Paralelização do Software de Análise Estrutural Frame3DD em Arquitetura Multicore

Renato P. Ferrari[1], **Andrea S. Charão**[1]

[1]Laboratório de Sistemas de Computação
Universidade Federal de Santa Maria – UFSM

rpferrari@gmail.com, andrea@inf.ufsm.br

Resumo. *Este trabalho apresenta uma abordagem de paralelização de um software de análise estrutural denominado Frame3DD. Este software tem código aberto e é utilizado em engenharia civil e áreas afins. A paralelização utiliza OpenMP e tem como alvo arquiteturas* multicore*. O desempenho obtido ficou abaixo do ideal, mas obteve-se redução no tempo de processamento com poucas alterações no código original.*

1. Introdução

A paralelização de um software é uma alternativa para reduzir seu tempo de execução, ou seja, aumentar seu desempenho, fazendo com que diferentes partes do código sejam executadas de forma simultânea em várias unidades de processamento.

Uma das abordagens de paralelização utiliza o conceito de *threads*, que em execução compartilham memória mas podem ser alocados a processadores diferentes. A programação de um software usando *threads* é uma prática antiga e implementada por várias linguagens de programação, tornando-se recentemente mais popular com a ampla disseminação de arquiteturas *multicore*.

O Frame3DD [Gavin 1992] é um software de código aberto, utilizado para análise de estruturas reticuladas. Dependendo da estrutura analisada, o tempo de processamento pode ser significativo. Este software foi projetado para arquiteturas mono-processadas e por isso não aproveita múltiplos núcleos de processamento que porventura estejam disponíveis. Sua adaptação para arquitetura *multicore* permitiria que o software fosse utilizado em estruturas complexas que apresentam maior número de detalhes estruturais e, em consequência, com resultados mais precisos. Com esta motivação, este trabalho apresenta uma abordagem de paralelização do Frame3DD, tendo como alvo arquiteturas *multicore*. A seção 2 apresenta uma breve fundamentação do trabalho, enquanto as seções 3 e 4 apresentam, respectivamente, o processo de paralelização e os resultados obtidos.

2. Fundamentação

2.1. Frame3DD

O software Frame3DD (*Static and Dynamic Structural Analysis of 2D and 3D Frames*) foi desenvolvido pelo Departamento de Engenharia Civil e Ambiental da Duke University e tem seu código-fonte aberto para estudo e alterações.

Em sua implementação, o software requer a resolução de sistemas lineares, através do algoritmo de decomposição LU. Esta tarefa demanda intenso esforço computacional,

que é proporcional às dimensões da estrutura analisada. Por exemplo, a análise de uma estrutura com 1932 barras e 976 nós requer 250 segundos de processamento em um computador com processador AMD FX(tm)-8120 Eight-Core Processor com frequência de 3.1 GHz e 8 GB de memória.

2.2. OpenMP

OpenMP [OpenMP Architecture Review Board 2011] (do inglês *Open Multi-Processing*) é uma interface para programação em arquiteturas paralelas com memória compartilhada, disponível para múltiplas linguagens e múltiplas plataformas. É constituída por um conjunto de diretivas de compilador, rotinas de biblioteca, e variáveis de ambiente que influenciam o comportamento do tempo de execução.

Neste trabalho, adotou-se OpenMP como ferramenta de paralelização pela sua facilidade de aplicação a um código já existente. De fato, as diretivas OpenMP causam mudanças mínimas no código fonte, podendo ser interpretadas como comentário caso o código seja compilado sem suporte a OpenMP. O compilador C/C++ utilizado neste trabalho foi o GCC que implementa a OpenMP 3.1 desde sua versão 4.7 e é distribuído de forma gratuita em http://gcc.gnu.org/.

3. Paralelização

A paralelização de código com OpenMP consiste basicamente em adicionar diretivas de compilação em pontos críticos, ou seja, que demandam maior tempo de processamento. Para descobrir estes pontos, pode-se usar ferramentas de *profiling*, que medem os tempos de execução das funções existentes em um programa, possibilitando através dessa análise a otimização do seu código fonte. Para realizar o *profiling* do Frame3DD, usou-se a ferramenta gprof, disponível no compilador GCC através da flag de compilação -pg. A estrutura utilizada como entrada para o programa foi aquela mencionada na seção 2.1. Os resultados de *profiling* são resumidos na tabela 1, onde visualiza-se o nome de algumas funções que dominam o tempo de execução.

Tabela 1. Resultados de *profiling* do aplicativo

Flat pro file:			
% time	cumulative seconds	self seconds	name
52.16	130.54	130.54	ldl_dcmp
28.88	202.82	72.28	ldl_mprove
9.76	227.25	24.42	xtAx
6.99	244.75	17.50	prodAB
1.45	248.39	3.64	pseudo_inv
0.20	248.89	0.50	main
0.14	249.25	0.36	assemble_K
0.06	249.41	0.16	end_forces
0.06	249.55	0.14	rotate

As funções escolhidas para paralelização encontram-se no arquivo matrix.c do código fonte do Frame3DD. Nessas funções, buscou-se analisar laços que pudessem ser paralelizados com diretivas OpenMP. A tabela 2 lista os trechos de código paralelizados em cada função. Em algumas funções na tabela 2, criou-se uma variável auxiliar

(aux) para que fosse possível aplicar a cláusula reduction, que não suporta A[j,i] como argumento. A cláusula reduction, faz operações, com a lista de variáveis localizadas nas threads colocando o resultado em uma cópia privada em cada thread. Com esta cláusula foi possível somar os valores de cada cópia calculados pelas threads e reduzi-los a um único valor. O código original é apresentado como comentário para fins de documentação das alterações feitas.

