Trabalho apresentado no XXXIX CNMAC, Uberlândia - MG, 2019.

Proceeding Series of the Brazilian Society of Computational and Applied Mathematics

Explorando Paralelismo *Dataflow* em Geometria de Distâncias Moleculares Intervalares

Sandro P. Vilela [1]
Programa de Engenharia de Sistemas e Computação, COPPE, UFRJ, Rio de Janeiro, RJ
Leandro A. J. Marzulo[2]
Departamento de Informática e Ciência da Computação, IME, UERJ, Rio de Janeiro, RJ
Felipe M. G. França[3]
Programa de Engenharia de Sistemas e Computação, COPPE, UFRJ, Rio de Janeiro, RJ

Resumo. Geometria de Distâncias corresponde a uma classe de problemas onde as posições dos pontos no espaço são determinadas através da informação de algumas distâncias entre estes. Um destes problemas, muito importante, consiste na reconstrução de estruturas tridimensionais de proteínas a partir de algumas distâncias intervalares obtidas através de experimentos de Ressonância Magnética Nuclear. Este trabalho propõe um abordagem paralela por *Dataflow* para o tratamento destes problemas aplicados à reconstrução de estruturas moleculares.

Palavras-chave. Geometria de Distâncias Moleculares, *iDMDGP*, *Dataflow*

1 Introdução

O Problema de Geometria de Distâncias (*DGP, Distance Geometry Problem*) tem sido objeto de muita pesquisa, possui várias aplicações, tais como na localização em redes de sensores, no reconhecimento de imagens, nas predições de estruturas moleculares, entre outras [2]. O *DGP* consiste na determinação das coordenadas de um determinado conjunto de pontos, a partir de algumas distâncias conhecidas entre os pares desses pontos [1]. Quando as distâncias entre alguns destes são dadas por intervalos, temos a versão intervalar do problema. Para o *DGP* relacionado a proteínas e empregando-se distâncias intervalares têm se o *Interval Discretizable Molecular Distance Geometry Problem (iDMDGP)*, proposto em Lavor *et al.* [3]. Neste trabalho iremos tratar da aplicação do *iDMDGP* na reconstrução de estruturas tridimensionais de proteínas, através do emprego de dados (intervalares) que simulam medidas experimentais obtidas por Ressonância Magnética Nuclear. Empregaremos, para isto, uma abordagem paralela para o *iDMDGP*, baseada no modelo de execução *Dataflow*, o qual, tem se mostrado uma boa opção para a programação paralela [6].

[1]sandro@cos.ufr.br
[2]leandro@ime.uerj.br
[3]felipe@cos.ufrj.br

 © 2020 SBMAC

2 Problema de Geometria de Distâncias Moleculares Intervalares (*iDMDGP*)

O DGP quando aplicado à moléculas, valendo-se de distâncias intervalares, é conhecido como *iDMDGP*. De uma maneira breve, ele pode ser formulado como:

Encontre as posições $x_1, ..., x_n \in R^3$, dos átomos da molécula, tais que

$$d_{inf} \leq |x_i - x_j| \leq d_{sup} \qquad (i,j) \in S, \tag{1}$$

onde S é um subconjunto dos pares de átomos cujas distâncias d_{inf} e d_{sup} correspondem, respectivamente, ao limite inferior e superior para a distância entre os átomos i e j e $|.|$ é a norma Euclidiana.

Como metodologia para encontrar as soluções (as posições atômicas) da equação (1), empregaremos um algoritmo *Branch & Prune* [4], o qual, explorará o espaço de busca (topologia em árvore), de maneira recursiva, verificando a cada novo ramo adicionado, se as soluções (posições atômicas) encontradas satisfazem as condições impostas pelos dados experimentais, *pruning phase* [5]. À cada nível que exploramos testamos várias posições, o *branching*. Ao se descobrir posições inviáveis reduz-se drasticamente o espaço de busca, pois assim que descobrimos uma posição não conforme deixamos de explorar aquele ramo (o ramo da árvore de busca que possui o nó em questão como raiz). No pior cenário, sem a poda, a árvore de busca crescerá exponencialmente.

