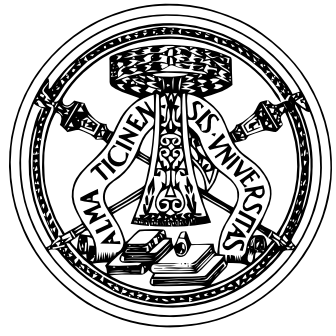
Substring Search with application to Genomics

Advanced Computer Architecture - Project

Diego Mastella & Matteo Ragni



University of Pavia

January 31^{th}, 2023

The Algorithm

- Rabin-Karp fingerprint search;
- Linear-time for substring searching `O(M+N)`;
- Constant time for hash computation of a M-character string.

```
        pat.charAt(j)
j       0  1  2  3  4
        2  6  5  3  5   % 997 = 613

                                  txt.charAt(i)
i       0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
        3  1  4  1  5  9  2  6  5  3  5  8  9  7  9  3
0       3  1  4  1  5   % 997 = 508
1          1  4  1  5  9   % 997 = 201
2             4  1  5  9  2   % 997 = 715
3                1  5  9  2  6   % 997 = 971
4                   5  9  2  6  5   % 997 = 442
5                      9  2  6  5  3   % 997 = 929      match
6 <-- return i = 6        2  6  5  3  5   % 997 = 613   <-
```

Rolling Hashing

- D: alphabet's dimension
- Q: large prime number

pat.charAt(j)

i	0	1	2	3	4
	2	6	5	3	5

0 2 % 997 = 2

1 2 6 % 997 = (2*10 + 6) % 997 = 26

2 2 6 5 % 997 = (26*10 + 5) % 997 = 265

3 2 6 5 3 % 997 = (265*10 + 3) % 997 = 659

4 2 6 5 3 5 % 997 = (659*10 + 5) % 997 = 613

D Q

Serial Program Functionalities

It has two major routines:

- `read_txt`: the one that reads the `txt` file and saves it into an array;
- `rabin_karp`: the one that develops Rabin-Karp substring search in the given text.

`gprof` outcome:

% time	comulative seconds	self seconds	calls	self s/call	total s/call	name
100.00	15.17	15.17	1	15.17	15.17	`rabin_karp`
0.00	15.17	0.00	1	0.00	0.00	`read_file`

Strategies for Parallelization

- Divide the `txt` string among several cores, allowing each core to look for a specific pattern within the text string's allocated part. Every process prints its results on the `stdout` file.
- Same as before but every process prints on its own file and then the master process merges all the files in a unique output one.
- Assign a single core to a single pattern (in the case of a multi-pattern search) and then search in the `txt` string for that specific pattern. *(not implemented)*.

Communication and Syncronization Strategy

These are the data sent by the master process to the other processes:

- `send`: an array containing `bufsize`, that is the length of text each core has to analyze, `M` that is the length of the `pat`, and the rest of $\frac{\texttt{N}}{\texttt{bufsize}}$ where `N` is the overall length of the `txt` string;
- `pat`: the actual string to be searched into the `txt` file;

When `M > bufsize + rest + 1`, the number of cores is decreased because otherwise the chunk of `txt` given to each core is smaller than the pattern to be found.

