Implementação paralela do Crivo de Eratóstenes utilizando Open MPI

Leonardo Nascimento

Centro de Matemática, Computação e Cognição (CMCC)

Universidade Federal do ABC (UFABC) – Santo André – SP – Brasil

nascimento.leonardo@aluno.ufabc.edu.br

1 Otimizações

Reduzimos o número de operações executadas ao marcar apenas os múltiplos de um número primo p a partir de p^2 e também ao evitar operações de multiplicação e divisão.

Diminuimos o consumo de memória listando somente os números da forma $6k\pm1$ e utilizando vetores de bits, além de otimizar seu acesso fazendo marcações apenas até um limite configurado, delimitando um bloco, e continuando após todo o bloco ser marcado.

2 Implementação sequencial

Foram utilizados diversos procedimentos para percorrer o crivo no formato $6k\pm1$ pelos seus índices, onde fórmulas foram definidas para encontrar o índice do quadrado de um número primo p e marcar seus múltiplos de maneira eficiente.

Por exemplo, para um número primo p em um índice par, o índice do seu quadrado é $2.(k.p - k) - 1$, onde k é o índice do par $6k\pm1$.

3 Implementação paralela

Usando como base a implementação sequencial e as otimizações aplicadas nele, distribuímos para cada processo o intervalo dos índices que estes devem marcar, cujo tamanho respeita o tamanho do cache L2 inferido.

4 Resultados

Tanto a implementação sequencial quanto a paralela do algoritmo é capaz de calcular a quantidade de números primos até 10^{11}, como mostra a Tabela 1.

O modelo de distribuição dos intervalos do crivo utilizado tornou bastante árdua a tarefa de exibir apenas os 20 últimos números primos gerados, e por este motivo ainda não foi concluída.

Tabela 1. Quantidade de números primos entre 1 e um limite.

Potência	Limite	Primos
1	10	4
2	100	25
3	1.000	168
4	10.000	1.229
5	100.000	9.592
6	1.000.000	78.498
7	10.000.000	664.579
8	100.000.000	5.761.455
9	1.000.000.000	50.847.534
10	10.000.000.000	455.052.511
11	100.000.000.000	4.118.054.813

5 Speedup e eficiência

As análises foram feitas em um computador portátil com processador Intel Core i7-8550U (Quad Core, 8M Cache, 1.8GHz, 15W) e 16GB de memória RAM (Dual In-Line Memory Module, 8GB, 2400Mhz, DDR4). Vale salientar que as análises foram feitas com o dispositivo conectado à tomada.

Para entradas $N = 10^i$, com i variando entre 3 e 10, verificamos que o algoritmo possui speedup máximo de 1,78 quando $N = 10^9$, com eficiência 0,45. Ainda, observamos valores baixos para speedup e eficiência quando mais de 4 processos são utilizados conforme esperado, uma vez que o processador possui apenas 4 núcleos.

Tabela 2. Tempo de execução, speedup e eficiência para a entrada 10^3.

Para $N = 10^3$					
Processos	1	2	4	8	16
Tempo (s)	0,23	0,23	0,23	0,24	0,29
Speedup	1,00	0,98	1,00	0,94	0,77
Eficiência	1,00	0,49	0,25	0,12	0,05

Tabela 3. Tempo de execução, speedup e eficiência para a entrada 10^4.

Para N = 10^4					
Processos	1	2	4	8	16
Tempo (s)	0,21	0,24	0,24	0,24	0,29
Speedup	1,00	0,88	0,87	0,87	0,71
Eficiência	1,00	0,44	0,22	0,11	0,04

Tabela 4. Tempo de execução, speedup e eficiência para a entrada 10^5.

Para N = 10^5					
Processos	1	2	4	8	16
Tempo (s)	0,21	0,24	0,22	0,24	0,31
Speedup	1,00	0,89	0,96	0,91	0,68
Eficiência	1,00	0,45	0,24	0,11	0,04

Tabela 5. Tempo de execução, speedup e eficiência para a entrada 10^6.

Para N = 10^6					
Processos	1	2	4	8	16
Tempo (s)	0,23	0,23	0,21	0,25	0,31
Speedup	1,00	0,98	1,08	0,92	0,74
Eficiência	1,00	0,49	0,27	0,12	0,05

Tabela 6. Tempo de execução, speedup e eficiência para a entrada 10^7.

Para N = 10^7					
Processos	1	2	4	8	16
Tempo (s)	0,26	0,25	0,26	0,29	0,33
Speedup	1,00	1,03	0,99	0,88	0,77
Eficiência	1,00	0,52	0,25	0,11	0,05

Tabela 7. Tempo de execução, speedup e eficiência para a entrada 10^8.

Para N = 10^8					
Processos	1	2	4	8	16
Tempo (s)	0,64	0,53	0,43	0,47	0,69
Speedup	1,00	1,22	1,48	1,36	0,94
Eficiência	1,00	0,61	0,37	0,17	0,06

Tabela 8. Tempo de execução, speedup e eficiência para a entrada 10^9.

Para N = 10^9					
Processos	1	2	4	8	16
Tempo (s)	4,54	3,43	2,55	3,08	4,67
Speedup	1,00	1,32	1,78	1,47	0,97
Eficiência	1,00	0,66	0,45	0,18	0,06

Tabela 9. Tempo de execução, speedup e eficiência para a entrada 10^{10}.

