Size Matters: Space/Time Tradeoffs to Improve GPGPU Applications Performance

12M38054 石原翔真

Outline

- Background
  - Read Alignment Problem
  - Suffix Tree & Suffix Array
- Offloading read alignment
  - Previous research: MUMmerGPU
  - MUMmerGPU++
  - Analysis of Space / Time Trade-off
- Experiments
- Discussion
- Conclusion

Read Alignment Problem

- Problem definition

  Find all maximal matches of query q on the reference sequence.

  - parameter: minimum match length

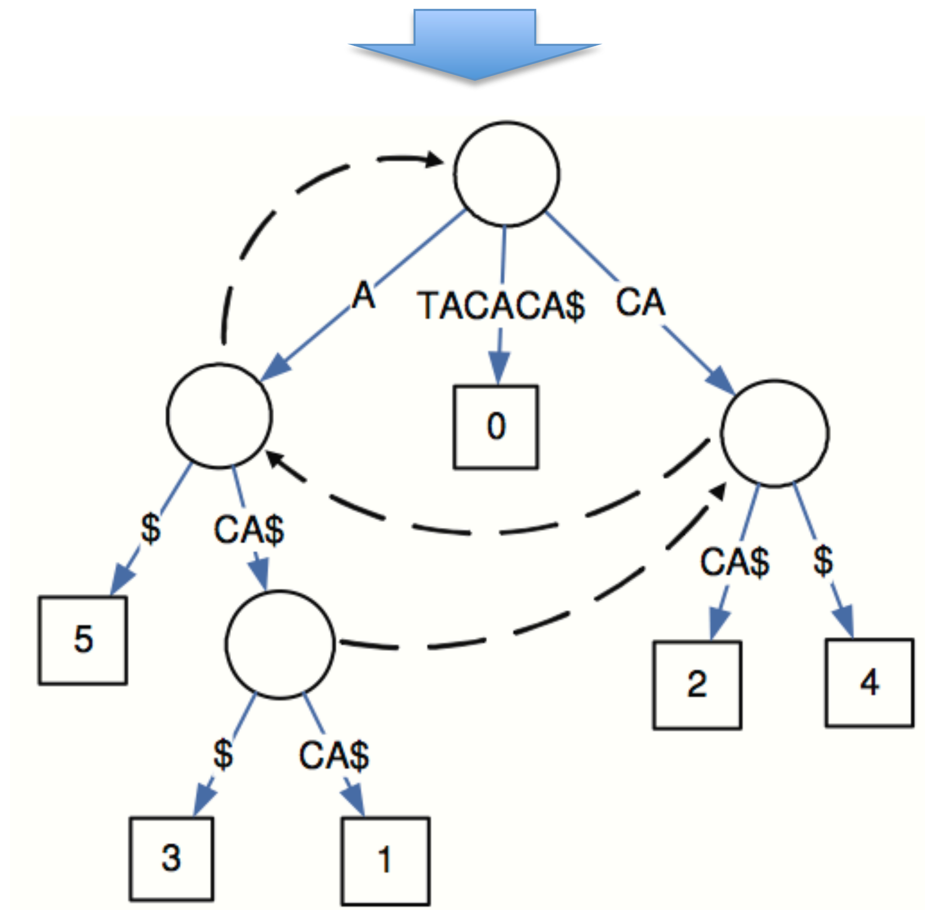
- Workload characteristics
  – Both of number of queries and reference sequence length are large
  – queries are short

Workload / Species	Reference sequence length	# of queries	Sequencing technology (read length)
HS1 - Homo sapiens chromosome 2	238,202,930	78,310,972	454 (~200)
HS2 - Homo sapiens chromosome 3	100,537,107	2,622,728	Sanger (~700)
MONO - L. monocytogenes	2,944,528	6,620,471	454 (~120)
SUIS - S. suis	2,007,491	26,592,500	Illumina (~36)

Suffix Tree

- trie-like structure…
- Time complexity:
  - search: O(m)
  - **suffix link**
- Space complexity:
  - # of nodes: O(n)
  - practically 20 * n Bytes

“TACACA”



m: query length
n : reference length
Q: queryset

Suffix Array

- similar matching operations to suffix tree, but less space complexity
- Time complexity:
  - search: O(m + log n)
  - **LCP Array**
- Space complexity:
  - O(n)
  - in practice, 3~5x less space than suffix tree

