Universidade de São Paulo
Instituto de Matemática e Estatística
MAC 5742 - Computação Paralela e Distribuída

Exercício Programa 1: OpenMP

Autores:

Diana Naranjo

Walter Perez

São Paulo

Abril 2015

Resumo

Nesse Exercício Programa o objetivo foi explorar a computação paralela com memoria compartilhada, para isso foi usado o padrão openMP. A primeira parte do trabalho explora o cuidado que deve-se ter no momento de realizar o desenvolvimento de programas usando as diretivas do openMP. É muito simples cometer erros quando ainda se está pensando de maneira sequencial, ao assumir algum comportamento ou quando não se conhece bem o comportamento padrão das diretivas usadas. A segunda parte do EP procura avaliar as melhoras (ou falta delas) no tempo de execução de um programa alvo, mult.c, que realiza a multiplicação de 2 matrizes. Para avaliar o desempenho da versão sequencial versus a paralela uma serie de experimentos foram realizados. Alguns deles involucraram a alterção do programa para criar distintas zonas paralelas e também a execução deles usando diferentes numeros de threads. A continuação presentamos os experimentos, resultados e conclusões.

Sumário

1 Introdução

A computação paralela consiste em executar um conjunto de cálculos de maneira simultanea [3]. A meta de este tipo de computação é diminuir o tempo de execução que exigem algumas aplicações, e.g. aquelas com fortes requerimentos de computo. Atualmente, existem computadores com múltiplos cores; além de isso também contam com tecnologias como hyper-threading, o que gera que o sistema operativo assuma que conta com uma maior quantidade de recursos de computo. Os recursos, por tanto, estão prontos para ser usados. Porém, para realmente obter uma melhoria no rendimento não é suficiente executar as aplicações em uma máquina multi-core, é necessário que elas troquem seus algoritmos sequências por uma versão paralela. De outra maneira, não estaria-se fazendo uso do processamento paralelo e seus beneficios.

O grande problema é que a troca de algoritmos sequências por paralelos não é um trabalho simples. Existem diversos problemas na geração de códigos paralelos de bom rendimento, alguns deles são [4]:

- Gargalos de comunicação. Debe ter-se em consideração a quantidade processadores que serão utilizados na execução e o overhead produto da comunicação entre estes processadores e a memoria [2]. Isto é importante pois pode acontecer que um programa em versão paralela tome mais tempo na execução do que na versão sequencial. O que se debe a que a memoria só pode ser acessada por um thread em qualquer momento e o custo de manter as caches de cada processador coerente também gera um overhead.
- Balanceamento de carga. Ao fazer uma divisão de trabalho o ideal é que cada uma das partes tenha uma carga igual. Se uma delas tem mais trabalho do que as outras então no final o problema volta a ser sequencial pois a maioria termina com sua carga e umas poucas continuam com o resto do trabalho.
- Problemas na construção do código. Os desenvolvedores estão acostumados a pensar em forma sequencial e por tanto é muito fácil cometer erros de concorrência. As operações simples, como por exemplo a assinação de valores a uma variável, causam resultados inesperados na versão paralela. Isto é devido as condições de corrida, i.e. quando o programa se desenvolve em um ordem diferente ao que o programador planeio.

Para evitar os erros produzidos por o pouco cuidado do desenvolvedor e para maximizar o paralelismo dos algoritmos que são sequências existem conjuntos de diretivas que criam um código paralelo executável. OpenMP é um conjunto de diretivas para o compilar, rutinas de libraria e variáveis de entorno que podem ser usadas para a geração de paralelismo em códigos Fortran e C/C++ [1]. O objetivo é garantir o correto funcionamento dos programas e obter os beneficios do paralelismo.

Neste trabalho exploramos os problemas persistentes no uso das diretivas do OpenMP. Tanto no desenvolvimento de código erróneo, como na geração de overhead pela geração de áreas paralelas ineficientes. Para isso desenvolvemos experimentos para obter o tempo promedio na execução de códigos gerados a partir de distintas áreas paralelas, em ambientes diferentes e com distintas quantidades de threads.