Theoretical Assessment

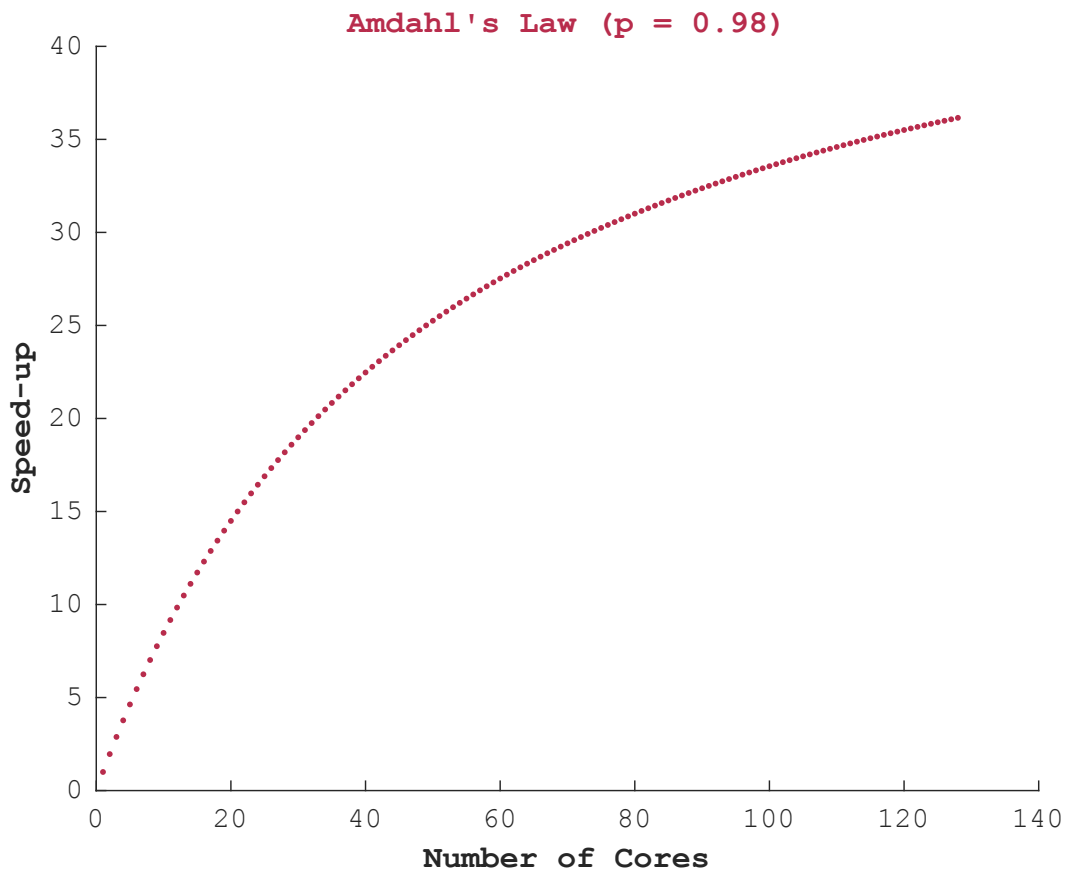


Figure 1: $speedup = \frac{n}{n+p(1-n)}$

Communication between cores

The master process is in charge of opening the communication between processes: it sends the necessary information to all the others to allow them to begin the execution. This was accomplished through the use of the `MPI_Scatter` and `MPI_Bcast` functions.

```
1 //Sending preliminary variables in send array
2     MPI_Bcast(send, 3, MPI_INT, 0, MPI_COMM_WORLD);
3
4 //Sending the pattern
5     MPI_Bcast(pat,pat_len,MPI_CHAR,0,MPI_COMM_WORLD);
6
7 //Sending the flag variable
8     MPI_Scatter(flag, 1, MPI_INT, &flag_send, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

File management

The file pointer is unique so that all cores could open the file through the `MPI_File_open` function. Then each process, once the file is opened, moves to the indicated location through the `MPI_File_set_view` function. Then reads its chunk of text thanks to `MPI_File_read` function.

```
1    MPI_File_open(MPI_COMM_SELF, text_name, MPI_MODE_RDONLY,MPI_INFO_NULL, &file_in);
```

```
1    MPI_File_set_view(file, position, MPI_CHAR, MPI_CHAR, "native", MPI_INFO_NULL);
2
3    MPI_File_read(file, seq, bufsize+M-1, MPI_CHAR, &status);
```

Frequency computation

Each process sends his frequency to the master process by using the `MPI_Gather` function.

