Ch.12 Working with Big Data in Bioinformatics

12.1 Introduction

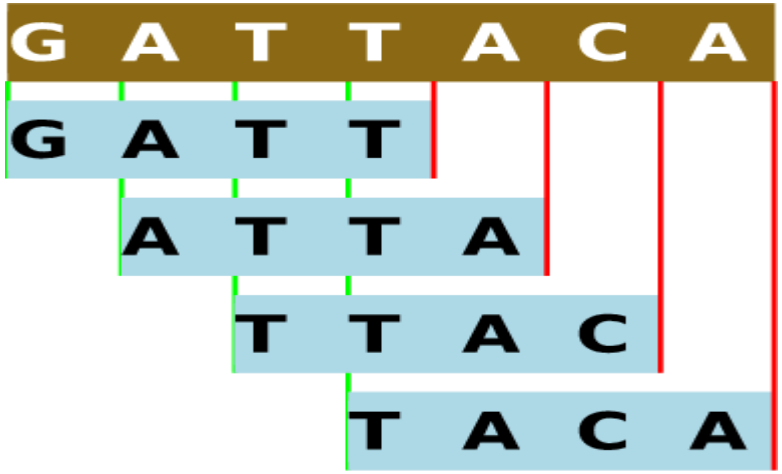
- Bioinformatics and Big Data
  - Sequencing
    - DNA sequencing
    - PCR
    - Nucleotide
      - G (Guanine) ⟷ C
      - A (Adenine) ⟷ T or U
      - T (Thymine) ⟷ A
      - C (Cytosine) ⟷ G
      - U (Uracil) not in DNA

12.1 Introduction

- Bioinformatics and Big Data
  - Sequencing
    - Big Data
    - HPC (High Performance Computing)
    - Pre processing for downstream analysis

12.1 Introduction

- What is the khmer software
  - K-mer vs. polymer (중합체)
    - K-mer : k개의 polymer (ex : 4-mer)



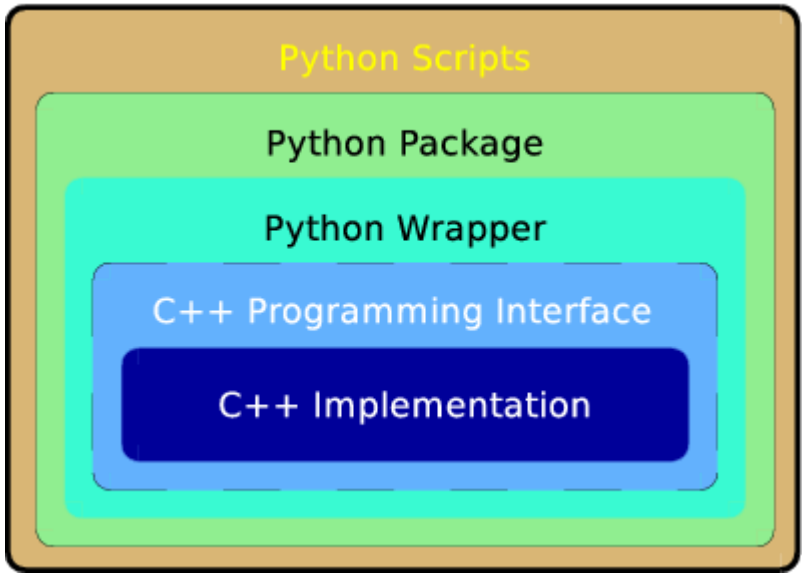
    - K-mer의 개수(count) = 뉴클레오티드 개수 -k+1
    - 4 = 7 - 4 + 1

12.1 Introduction

- What is the khmer software
  - Core function : k-mer의 개수 세기 (counting)
  - 자료구조 : Bloom Filter
    - 해시 테이블
    - 확률적 자료구조