A estratégia que empregaremos para percorrermos a árvore de busca de maneira eficiente será o paralelismo por *Dataflow*, de tal maneira que a árvore será particionada a partir de um determinado nível. Cada subárvore, ramo, tornar-se-á assim uma partição e será submetida a um *worker* (*core*), o qual encontrará as soluções associadas, fig. 1.

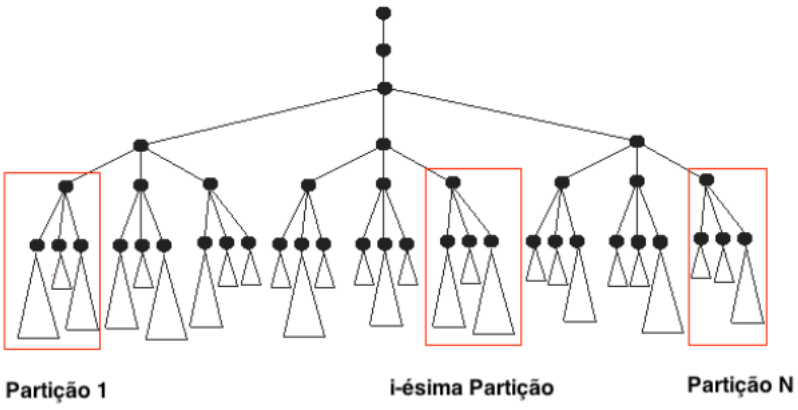

Figura 1: Particionamento da árvore de busca do *iDMDGP*.

Uma vez que se tem as soluções encontradas por todos os *workers* seleciona-se a solução que possua o menor valor para o *LDE, Largest Distance Error*, uma medida da qualidade da solução.

DOI: 10.5540/03.2020.007.01.0383 © 2020 SBMAC

3 O Paralelismo *Dataflow*

O modelo de execução *Dataflow* consiste em um paradigma de computação que explora o paralelismo em aplicações de uma maneira muito natural. As instruções são executadas de acordo com o fluxo de dados, ou seja, assim que todos os seus operandos de entradas estiverem disponíveis. Um programa *Dataflow* é descrito por um grafo direcionado, onde cada vértice representa uma tarefa e cada aresta as dependências de dados entre as tarefas. As tarefas são executadas de acordo com as suas dependências, o que permitirá a execução concorrente sem a necessidade de um contador de programa. De fato, este modelo é adotado em máquinas de *Von Neumann* com o intuito de extrair paralelismo, implementado como um mecanismo de execução fora-de-ordem com escalonamento dinâmico baseado em fluxo de dados. No entanto, o paralelismo explorado está limitado pela emissão das instruções as quais continuam seguindo o fluxo de controle.

3.1 *Sucuri*

Neste trabalho empregamos a biblioteca *Dataflow Sucuri* como base para implementarmos o paralelismo no *iDMDGP*. Nesta seção a descrevemos brevemente.

A biblioteca *Sucuri* descrita em Alves et al. [6], escrita na linguagem Python, permite a implementação em alto nível de algoritmos paralelos seguindo o modelo *Dataflow*. De uma maneira simples e intuitiva ela possibilita a construção de um grafo *Dataflow*, da aplicação em questão, através da criação de nós e arestas.

Resumidamente, podemos afirmar que a *Sucuri* é composta por três estruturas principais descritas a seguir:

Grafo: Consiste em uma estrutura composta por um conjunto de nós e arestas, onde cada nó possui: a função a ser executada ao receber todos os operandos de entrada, a lista de nós destinatários que dependem dos operandos resultantes e a lista de operandos recebidos aguardando casamento.
Escalonador: Responsável pela comunicação entre os nós do Grafo, bem como pelo envio de tarefas aos *Workers*. É composto por uma unidade de casamento e uma fila de prontos.
Worker: Responsável por processar as tarefas enviadas pelos escalonadores.

