Avaliação do Suporte à Programação *Multithread* com OpenMP no Compilador GCC

Rodolfo Leffa de Oliveira[1], **Tiago de Albuquerque Reis**[1], **Matheus Anversa Viera**[1,2], **Andrea Schwertner Charão**[1,2]

[1]Laboratório de Sistemas de Computação (LSC)

[2]Programa de Educacão Tutorial (PET) – Curso de Ciência da Computação
Universidade Federal de Santa Maria (UFSM)

{rodox, reis, matheus, andrea}@inf.ufsm.br

Abstract. *OpenMP is an emerging standard for multithreaded programming in shared-memory multiprocessor architectures. In this work, we evaluate the OpenMP support in GCC compiler, which will be officially released as part of GCC version 4.2. To this end, we carried out a performance comparison of OpenMP extensions in GCC and Intel ICC, which supports OpenMP since some years ago. The results indicate that GCC is able to achieve satisfactory performance, thus representing an important Free Software alternative to proprietary compilers for developping OpenMP-based applications.*

Resumo. *OpenMP é um padrão emergente para programação multithread em arquiteturas multiprocessadas com memória compartilhada. Neste trabalho, avalia-se o atual suporte a OpenMP no compilador GCC, que terá essa funcionalidade oficialmente incorporada a partir de sua versão 4.2. Para isso, realizou-se uma comparação de desempenho dos suportes a OpenMP no GCC e no compilador ICC da Intel, que já possui suporte a OpenMP há mais tempo. Os resultados obtidos indicam que o GCC é capaz de obter desempenho satisfatório, constituindo assim uma importante alternativa de Software Livre a compiladores proprietários para desenvolvimento de aplicações com OpenMP.*

1. Introdução

Com a proliferação das arquiteturas *multicore* e SMP (*Symmetric Multiprocessing*), que possibilitam diversos fluxos de execução simultâneos, faz-se cada vez mais necessário o uso de técnicas de programação que explorem eficientemente o potencial destas arquiteturas. De fato, não basta que o *hardware* possua suporte a concorrência, pois o *software* também precisa ser desenvolvido focando esta característica. Em arquiteturas deste tipo, uma técnica comum para exploração de concorrência é a programação *multithread*.

Dentre as alternativas para desenvolvimento de aplicações *multithread*, destaca-se o padrão OpenMP [OpenMP 2006]. Basicamente, OpenMP consiste em uma API (*Application Programming Interface*) para facilitar a programação de aplicações *multithread* com portabilidade e desempenho, nas linguagens C/C++ e Fortran. O suporte a esse padrão foi inicialmente implementado por compiladores proprietários, de fabricantes tais como Intel e Portland Group. Recentemente, o suporte a OpenMP começou a ser incorporado ao compilador GCC, um dos mais importantes projetos de Software Livre da atualidade.

Este trabalho tem como objetivo avaliar o suporte a OpenMP disponível para o compilador GCC, através de uma comparação de desempenho com o compilador ICC da Intel. Esta avaliação é relevante para a comunidade de Software Livre, pois constitui uma experiência de utilização desta nova funcionalidade do GCC. Além disso, este trabalho fornece subsídios quantitativos para programadores que desenvolvem aplicações *multithread* e desejam explorar novas alternativas de programação.

Este artigo encontra-se organizado da seguinte maneira: primeiramente, faz-se uma breve introdução à programação *multithread* utilizando OpenMP. Em seguida aborda-se o compilador GCC, discutindo-se também o suporte ao OpenMP que será disponibilizado a partir da versão 4.2 deste compilador. Na seqüência apresenta-se a avaliação dos compiladores, descrevendo-se os experimentos realizados e os resultados obtidos. Por fim apresenta-se as considerações finais sobre o trabalho.

2. Programação *Multithread* com OpenMP

A programação *multithread* é uma técnica que permite a execução simultânea ou concorrente de múltiplos fluxos de código em um mesmo programa [Berg and Lewis 1996]. Esta abordagem é adequada para a programação em arquiteturas que dispõem de múltiplas unidades de processamento, como máquinas SMP e os atuais processadores *multicore*. Nestas arquiteturas, os processadores ou núcleos compartilham a mesma memória global, e se comunicam através de escritas e leituras nesta memória.

