Programação Paralela - Algoritmo SYMM utilizando UPC + Polybench

Thiago I. Yasunaka RA:103069

I. Introdução

Linguagens de programação quando compiladas e executadas, geralmente são executadas sequencialmente, isso significa que durante a sua execução, existirá apenas um fluxo a ser seguido por este algoritmo implementado. No entanto, a fim de buscar melhores resultados e desempenho nos computadores, surgiram modelos de pro gamação paralela, este paradigma de programação faz com que diversos fluxos sejam executados simultaneamente por um determinado programa, fazendo com que o seu desempenho seja melhorado em grande escala.

Programação paralela é suportada por diversas linguagens, por exemplo, a linguagem C, C++ e entre outras. No entanto, algumas linguagens são construídas especificamente para a programação paralela, como é o caso da linguagem *Unified Parallel C (UPC)*, de UPC. Baseado fortemente na própria linguagem C, o UPC possui diversas ferramentas para trabalhar com o paralelismo, inclusive ele possui suporte para *networks* do tipo MPI, UDP, SMP e IBV. Neste trabalho, o objetivo é utilizar o algoritmo *SYMM* disponível no pacote do *Polybench*Pol e implementar a paralelização por meio da linguagem UPC. O algoritmo *SYMM* nada mais é do que uma multiplicação de matrizes simétrica.

12 de setembro de 2021

II. Metodologia

Para a implementação do algoritmo e para os experimentos realizados, foram utilizados algumas ferramentas que serão descritas na tabela e tópicos abaixo:

- Linguagem utilizada: Unified Parallel C
- Versão UPCC: 2021.4.0, built on Nov 28 2021 at 11:33:02
- Makefile para compilação e execução do algoritmo

Tabela I
Configurações da máquina

S.O	Ubuntu 21.04
Processador	11th Gen Intel® Core™ i7-1165G7 @ 2.80GHz × 4
Memória RAM	16 GB's
Placa de Vídeo	GeForce MX330/PCIe/SSE2

Para simplificar todo o processo de compilar, executar e passar os parâmetros, existe no arquivo Makefile a configuração que faz todo esse processo de forma mais rápida. Para isso, execute este comando no terminal:

```
make clean && make && make upc-symm-run
```

Este comando irá limpar os arquivos compilados, compilá-los novamente e depois exibir o tempo de execução juntamente com a execução do programa.

Também é possível compilar e executar o comando diretamente pela linha de comando, para isso faça:

- **Para compilação com 8 *threads* e network UDP:**

  ```
  upcc symm-par.c utilities/polybench.c -T=8 –network=udp -I utilities -I . -DPOLYBENCH_TIME -o upc_symm_time
  ```

- **Para execução:**

  ```
  upcrun -localhost ./upc_symm_time
  ```

Outra informação relevante são os tamanhos da entrada utilizada para os testes, elas foram alteradas em relação ao padrão *polybench* para que fosse possível medir o tempo de cada execução em um intervalo de tempo maior. Vale destacar que as matrizes são todas de ordem quadrática. Abaixo segue os valores de cada entrada:

- MEDIUM_DATASET: 1250
- LARGE_DATASET: 1500
- EXTRALARGE_DATASET: 1750
- EXTRA_EXTRALARGE_DATASET: 2000

III. Resultados e Discussão

Os resultados adquiridos utilizando a arquitetura de programação paralela fornecida pela linguagem UPC resultaram em uma melhora significativa se comparada com a execução do algoritmo de forma sequencial, porém em diversos casos não obteve-se o resultado esperado com aquilo que seria o esperado teoricamente. Diversos tamanhos de entrada que simulam o processamento foram testados, observe abaixo:

A interface do MPI não suporta que um mesmo computador compartilhe diferentes núcleos virtuais. Por se tratar de um modelo de troca de mensagens e que só é possível simular com núcleos físicos, a única quantidade de *threads* disponíveis para este caso foram quatro *threads*. Quantidades de *threads* menores do que isso não foram suportados pelo UPC devido ao tamanho do bloco de memória para a alocação da matriz. Observe a figura 1, ela apresenta os resultados para o teste MPI com as entradas disponíveis. O resultado do gráfico de *speedup*