"TACACA"

Index	Suffix	Suffix Array	LCP Array	Rank Array (Suffix Array $^{-1}$)
0 (smallest)	A	5	0	5
1	ACA	3	1	2
2	ACACA	1	3	4
3	CA	4	0	1
4	CACA	2	2	3
5 (largest)	TACACA	0	0	0

m: query length
n : reference length
Q: queryset

GPGPU Programming

- three stages:
    1. transfer input data to the GPU’s internal memory
    2. launch the processing “kernel”
    3. transfer output
- GPU has no direct access to the host’s memory nor to its i/o devices.
    - need to allocate i/o buffers on local memory

Challenges

1. Limited onboard GPU memory
   – 60GB data (3Gbp DNA, suffix tree) >> 1.5GB memory (GeForce GTX 480)
2. Limited access to other I/O devices
   – need i/o buffers on GPU local memory
   – output size is unpredictable because of multiple alignments (max $O(mn|Q|)$)

Previous Effort: MUMmerGPU

- use suffix tree
- Divide
  - dividing the long reference string into shorter overlapping segments.
  - dividing the query set into smaller sized subsets.
  - reporting a complressed representation of the results
- **4-step**
  - **Copy in** : transfer the query subset and suffix tree to the GPU
  - **Matching :** queries of a query subset are aligned to the tree.
  - **Copy out:** transfer back.
  - **Post-Processing:** decompress the results and find other matches.

MUMmerGPU++

- almost the same as the MUMmerGPU, but using suffix array
- Matching:

```
/* Assumes SA, LCP and l global variables */
procedure Match(q, qlen) {
   i = 0
   while i ≤ qlen - 1 do {
      (si, ml) = BinarySearch(qi)
      RecordResult(qi, si, ml)
      i = i + 1
      while si != NULL and i ≤ qlen - 1 do {
   /* phase 1: cut the search space */
         i = i + 1
         s = ml - 1
         si = Rank[SA[si] + 1]
         j = SA[si] + s
         (r, ml) = Comp(Sj, qi+s)
   /* phase 2: find the longest */
         if r > 0 then {
            (si, ml) = ScanUp(s+ml, qi)
         } else {
            (si, ml) = ScanDown(s+ml, si, qi)
         }
         RecordResult(qi, si, ml)
         i = i + 1
      }
   }
}
procedure ScanUp(s, si, qi) {
   r = 1
   while LCP[si] > s and r > 0 do {
      si = si - 1
      j = SA[si] + s
      (r, ml) = Comp(Sj, qi+s)
      s = s + ml
   }
   return (si, s)
}
```

MUMmerGPU++

- Post-Processing:

```
/* Assumes SA, LCP and l global variables */
procedure PrintSubQueryAlignments(i, si, ml){
   /* print the longest one */
   PRINT(SA[si], i, ml)
   /* Scan up */
   v = si
   m = ml
   while v > 0 and m ≥ l do {
   /* the lcp could be longer than the
      match length, hence the minimum */
      m = MIN(m, LCP[v])
      v = v - 1
      PRINT(SA[v], i, m)
   }
   /* Scan down */
   v = si + 1
   m = MIN(ml, LCP[v])
   while v < reflen and m ≥ l do {
      PRINT(SA[si], i, ml)
      v = v + 1
      m = MIN(m, LCP[si])
   }
}
```

Analysis of Space/Time Tradeoffs(1)

- Matching Stage
  - Time complexity is expressed as follows:

$$T_d = kc_d t_d \alpha$$

  - suffix tree

$$T_{tree} = kc_{tree} \alpha O(m)$$

  - suffix array

$$T_{array} = kc_{array} \alpha O((m + \log(n/c_{array}))/r_{array})$$

    - r: efficiency of calculating the subqueries of a query

t: time complexity of each query
k: # of query subsets
c: # of segments
α: ratio (# of queries / # of SIMD processors)

Analysis of Space/Time Tradeoffs(1)

- Matching Stage

$$Speedup = \frac{T_{tree}}{T_{array}} = \frac{c_{tree}}{c_{array}} \times \frac{O(m)}{O((m + \log(n / c_{array})) / r_{array})}$$

t: time complexity of each query
k: # of query subsets
c: # of segments
α: ratio (# of queries / # of SIMD processors)

  - thee main factors
    - space ratio (3)
    - query to segment length ratio(1/2 – 1)
    - efficiency of calculating maximal matches
      - depends on workload

Analysis of Space/Time Tradeoffs(2)

- Post-Processing Stage
  - MUMmerGPU / Suffix tree
    - using GPU
    - need to know the result size
      - using additional information on suffix tree
    - stackless DFS
  - MUMmerGPU++ / Suffix array
    - scan the LCP array directly
  - latter approach is more efficient!

