Estudo de Linguagens Paralelas para Arquiteturas Multicore

Conference Paper · April 2015

CITATIONS
0

READS
27

1 author:

Raffael Bottoli Schemmer

10 PUBLICATIONS **2** CITATIONS

SEE PROFILE

Estudo de Linguagens Paralelas para Arquiteturas Multicore

Raffael B. Schemmer, André L. Atibola, Júnior F. Barros, Júlio C. S. Anjos
Cláudio F. R. Geyer

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{raffael.schemmer,altibola,jfbarros,jcsanjos,geyer}@inf.ufrgs.br

Resumo. *Este trabalho apresenta um estudo de quatro linguagens paralelas para uma arquitetura multicore usando um problema de multiplicação matricial como estudo de caso. O trabalho conclui que a linguagem Go é aquela que apresenta menor esforço e melhor tempo de execução sendo está uma alternativa a linguagem C com uso de Pthreads.*

1. Introdução, Motivação e Objetivos do Trabalho

A disponibilidade tecnológica permitiu que a partir de um processador que executava instruções de maneira atômica, surgisse o conceito de pipeline e de arquitetura superescalar, que possibilitou a execução concorrente de um mesmo fluxo de instruções. Na atualidade a arquitetura multicore, composta por vários processadores permite a execução paralela de um fluxo de instruções de forma independente. Esta arquitetura possui uma série de vantagens comparada às demais, desde o reuso de hardware do processador durante seu projeto até regularidade do layout. Estas vantagens possibilitam redução de custos e uma melhor eficiência energética.

A proposta da arquitetura multicore é projetada no sentido de que cada módulo de processamento, execute um ou mais fluxos de instruções de maneira concorrente ou paralela. Um processador superescalar explora o paralelismo em nível de instrução, já um processador multicore, em nível de fluxos de instruções (threads) que compõem um processo. O paralelismo em nível de instrução é na grande maioria dos casos explorado de maneira sub ótima pelo compilador. No modelo de desenvolvimento atual é da responsabilidade do programador explorar e encapsular o paralelismo da aplicação em threads ou processos de instruções independentes.

A literatura apresenta diferentes estratégias e soluções tanto em linguagens como em frameworks para exploração do paralelismo em nível de threads e de processos. Também, diferentes são as propostas de arquiteturas multicore existentes, tanto em número de recursos como em complexidade de programação. Este trabalho pretende através destas motivações, realizar um estudo de quatro linguagens paralelas sendo elas (i) C com uso de Pthreads; (ii) Go; (iii) Cilk; (iv) UPC; para um processador de arquitetura multicore. O trabalho têm como objetivo verificar a relação de esforço necessário para paralelização de um problema de multiplicação matricial fazendo uso das linguagens. Também, a aceleração do código paralelo fazendo uso de um processador multicore.

2. Avaliações, Resultados e Conclusões

A escolha das linguagens paralelas foi feita de maneira oportuna, partindo de citações e aceitações em revisões de literatura. Também, por terem correlação com a sintaxe de C e

por serem procedurais. Das linguagens avaliadas apenas C precisou de bibliotecas externas para implementação da concorrência e sincronização. A avaliação das linguagens foi realizada através da implementação de um problema clássico de multiplicação matricial de duas matrizes quadráticas como estudo de caso. A implementação inicial foi proposta em linguagem C com uso da biblioteca Pthreads residindo a complexidade na divisão do problema como na manipulação das threads.

A codificação do problema em Go foi semelhante a codificação em C, uma vez que cada thread também deverá ser encapsulada em uma função do programa. O não uso de ponteiros e a necessidade de apenas chamar a palavra reservada Go para geração de threads foi algo que facilitou o desenvolvimento da aplicação nesta linguagem. Em Cilk, o compilador é quem realiza a geração das threads pela paralelização de estruturas de repetição. A codificação necessita apenas que o programador anote as estruturas de repetição que devem ser paralelizadas através do uso de cilk_for. A última codificação foi feita em UPC. Diferente das demais, UPC dá suporte ao paralelismo em nível de processos e não de threads. Cada processo possui um identificador único, sendo este usado no código fonte para definir quais pontos da matriz devem ser computados.

A avaliação de resultados fez uso de um processador multicore de 4 núcleos de processamento e de matrizes de entrada de um milhão de pontos cada, variando o número de threads dos problemas em uma, duas, quatro, oito e dezesseis threads. A Figura 1 ilustra as duas métricas avaliadas sendo o tempo de execução em segundos (a) e o fator de aceleração (b), ambos normalizados em 30 execuções para redução do desvio padrão.

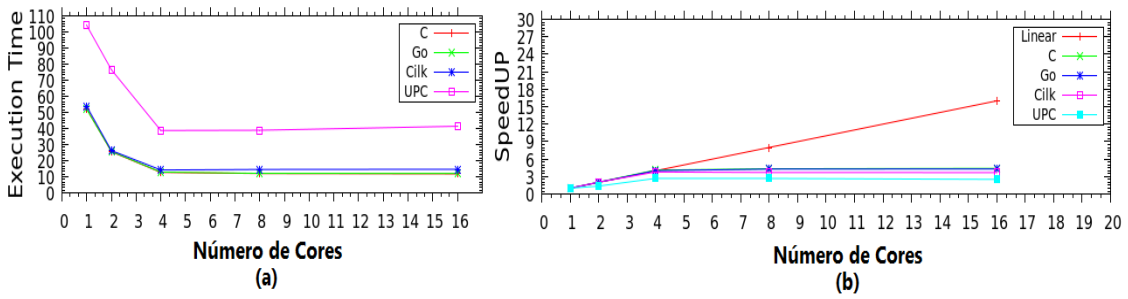

Figura 1. (a) Tempo de execução; (b) Fator de aceleração (SpeedUP);

Das linguagens avaliadas, C e Go apresentaram o mesmo desempenho e aceleração ideal dado a disponibilidade da arquitetura multicore utilizada. Cilk apresentou desempenho próximo a C e Go tanto em tempo como em aceleração. UPC ao contrário das demais, não apresentou tempo de execução e aceleração comparado às demais linguagens avaliadas. C e Go foram as linguagens que consumiram mais linhas de código, também, as que demandaram maior esforço de codificação, porém, as que reportaram melhores resultados em relação a aceleração ideal. Cilk e UPC pelo contrário, consumiram menor esforço em codificação mas não atingiram os mesmos resultados em relação a aceleração ideal comparados às demais linguagens. Os resultados demostram existirem propostas de linguagens sob diferentes óticas, umas no objetivo de buscar menor esforço e flexibilidade e outras no sentido de obtenção máxima de desempenho. Considera-se Go como sendo a linguagem que demonstrou flexibilidade, funcionalidades e desempenho equivalente a ponto de ser utilizada como linguagem moderna e substituta a C com uso de Pthreads.