Mergesort Paralelo com MPI

Gissely de Souza
Guilherme Manteuffel Bettu
UFPR - Ciência da Computação
Programação Paralela 2019/1

Problema de ordenação

Problema clássico de computação

Muitas aplicações práticas

Muitas estratégias sequenciais já estudadas

Algoritmo adotado: Quicksort sequencial e Mergesort paralelo

Complexidade

Quicksort:

Pior caso é $O(n^2)$, mas no geral é O(n logn), por isso é usado

Mergesort:

Função *merge* é linear e chamada a cada nível da árvore de recursão, resultando em O(n logn) operações.

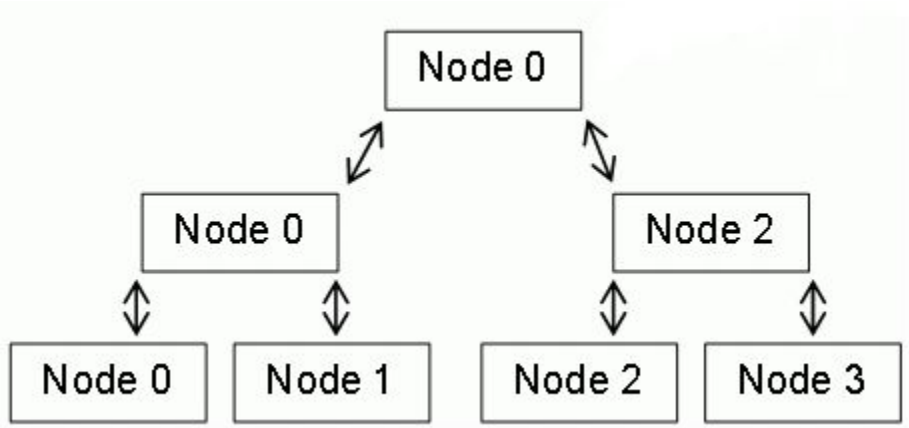
Como o Merge é serial a implementação paralela é O(n)

Mergesort paralelo

1. Cria um processo para cada folha da árvore de recursão
2. Processo raiz entra na função, os outros esperam receber de um pai
3. Quicksort é executado nas folhas sequencialmente (N/p)
4. Etapa de *merge* é executada depois da recursão e de receber a resposta

Mergesort paralelo - árvore

Processos vão descendo pela esquerda
Raiz tem altura máxima e folhas são de altura 0



Mergesort paralelo - Divisão

- Divisão: Cada chamada divide o vetor recebido no meio, envia a metade da direita para o processo filho à direita na árvore e usa recursão com a metade da esquerda

```
Paramerge(vetor, altura)
  if (altura != 0):
    MPI_Send(vetor.dir, filho_dir)
    Paramerge(vetor.esq, altura-1)
    MPI_Recv(vetor.dir, filho_dir)
    Merge(vetor, vetor.esq, vetor.dir)
  else: // altura == 0 => folha
    Quicksort(vetor)
  if (pai):
    MPI_Send(vetor, pai)
```

MPI - Message Passing Interface

Padrão de comunicação de dados para programação paralela

Processos separados e memória distribuída

Baseado no exemplo:
http://penguin.ewu.edu/~trolfe/ParallelMerge/ParallelMerge.html