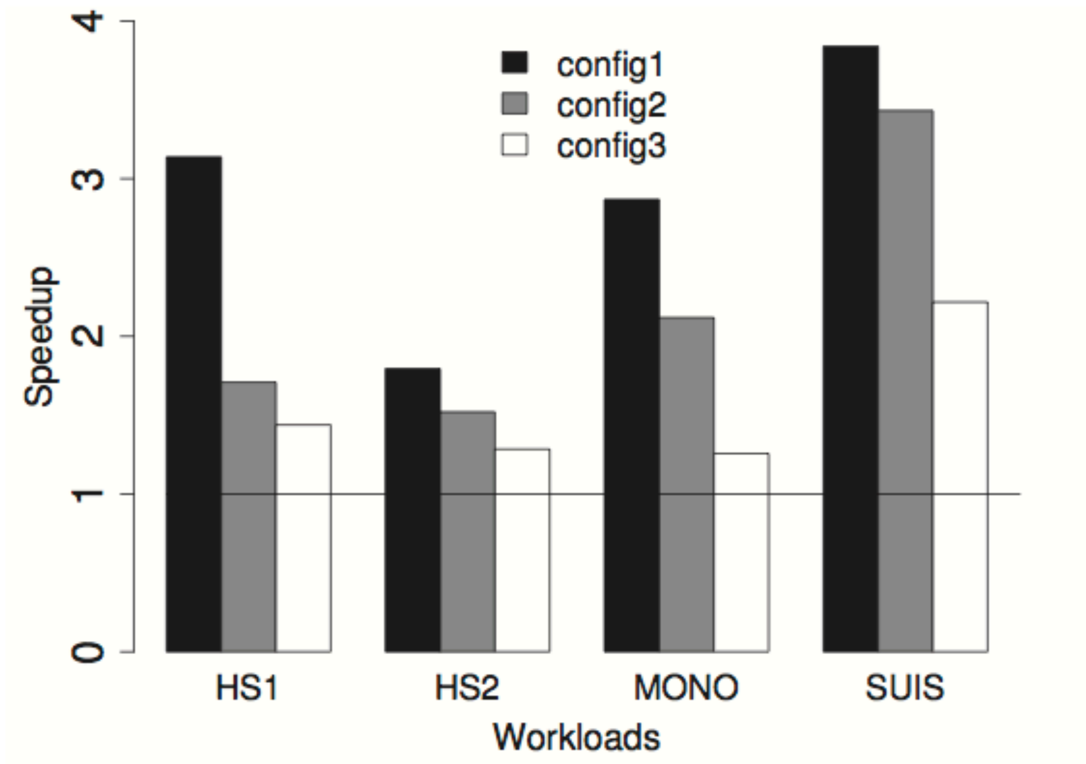
Analysis of Space/Time Tradeoffs(3)

- Data Transfer
  - 20 % of total exec. time on MUMmerGPU
  - suffix array reduces the cost because of better space efficiency
  - extra data transfer in suffix tree based approach.

