Análise de compiladores com suporte a OpenMP

Adriano Bonat, Filipe Giusti

[1]Departamento de Informática – Universidade Federal de Pelotas (UFPel)
Pelotas – RS – Brazil

{adrianob_ifm,fgiusti.ifm}@ufpel.edu.br

Resumo. *Este artigo faz uma comparação, quanto ao suporte a OpenMP, dos três compiladores mais utilizados hoje em dia. Faz também uma análise dos dados extraidos de benchmarks com os três compiladores, variando os níveis de otimização providos para cada compilador.*

1. Introdução

Com a crescente oferta de processadores multi-core pelo mercado, há também a crescente busca pelos desenvolvedores de software para que seus programas rodem com a máxima performance oferecida pela disponibilidade de múltiplos cores de processamento. Há também outras arquiteturas de computadores como SMPs e clusters, e desenvolver programas com suporte a várias arquiteturas, incluindo a multi-core, é uma tarefa que demanda o estudo de cada arquitetura e a adaptação do código-fonte para cada uma delas, uma tarefa complexa e demorada.

A especificação OpenMP define um conjunto de diretivas de compilação, chamadas de função e variáveis de ambiente, de forma que a tarefa de paralelizar o código para uma arquitetura diferente seja uma responsabilidade delegada ao compilador dessa arquitetura em que se deseja rodar o programa, dessa forma não há alteração significativa no algoritmo para que o mesmo obtenha os benefícios de arquiteturas que ofereçam paralelismo, mas que também permita que o mesmo programa siga funcionando de forma seqüencial.

Este artigo apresenta algumas experiências no uso de três compiladores que oferecem suporte a especificação OpenMP, são eles: Intel Compiler, GCC OpenMP e Sun Studio. Todas as experiências com estas ferramentas ocorreram em uma máquina dual-core rodando o sistema operacional GNU/Linux.

Para tanto foi utilizado o algoritmo de Jacobi para resolução de sistemas, este foi compilado nos três compiladores utilizando diferentes níveis de otimização, e comparando os tempos de execução entre estes.

2. Características dos compiladores

Nesta seção iremos fazer uma comparação dos recursos oferecidos por cada um dos compiladores. E as suas implementações da especificação OpenMP.

2.1. Intel Compiler

Este compilador pode ser obtido pelo seguinte endereço http://www.intel.com/cd/software/products/asmo-na/eng/284132.htm. O registro

precisa apenas de nome e e-mail e com isso consegue-se uma licença para utilização de 30 dias.

O código executável gerado por este compilador possui referencia para ligar em tempo de execução com as bibliotecas libguide e libpthread.

Para compilar algum código-fonte, primeiro deve-se setar as variáveis de ambiente necessárias para o compilador:

$ source diretório bin da instalação do compilador/iccvars.sh

Após, para realizar a compilação com suporte a OpenMP, utiliza-se o seguinte comando:

$ cc codigo.c -o saida -openmp

2.2. GOMP: GCC OpenMP

Apartir da versão 4.2 do GCC foi adicionado suporte a uma implementação de OpenMP para as linguagens C, C++ e Fortran 95. Para se compilar um código-fonte que utilize OpenMP utiliza-se o seguinte comando:
$ gcc -o saida programa.c -fopenmp

Desta forma o executável ”saida”estará ligado a biblioteca ”libgomp”e a sua execução pode ser modificada através das variáveis de ambiente que a especificação OpenMP define, entre outras que a GOMP provê fora da especificação, como: GOMP_CPU_AFFINITY e GOMP_STACKSIZE. A primeira permite associar a execução de threads a processadores, e a segunda permite alterar o tamanho padrão da pilha das threads criadas.

2.3. Sun Studio

O compilador da Sun também é gratuito e pode ser obtido no seguinte endereço http://developers.sun.com/sunstudio/downloads/index.jsp, é exigido apenas que o usuário se cadastre no programa SDN da Sun.

O binário gerado por este compilador liga-se em tempo de execução com duas bibliotecas, libmtsk e libpthread.

A compilação com suporte OpenMP pode ser feita utilizando o seguinte comando:
$ cc codigo.c -o saida -xopenmp

Uma funcionalidade interessante provida por este compilador é a possibilidade de se produzir um algorítmo seqüêncial e deixar a tarefa de paralelização do código para o compilador [van der Pas 2007], este analisa o código-fonte a procura de padrões conhecidos que possam ser paralelizados, fazendo também a identificação das dependências entre variáveis, e então decidindo se a paralelização do código identificado é interessante ou não. Essa funcionalidade é utilizada através da seguinte forma:

$ cc codigo.c -o saida -xautopar

3. Benchmarks

Para realizar as medições foi utilizado um computador Intel Centrino Core Duo 1,67GHz, 2GB DDR2 rodando GNU/Linux com kernel versão 2.6.23.1, com suporte a SMP e kernel com suporte a preempção do código do kernel.

A implementação é um algoritmo interativo de Jacobi que possui alto grau de paralelismo e utiliza intensivamente o processador.