Através do emprego destas três estruturas básicas o desenvolvedor será capaz de criar programas que exploram o paralelismo por *Dataflow*, para isto bastará implementar as funções do seu programa e associá-las aos objetos *Node* (nó) os quais são conectados através de arestas, de acordo com as suas dependências de dados. Os nós são inseridos em um objeto *Graph* (grafo) o qual é passado como parâmetro para o escalonador da *Sucuri* (responsável pela execução de acordo com a regra de disparo *Dataflow*). Enfim, se duas funções não tiverem dependências entre elas, estas serão potencialmente executadas em paralelo, dependendo da disponibilidade de *cores* (*workers*) ociosos nas máquinas alocadas. Assim, depois que os nós e suas respectivas dependências forem instanciados, o programa poderá ser executado em uma única máquina *multicore* ou em um grupo delas.

DOI: 10.5540/03.2020.007.01.0383 © 2020 SBMAC

3.2 Resolvendo o *iDMDGP* por *Dataflow*

A figura 2 mostra o tratamento utilizado na paralelização do *iDMDGP* valendo-se da biblioteca *Sucuri*, em específico, o grafo empregado.

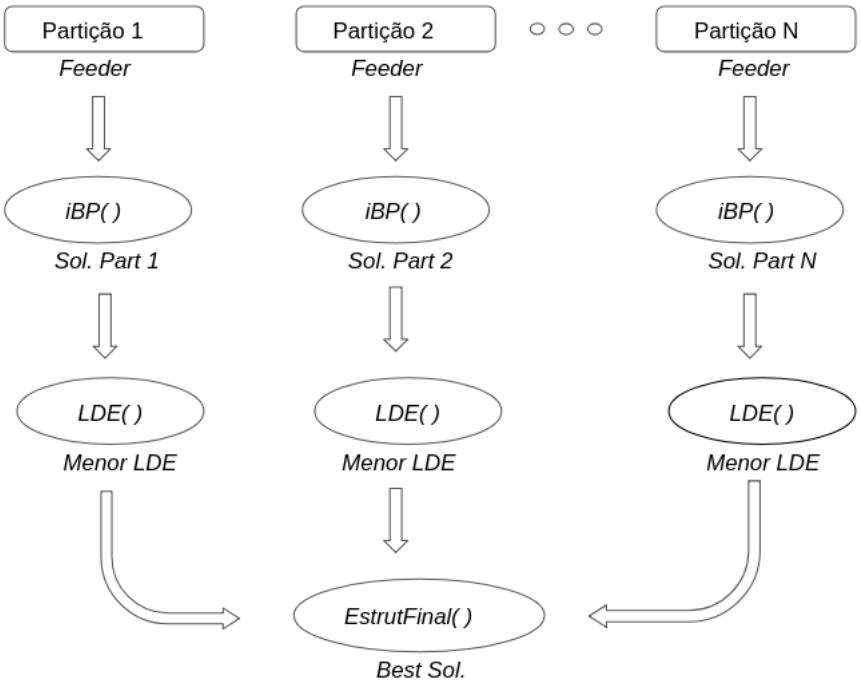

Figura 2: Particionamento da árvore de busca do *iDMDGP*.

Inicialmente exploraremos a árvore de busca, empregando o algoritmo *Branch & Prune*, descrito em Lavor *et al.* [4], até um determinado nível, a partir deste nível particionaremos a árvore de busca em função do número de soluções encontradas até o presente nível. As informações referentes a cada partição serão atribuidas aos nós *Feeders* (nós especializados) os quais as repassarão aos nós que executam a rotina *iBP*, obtendo-se assim as soluções associadas a cada uma das partições. Estas soluções serão direcionadas aos nós que selecionarão as soluções com o menor *LDE* e as repassarão a um único nó, o qual será capaz de selecionar globalmente a solução com o menor *LDE*, a melhor solução.

