Uso do C-XSC com OpenMP em Plataformas Multicore

Viviane Linck Lara[1], Luis Paulo Arendt[1], Carlos Amaral Hölbig[1]

[1]Curso de Ciência da Computação – ICEG
Universidade de Passo Fundo (UPF)
Campus 1 – BR 285 – Bairro São José – CEP 91501-970 – Passo Fundo – RS – Brasil

vivillara@hotmail.com, luigi_piazao@yahoo.com.br, holbig@upf.br

Resumo. *Com o uso da biblioteca intervalar e de alta exatidão C-XSC em ambientes paralelos (ambientes do tipo clusters e utilizando a biblioteca de troca de mensagens MPI) sendo consolidado e com a popularização das plataformas multicores observou-se a possibilidade de integrar a biblioteca C-XSC neste tipo de ambiente computacional. Devido a estes fatores foi realizado um estudo sobre programação paralela e ferramentas que possibilitariam a paralelização de programas. Dentre as ferramentas pesquisadas foi escolhida a API OpenMP e a partir da escolha foi feita a integração da OpenMP com o C-XSC. Testes foram elaborados para comprovar a validade e eficiência da integração.*

1. Introdução

A biblioteca C-XSC foi elaborada em C++ e é direcionada ao desenvolvimento de algoritmos que necessitam de resultados em alta exatidão (Hölbig, 2005). Esta biblioteca trabalha com classes que implementam vários tipos de dados numéricos que foram desenvolvidos para o trabalho com valores em alta exatidão e com intervalos de dados, pois os tipos de dados convencionais não suportam estes tipos de dados como, por exemplo, o tipo de dado `interval` (intervalo de reais), `complex` (número complexo), `rvector` (vetor de números reais), `dotprecision` (acumulador real de alta precisão), entre outros. Para estes tipos de dados também foram desenvolvidos operadores especiais. Nos últimos anos, pesquisas que abordaram o uso da biblioteca C-XSC em ambientes paralelos tem direcionado seus esforços à sua utilização em clusters de computadores, integrada com a biblioteca de troca de mensagens MPI, conforme pode ser encontrado em diversos trabalhos desenvolvidos em universidades alemãs e brasileiras (Grimmer 2005, Zimmer 2009, Holbig 2005).

Atualmente, com a popularização de computadores com processadores multicore e com a grande evolução desta tecnologia foi proposta a ideia de unir a biblioteca C-XSC e aproveitar as vantagens das plataformas Multicore. Existem vários tipos de API (*Application Program Interface*) que podem ser utilizadas para implementar programas em paralelo, entre elas foi optado pelo estudo da Phtreads e da OpenMP (Chapman, 2007).

Na seção 2 é apresentado o estudo que foi realizado sobre a integração entre o C-XSC e a API OpenMP. A seção 3 apresenta os testes realizados com os algoritmos propostos. E, complementando a análise dos testes realizados, a sessão 4 apresenta as conclusões e sugestões de continuidade desta pesquisa.

2. Integração CXSC e OpenMP

Para compilar o C-XSC com a OpenMP apenas é necessário unir os comandos dos dois, abaixo é apresentado o exemplo de como deve ser feita a compilação:

```
g++ -fopenmp -I/usr/local/cxsc/include -L/usr/local/cxsc/lib -Wl,-R/usr/local/cxsc/lib <<nome_do_programa>>.cpp -lcxsc -o <<nome_do_executável>>
```

