Tarefa 4 de SemináriosII

Gustavo Lopes Rodrigues

1. Como ficou o tempo de execução da versão paralelizada com *reduction* em relação à versão ante

Comparando o código paralelizado com redução, com o código inicial, o código paralelizado ficou aproximadamente 4 vezes mais rápido

gustavolr@gustavolr-340XAA-350XAA-550XAA:~/Área de Trabalho/Exercicios/Seminario
s II/TarefaIII$ time ./a.out

Codigo não paralelizado
Número total de primos: 148933

real 0m0,834s
user 0m0,832s
sys 0m0,000s
gustavolr@gustavolr-340XAA-350XAA-550XAA:~/Área de Trabalho/Exercicios/Seminario
s II/TarefaIII$ gcc primos.c -lm -fopenmp
gustavolr@gustavolr-340XAA-350XAA-550XAA:~/Área de Trabalho/Exercicios/Seminario
s II/TarefaIII$ time ./a.out

Codigo paralelizado
Número total de primos: 148933
speedup: 4

real 0m0,204s
user 0m1,362s
sys 0m0,032s

2. Como ficou o tempo de execução da versão paralelizada com a seção crítica em relação ao que usou a redução (*reduction*)? Rode pelo menos 10 vezes e compare o tempo médio (computado e impresso em seu código, como na tarefa do cálculo de Pi).

O codigo paralelizado com redução e com secção crítica tem tempos muito parecidos, no tempo real e do usuario, a da secção crítica possui o tempo um pouco mais alto, pois há mais chamadas ao sistema, e sem contar que quando há uma secção crítica, as threads precisam esperar as outras threads terminar os calculos, por isso, esse tipo de programa demora mais um pouco, já que quando tem reduction, as threads criam uma variavel separado e depois soma elas separadamente

Codigo do primos.c com critical

gustavolr@gustavolr-340XAA-350XAA-550XAA:~/Área de Trabalho/Exercicios/Seminarios
II/TarefaIII$ time ./a.out
Número total de primos: 148933
tempo de execução = 0.207633

real 0m0,211s
user 0m1,416s

```
sys     0m0,009s
gustavolr@gustavolr-340XAA-350XAA-550XAA:~/Área de Trabalho/Exercicios/Seminarios
II/TarefaIII$
gustavolr@gustavolr-340XAA-350XAA-550XAA:~/Área de Trabalho/Exercicios/Seminarios
II/TarefaIII$ time ./a.out
Número total de primos: 148933
tempo de execução = 0.215664

real    0m0,219s
user    0m1,459s
sys     0m0,004s
gustavolr@gustavolr-340XAA-350XAA-550XAA:~/Área de Trabalho/Exercicios/Seminarios
II/TarefaIII$ time ./a.out
Número total de primos: 148933
tempo de execução = 0.223805

real    0m0,228s
user    0m1,484s
sys     0m0,004s
gustavolr@gustavolr-340XAA-350XAA-550XAA:~/Área de Trabalho/Exercicios/Seminarios
II/TarefaIII$ time ./a.out
Número total de primos: 148933
tempo de execução = 0.217656

real    0m0,220s
user    0m1,458s
sys     0m0,000s
```

Codigo do primos.c com reduction

```
gustavolr@gustavolr-340XAA-350XAA-550XAA:~/Área de Trabalho/Exercicios/Seminarios
II/TarefaIII$ gcc primosReduction.c -lm -fopenmp
gustavolr@gustavolr-340XAA-350XAA-550XAA:~/Área de Trabalho/Exercicios/Seminarios
II/TarefaIII$ time ./a.out
Número total de primos: 148933
tempo de execução = 0.202148

real    0m0,205s
user    0m1,376s
sys     0m0,004s
gustavolr@gustavolr-340XAA-350XAA-550XAA:~/Área de Trabalho/Exercicios/Seminarios
II/TarefaIII$ time ./a.out
Número total de primos: 148933
tempo de execução = 0.213289

real    0m0,217s
user    0m1,461s
sys     0m0,012s
```

gustavolr@gustavolr-340XAA-350XAA-550XAA:~/Área de Trabalho/Exercicios/Seminarios II/TarefaIII$ time ./a.out
Número total de primos: 148933
tempo de execução = 0.191258

real 0m0,194s
user 0m1,310s
sys 0m0,016s
gustavolr@gustavolr-340XAA-350XAA-550XAA:~/Área de Trabalho/Exercicios/Seminarios II/TarefaIII$

---

3. Explique porque o tempo com a seção crítica ficou "bom" ("comparável" ao tempo com a redução). Pense na carga de trabalho que cada *thread* recebe ao serem divididas as iterações do *loop* paralelizado.

Para entender o tempo entre os dois códigos, é preciso entender que quando há uma secção crítica, as threads fazem os cálculos, e quando chegam na secção crítica, eles esperam as outras threads chegarem para então continuarem. O código com reduction significam que todas as threads teram uma cópia da varíavel (nesse caso a varíavel "soma") e depois todas seram somadas(já que o comando é "+").

Levando em consideração isso tudo, é compreensível porque o tempo do código com secção ficou bom, pois a redutiva possui mais uma váriavel, dando um pouco mais de peso ao código.

---

4. Em relação ao balanceamento de carga, compare os tempos de execução usando a redução (sem seção crítica) e cada uma das três políticas (*static, dynamic e guided*, estas duas últimas com *chunk size* igual a 100). Use LIMITE_MAX igual a 2 milhões e 4 milhões. Novamente, rode pelo menos 10 vezes com cada parâmetro e reporte algo parecido com:

Número de execuções: 10(para cada tipo de política)
Tempo médio nas execuções: 0,210666
LIMITE_MAX: 2000000

Política	**static**	**dynamic**	**guided**
Tempo médio	0,200	0,182	0,250

Número de execuções: 10(para cada tipo de política)
Tempo médio nas execuções: 0,478333
LIMITE_MAX: 4000000

Política	**static**	**dynamic**	**guided**
Tempo médio	0,510	0,445	0,480

---

5. Varie o *chunk* de 50 em 50 até um valor máximo, MAX_CHUNK, que você definir, e tente determinar o *chunk* ótimo para as políticas *dynamic* e *guided*. Novamente, reporte o tempo médio para, pelo menos, 10 execuções. Ou seja, para cada valor de *chunk* seu código deve rodar pelo menos 10 vezes e reportar o tempo médio para verificação. Mantenha variáveis para armazenar o "tempo mínimo global", isto é, o menor tempo entre todas as execuções com todas as políticas com todos os valores de *chunk*, e o "tempo mínimo global médio", isto é parecido com o anterior, mas considera as 10 (pelo menos) execuções. Use LIMITE_MAX de 4 milhões.

Número de execuções: 10 (para cada tipo de política)
Tempo médio nas execuções:0,396666667
LIMITE_MAX: 4000000

Política	**static**	**dynamic**	**guided**
Tempo médio	0,410	0,380	0,400

MAX_CHUNK:

guided: 100

static: 150

dynamic: 50