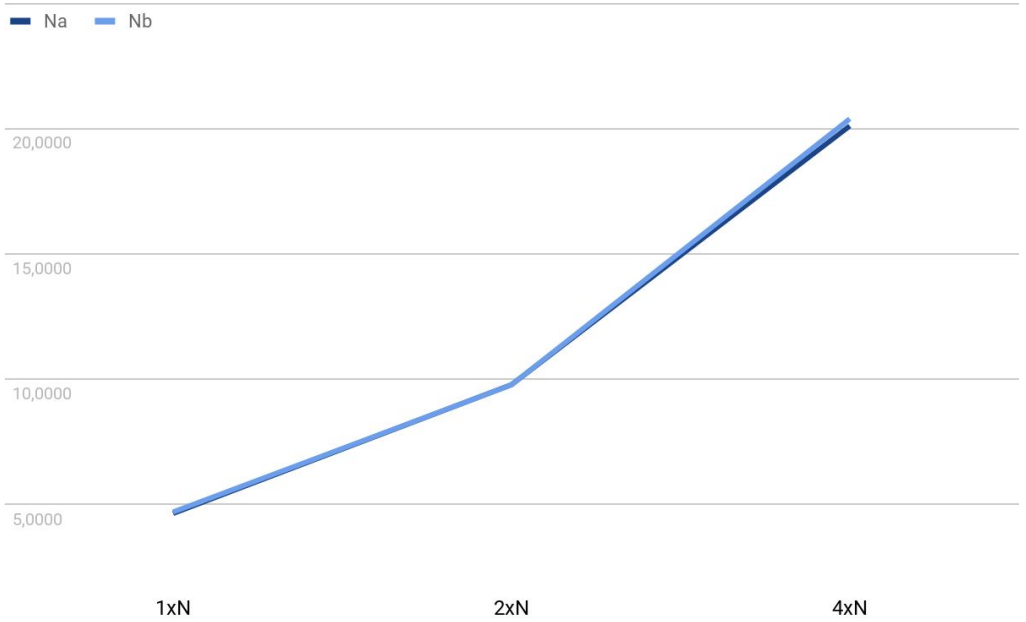
Ambiente de Testes

Intel(R) Core(TM) i5-8400 CPU @ 2.80GHz
6 cpus
L1d cache: 32K
L1i cache: 32K
L2 cache: 256K
L3 cache: 9216K
8 GiB de RAM

Duas entradas para cada N, com pouca diferença nos resultados
Cada teste repetido 20 vezes, com média e desvio padrão calculados
1xN = 50 000 000
2xN = 100 000 000
4xN = 200 000 000

Quicksort

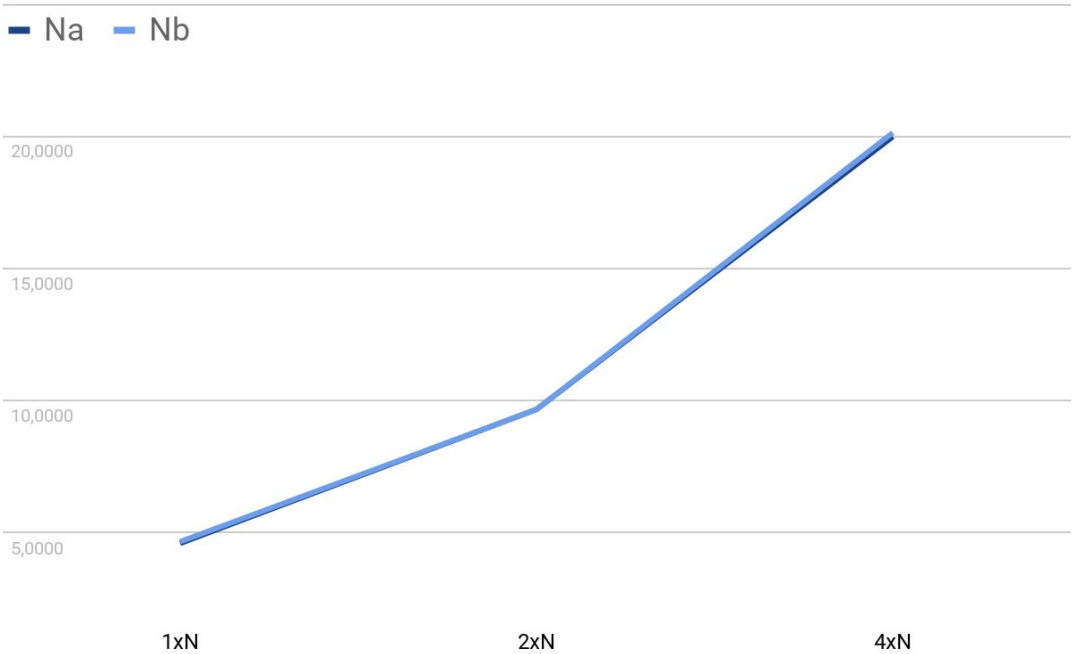
	Na	desvio	Nb	desvio
1xN	4,6485	0,0123	4,7064	0,0080
2xN	9,7824	0,0505	9,7744	0,0569
4xN	20,1004	0,0915	20,3834	0,1004



