XXI ENMC
ENCONTRO NACIONAL DE MODELAGEM COMPUTACIONAL
IX ECTM
ENCONTRO DE CIÊNCIA E TECNOLOGIA DE MATERIAIS

08 a 11 de Outubro de 2018
Instituto Federal Fluminense
Búzios - RJ

UMA IMPLEMENTAÇÃO SIMPLES POR OPENMP DE VARIANTE DA BUSCA EM LARGURA

Jean Antonio Ribeiro - jeankusanagi@posgrad.ufla.br
Sanderson Lincohn Gonzaga de Oliveira - sanderson@dcc.ufla.br
[1]Universidade Federal de Lavras, Lavras, MG, Brasil

Abstract. *Recentemente, foi proposta uma variação sequencial da busca em largura ao ser aplicada na reordenação de linhas e colunas de sistemas de equações lineares de forma a reduzir o custo de execução total de resolutores de sistemas de equações lineares. Neste trabalho, mostramos que uma implementação simples por OpenMP dessa variante da busca em largura não obtém aceleração em relação a sua versão sequencial.*

Keywords: *Redução de largura de banda, OpenMP, matrizes esparsas.*

1. INTRODUÇÃO

Heurísticas para redução de largura de banda de matrizes são utilizadas, principalmente, para melhorar o desempenho computacional na resolução de sistemas de equações lineares do tipo $Ax = b$, em que A é uma matriz esparsa de grande porte. x é o vetor de incógnitas e b é o vetor de termos independentes. A redução de largura de banda de matrizes consiste em realizar permutações de linhas e de colunas, deixando a matriz com uma estrutura em banda. Largura de banda de uma matriz simétrica, denotado por β, é a quantidade de subdiagonais com coeficientes não nulos da matriz triangular inferior.

Seja $A = [a_{ij}]$ uma matriz simétrica, $n \times n$, associada a um grafo não-direcionado $G = (V, E)$, composto de um conjunto V de vértices e um conjunto E de arestas. A largura de banda da linha i da matriz A é $\beta_i(A) = i - \min_{1 \leq j \leq i} [j : a_{ij} \neq 0]$. Com isso, a largura de banda da matriz A é definida como $\beta(A) = \max[\beta_i(A)]$. De forma equivalente, a largura de banda do grafo G, para uma numeração $S = \{s(v_1), s(v_2), \cdots, s(v_{|V|})\}$, isto é, uma bijeção de V para o conjunto $\{1, 2, \cdots, |V|\}$, é $\beta(G) = \max_{v \in V}[(\{v, u\} \in E)\ |s(v) - s(u)|]$.

Com a utilização de processamento paralelo, pode-se obter um custo muito menor nas resoluções de sistemas de equações lineares de grande porte quando comparados com resoluções sequenciais. Neste artigo, o principal objetivo é avaliar o desempenho computacional da heurística RBFS-GL [Gonzaga de Oliveira et al., 2018] em arquitetura *multicore*, implementada por meio da biblioteca OpenMP. Essa heurística é o atual estado da arte em reordenação de linhas e colunas de matrizes de sistemas de equações lineares oriundos de áreas de aplicação como análises de estruturas e acústica.

A seguir, a implementação do algoritmo é descrita na Seção 2. Os experimentos são mostrados na Seção 3. Por fim, a conclusão e os trabalhos futuros são apresentados na Seção 4.

2. IMPLEMENTAÇÃO

Nesta seção, são descritas as paralelizações da heurística RBFS-GL [Gonzaga de Oliveira et al., 2018]. Essa heurística numera os vértices do grafo na ordem inversa dada pela busca em largura. O vértice inicial é dado pelo algoritmo de George and Liu [1981].

A abordagem de paralelização por OpenMP adotada na implementação do algoritmo é simples. A implementação sequencial, por sua vez, é eficiente. Foi utilizada a linguagem de programação C++.

A busca em largura com reordenação inversa (RBFS-GL) é um método que pode ser utilizado como um algoritmo de reordenação de linhas e colunas de matrizes. Inicialmente, um vértice pseudoperiférico é escolhido como vértice inicial. A partir desse vértice, a cada vértice percorrido, é considerada a distância a partir do vértice inicial. Por fim, a numeração gerada pela busca em largura, correspondente à ordem em que os vértices são percorridos é invertida.

Na heurística sequencial RBFS-GL, um vértice no nível l (ou seja, a distância para o vértice inicial) deve ser computado antes de qualquer outro vértice em nível $l + 1$. Essa ordem produz uma implementação eficiente, mas restringe o paralelismo. Embora todos os vértices no nível l possam ser computados simultaneamente, nenhum vértice no nível $l + 1$ poderá ser computado até que todos os vértices no nível l sejam computados. Consequentemente, as *threads* deverão ser sincronizadas nível à nível na estrutura de nível enraizada no vértice inicial.

Na implementação paralela do algoritmo, os vértices de cada nível foram particionados entre as *threads*. Operações como renumeração de vértices e escolha do próximo vértice a ser computado foram implementadas como operações atômicas. Além disso, a ordem de escolha dos vértices é aleatória. Com isso, as soluções geradas, nas execuções do algoritmo, poderão ser diferentes umas das outras.