Experiment

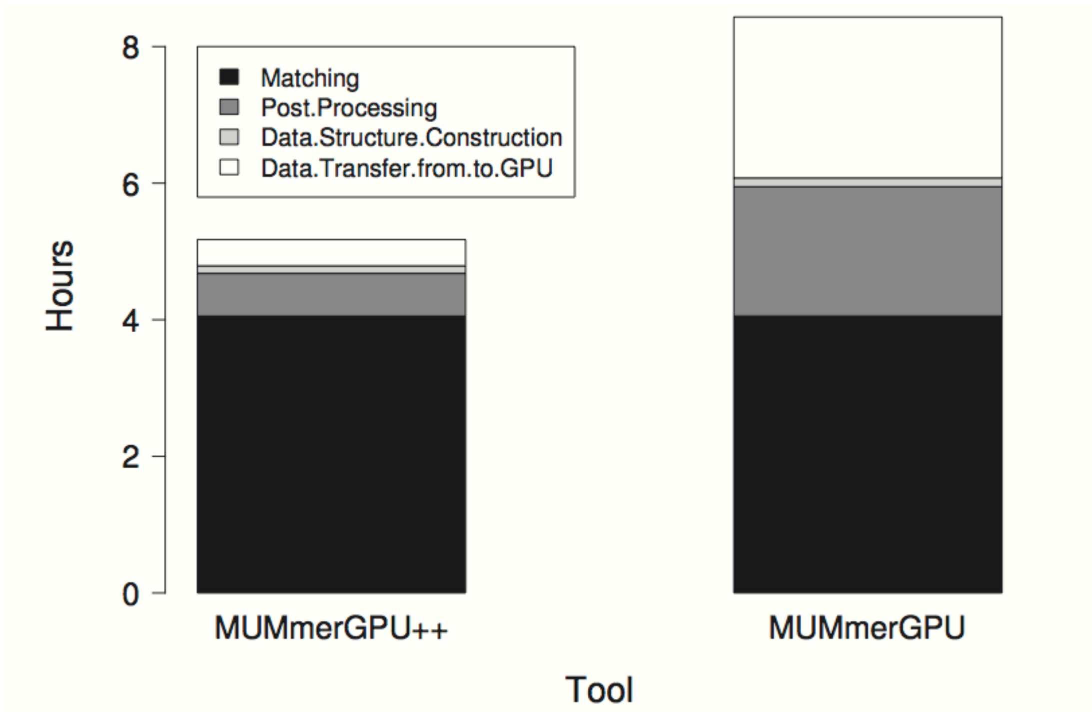
- machine characteristics:
  - Intel Core 2 Quad (Q6700 2.66GHz)
  - host's memory: 8GB
  - NVIDIA GeForce 9800GX2
    - dual gpu, 128 core x 2, 1500MHz , 1GB memory
    - PCIe 2.0 x16 bus
- memory division strategy: maximizing segments size
- MUMmerGPU++ does not aggressively optimize
  - focus on core data structure

Overall Speedup



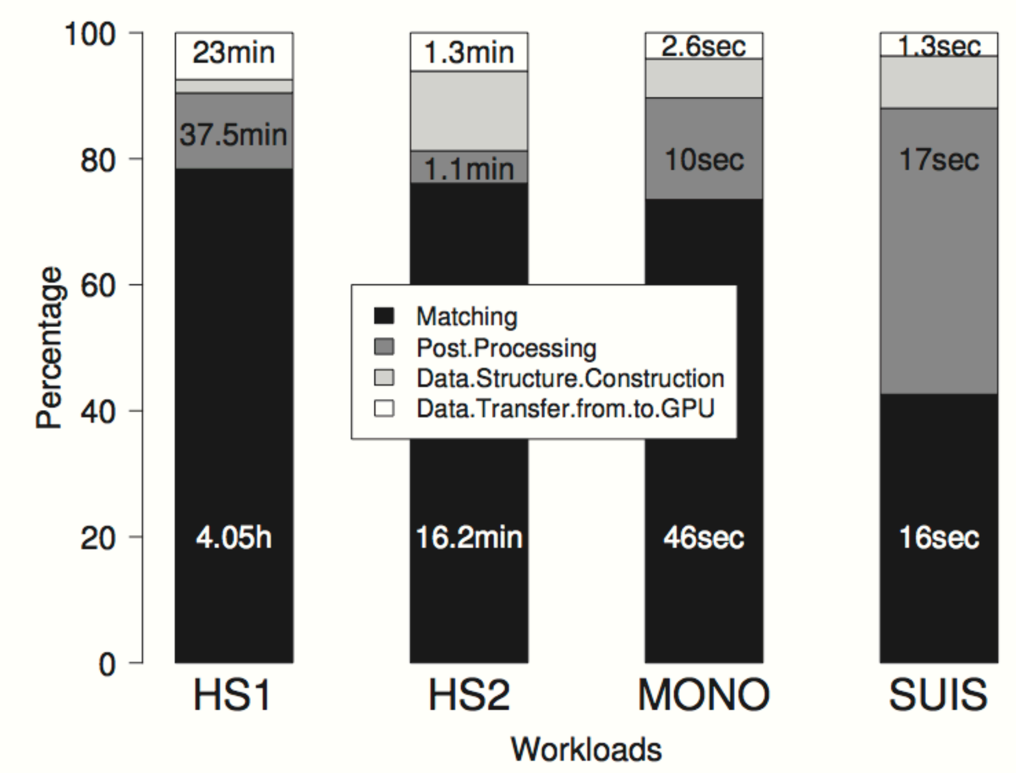
- achieve 1.52~3.43x speedup (config2)
- lower speedup in longer minimum-match length (config3)
  - number of alignment is decreasing, and matching st