4 Experimentos e Resultados

Realizamos uma bateria de experimentos em um computador *multicore*, processador *Intel(R) Core(TM)* i9-7900X, velocidade de 3.30 GHz, 10 *cores*. Os resultados são apresentados na Tabela 1.

Os experimentos foram realizados com instâncias de 200, 300 e 500 átomos. As instâncias de 500 átomos, no caso duas, são diferentes entre si. Empregamos um fator de *branching* igual a 5 e particionamos as árvores de busca a partir do sexto nível. Na

DOI: 10.5540/03.2020.007.01.0383 © 2020 SBMAC

Tabela 1: Dados referentes aos experimentos realizados.

Qtd. Átomos	Partições	T. Serial[s]	T. Paralelo[s]	*Speedup M.*
200	24	76.378	11.350	6.74
300	62	268.488	38.750	6.93
500	26	111.420	17.787	6.26
500	122	2879.756	549.066	5.24

tabela 1 temos os tempos de execução encontrados nos experimentos, a coluna *T. Serial[s]* corresponde ao tempo gasto por apenas 1 *core*, enquanto a coluna *T. Paralelo[s]* ao tempo consumido empregando os 10 *cores*. A coluna *Partições* corresponde ao número de particionamentos da árvore de busca que empregou-se em cada instância e a coluna *Speedup M.* contém os *Speedups Máximo* alcançados pelo nosso programa nos experimentos.

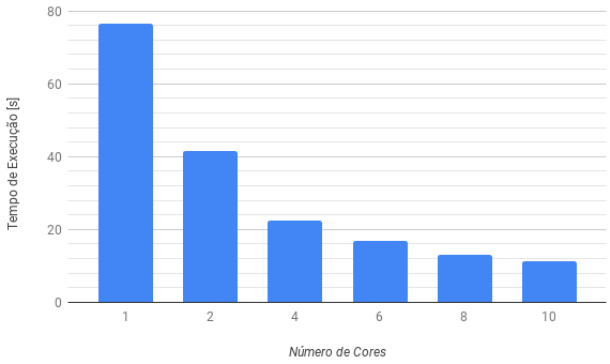

Figura 3: Tempo de Execução versus Número de *Cores*, instância de 200 átomos.

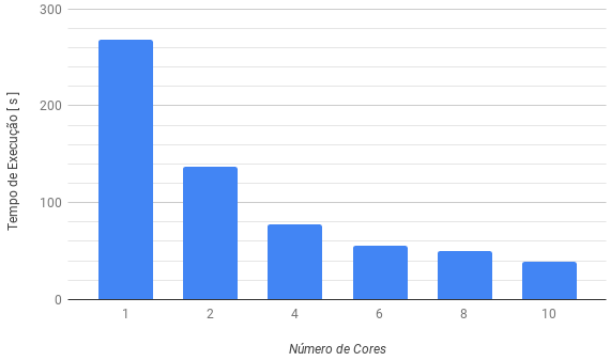

Figura 4: Tempo de Execução versus Número de *Cores*, instância de 300 átomos.

DOI: 10.5540/03.2020.007.01.0383

© 2020 SBMAC

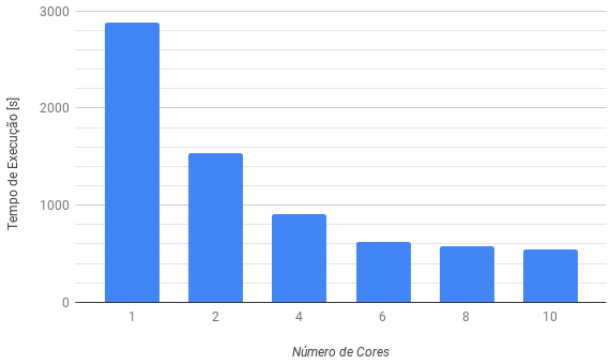

Figura 5: Tempo de Execução versus Número de Cores, instância de 500 átomos e 122 partições.