3. RESULTADOS

Nos experimentos, foram utilizados computadores com processador Intel® Core(TM) i7-4770 de 3,40Ghz com (4 núcleos e 2 *threads* por núcleo) e com 8Mb de memória *cache*, 8Gb de memória RAM DDR3 1333MHz. Foi utilizada uma distribuição Linux Ubuntu 16,04 LTS 64 bits com *kernel* versão 4.10.0-42-$generic$. O compilador empregado foi o g++ versão 5.4.0, e foi utilizada a *flag* -O3 na fase de compilação.

Na tabela 1, são mostradas informações das 11 instâncias da base *SuiteSparse Matrix Collection* utilizadas nos experimentos [Davis and Hu, 2011]. Essas instâncias são compostas por matrizes simétricas.

Para cada instância, foram realizadas cinco execuções. Foram estabelecidas de 1 a 5, 8 e 12 *threads* para cada execução da heurística em paralelo.

Nas tabelas 2, 3 e 4, são apresentados os resultados obtidos com os experimentos. Nas últimas colunas, são mostrados os *speed-downs* obtidos com a paralelização do algoritmo por uma estratégia simples por OpenMP.

Table 1- Instâncias utilizadas nos experimentos.

Instância	Dimensão	Coeficientes não nulos	β	*Profile*
offshore	259.789	4.242.673	237.738	3.588.201.815
G2_circuit	150.102	726.674	93.719	1.098.802.048
shipsec1	140.874	3.568.176	5.237	431.771.001
boneS01	127.224	5.516.602	3.722	331.330.356
cfd2	123.440	3.085.406	4.501	156.107.322
thermomech_TC	102.158	711.558	102.138	2.667.823.445
thermomech_TK	102.158	711.558	102.138	2.667.823.445
2cubes_sphere	101.492	1.647.264	100.407	483.241.271
thermal1	82.654	574.458	80.916	175.625.317
qa8fm	66.127	1660579	1.048	64.862.523
Andrews	60.000	760.154	59.925	1.501.971.521

Table 2- Resultados de execuções de implementação simples por OpenMP da heurística RBFS-GL aplicada a três instâncias.

Instância	#	β	*Profile*	Tempo (s)			Speed-down
				RBFS	GL	RBFS-GL	
offshore	1	20274	2654837432	0,09	0,4	0,5	–
	2	33312	2923489977	0,30	0,4	0,7	0,7
	3	31231	2767033704	0,32	0,3	0,6	0,8
	4	36034	2718727645	0,42	0,3	0,7	0,7
	5	33034	2730808793	0,51	0,3	0,8	0,6
	8	37823	2745345002	0,64	0,4	1,0	0,5
	12	42663	2991075925	0,61	0,4	1,0	0,5
G2_circuit	1	1959	181180439	0,02	0,1	0,2	–
	2	2899	184203242	0,05	0,1	0,2	1,0
	3	2833	183451437	0,06	0,1	0,2	1,0
	4	3224	183865030	0,08	0,1	0,2	1,0
	5	3352	184125835	0,09	0,2	0,3	0,7
	8	3657	184146991	0,11	0,2	0,3	0,7
	12	3705	184157214	0,12	0,2	0,3	0,7
cfd2	1	2386	164048704	0,04	0,1	0,1	–
	2	3325	178682141	0,21	0,1	0,3	0,3
	3	3657	184234122	0,23	0,2	0,4	0,3
	4	4009	187113163	0,29	0,2	0,5	0,2
	5	3786	187254138	0,35	0,2	0,5	0,2
	8	4121	191803276	0,45	0,3	0,7	0,1
	12	4307	193897771	0,40	0,2	0,6	0,2

4. CONCLUSÃO

Neste artigo, foi avaliada uma implementação, por meio da biblioteca OpenMP, da heurística RBFS-GL [Gonzaga de Oliveira et al., 2018], utilizada para o reordenamento de linhas e colunas de matrizes. A busca em largura é um algoritmo irregular. Mostra-se que uma paralelização simples por OpenMP obtém resultados ruins.

Table 3- Resultados de execuções de implementação simples por OpenMP da heurística RBFS-GL aplicada a seis instâncias.