A maioria dos tipos de dados do C-XSC funciona corretamente após integrar o OpenMP com C-XSC. Entretanto para manter a qualidade numérica são necessários cuidados especiais. As variáveis do tipo `opdotprec` (que definem a precisão a ser utilizado pela biblioteca) se declaradas dentro de uma região paralela agem sobre as todas as threads e isso pode acarretar perda de valores ou gerar valores indefinidos (Zimmer, 2009). Outro cuidado que se deve ter é apresentado na figura 1. Em um código sequencial onde é utilizanda a função `accumulate` para se obtenha o resultado de um produto escalar ótimo (com apenas um arredondamento) seria apenas necessário arredondar a variável `accu` (variável de alta precisão) utilizando a função `rnd` (arredonda o resultado armazenado em varáveis de alta exatidão para uma variável de 64 bits, de acordo com o padrão IEEE-754). Porém na hora de implementar um código paralelo são necessárias modificações, pois se a lógica for mantida a variável `result` (do tipo real – 64 bits) receberá o valor apenas da ultima thread e não do somatório em si. Então após todo o somatório há uma região crítica em que as threads adicionam seus resultados nessa variável que após todas as inserções será arredondada mantendo assim a qualidade numérica.

```
void dinamico(int threads) {
   int chunk = CHUNK;
   accu = 0.0;
   accu2= 0.0;
   result = 0;
   #pragma omp parallel num_threads(threads) shared(A,B,chunk,result,accu2) private(accu,j)
   {
    #pragma omp for schedule(static,chunk)
      for (j=1; j<=n; j++){

          accumulate(accu, A[j],B[j]);
     }
          #pragma omp critical
          accu2 = accu2 + accu;

    }
    result = rnd(accu2);
}
```

Figura 1. Código representando a utilização da região crítica para o resultado permanecer com qualidade numérica.

Integrar OpenMP e C-XSC é possível sem muita complexidade, porém é necessária atenção durante a implementação, por isso, é importante saber sobre as diretivas, pragmas, rotinas e variáveis de ambiente. Só através deste conhecimento que se torna viável a obtenção da qualidade numérica aliada ao alto desempenho utilizando programação paralela.

3. Testes e Resultados

Os testes foram realizados em um computador com processador Intel i7 920 com 2.66 GiHZ de frequência de clock, 8 MiB de memória Cache, e 8GiB de memória RAM. O sistema operacional utilizado foi o Ubuntu, versão 10.04 para sistemas de 64 bits. O compilador utilizado é o GCC 4.4.3. A versão do C-XSC instalada foi a 2.4.0 e a versão da OpenMP atual é a 3.0.

Para comprovar a integração entre o C-XSC e a OpenMP foram elaborados diversos testes comparando o tempo de execução entre um programa sequencial e um em paralelo, comparando também diferenças no histórico das CPUs destes dois tipos de programas, além de outros testes comparando o uso das diretivas `for` e `sections`. O programa utilizado para demonstrar os resultados efetua um cálculo da multiplicação entre vetores de tipo intervalar de ordem 100.000.000. O programa foi executado utilizando de 1 a 8 processadores. A função `dinâmico(threads)` que recebe como parâmetro o valor da variável inteira i que é utilizado na cláusula `num_threads` possibilita forçar a primeira execução apenas com uma CPU sendo utilizada. Deste modo a execução do programa aparenta ser sequencial, porque só se utiliza uma CPU ao máximo. Após a primeira execução do cálculo a variável `i` que envia o parâmetro da função é incrementada e assim a função recebe agora o valor dois como parâmetro e através disso inicia a paralelização, porém utilizando apenas dois processadores enquanto os outros 6 estão ociosos, representando um computador com apenas dois núcleos de processamento. E assim incrementa-se o número de processadores utilizados de um a oito, possibilitando assim a comparação de tempo entre a relação de CPUs utilizadas em paralelo e o tempo de execução.

A Figura 2 apresenta o speedup[1] em relação a concentração de CPUs entre cinco tipos de programas. As três utilizando a diretiva `for` onde cada programa possuiu um tipo de escalonamento (`dynamic`, `static` e `guided`). Um programa foi implementado utilizando a diretiva `sections` e outro foi implementado em pthreads. Foram realizadas 8 execuções de cada programa onde a primeira utilizava apenas uma CPU concentrada, simulando assim uma execução sequencial. A média de tempo de execução sequencial foi de 7,8 segundos, Após cada execução incrementava-se uma CPU concentrada, onde a primeira execução era com uma CPU, a segunda com duas, até a oitava onde se se utilizou todas as CPUs disponíveis. A qualidade numérica em todos permaneceu igual à execução sequencial. O menor tempo de execução obtido (1,32 segundos) ocorreu quando utilizamos oito processadores concentrados através do uso da OpenMP com a diretiva `for` e escalonamento do tipo `guided`. O *speedup* desta execução foi de 5,89.