Paramerge 1p

Pouca diferença entre Na e Nb

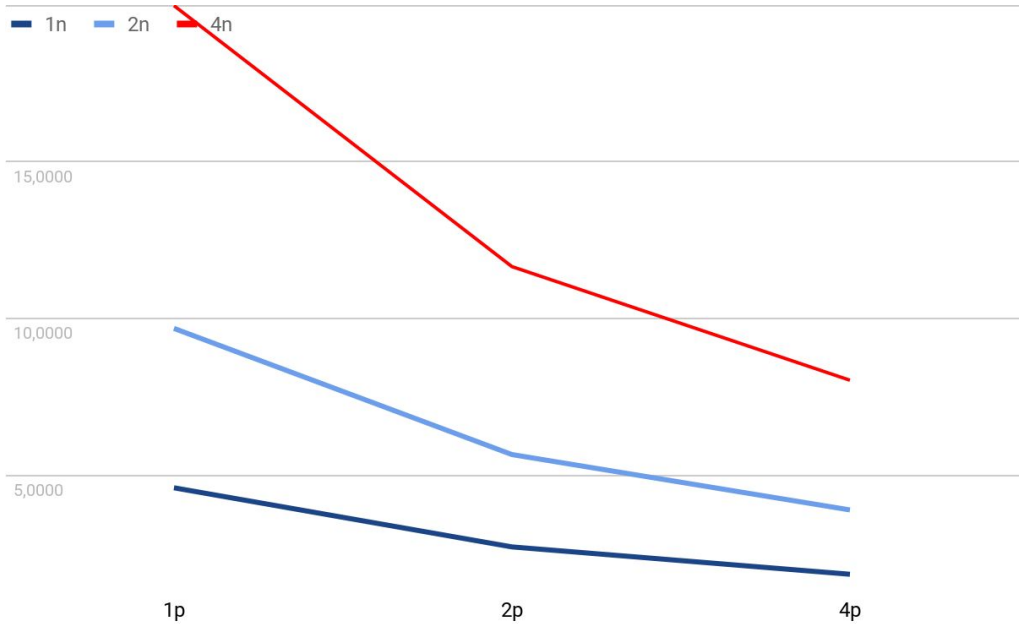
	Na	desvio	Nb	desvio
1xN	4,6019	0,0105	4,6651	0,0127
2xN	9,6686	0,0593	9,6665	0,0580
4xN	19,9820	0,1145	20,1325	0,0988



Paramerge com N e P variando

Processadores x Tempo
com 1xN, 2xN e 4xN

	1n	2n	4n
1p	4,6019	9,6862	19,9820
2p	2,7158	5,6616	11,6572
4p	1,8433	3,8954	8,0326

