PCP Trabalho 4

Matheus Coelho Ambrozio (1812794) e Renan Almeida (1812805)

Solução

Para este trabalho, utilizamos uma solução estática no número de nós do MPI e dinâmica dentro de cada nó, de forma que cada nó cria *threads* que compartilham tarefas entre si.

Primeiramente, o programa inicializa um nó mestre, que executa uma BFS a partir da cidade de partida para achar pelo menos NP tarefas (onde NP é o número de processos/nós do MPI). O nó mestre, então, distribui de forma balanceada as tarefas entre os nós trabalhadores. Dentro de cada trabalhador (função *worker*), ocorre uma nova divisão de tarefas a serem distribuídas para NT *threads* (NT é um valor inteiro passado como argumento para o programa).

Enquanto os trabalhadores expandem o grafo, procurando por soluções, o nó mestre espera por dois possíveis tipos de mensagens: MPI_TAG_SENDING_TOUR e MPI_TAG_DONE. A primeira indica que um trabalhador achou um caminho local ótimo, e sincroniza o caminho ótimo desse trabalhador com o do nó mestre (se o global não for ótimo, atualiza o global e, caso o global seja ótimo, atualiza o local). Nossa solução utiliza o nó mestre somente como controlador do caminho global ótimo, sem que o mesmo realize as computações sobre o grafo. Escolhemos esse modelo pela sua simplicidade e facilidade de desenvolvimento.

Enquanto procuram por caminhos ótimos, cada trabalhador insere e remove caminhos parciais de uma pilha local de tarefas. Quando essa pilha local está vazia, o trabalhador busca tarefas em uma pilha global, compartilhada entre todos os trabalhadores. Da mesma forma, quando sua pilha local está cheia, o trabalhador adiciona tarefas à pilha global. A concorrência sobre a pilha global é protegida por um *mutex* e a sincronização é garantida por duas variáveis de condição (*empty* e *full*). Escolhemos compartilhar tarefas dessa forma para evitar o uso excessivo de *locking* a cada passo do algoritmo (o que poderia acarretar em perda de desempenho).

Uma vez que um trabalhador termina, ele envia uma mensagem para o nó mestre que, por sua vez, está esperando pelo término de todos os trabalhadores. Quando isso ocorre, o nó mestre imprime o melhor caminho e o programa termina.

Compilação & Execução

Para compilar o programa, basta executar o `make`, atentando-se às opções de configuração fornecidas. Para

GRAPH	caminho (relativo à pasta `./bin`) para o arquivo contendo o grafo.
NTHREADS	número de *threads* a serem criadas em cada nó MPI.
NP	número de processos MPI (sempre deve ser igual ao número de nós).
MPIRUN	comando para executar MPI (`mpiexec` por default).
HOSTFILE	caminho para o arquivo contendo as informações dos nós MPI.

Resultados

Executamos o programa com 2, 3 e 5 processos MPI pois nossa solução usa um dos nós apenas como controlador do melhor caminho, sem que ele execute as computações do algoritmo. Com 2, 3 e 5 processos temos, respectivamente, 1, 2 e 4 nós trabalhadores.

Para os resultados com quatro *threads*, fizemos três repetições e calculamos a média do tempo. Para os resultados com uma *thread*, medimos apenas uma execução (devido ao tempo). Não conseguimos medir o teste com dois processos MPI e uma *thread* pois ele demorava demais.

	2 processos MPI	**3 processos MPI**	**5 processos MPI**
1 thread	∞	1253.47	933.14
4 threads	441.62	344.95	315.52

Percebemos pelo resultado que aumentar a quantidade de *threads* altera de forma significativa o desempenho. Aumentar a quantidade de processos MPI, apesar de acelerar o término do programa, não produz os mesmos ganhos.

Acreditamos que a frequência com que atualizamos o menor caminho global possa ter influenciado os ganhos pequenos ao aumentar a quantidade de processos MPI. Se sincronizássemos o menor caminho global com maior frequência, talvez os ganhos aumentando o número de processos MPI fossem maiores.