Otimização do Crivo de Erastótenes Considerando Arquitetura de Memória

Descrição da otimização

O algoritmo foi modificado para atualizar o crivo em blocos de tamanho pouco menores que a cache, de forma a poder carregar o bloco inteiro cada vez que for necessário, minimizando os misses na cache causados por ter que carregar a proxima parte do crivo.

Para isso, é necessário armazenar uma lista de primos já encontrados de forma a atualizar cada bloco novo carregado do crivo. Só é necessário armazenar os primos menores ou iguais à raiz quadrada do numero máximo a ser checado. Esses primos foram armazenados numa lista ligada.

A cada bloco carregado, atualiza-se com os primos já encontrados e se itera sobre o bloco procurando novos primos como no algoritmo original e limpando o bloco dos multiplos desse primo.

O uso de memória poderia ser ainda mais eficiente se em vez de uma lista ligada, fosse utilizada uma estrutura mais eficiente de memória, porem a ordem de grandeza da lista é substancialmente menor e essa otimização foi ignorada.

Arquivo de saída

```
// Out_Crivo

Calcular primos menores que 1000000000

Encontrou 50847534 primos
Tempos: total=5.038713, init=1.188831, comp=3.085773, resto=0.764109
```

Comentários

Se observarmos os resultados obtidos no exercício 3 sem e com otimizações do compilador, observamos que levar a arquitetura de memória em consideração desde o começo do problema (e não só contar com as otimizações do compilador), podem levar a melhoras maiores até mesmo que a paralelização.

O código implementado nesse exercício é mais rapido que o algoritmo não otimizado rodando em até 24 threads e que o otimizado rodando em 8, e mesmo para 24 threads o speed-up não é tão elevado: 1,234.

Assim, percebe-se que para problemas com grande quantidade de memória envolvida, levar em conta a arquitetura de memória pode ser tão relevante ou mais que paralelizar o mesmo.