Parallel String Search Algorithm

ODTÜ METU

Başar Kütükcü

Problem Statement

The main usage of string searching is to find particular strings in small files in personal computers. This type of usage directly serves its results to people. It is fast enough to be seen instant by people. Even though it would take twice as much time, people could not tell the difference in many cases. However, there are other usages of string searching and even though they are not common as much as the ones in personal computers, they are not less important. These other usages, such as intrusion detection systems, search engines and plagiarism detection, are open and required to improvements in performance. This need can be filled using parallel algorithms in GPU. While the suggested algorithm in this poster does not directly address these problems, it could be adopted and improved to be used in such problems.

Algorithm

Every thread checks the character they are mapped to. If the checked character is not the first character of the key string, that thread's task is done for the current kernel call. If the checked character is the first character of the key string, the thread starts checking the following characters to see if it is indeed mapped to the first character of the key string. If all the following characters are matched with the key string's characters, the thread saves the index of the first character.

Algorithm 1 Parallel String Search Kernel

```
1: procedure KERNEL(text, key, keySize, keyIndexes) ▷ Find key in the text
2:     if text[idx] = key[0] then
3:         save ← 1
4:         for i ← 1 to keySize do
5:             if text[idx + i] ≠ key[i] then
6:                 save ← 0
7:                 break
8:         if save = 1 then
9:             keyIndexes[++CurrInd] ← idx      ▷ AtomicAdd to increment CurrInd
```

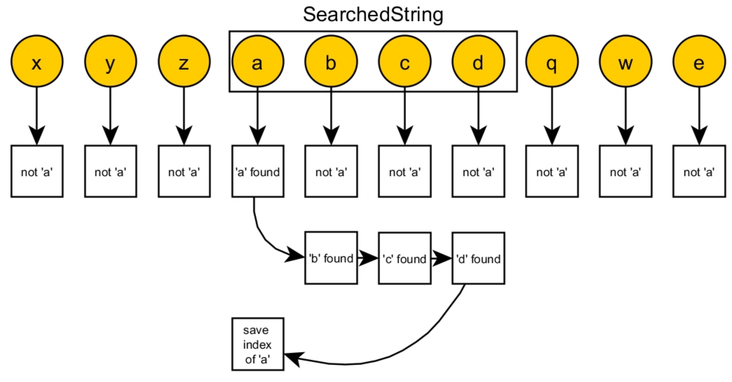

Figure 1 Visualization of the algorithm

Experiments

The experiments are done on the book 'The Hobbit'. Time is taken to be the only parameter to evaluate the algorithm. Time elapsed at transferring data to/from GPU is also taken into account since the comparison is done against 'findstr' and this command cannot be split into parts for benchmark purposes. First benchmark is done by searching the word 'Bilbo' in the book. 'Bilbo' appears 556 times in the book. Second benchmark is done by searching the word 'METU' which is inserted by editing the original text. 'METU' appears 10 times in the edited book. The same benchmarks are also done with a larger text which is 20 times copied version of the book. 3 different versions of parallel string search and 1 findstr are benchmarked with original book. 2 different versions of parallel string search and 1 findstr are benchmarked with edited (20x) book.

	"METU" (10 findings)	"Bilbo" (555 findings)
Findstr	32,67ms	33,02ms
Data->Global, key->Global	1,92ms	2,14ms
Data->Global, key->register	2,22ms	2,28ms
Data->Global, key0->register	1,82ms	2,01ms

Table 1 The results for search in original book

	"METU" (200 findings)	"Bilbo" (11120 findings)
Findstr	38,15ms	34,33ms
Data->Global, key->Global	13,84ms	17,09ms
Data->Global, key0->register	13,53ms	16,67ms

Table 2 The results for search in edited book

While the best performance increase is approximately 18x with original sized book, the speedup gets low down to approximately 2x as the size of the text is increased by 20 times. While FS's performance does not change much by increasing size of text, parallel implementations' performances decrease drastically. The reason for that is memory copies between CPU and GPU. GM showed the best performance among parallel implementations.

Limitations

- The length of the array, that holds the indexes of which found key string in the text, is unknown before running the kernel.
- The first CUDA call takes huge time. It is so huge that it makes the performance improvements disappear.

The Code Repository

Conclusion

- It should be noted that while parallel string search algorithm shows better performance than CPU string search algorithm, it still cannot utilize the GPU completely. The very large part of elapsed time is from memory copy operations. Therefore, we could say that parallel string search algorithm is I/O bounded as well as CPU search algorithms but still GPU version is faster.
- In overall, the parallel string search algorithm showed a significant performance increase over the fastest CPU string search algorithm. Even though it has some limitations and problems, it could lead to future research and optimization. It has a potential to be used on real world problems.