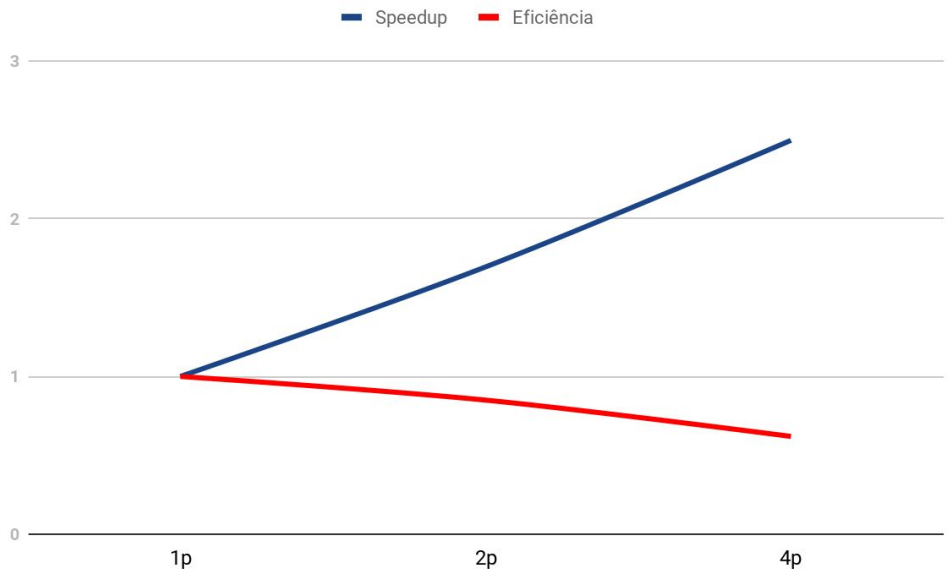


Paramerge Speedup e Eficiência (para N1, N2 e N4)

Pouco fortemente escalável, eficiência cai muito com P maior

N	Speedup	Eficiência
1p	1	1
2p	1,69	0,85
4p	2,50	0,62

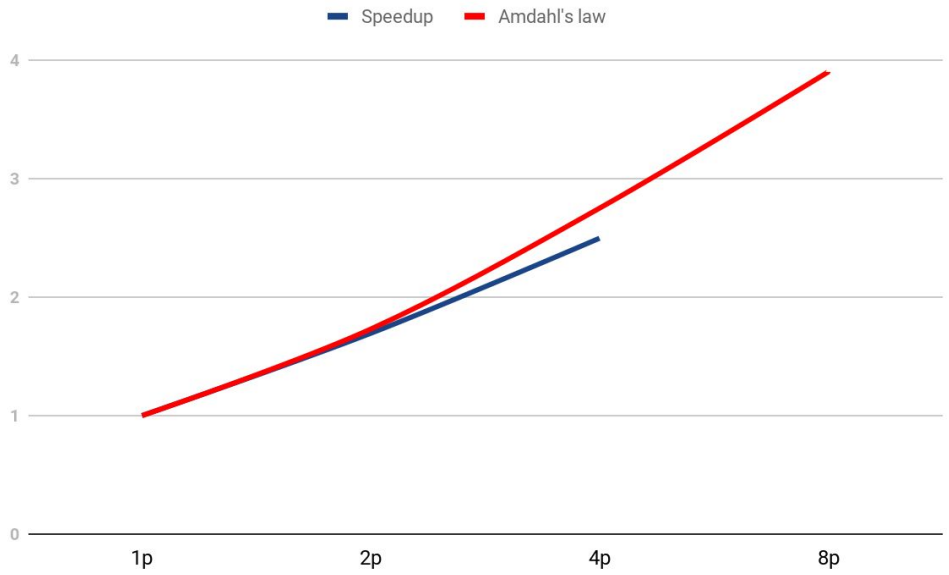
Pouco fracamente escalável, speedup não se mantém ao aumentar N e P juntos



lei de amdahl

Assumindo 85% paralelizável

Speedup com P->infinito: 6,66



Conclusão e Fontes

Uma mudança simples no algoritmo já faz uma boa diferença

Se for possível melhorar a função *merge* para ser paralelizada parcialmente, a eficiência poderia aumentar

http://penguin.ewu.edu/~trolfe/ParallelMerge/ParallelMerge.html

https://github.com/gmb18/paralela19/