absolute computation time



- Matching stage: MUMmerGPU ≦ MUMmerGPU++
- Post-Processing stage: MUMmerGPU > MUMmerGPU++
- Data Transfer: MUMmerGPU >> MUMmerGPU++

percentage of execution time in each stage



- i/o reduction on MUMmerGPU++
- allow optimizations on matching stage only ?

Discussion(1)

- Are the speedup offered by MUMmerGPU++ significant?
  - **YES :**
- Is it fair to use MUMmerGPU as a baseline to evaluate the advantages of the suffix array?
  - the analysis is solely based on the characteristics of data structure
  - MUMmerGPU is well optimized
  - MUMmerGPU++ is not specifically optimized

Discussion(2)

- Can the data transfer overheads be hidden by overlapping the transfers with the GPU kernel execution?
  - **NO:** because data transfer requires i/o buffers.

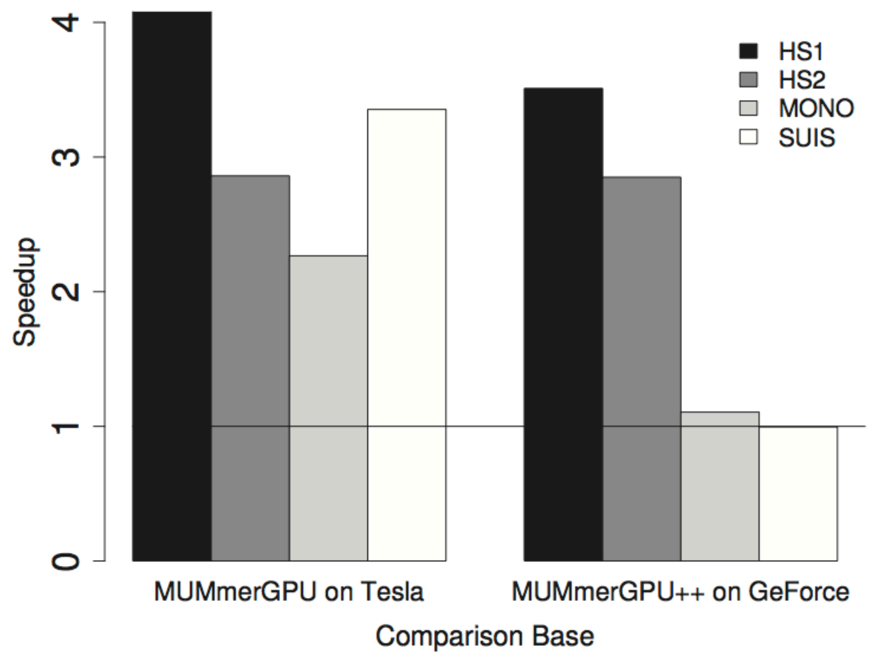
Energy comsumption

Tool	*kWh*	*Running time (minutes)*	*Watt*
MUMmerGPU++	**0.07**	21	200
MUMmerGPU	**0.12**	36	200
MUMmer	**0.76**	256	178

workload: HS2 / config2

- energy comsumption is linearly proportional to the computation time
- CPU-based tool uses energy at a lower rate
- (only 13% better performance on the hybrid architecture)

Comparison with high-end GPU



•

Conclusion

- GPUs have different characteristics
  - high memory access bandwidth, computational power
  - low internal memory space
- so we need to reconsider the choice of the data structures on GPU-supported platforms.