Tabela 2. Trechos de código paralelizados

Função	Linha do código	Fragmento do código

lu_dcmp	88	for (i = k+1; i <= n; i++) {
		A[i][k] /= pivot;
		#pragma omp parallel for private(j)
		for (j=k+1; j <= n; j++) A[i][j] -= A[i][k]*A[k][j];
		}

ldl_dcmp	157	#pragma omp parallel for private(k) reduction (-:aux)
		for (k=m; k < i; k++){
		// A[j][i] -= A[j][k]*A[i][k];
		aux -= A[j][k]*A[i][k];
		}

ldl_dcmp	167	#pragma omp parallel for private(i) reduction (-:aux)
		for (i=m; i < j; i++) {
		//d[j] -= A[j][i]*A[j][i]/d[i];
		aux -= A[j][i]*A[j][i]/d[i];
		}

ldl_dcmp	190	#pragma omp parallel for private(j) reduction (-:aux)
		for (j=1; j < i; j++){
		aux -= A[i][j]*x[j]; //x[i] -= A[i][j]*x[j];
		x[i]=aux; }

ldl_mprove	249	#pragma omp parallel for private(j) reduction (-:sdp)
		for (j=1;j<=n;j++) {
		if (i <= j) sdp -= A[i][j] * x[j];
		else sdp -= A[j][i] * x[j];
		}
		r[i] = sdp;

xtAx	507	#pragma omp parallel for reduction (+:aux)
		for (k=1; k<=N; k++) {
		if (i <= k) aux += A[i][k] * X[k][j]; // AX[i][j] += A[i][k] * X[k][j];
		else aux += A[k][i] * X[k][j]; // AX[i][j] += A[k][i] * X[k][j];
		}
		AX[i][j]=aux;

xtAx	519	#pragma omp parallel for reduction (+:aux)
		for (k=1; k<=N; k++)
		aux += X[k][i] * AX[k][j]; //
		// C[i][j] += X[k][i] * AX[k][j];
		C[i][j]= aux;

4. Resultados

O software paralelizado foi executado em um computador com 8 *cores*. Os tempos médios de 10 execuções, variando-se o número de threads de 1 a 8, são mostrados na tabela 3. Como métrica para avaliar o desempenho do programa paralelo, utilizou-se a aceleração ou *speedup*, dado por $S(N) = T(1)/T(N)$, onde $T(1)$ é o tempo de execução com 1 processador e $T(N)$ é o tempo de execução em paralelo com N processadores (núcleos). O gráfico na figura 1 apresenta o *speedup* obtido para cada número de *threads*.O *speedup* obtido foi abaixo do ideal, embora tenha-se obtido redução no tempo de execução.É importante ressaltar que a abordagem de paralelização utilizada concentrou-se apenas em laços na parte crítica do programa, exigindo alterações mínimas no código original. Outra abordagem possível seria reprojetar o algoritmo utilizado para resolução de sistemas lineares (decomposição LU), transformando-o num algoritmo paralelo. Essa abordagem,

Tabela 3. Tempos de processamento

Threads	Tempo (seg)					
	lu_dcmp	ldl_dcmp	ldl_mprove	xtAx	Total	Desvio padrão
1	7	128	75	25	250	2.81
2	5	90	70	20	214	2.72
3	4	77	66	18	202	2.55
4	4	68	62	17	177	2.41
5	4	74	67	17	175	2.20
6	4	67	70	15	160	2.40
7	3	60	72	15	155	2.18
8	5	65	71	18	168	2.13

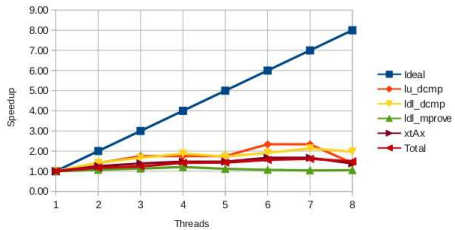

Figura 1. *Speedup* do programa paralelizado

no entanto, exigiria um maior esforço de alteração do código. Levando-se em conta que o software não foi projetado para executar em paralelo e que somente inseriu-se diretivas OpenMP no código existente, os resultados podem ser considerados satisfatórios.

5. Conclusão

Neste trabalho, utilizou-se OpenMP para reduzir o tempo de processamento de um software de análise estrutural. O aumento da velocidade de execução de um sistema abre possibilidades na área da engenharia, às vezes pouco exploradas em programas desenvolvidos para computadores mono-processados. Como trabalhos futuros para melhorar o desempenho do Frame3DD, pode-se investigar o emprego de bibliotecas paralelas de resolução de sistemas lineares, em substituição ao código original. Outra linha de investigação seria o emprego de GPU para acelerar o processamento numérico no Frame3DD.

Referências

Gavin, H. P. (1992). Frame3DD – static and dynamic structural analysis of 2d and 3d frames. Disponível em: `http://frame3dd.sourceforge.net` Acesso em: 18 jul. 2012.

OpenMP Architecture Review Board (2011). OpenMP application program interface version 3.1. Disponível em: `http://www.openmp.org/mp-documents/OpenMP3.1.pdf` Acesso em: 18 jul. 2012.