O desenvolvimento de programas *multithread* pode ser feito através de linguagens com suporte nativo a *threads* (p.ex. Java), ou ainda através de APIs tais como POSIX Threads [Nicols et al. 1996], que adicionam o suporte a *threads* a linguagens puramente seqüenciais. Com estas abordagens, faz-se necessário a criação, terminação e sincronização explícita das *threads*.

Com o objetivo de fornecer uma maneira simples de explorar paralelismo sem interferir na estrutura do algoritmo, diferentemente do que ocorre com o uso de POSIX Threads, surgiu o padrão OpenMP. Um programa utilizando OpenMP compila e executa tanto em ambiente de execução paralela como em um seqüencial [GOMP 2006]. O uso de OpenMP também facilita o ato de conversão de um programa seqüencial em um concorrente, possibilitando a paralelização de forma incremental.

Nas seções seguintes apresenta-se um histórico do padrão OpenMP e uma visão geral dos recursos de OpenMP para programação *multithread*.

2.1. Histórico

Devido à falta de um padrão para programação concorrente e paralela em arquiteturas multiprocessadas com memória compartilhada, e após um primeiro esforço não concluido de padronização ANSI em 1994, surge uma iniciativa para o desenvolvimento do OpenMP. Diversos fabricantes, entre eles Compaq, HP, Intel, IBM, Silicon e Sun, formaram o OpenMP Architecture Review Board (ARB) e participaram do trabalho de especificação de sua API.

Embora o padrão ANSI 1994 não tenha obtido sucesso, seus conceitos básicos eram válidos, e o projeto OpenMP foi implementado a partir destes, com adição das características que deixavam a desejar no seu antecessor, como escalabilidade e portabilidade. A primeira versão do padrão, OpenMP para FORTRAN 1.0, foi lançada em outubro

de 1997, sendo a versão para C/C++ lançada no ano seguinte. No ano de 2000 surgiu a versão 2.0 para FORTRAN, e em 2002 esta para C/C++. A versão atual é a 2.5, que combina suporte a FORTRAN e C/C++ e foi lançada em 2005. A versão 3.0 não possui data de lançamento, mas está definida e sendo implementada pelo comitê de desenvolvimento do OpenMP [OpenMP 2006].

2.2. Visão geral de um programa OpenMP

Um programa utilizando a ferramenta OpenMP sofre poucas alterações quando comparado com um programa seqüencial. A programação é baseada em diretivas de compilação, que são inseridas no programa seqüencial indicando para o compilador as partes que podem ser paralelizadas. Esse compilador tem que possuir suporte a OpenMP, e é ele o responsável pela criação das *threads*.

Uma aplicação com OpenMP utiliza o modelo de programação *fork-and-join*, com uma *thread* principal sendo criada no início do programa e a mesma executando sozinha até localizar uma diretiva que permita uma execução paralela, voltando a *thread* principal a executar sozinha após o termino da região paralela [Wilkinson and Allen 1998]. O número de *threads* pode ser especificado através de uma variável de ambiente definida pelo usuário, ou por uma chamada de função do OpenMP. Entre as principais diretivas podemos citar:

- `#pragma omp parallel`: permite a paralelização do bloco de código que se encontra após essa diretiva. No momento em que esta diretiva é invocada, o bloco é executado paralelamente com as *threads* criadas pela *thread* principal, incluindo ela mesma.
- `#pragma omp for`: o laço que se encontra após essa diretiva é executado de forma paralela entre várias *threads*. Nessa diretiva existe uma sincronização implícita, podendo ser assíncrono através do comando *nowait*.

O trecho de código apresentado na figura 1 ilustra o uso de uma diretiva de OpenMP.

```
#include <omp.h>
int main() {
    ...
    #pragma omp parallel
    {
        do_work();
    }
    ...
}
```

Figura 1. Exemplo da utilização de OpenMP

As diretivas apresentadas permitem que o programador defina a distribuição das tarefas. Além disso, existe a diretiva `#pragma omp parallel for` em que o compilador é quem faz a distribução do trabalho [OpenMP, ARB 2005]. Existem outras diretivas, bem como formas de sincronização, mas não é o escopo desse trabalho descrevê-las.