```
1 MPI_Gather(&freq, 1, MPI_INT, rec_freq, 1, MPI_INT, 0, MPI_COMM_WORLD);
2 if(myrank == 0){
3     int total_freq = 0;
4     for(int i = 0; i<size;i++){
5       total_freq = total_freq + rec_freq[i];
6     }
7     printf("PATTERN FOUND %d TIMES\n",total_freq);
```

Strong scaling analysis

1. Light cluster, multi-region, 16 cores each 2 vCPUs (N2 series).

Stato	Nome	Zona	Suggerimenti	Utilizzato da	IP interno	IP esterno	Connetti
✓	lc-node-1	us-central1-a			10.128.0.14 (nic0)	34.27.92.111 (nic0)	SSH
✓	lc-node-10	southamerica-west1-a			10.194.0.3 (nic0)	34.176.234.12 (nic0)	SSH
✓	lc-node-11	southamerica-west1-a			10.194.0.4 (nic0)	34.176.57.48 (nic0)	SSH
✓	lc-node-12	southamerica-west1-a			10.194.0.5 (nic0)	34.176.10.208 (nic0)	SSH
✓	lc-node-13	southamerica-east1-b			10.158.0.2 (nic0)	35.199.101.20 (nic0)	SSH
✓	lc-node-14	southamerica-east1-b			10.158.0.3 (nic0)	35.247.225.155 (nic0)	SSH
✓	lc-node-15	southamerica-east1-b			10.158.0.4 (nic0)	35.199.87.122 (nic0)	SSH
✓	lc-node-16	southamerica-east1-b			10.158.0.5 (nic0)	35.247.245.195 (nic0)	SSH
✓	lc-node-2	us-central1-a			10.128.0.15 (nic0)	35.226.58.181 (nic0)	SSH
✓	lc-node-3	us-central1-a			10.128.0.16 (nic0)	35.239.57.59 (nic0)	SSH
✓	lc-node-4	us-central1-a			10.128.0.17 (nic0)	34.71.212.159 (nic0)	SSH
✓	lc-node-5	us-east1-b			10.142.0.2 (nic0)	34.73.209.130 (nic0)	SSH
✓	lc-node-6	us-east1-b			10.142.0.3 (nic0)	104.196.120.178 (nic0)	SSH
✓	lc-node-7	us-east1-b			10.142.0.6 (nic0)	35.237.86.8 (nic0)	SSH
✓	lc-node-8	us-east1-b			10.142.0.7 (nic0)	34.138.43.184 (nic0)	SSH
✓	lc-node-9	southamerica-west1-a			10.194.0.2 (nic0)	34.176.200.51 (nic0)	SSH

Righe per pagina: 30 1 – 16 di 16

Strong scaling analysis (cont.)

N. of cores	Execution time (sec.)
1	17.96
2	10.97
3	10.00
4	6.96
5	5.97
6	4.84
7	4.23
8	3.68
9	3.36
.	.
.	.
31	1.35
32	1.23

Strong scaling analysis (cont.)

2. Light cluster, single-region (us-central1-a), 4 cores each 2 vCPUs (N2 series). Execution time $\sim$ 3.15 sec.
3. Light cluster, 4 cores each 2 vCPUs (N2 series) each core in one different region. Execution time $\sim$ 3.50 sec.
4. Fat cluster, multi-region, 2 cores each 16 vCPUs (N2 series). Execution time $\sim$ 1.35 sec.

Stato	Nome ↑	Zona	Suggerimenti	Utilizzato da	IP interno	IP esterno	Connetti
✓	fc-node-1	us-central1-a			10.128.0.14 (nic0)	104.197.9.48 (nic0)	SSH
✓	fc-node-2	northamerica-northeast2-a			10.188.0.2 (nic0)	34.130.50.82 (nic0)	SSH

5. Fat cluster, single-region (southamerica-west1), 2 cores each 16 vCPUs (N2 series). Execution time $\sim$ 1.35 sec.

Strong scaling analysis (cont.)

cloudquota@google.com
a me

inglese > italiano Traduci messaggio

Hello,

Your quota request for rosy-fiber-332508 has been approved and your project quota has been adjusted according to the following requested limits:

NAME	DIMENSIONS	REGION	REQUESTED LIMIT	APPROVED LIMIT
CPUS_ALL_REGIONS		GLOBAL	32	32
N2_CPUS	region=southamerica-west1	southamerica-west1	32	32

After approved, Quotas can take up to 15 min to be fully visible in the Cloud Console and available to you.

Weak scaling analysis

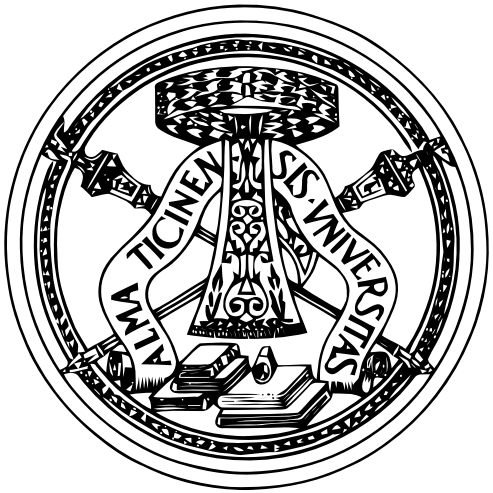
We performed a few tests with different text file without changing the setup of the VMs instances (fat cluster, 2 cores each 16 vCPUs (N2 series)).

Text size	Execution time (sec.)
1 KB	0.01
100 KB	0.02
1 MB	0.02
2 GB	1.35

Table 1: Weak scalability

Consideration about speed-up

Given that the serial program run with one vCPU takes almost 18 seconds and the fastest VM configuration takes 1.23 seconds, we can estimate that running a parallel program produce a speed-up of ~ 14.6, almost as the theoretical speed-up of the Amdahl's Law.



Thanks for your attention