\_par\_sum\_set, A', B')$ ▷ Actualizar suma parcial
14: $\mathbf{update\_diff}(A_c, B_c, i, j, s, GM, G)$ ▷ 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: $(A^\star, B^\star) = max\_par\_sum\_set$
18: $A = (A/A^\star) \cup B^\star$
19: $B = (B/B^\star) \cup A^\star$
20: **end if**
21: **return**
22: **end function**

4. Ejemplo

El siguiente modelo, presentado en [1] representa una ecuación de Advección-Difusión–Reacción (ADR) en una dimensión, que se definir en Modelica como:

```
model adv_dif_reac
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;
```

La Tabla 1 muestra los resultados obtenidos para diferentes tamaños del modelo, como así también los resultados obtenidos utilizando Metis. Donde podemos observar que el tiempo de inicialización y particionado se mantiene constante para los algoritmos SBG, mientras que crece de manera lineal para Metis. .[1]

Tabla 1: Tiempos de ejecución para diferentes valores de N

Tamaño	**SBG Partitioner**		**Metis**	
	Construcción (ms)	**Particionado (ms)**	**Construcción (ms)**	**Partitionado (ms)**
1000	$0,52$	$4,73$	$1,2$	0,9
10000	$0,48$	$5,31$	$12,9$	$7,4$
100000	$0,51$	$5,51$	139	141
1000000	$0,79$	$5,42$	1350	1422

5. 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 del los arreglos. Los resultados obtenidos también son representados como conjuntos definidos por intesión, lo que permite que en la etapa de simulación los mismos puedan ser tratados de manera eficiente.

Referencias

[1] F. Bergero, J. Fernández, E. Kofman, and M. Portapila, *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 (2016), pp. 47–61.

[2] A. Boukerche and S. Das, *Load balancing strategies for parallel simulations on a multiprocessor machine*, Advanced Computer Performance Modeling and Simulation. K. Bagchi, J. Walrand und GW Zobrist, (1998), pp. 135–164.

[3] R. M. Fujimoto, *Parallel discrete event simulation*, Communications of the ACM, 33 (1990), pp. 30–53.

[4] G. Karypis and V. Kumar, *Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices*, tech. rep., University of Minnesota, Department of Computer Science and Engineering, Army HPC Research Center, Minneapolis, MN, 1997.

[5] B. W. Kernighan and S. Lin, *An efficient heuristic procedure for partitioning graphs*, The Bell system technical journal, 49 (1970), pp. 291–307.

[6] T. Rauber and G. Rünger, *Parallel programming*, Springer, 2013.

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

[1] Los algoritmos y ejemplos presentados fueron implementados un repositorio de código abierto: https://github.com/CIFASIS/sbg-partitioner