3. OpenMP no Compilador GCC

O GNU Compiler Collection (GCC) é um dos principais pacotes de Software Livre ligados ao projeto GNU. O GCC começou a ser desenvolvido em 1985 por Richard Stallman, como um compilador para a linguagem C. Atualmente, o GCC oferece suporte a várias outras linguagens tais como C++, Objective-C, Fortran e Java. A versão oficial disponível atualmente é a 4.1.2, mas a versão 4.2.0, que é a utilizada neste trabalho, já está disponível e será lançada em breve. É a partir da versão 4.2.0 que o GCC estará oficialmente implementando o suporte a OpenMP. A próxima versão, 4.3.0, também já está em desenvolvimento [GNU 2006].

O suporte a OpenMP no GCC começou a ser desenvolvido por volta de 2002, com uma ramificação do projeto GCC denominada GOMP (GNU OpenMP) [GOMP 2006]. Uma das motivações para o projeto GOMP foi oferecer uma alternativa de Software Livre para o desenvolvimento de aplicações com OpenMP, contribuindo também para manter o GCC alinhado com novas tecnologias e tendências dos compiladores proprietários. Em 2005, os resultados do projeto GOMP foram considerados prontos para serem incorporados ao ramo principal do GCC. Basicamente, o suporte a OpenMP se dá através de uma biblioteca (libgomp) e de extensões aos pré-processadores para as linguagens C, C++ e Fortran.

4. Avaliação de Desempenho

Esta avaliação tem como objetivo uma comparação de desempenho dos suportes a OpenMP nos compiladores GCC e Intel ICC, respectivamente em suas versões 4.2.0 e 9.0. Para esta avaliação, realizou-se três experimentos que são apresentados nas seções a seguir. Todos os experimentos foram realizados em uma máquina com 2 processadores Pentium III de 1Ghz, com 1 GB de memória RAM, executando sistema operacional Linux versão 2.6.6, com distribuição Gentoo.

4.1. Criação de *Threads*

O primeiro experimento realizado foi a execução de um programa que cria uma região paralela, ou seja, cria uma *thread* para executar um bloco de código. Esta operação é feita repetidamente dentro de um laço, a fim de avaliar a criação e inicialização de um fluxo paralelo de execução no código gerado por ambos os compiladores. Também executou-se o mesmo programa utilizando 2 *threads* e, em seguida, realizou-se a execução desse mesmo programa sem OpenMP, para observar o custo que cada compilador agrega à execução.

O número de repetições atribuídas ao laço foram de 1000, 10000 e 100000. Os resultados obtidos, tanto para o GCC como para o ICC são mostrados na tabela 1.

Tabela 1 - Tempos de criação de *threads*

Compilador	1000 repetições	10000 repetições	100000 repetições
GCC (1 thread)	0,001737 s	0,017184 s	0,172491 s
ICC (1 thread)	0,001280 s	0,010572 s	0,103790 s
GCC (2 threads)	0,013855 s	0,136291 s	1,282762 s
ICC (2 threads)	0,003197 s	0,023717 s	0,227047 s
GCC (sem OpenMP)	0,000047 s	0,000465 s	0,004633 s
ICC (sem OpenMP)	0,000007 s	0,000061 s	0,000607 s

Pode-se observar, na tabela 1, que o uso de OpenMP gerou uma sobrecarga na execução do programa quando executado em um só processador (uma *thread* e sem OpenMP), aumentando o tempo de execução em 37 vezes para o GCC com 100000 repetições. Observa-se, também, que para criar a *thread* que será executada no segundo processador (comparando o tempo de uma *thread* com o de duas) o tempo de execução aumentou em até 8 vezes para o GCC com 1000 repetições.

Comparando os compiladores, o ICC se mostrou mais eficiente para esse experimento. De fato, os tempos de execução para o GCC chegaram a ser mais de 5 vezes maiores nos casos com 10000 e 100000 *threads*.