[1] SpeedUp – É obtido, através da divisão entre o tempo sequencial e o tempo paralelo. Representa a aceleração obtida

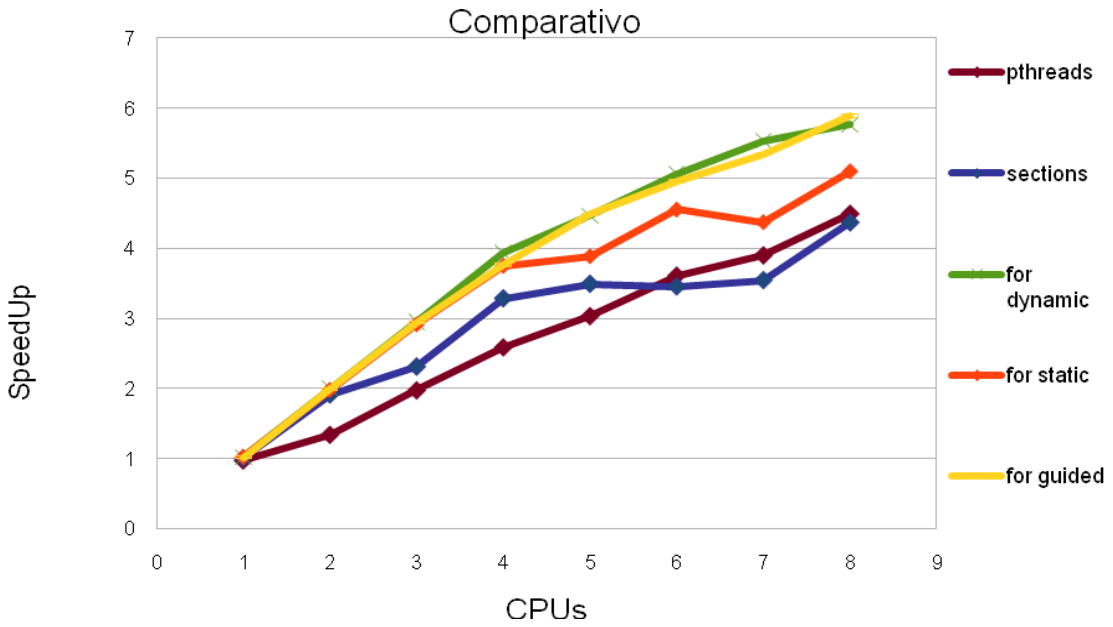

Figura 2. Comparativo entre o speedup das cláusulas e diretivas relacionadas ao número de processadores utilizados.

4. Conclusões

É possível utilizar a biblioteca C-XSC e a API OpenMP integradas. O aumento do desempenho é considerável como pode-se observar com os resultados de *speedup* obtidos. A qualidade numérica é mantida se houver cuidados especiais em relação as variáveis que devem ser privadas, compartilhadas e as regiões críticas para não se perder valores no decorrer da paralelização. Então futuramente podemos analisar o desenvolvimento de funções que fazem a paralelização das operações sobre matrizes e vetores do C-XSC automaticamente. Também pode-se pensar em utilizar a OpenMP para aumentar o desempenho de implementações já feitas no grupo de pesquisa atualizando assim o Solver BAND para a resolução de sistemas lineares esparsos.

Referenciais Bibliográficos

CHAPMAM, B., JOST, G., PAS, R. V. D. Using OpenMP. Cambridge: MIT Press, 2007.

GRIMMER, M. An MPI Extension for the Use of C-XSC in Parallel Environments. Universität Wuppertal, Preprint BUW - WRSWT 2005/3, 2003. Wuppertal, 2005.

HÖLBIG, C. A. Ambiente de Alto Desempenho com Alta Exatidão para a Resolução de Problemas. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2005.

ZIMMER, M;KRÄMER, W; HOFSHUSTER,W; Using C-XSC in High Performance Computing, BUW-WRSWT 2009/5, Universität Wuppertal, 2009