2 Exercício 1

Escolher um código na internet que use as diretivas de compilação do openMP, esse código deve ser procurado nos respetivos tutoriais e manuais desse padrão de programação multiprocessamento. A ideia e encontrar erros nessas implementações fornecidas ou apresentadas nos tutoriais consultados. Apresente o código, aponte os problemas e descreva quais são as correções feitas para tirar o erro da aplicação.

2.1 Código com erro

Comando para preservar a formatação do texto.

```
#include <iostream>         // < > is used for standard libraries.
void main(void)             // ''main'' method always called first.
{
 cout << ''This is a message.'';
                            // Send to output stream.
}
```

2.2 Problemas encontrados

2.3 Correções

Comando para preservar a formatação do texto.

```
#include <iostream>         // < > is used for standard libraries.
void main(void)             // ''main'' method always called first.
{
 cout << ''This is a message.'';
                            // Send to output stream.
}
```

2.4 Conclusões

3 Exercício 2

Modique o programa mult.c, que realiza a multiplicação de 2 matrizes. Modique este código para que ele realize a multiplicação utilizando as primitivas de paralelização de openMP. Compare o desempenho com 1, 2, 3 4, 8 e 16 threads. Tente realizar a paralelização no laço for das variáveis i, j e k, explicando no relatorio se o comportamento obtido está correto ou não. Apresente e descreva no relatorio grafos, tabelas e estatísticas dos tempos de execução.

- **Compare o desempenho obtido, explicando por que melhorou ou piorou e compare também a execução do programa em sua versão sequencial.**
- **Um dos objetivos e verificar se algum overhead e inserido pelo ambiente de execução (runtime openMP) quando a versão paralela do programa em openMP executa apenas com uma (1) thread, comparando-se com a versão sequencial (mult.c).**
- **Esse programa deve ser executado em pelo menos dois processadores diferentes, para efeitos de encontrar os intervalos de confiança cada execução deve ser repetida pelo menos 10 vezes.**
- **Outras informacões que julgarem pertinentes ao contexto do trabalho podem ser adicionadas, e poderão ser somadas como pontos adicionais do EP.**

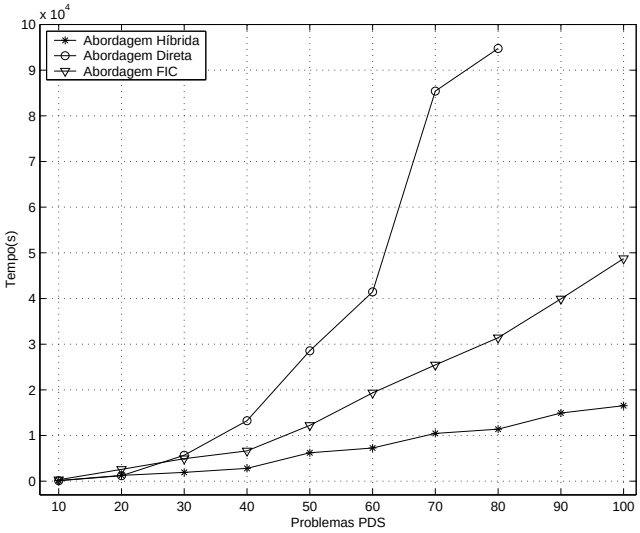
3.1 Experimentos

3.2 Resultados

Problem	CCF preconditioner		Number of nonzeros	
	η	$\frac{n(AD^{-1}A^T)}{nrow}$	FCC	Cholesky
ELS-19	-11	31	87750	3763686
SCR20	-12	31	103179	2591752
NUG15	-12	32	54786	6350444
PDS-20	15	5	625519	7123636

Tabela 1: Título da Tabela.

Referenciando a tabela 1.



3.3 Conclusões

4 Conclusões

Apresentar as conclusões finais.

Referências

[1] OpenMP ARB Corporation. Frequently asked questions openmp, 2013.

[2] F. Gebali. *Algorithms and parallel computing*. Wiley, 2011.

[3] A. Gottlieb and G. Almasi. *Highly parallel computing*. Benjamin-Cummings Publishing Co., 1989.

[4] N. Matloff. *Programming on parallel machines*. University of California, Davis, 2014.

A Anexo I

Comando para preservar a formatação do texto.

```
#include <iostream>          // < > is used for standard libraries.
void main(void)              // ''main'' method always called first.
{
 cout << ''This is a message.'';
                             // Send to output stream.
}
```