Instância	#	β	*Profile*	Tempo (s)			Speed-down
				RBFS	GL	RBFS-GL	
boneS01	1	7697	411029844	0,06	0,2	0,3	–
	2	10984	437647729	0,38	0,2	0,6	0,5
	3	11453	407439060	0,51	0,2	0,7	0,4
	4	13487	453377638	0,62	0,2	0,8	0,4
	5	11505	424156567	0,81	0,2	1,0	0,3
	8	14998	467417688	1,02	0,2	1,2	0,3
	12	14654	461602840	0,91	0,2	1,1	0,3
thermomech_TC	1	260	17795049	0,02	0,1	0,1	–
	2	394	18196102	0,05	0,1	0,1	1,0
	3	393	18212930	0,06	0,1	0,2	0,5
	4	473	18381001	0,07	0,1	0,2	0,5
	5	470	18431960	0,09	0,1	0,2	0,5
	8	487	18505808	0,11	0,1	0,2	0,5
	12	471	18412727	0,11	0,1	0,2	0,5
thermomech_TK	–	102138	2667823445	–	–	–	–
	1	277	17908476	0,02	0,1	0,1	–
	2	410	18234174	0,05	0,1	0,1	1,0
	3	428	18312035	0,06	0,1	0,2	0,5
	4	468	18456816	0,08	0,1	0,2	0,5
	5	486	18523021	0,09	0,1	0,2	0,5
	8	473	18497516	0,11	0,2	0,3	0,3
	12	485	18468534	0,10	0,2	0,3	0,3
2cubes sphere	1	4937	270318206	0,03	0,1	0,1	–
	2	6606	286184506	0,12	0,1	0,2	0,5
	3	6642	283332237	0,13	0,1	0,2	0,5
	4	8080	282900409	0,16	0,1	0,3	0,3
	5	8428	291220902	0,19	0,1	0,3	0,3
	8	8713	289092212	0,25	0,2	0,4	0,3
	12	8683	289491306	0,22	0,2	0,4	0,3
shipsec1	1	5900	451454163	0,07	0,2	0,3	–
	2	8702	470534586	0,44	0,2	0,6	0,5
	3	9431	473917632	0,58	0,2	0,8	0,4
	4	9355	477165207	0,72	0,2	0,9	0,3
	5	9407	479416239	0,91	0,2	1,1	0,3
	8	10334	482618706	1,13	0,3	1,4	0,2
	12	10186	483220152	1,02	0,2	1,2	0,3
thermal1	1	237	12279187	0,01	0,1	0,1	–
	2	336	12342710	0,04	0,1	0,1	1,0
	3	324	12294996	0,05	0,1	0,1	1,0
	4	409	12533236	0,06	0,1	0,2	0,5
	5	408	12442987	0,07	0,1	0,2	0,5
	8	416	12456548	0,09	0,1	0,2	0,5
	12	391	12423054	0,10	0,1	0,2	0,5

Table 4- Resultados de execuções de implementação simples por OpenMP da heurística RBFS-GL aplicada a duas instâncias.

Instância	#	β	*Profile*	Tempo (s)			Speed-down
				RBFS	GL	RBFS-GL	
qa8fm	1	1992	76090237	0,02	0,1	0,1	–
	2	2792	83953119	0,10	0,1	0,2	0,5
	3	2925	85579082	0,12	0,1	0,2	0,5
	4	3416	88884503	0,15	0,1	0,2	0,5
	5	3568	89835420	0,19	0,1	0,3	0,3
	8	3640	91523063	0,23	0,1	0,3	0,3
	12	3678	91896695	0,24	0,1	0,3	0,3
Andrews	1	16625	493969130	0,02	0,1	0,1	–
	2	19368	472553205	0,08	0,1	0,1	1,0
	3	22990	514339951	0,09	0,1	0,1	1,0
	4	27698	529801374	0,10	0,1	0,1	1,0
	5	25803	497314083	0,10	0,1	0,2	0,5
	8	29104	530075752	0,12	0,1	0,2	0,5
	12	29211	540165748	0,13	0,1	0,2	0,5

Como trabalhos futuros, pretendemos melhorar essa implementação paralela das heurística RBFS-GL [Gonzaga de Oliveira et al., 2018]. Com a utilização da estrutura *bags* [Leiserson and Schardl, 2010] na implementação da heurística, são esperadas melhorias no desempenho computacional do algoritmo. Pretende-se também implementar em paralelo o algoritmo de George and Liu [1981], utilizado para encontrar vértices pseudo-periféricos.

Agradecimentos

Este trabalho foi realizado com o apoio da Fundação de Amparo à Pesquisa do Estado de MG (FAPEMIG).

REFERÊNCIAS

T. A. Davis and Y. Hu. SuiteSparse Matrix Collection. *ACM Transactions on Mathematical Software*, 38(1):1–25, 2011. doi: 10.1145/2049662.2049663.

A. George and J. W. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice Hall Professional Technical Reference, New York, USA, 1981.

S. L. Gonzaga de Oliveira, J. A. B. Bernardes, and G. O. Chagas. An evaluation of reordering algorithms to reduce the computational cost of the incomplete Cholesky-conjugate gradient method. *Computational & Applied Mathematics*, 37(3), 2018.

C. E. Leiserson and Tao B. Schardl. A work-efficient parallel breadth-first search algorithm (or how to cope with the nondeterminism of reducers). In *SPAA*, pages 303–314, New York, NY, USA, 2010. ACM. doi: 10.1145/1810479.1810534.

A simple OpenMP-based implementation of a variant of the breadth-first search

Abstract. *A recent publication proposed a serial variant of the breadth-first search ordering with the objective of reducing the total execution cost of linear system solvers. This work shows that a simple OpenMP-based implementation of this variant of the breadth-first search procedure does not speed up its serial version.*

Keywords: *Bandwidth reduction, OpenMP, sparse matrices.*