UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

CURSO DE BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

Jessé Pires Barbato Rocha

Jhonatan Guilherme de Oliveira Cunha

ORDENAÇÃO DE ARQUIVOS UTILIZANDO INTERCALAÇÃO EM K-VIAS

CAMPO MOURÃO

2020

1. Relatório

Primeiramente, a ideia geral do trabalho foi discutida. Isso, para que tivéssemos o mesmo nível de conhecimento a respeito do problema proposto e como ele poderia ser resolvido. Aqui também é importante destacar a reunião online feita com o professor que auxiliou muito nesta etapa.

Além das implementações solicitadas explicitamente no trabalho, também notamos a necessidade de implementar funções para particionar o arquivo de entrada. Também notou-se a necessidade de implementar o algoritmo de ordenação *quick sort* (para ordenar as partições), bem como funções para verificar se as partições e o arquivo de saída estavam ordenados.

Após isso ser feito, partimos para a divisão do trabalho. Jhonatan ficou responsável pela implementação do *buffer* de entrada, das funções de particionamento do arquivo de entrada e da função de intercalação em *k*-vias.

Já eu, Jessé, fiquei responsável pela implementação do *buffer* de saída e do algoritmo *quicksort*. Também implementei uma função para verificar se a partição está ordenada e outra que verifica a ordenação do arquivo de saída. As duas últimas funções citadas (que foram feitas apenas para auxiliar no desenvolvimento do trabalho), não são usadas na versão final do código, uma vez que informar tais coisas ao usuário não é a proposta do programa.

Por fim, a função de ordenação externa e a função *main* foram implementadas em conjunto.

Tanto a função de intercalação em k-vias quanto a de ordenação externa foram implementadas após os buffers e a rotina de particionamento estarem concluídas. Julgamos mais prático ter todas as outras partes já implementadas e com certa garantia de que estavam corretas (sempre podem haver problemas que passam despercebidos) primeiro. Assim, quando fomos implementar as funções de intercalação em k-vias e de ordenação externa, pudemos usar essas “partes” com mais tranquilidade.

Vale ressaltar que, após finalizada a implementação completa do programa, começamos a refatorar o código, buscando deixá-lo mais limpo e organizado. Nesta etapa, não houve divisão do trabalho. Cada um fez as alterações que considerou necessárias.

Por fim, é importante salientar que todo o processo de desenvolvimento do código foi intensamente facilitado pela utilização do Git e GitHub, uma vez que estamos distantes. O repositório do código encontra-se em https://github.com/jhonatancunha/ed2_k-way_merge

2. Análise

Como solicitado na descrição do trabalho, foi coletado o tempo de execução, em segundos, com as informações dispostas nas tabelas abaixo. Vale ressaltar que o programa foi executado em uma máquina com HDD.

Nota-se um padrão durante a execução dos testes com qualquer uma das quantidades de registros. Em todos os tamanhos de B, segue-se uma sequência em que o tempo de execução em $B/2$ é menor que em $B/4$ que, por sua vez, é menor que o tempo de execução em $B/8$.

Tal fenômeno se deve ao tamanho do *buffer* que fica maior conforme o divisor de B diminui. Logo, a quantidade de chamadas da função *BUFFER_SAIDA_despejar()* é menor. Como esta função escreve os itens do *buffer* de saída no arquivo, chamadas de sistema serão feitas. Quanto menos vezes *BUFFER_SAIDA_despejar()* for chamada, menos chamadas de sistema (que gastam muito tempo), por conseguinte, serão feitas. Consequentemente, o tempo de execução também é reduzido.

TEMPO EM SEGUNDOS		S		
		B/8	B/4	B/2
B	8388608 (8MB)	1.703875	1.532878	1.530273
	16777216 (16MB)	1.678866	1.591172	1.598573
	33554432 (32MB)	1.695141	1.681036	1.664001

Tabela 1: Tempo De Execução Para Ordenar Arquivo com 256000 Registros

TEMPO EM SEGUNDOS		S		
		B/8	B/4	B/2
B	16777216 (16MB)	3.257904	3.250158	3.248757
	33554432 (32MB)	3.352030	3.340549	3.336039
	67108864 (64MB)	3.731147	3.690805	3.687860

Tabela 2: Tempo De Execução Para Ordenar Arquivo com 512000 Registros

TEMPO EM SEGUNDOS		S		
		B/8	B/4	B/2
B	67108864 (64MB)	6.743271	6.389229	6.335587
	134217728 (128MB)	6.798348	6.785877	6.704722
	268435456 (256MB)	7.296477	7.252677	7.185861

Tabela 3: Tempo De Execução Para Ordenar Arquivo com 921600 Registros

TEMPO EM SEGUNDOS		S		
		B/8	B/4	B/2
B	67108864 (64MB)	11.381271	11.105842	11.091826
	134217728 (128MB)	11.785119	11.730616	11.599802
	268435456 (256MB)	11.889736	11.885930	11.614078

Tabela 4: Tempo De Execução Para Ordenar Arquivo com 1572864 Registros

3. Conclusão

Neste relatório, a partir de todo o processo de implementação e análise dos resultados, verificamos que, na computação, pode-se trabalhar de maneira eficiente mesmo em cenários em que os recursos são limitados. No caso em específico, foi possível ordenar um dado arquivo (com testes em que o arquivo que deve ser ordenado tem até mais ou menos 1GB), de mesmo que se tenha pouca memória *ram*.

Assim, utilizando estratégias de particionamento do arquivo e intercalação, ou seja, a ideia de dividir para conquistar, conseguimos concluir o objetivo principal do trabalho tal qual nos foi proposto.