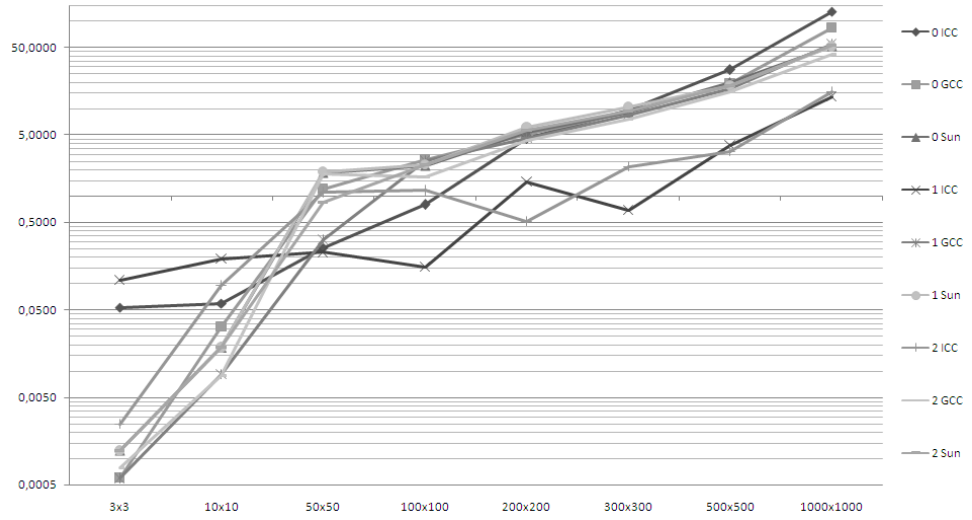

Figura 1. Comparação da performance dos compiladores com níveis de otimização 0, 1 e 2

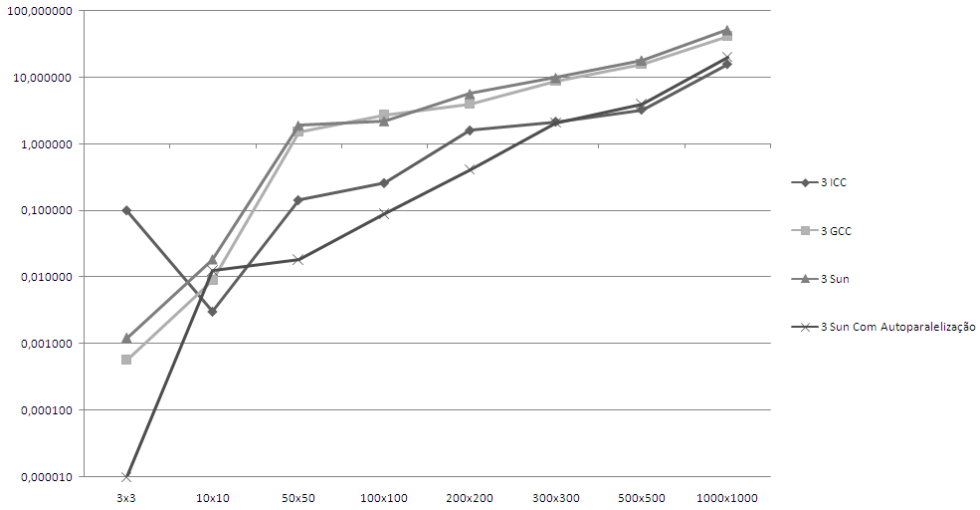

Figura 2. Comparação da performance dos compiladores com nível de otimização 3

As figuras 1 e 2 apresentam os tempos obtivos para a execução com 2 threads de serviço. A figura 2 destaca os tempos obtidos com o nível 3 de otimização juntamente

com o desempenho obtido utilizando o recurso de auto-paralelização do compilador da Sun.

O compilador da Sun sempre compila utilizando otimização O3 quando compilando com suporte a OpenMP.

Métodos numéricos iterativos buscam encontrar a solução de um sistema Ax = b, onde A é uma matriz de coeficientes, x é um vetor de incognitas a ser determinado e b é o vetor de termos independentes. Através de sucessivas aproximações iterativas os métodos conseguem atingir uma solução aproximada da exata. No entanto, este processo pode ser bastante demorado, dependendo do tamanho do sistema a ser resolvido. Uma solução para melhorar o desempenho de métodos numéricos iterativos é a utilização de implementações paralelas, o que possibilita encontrar a solução de forma mais rápida, dividindo a carga de trabalho entre um número maior de processos [Keys 2000].

4. Conclusões

Como o problema de encontrar a solução de um sistema pelo método de Jacobi é um problema de paralelização de dados, a utilização de OpenMP é indicada, visto que através de suas diretivas, é muito simples se paralelizar os cálculos, obtendo bons resultados muito rapidamente. A geração dos blocos de código, geralmente utilizando pthreads, é totalmente deixada a cargo do compilador. Outra vantagem de se ter uma abstração maior, é que o código que o compilador gera ao analisar as diretivas OpenMP pode utilizar simples diretivas fork/join, como também utilizar a biblioteca pthreads, que é o caso nos testes realizados neste artigo.

Em relação aos resultados dos testes, o compilador que obteve os melhores tempos com uma paralelização explícita, ou seja, onde o algoritmo teve de ser alterado, como para a inserção das diretivas OpenMP, foi o compilador da Intel, seguido pelo novato compilador GCC. Temos que ressaltar também um resultado muito bom através da funcionalidade de auto-paralelização provida pelo compilador da Sun, que se mostra como uma boa solução quando se necessita testar se algum algoritmo pode obter benefícios pela paralelização da execução.

Referências

Keys, D. E. (2000). *Four horizons for enhancing the performance of parallel simulations based on partial differential equations*.

van der Pas, R. (2007). Sun tools for openmp programming.