Francisco Tassinari Fajardo - Trabalho 7 de Programação Paralela

---

SPECS:

Arquitetura: x86_64
Modo(s) operacional da CPU:32-bit, 64-bit
Ordem dos bytes: Little Endian
CPU(s): 4
Lista de CPU(s) on-line:0-3
Thread(s) per núcleo: 2
Núcleo(s) por soquete:2
Soquete(s): 1
Nó(s) de NUMA: 1
ID de fornecedor: GenuineIntel
Família da CPU: 6
Modelo: 61
Nome do modelo: Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz
Step: 4
CPU MHz: 2182.409
CPU MHz máx.: 2700,0000
CPU MHz mín.: 500,0000
BogoMIPS: 4391.85
Virtualização: VT-x
cache de L1d: 32K
cache de L1i: 32K
cache de L2: 256K
cache de L3: 3072K
CPU(s) de nó0 NUMA: 0-3

OBS: na verdade são 2 cores e 2 hiperthreads mas o linux não pega

---

RESULTADOS:

N	PROCESSADORES		
	1	2	4
752			
500	4.65	2.6s	2.3s
252	0.61s	0.4s	0.4s
100	0.55s	0.34s	0.5s(muita variação)
12	0.0006s	0.0003s	0.01s|

N	SPEEDUP		
	(1)	(2)	(4)
500	1	1.8	2.0
252	1	1.5	1.5
100	1	1.6	1.1

Análise:

Pode se notar que para um número considerável de grafos totais, ou como o programa chama cidades totais, ocorre um speedup só com a divisão da carga de trabalho no laço for que faz as chamadas para a função que calcula o menor caminho entre todos os gráficos, observa-se que o código utilizado para gerar os tempos só está trabalhando com números múltiplos na relação entre cidades totais e o número de processadores, Isso não foi tratado devido ao fato de que utilizando a ferramenta Openmpi não consegui implementar a diferença entre tempos de quando termina o último processo menos a criação do primeiro processo, então mesmo que tratado esse detalhe eu não conseguiria medir o tempo, o problema do tempo acarreta em um speedup maior do que o real, o quão maior depende da diferença entre a criação do 1 e o último processo ,mas se for pequeno não é uma diferença que torna os dados inválidos.

Partindo para a Análise do Código percebe-se que após vários experimentos realizados como o gerador de ligações é random não se tem sempre um caso especifíco do Dijkstra(pior caso onde precisa percorrer todos os grafos), mas percebe-se que o paralelismo usado nesse código serve para apenas acelerar o tempo total, é possível usar o paralelismo para melhorar o Dijkstra em seu desepenho, no qual é aplicavel para n grandes, como não foi especificado se o objetivo era reduzir esse tempo com o paralelismo, não se tentou essa implementação, visto que para casos onde temos um n total pequeno o desempenho do algorítmo que é n^2 não influência tanto(baixos tempos).Nota-se que para n 752 por exemplo ocorre muita demora, possívelmente pois o random gerou perto do pior caso(o random pode gerar como ocorre para o 500 muitos grafos desconexos o que facilita a busca) e como o n já está consideravél ocorre uma demora abusiva, para casos assim é necessário implementar a versão do Dijkstra que particiona a tabela de distâncias usando variáveis globais que são atualizadas o que parece mais fácil de ser feito em um API tipo o openMP.

Obs final: dificuldades para obter os tempos usando openMpi e quando o programa faz alguma execução errada é necessário reiniciar o computador.