Para N = 10^{10}					
Processos	1	2	4	8	16
Tempo (s)	49,57	38,11	28,59	33,37	51,75
Speedup	1,00	1,30	1,73	1,49	0,96
Eficiência	1,00	0,65	0,43	0,19	0,06

6 Escalabilidade

Analisando as tabelas listadas abaixo, é possível afirmar que o algoritmo não possui escalabilidade forte, uma vez que a eficiência não é mantida à medida que o número de processos cresce sem um aumento no tamanho do problema.

Entretanto, o algoritmo possui escalabilidade fraca, já que a eficiência é mantida à medida que o número de processos cresce quando também há um aumento proporcional do tamanho do problema. Assim sendo, o algoritmo é escalável.

Tabela 10. Tempo de execução, speedup e eficiência para a entrada 10^8, sua metade e dobro.

Para N = 10^8						
	Processos	1	2	4	8	16
N/2	Tempo (s)	0,44	0,38	0,37	0,36	0,52
	Speedup	1,00	1,14	1,18	1,22	0,84
	Eficiência	1,00	0,57	0,30	0,15	0,05
N	Tempo (s)	0,64	0,53	0,43	0,47	0,69
	Speedup	1,00	1,22	1,48	1,36	0,94
	Eficiência	1,00	0,61	0,37	0,17	0,06
2*N	Tempo (s)	1,04	0,82	0,67	0,68	1,01
	Speedup	1,00	1,27	1,56	1,53	1,03
	Eficiência	1,00	0,64	0,39	0,19	0,06

Tabela 11. Tempo de execução, speedup e eficiência para a entrada 10^9, sua metade e dobro.

Para N = 10^9						
	Processos	1	2	4	8	16
N/2	Tempo (s)	2,35	1,78	1,31	1,48	2,37
	Speedup	1,00	1,32	1,79	1,59	0,99
	Eficiência	1,00	0,66	0,45	0,20	0,06
N	Tempo (s)	4,54	3,43	2,55	3,08	4,67
	Speedup	1,00	1,32	1,78	1,47	0,97
	Eficiência	1,00	0,66	0,45	0,18	0,06
2*N	Tempo (s)	9,53	7,15	5,12	5,91	9,04
	Speedup	1,00	1,33	1,86	1,61	1,05
	Eficiência	1,00	0,67	0,47	0,20	0,07

Tabela 12. Tempo de execução, speedup e eficiência para a entrada 10^{10}, sua metade e dobro.

Para $N = 10^{10}$						
	Processos	1	2	4	8	16
N/2	Tempo (s)	22,91	17,95	13,10	15,35	25,28
	Speedup	1,00	1,28	1,75	1,49	0,91
	Eficiência	1,00	0,64	0,44	0,19	0,06
N	Tempo (s)	49,57	38,11	28,59	33,37	51,75
	Speedup	1,00	1,30	1,73	1,49	0,96
	Eficiência	1,00	0,65	0,43	0,19	0,06
2*N	Tempo (s)	102,05	89,90	65,74	76,98	Killed[1]
	Speedup	1,00	1,14	1,55	1,33	N/A
	Eficiência	1,00	0,57	0,39	0,17	N/A

7 Balanceamento de carga

Quando N é menor do que $b = 2.10^6$, valor encontrado experimentalmente que sugere o melhor desempenho do algoritmo, coincidindo com aproximadamente o tamanho do cache L2, não há balanceamento de carga.

Para valores maiores do que *b*, a distribuição para os procesos é feita por round-robin em blocos de tamanho *b*.

Essa divisão, visando um melhor desempenho para valores de N grande, dificultou a tarefa de exibir apenas os 20 últimos números primos gerados, todavia apresentou na prática o melhor balanceamento de carga, por exemplo, foi melhor do que a divisão por tamanhos iguais e as progressões geométricas decrescentes testadas.

8 Considerações finais

Desenvolver esse projeto foi bastante desafiador e prazeroso, apesar de não ter sido cumprido em sua totalidade e demandar cerca de 120 horas para chegar a um entregável.

Esse projeto demandou analisar o desempenho real do algoritmo considerando o hardware, e fomos incentivados a propor formas de melhorar o desempenho de diversas

[1] O processo foi morto pelo sistema operacional

maneiras, às vezes contraintuitivas, onde mesmo um aumento no número de operações foi capaz de representar um ganho de desempenho considerável.

Houveram conversas em alto nível com os alunos Anderson Chaves Faria e Guilherme Matheus Costa sobre a resolução do problema que podem se manifestar no nosso trabalho.

Sem dúvida foi o projeto mais legal e mais estressante que fiz na universidade.

Referências

Notas de aula de Programação Paralela. Disponível em <http://professor.ufabc.edu.br/~e.francesquini/2019.q1.pp/> Acesso em: 21 abr. 2019.

How Many Primes Are There?. Disponível em <https://primes.utm.edu/howmany.html> Acesso em 8 abr. 2019.

Array of bits. Disponível em <http://www.mathcs.emory.edu/~cheung/Courses/255/Syllabus/1-C-intro/bit-array.html> Acesso em: 16 abr. 2019.