4.2. Multiplicação de Matrizes

O segundo experimento consistiu na execução paralela de um programa que faz a multiplicação entre duas matrizes quadradas. Foram testados três tamanhos diferentes de matrizes, com 512 x 512, 1024 x 1024 e 2048 x 2048 elementos. As matrizes são inicializadas com valores calculados a partir dos índices de linha e coluna de cada elemento. O trabalho é dividido entre 2 *threads*, de forma que cada *thread* calcula um dos elementos da matriz resultante por vez. Para determinar-se o ganho de desempenho obtido com a utilização de OpenMP, executou-se também o mesmo programa com apenas 1 *thread*, de modo que a multiplicação foi feita seqüencialmente. As tabelas 2 e 3 apresentam os tempos de execução obtidos utilizando-se o GCC e o ICC, respectivamente. Na seqüência, a tabela 4 apresenta uma comparação entre os ganhos de desempenho obtidos com cada compilador (relação entre os tempos de execução com 1 e 2 *threads* para cada caso).

Tabela 2 - Tempos de execução utilizando 1 e 2 *threads* com o GCC

Nº *threads*	512 x 512	1024 x 1024	2048 x 2048
1	11,124171 s	90,691185 s	756,789741 s
2	6,546238 s	53,003747 s	491,308279 s

Tabela 3 - Tempos de execução utilizando 1 e 2 *threads* com o ICC

Nº *threads*	512 x 512	1024 x 1024	2048 x 2048
1	8,049439 s	64,876937 s	548,893785 s
2	4,804297 s	38,880329 s	352,978104 s

Nas tabelas 2 e 3, nota-se que ambos os programas, tanto o compilado pelo GCC, quanto o pelo ICC, obtiveram uma diminuição no seu tempo de execução, devido ao fato de terem dois fluxos de execução simultâneos. Os programas paralelizados com OpenMP tiveram *speed-up* entre 1,5 e 1,7, o que confirma a validade desta ferramenta em ambientes onde se faz necessário o alto desempenho das aplicações.

Tabela 4 - Comparação de *speed-up* entre os dois compiladores

Compilador	512 x 512	1024 x 1024	2048 x 2048
GCC	1,70	1,71	1,54
ICC	1,67	1,67	1,59

A tabela 4 mostra que o compilador ICC apresentou uma vantagem em relação ao tempo de execução, também para esta abordagem. Este resultado motivou a comparação dos compiladores com a utilização de *flags* de otimização.

4.3. Compilação utilizando *flags* de otimização

O terceiro experimento realizado foi a execução do mesmo programa de multiplicação de matrizes utilizado anteriormente, desta vez compilando os códigos-fonte com a *flag* de otimização -O3, para ambos os compiladores. A tabela 5 apresenta os resultados da execução dos códigos otimizados.

Tabela 5 - Resultados da execução dos códigos otimizados

Compilador	512 x 512	1024 x 1024	2048 x 2048
GCC	4,849266 s	39,406653 s	347,324485 s
ICC	4,803244 s	38,811151 s	344,681492 s

Na tabela 5, nota-se que com a utilização da *flag* de otimização, o GCC conseguiu equiparar o tempo de execução do programa com o do ICC, que por sua vez obteve pouca melhora.

5. Conclusão

Este artigo apresentou experimentos com o suporte do GCC para programação *multithread* com OpenMP, através da biblioteca GOMP, que estará disponível a partir de sua versão 4.2. Como contribuição, este artigo mostrou que o projeto GOMP, apesar de recente, coloca o GCC como uma alternativa livre viável para os compiladores proprietários, pois já apresenta estabilidade e tem desempenho similar aos mesmos, com a perspectiva de melhora com a maturação do projeto. Os testes realizados podem ser utilizados como referência para desenvolvedores de aplicações *multithread*. Como sugestão para trabalhos futuros, pode-se aprofundar a avaliação realizada através de um comparativo entre POSIX Threads e OpenMP usando o GCC.

Referências

Berg, D. and Lewis, B. (1996). *Threads Primer: A Guide to Multithreaded Programming*. SunSoft Press.

GNU (2006). GCC website. `http://gcc.gnu.org/`.

GOMP (2006). GNU OpenMP. `http://gcc.gnu.org/projects/gomp/`.

Nicols, B., Buttlar, D., and Farrell, J. (1996). *Pthreads Programming*. O'Reilly & Associates, Inc., Cebastopol, CA.

OpenMP (2006). Openmp website. `http://www.openmp.org/`.

OpenMP, ARB (2005). OpenMP Application Program Interface. Technical report, OpenMP Architecture Review Board. `http://www.openmp.org/specs`.

Wilkinson, B. and Allen, M. (1998). *Parallel programming: techniques and applications using networked workstations and parallel computers*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA.