Trabalho Prático MPI

Artur Temporal Coelho
CI316 – Programação Paralela – DINF UFPR
Prof. Marco Antônio Zanata Alves

1. Sobre o Algoritmo

Nosso objetivo é fazer a busca de trechos de DNA em um genoma, o genoma esta dividido em seções (sections), com aproximadamente 10 mil bases em cada uma das 10 seções, as sequências a serem pesquisadas (query) está organizada em 100 mil linhas de aproximadamente mil bases cada.

Vamos utilizar o algoritmo de Boyer–Moore–Horspool-Sunday (bmhs), para fazer as comparações entre cada linha (query) em cada section obtendo como resultado a posição inicial onde a query foi achada em section.

O algoritmo possui a seguinte estrutura básica:

```
// Alocação e abertura de arquivos
alloc_arrays();
open_files();

// Laço
while(!feof(query)) {

    // Leitura
    query = read_query();

    while(!feof(dna)) {

        // Leitura
        dna = read_dna();

        // BHMS
        result = bhms(dna, query);

        print(result);
    }
}
```

5. Métodos de medição

Para medir com precisão o tempo de execução interno de cada programa será utilizada a função ‘clock’ definida em ‘time.h’. Será utilizada também a função ‘time’ para medir o tempo externo do programa.

Os programas foram compilados com o gcc 9.3.0, utilizando a flag de otimização -O3

A máquina utilizada:

- Processador: AMD Phenom(tm) II X4 955 Processor
- Memória: 7955MiB (8GB)
- Linux: Pop!_OS 20.04 LTS
- Kernel: 5.11.0-7614-generic

Foram utilizados os arquivos de entrada:

query.in: 100 mil linhas de mil bases cada

dna.in: 10 seções de 10 mil bases cada

Para realizar as medições a máquina foi reduzida apenas as funções necessárias, para podermos chegar a eficiência máxima e padronizar as medições. Foram feitas 20 medidas para obter a média de tempo entre elas, para obter resultados mais consistentes.

7. Paralelização MPI

Para podermos paralelizar corretamente algumas modificações são necessárias, utilizaremos apenas o processo master (rank 0) para ler e enviar os dados para os demais processos, que por sua vez vão receber as mensagens via MPI, ao invés de abrir e ler o arquivo em si.

Por fim os demais processos enviam os resultados ao processo master, que os compila e escreve no arquivo de saída final

8. Implementação MPI

A estrutura agora do algoritmo:

```
// Alocação e abertura de arquivos
if (rank == MASTER) {
    open_files();
    send_data(num_procs);
} else {
    get_data();
}

alloc_strings();
map_strings();

calculate_init_end_point(rank, num_procs);

// Laço (parte paralelizável)
for (int i = init; i < end; i++) {
    for (int j = 0; j < size_dna; j++) {

        // BHMS
        results[i, j] = bhms(dna[j], query[i]);

    }
}

// Escrita
if (rank == MASTER) {
    get_results();
    write_file(results);
} else {
    send_results();
}
```

- Alocação e arquivos: 0.42 s
- BHMS: 26.31 s
- Escrita: 0.02 s

- Total: 27.350 s

6.1 Modelo de Amdhal

Com os dados obtidos podemos descobrir o speedup teórico do algoritmo:

parte sequencial = 1%
parte paralelizável = 99%

Num processadores	1	2	3	4	8	infinitos
Speedup teórico	1	1.98	2.94	3.88	7.47	100

8.1. Tempo paralelo

Foram utilizados os mesmos requisitos de medição do dna0, porém com as modificações de paralelismo

Numero de threads	1	2	3	4
Total	27.350 s	14.454 s	10.170 s	8.015 s
CPU	99%	195%	290%	382%
Speedup (T0 / Total)	100%	189%	269%	341%
Eficiência	1	0.95	0.89	0.85

A diferença entre o tempo CPU para o speedup se deve ao tempo de sincronização inicial necessário para o envio de dados entre as threads, que é síncrona e bloqueante.

8.2 Tamanho de carga e eficiência

Entrada(query.in) / Processos	1	2	4
50k	14.080	7.662	4.389
100k	26.961	14.465	7.925
200k	54.261	28.041	15.429
400k	107.84	55.416	30.746

Em ambas as diagonais coloridas é possível notar que a eficiência se mantém razoavelmente consistente, com o aumento no tempo decorrente da transmissão dos dados, que não é paralelizada mas pode ser otimizada.

9. Conclusões

Com essa paralelização, dados de entrada e tempo obtidos observamos um tempo interno ao programa de speedup linear. Essa diferença se da pelo tempo adicional de sincronização inicial e final, após a conclusão das buscas, assim como o tempo adicional de transmissão dos dados, e em parte mínima por conta da parte serial do programa.

Pelo meio das observações pode-se dizer que o algoritmo gerado é fortemente paralelizável. Sendo as diferenças no tempo real notadas pela necessidade de sincronizar e transmitir os dados.

Além disso temos o speedup medido proporcional ao speedup teórico calculado com o modelo de Amdhal.