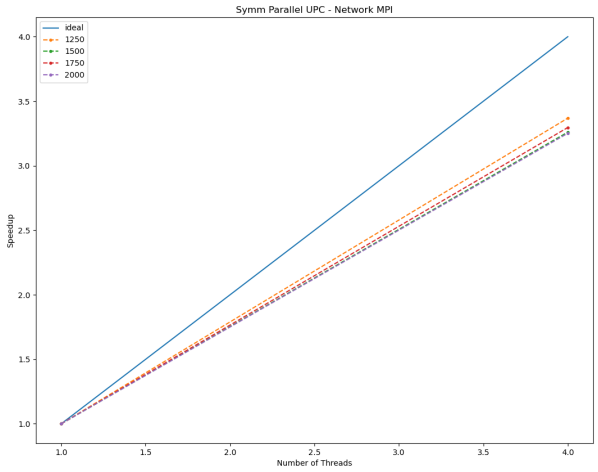

Figura 1. Speedup para network MPI

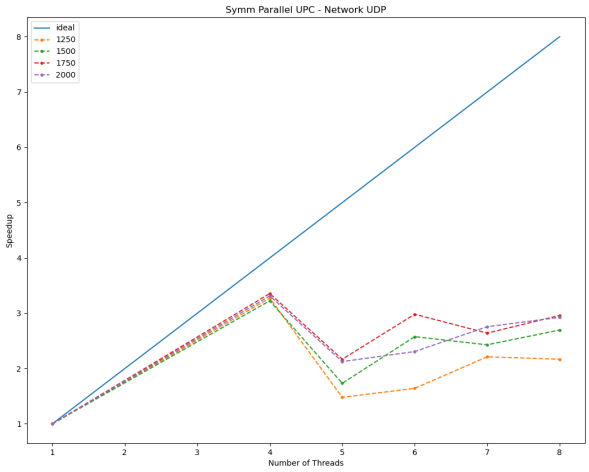

Figura 3. Speedup para network UDP

em 1 não foi linear, porém houve um ganho expressivo no tempo de execução do programa quando comparado apenas com o sequencial.

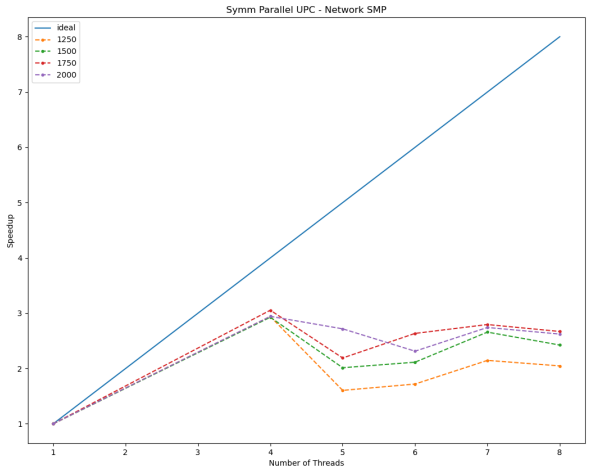

Figura 2. Speedup para network SMP

Olhando a figura 2 que representa os resultados obtidos para a interface SMP, é fato que também houve melhora para quatro núcleos físicos do computador. No entanto, o mesmo não se repetiu para quando houve o uso dos núcleos virtuais do processador, isto é, utilização de cinco ou mais *threads*. O melhor desempenho obtido, mesmo que superficialmente, durante os testes se deve para a interface UDP, representado na figura 3. No entanto, o comportamento do gráfico obtido se assemelha aos resultados obtido em 2.

Um ponto que vale atenção é que em algumas execuções ocorreu casos em que*threads* de uma execução trabalhavam mais do que outras. Com isso, essas *threads* levavam um tempo maior para terminar o seu processamento, causando um resultado final pior. Por exemplo, a seguinte saída era retornada pelo algoritmo:

Tempo de execução

- T1: 3.567905s
- T2: 3.751191
- T3: 3.779177
- T4: 9.463244
- T5: 9.725925

Caso fosse possível criar uma média entre os tempos de execução e atribuir igualmente para todas as *threads*, seria possível diminuir consideravelmente o tempo utilizado para os resultados obtidos, conseguindo assim, um gráfico de *speedup* mais próximo do ideal.

IV. Conclusão e Trabalhos Futuros

É extremamente importante ter o conhecimento de programação paralela, pois a importância dela para os computadores modernos é de grande relevância. A linguagem UPC é uma ótima ferramenta para a criação de programas paralelos, porém a comunidade que desenvolve em UPC não é tão ativa quanto, por exemplo, a comunidade da linguagem C. Quanto ao algoritmo SYMM modificado neste trabalho, existem algumas melhorias que podem ajudar na obtenção de melhores resultados. Por exemplo, diminuir as responsabilidades de variáveis compartilhadas, isto é, utilizar variáveis compartilhadas somente para atribuir ou receber um valor, sem necessitar de inserir cálculos diretamente nela. Outro ponto é fazer melhores distribuições entre as *threads*, pois como foi visto neste trabalho, algumas *threads* acabaram realizando mais processamento do que outras.

Referências

Polybench/c. http://web.cse.ohio-state.edu/~pouchet.2/software/polybench/. Accessed: 2021-12-09.

Berkeley upc - unified parallel c. BerkeleyUPC-UnifiedParallelC. Accessed: 2021-12-09.