Nas figuras 3, 4 e 5 apresentamos os (gráficos) tempos de execução em função do número de *cores (workers)* para as instâncias de 200, 300 e 500 átomos.

5 Análise e Conclusões

Os resultados apresentados na tabela 1 mostram que a abordagem envolvendo o paralelismo apresentou uma redução no tempo de execução considerável, quando comparado à versão serial. Dos gráficos, figuras 3, 4 e 5, notamos também uma queda no tempo de computação, notadamente para a instância de 300 átomos. Os resultados mostram a efetividade do emprego da biblioteca *Dataflow Sucuri* no tratamento do *iDMDGP*.

Um dos fatores que consideramos como razão para não termos encontrado melhores valores para os *speedups*, o maior valor encontrado foi 6.93 com o emprego de 10 *cores*, consiste no fato de que as partições não são bem balanceadas. Por exemplo, para a instância de 500 átomos com 122 partições investigadas, verificamos que dentre estas a grande maioria foi "resolvida", podada, em menos de um milésimo de segundo de processamento. Ao final do processo de exploração da árvore de busca, associada à esta instância, observamos que apenas 6 partições (em cada partição foram encontradas 1514 soluções) dominaram a maior parte do processamento, as 6 consumiram aproximadamente 338 segundos, enquanto isto 4 *cores* permaneceram inativos. Assim, como trabalhos futuros, pretendemos atacar o *iDMDGP* com uma abordagem que envolva o escalonamento dinâmico de tarefas, escolha mais adequada em casos desta natureza (desbalanceamento de carga), neste trabalho empregamos o escalonamento estático.

Desejamos, também, investigar o particionamento a partir de níveis com valores maiores do que seis (mais profundos) e o uso do fator de *branching* com valores maiores do que 5. O *branching* possui uma importância capital no tempo total de processamento pois ele determina a quantidade de posições atômicas a serem exploradas, por exemplo: um fator igual a 5, em uma instância de 500 átomos, implicará numa quantidade de aproximadamente 5^{497} posições atômicas a serem investigadas, um número maior do que a quantidade de átomos do universo ($\sim 10^{87}$). A exploração de uma árvore de busca de

DOI: 10.5540/03.2020.007.01.0383

© 2020 SBMAC

tal tamanho ($\sim 5^{500}$ nós) somente foi possível graças ao emprego do algoritmo *Branch & Prune*, através dele "*podamos*"vários ramos da árvore, reduzindo drasticamente o espaço de busca e tornando a exploração factível.

Referências

[1] A. Mucherino, C. Lavor, L. Liberti, N. Maculan. Distance Geometry. *Theory, Methods, and Applications*, Springer New York Heidelberg Dordrecht London. ISBN: 978-1-4614-5127-3. DOI: 10.1007/978-1-4614-5128-0, 2013.

[2] C.Lavor, L. Liberti, A. Mucherino. Recent advances on the discretizable molecular distance geometry problem, *European Journal of Operational Research*, 219:698-706, 2012.

[3] C. Lavor, L. Liberti, A. Mucherino. The Interval Branch-and-Prune algorithm for the discretizable molecular distance geometry problem with inexact distances. *Journal of Global Optimization,* pp. 1-17, 2011.

[4] C. Lavor, L. Liberti, N. Maculan, A. Mucherino. The Discretizable Molecular Distance Geometry Problem. *Computational Optimization and Applications* 52, 115-146, 2012.

[5] D. S. Gonçalves, A. Mucherino, C. Lavor. Recent advances on the interval distance geometry problem. *Journal of Global Optimization.* DOI 10.1007/s10898-016-0493-6, 2017.

[6] T. Alves, B. Goldstein, L. A. J. Marzulo. A minimalistic dataflow programming library for python. *Computer Architecture and High Performance Computing Workshop (SBAC-PADW),* International Symposium, 2014.

 © 2020 SBMAC