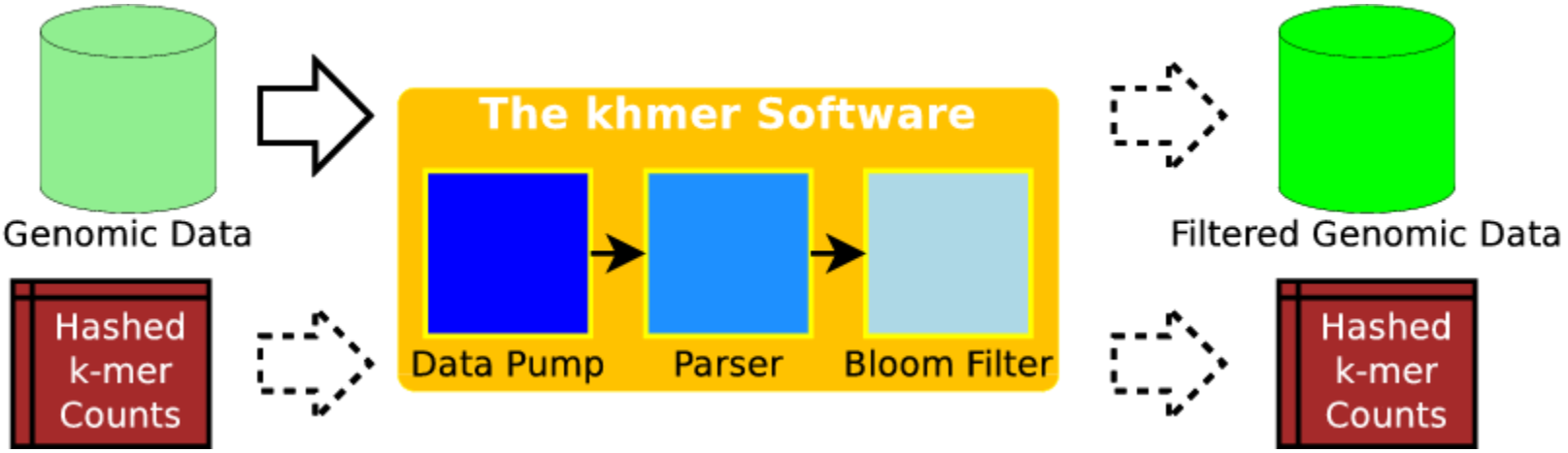
12.2 khmer의 구조와 성능상의 고려사항

- Software 구조 (a layered view of the khmer)



12.2 khmer의 구조와 성능상의 고려사항

- 자료흐름 (data flow)



12.2 khmer의 구조와 성능상의 고려사항

- 자료펌프 관련 유의 사항 (책 p.259)
  - Are we fully exploiting the fact that the data is accessed sequentially?
  - Are enough pages of data being prefetched into memory to minimize access latency?
  - Can asynchronous input be used instead of synchronous input?
  - Can we efficiently bypass system caches to reduce buffer-to-buffer copies in memory?
  - Does the data pump expose data to the parser in a manner that does not create any unnecessary accessor or decision logic overhead?

12.2 khmer의 구조와 성능상의 고려사항

- 파서 관련 고려 사항 (책 p.260)
  - Have we minimized the number of times that the parser is touching the data in memory?
  - Have we minimized the number of buffer-to-buffer copies while parsing genomic reads from the data stream?
  - Have we minimized function call overhead inside the parsing loop?
  - The parser must deal with messy data, including ambiguous bases, too-short genomic reads, and character case. Is this DNA sequence validation being done as efficiently as possible?
  - For iterating over the k-mers in a genomic read and hashing them, we could ask:

12.2 khmer의 구조와 성능상의 고려사항

- 해싱 코드 관련 고려사항 (책 p.260)
  - Can the k-mer iteration mechanism be optimized for both memory and speed?
  - Can the Bloom filter hash functions be optimized in any way?
  - Have we minimized the number of times that the hasher is touching the data in memory?
  - Can we increment hash counts in batches to exploit a warm cache?

12.3 Profiling and Measurement

- Program 각 부분이 소비하는 시간의 계량화 필요
  - Profiling 도구의 사용
    - GNU profiler (gprof)
    - TAU (Tuning and Analysis Utilities)
    - Instrumentation

12.3 Profiling and Measurement

- Code Review 결과 (책 p.262)
  - We expected the highest traffic to be in the k-mer counting logic.
  - Redundant calls to the **toupper** function were present in the highest traffic regions of the code.
  - Input of genomic reads was performed line-by-line and on demand and without any readahead tuning.
  - A copy-by-value of the genomic read struct performed for every parsed and valid genomic read.

12.3 Profiling and Measurement

- Profiling Tools
  - /usr/bin/time
  - gprof =➔ not used (because of OpenMP)
    - gprof does not support multi-thread
  - TAU ('Tuning and Analysis') by University of Oregon
    - MPI (message passing interface) support
    - But khmer does not use MPI
    - OpenMP support
    - Instrumental profiling 가능

12.3 Profiling and Measurement

- Manual instrumentation
  - IPerformanceMetrics class 사용

- (참조) profiling
  - Instrumentation 방식
  - Sampling 방식
    - Program counter (register) monitoring

12.4 Tuning

12.5 General Tuning

- 용어
  - Aggressive optimization
    - O2, O3
    - ==➔ 이식성 문제
  - Profile guided optimization
    - PGO option in Visual Studio
    - Code 수정의 위험성
- Tuning 작업의 방향
  - algorithmic improvement !!

12.5 General Tuning

- Data pump and Parser Operations
  - K-mer counting time > input (from storage)time
  - 따라서 counting time 최적화부터 해야 함
  - But, data pump와 parser도 개선 이유 있음
    - Multi thread 化 for scalability
    - '메모리 대 메모리 복사 횟수 감축' for 'bloom filter 효율성 증가'
    - Counting time이 상당히 줄어 들었을 경우에 대비하여 input time도 개선 필요
    - 유지보수, 확장성
  - 0_DIRECT 플레그를 통해 OS 가 관리하는 cache 우회
  - Posix_fadvise(2)를 통해 OS가 순차적 접근 자료를 미리 읽어오도록 hint 를 줌

12.5 General Tuning

- Bloom filter operations
  - 'toupper' in C library

```
#define is_valid_dna(ch) \
    ((toupper(ch)) == 'A' || (toupper(ch)) == 'C' || \
     (toupper(ch)) == 'G' || (toupper(ch)) == 'T')
```

and:

```
#define twobit_repr(ch) \
    ((toupper(ch)) == 'A' ? 0LL : \
     (toupper(ch)) == 'T' ? 1LL : \
     (toupper(ch)) == 'C' ? 2LL : 3LL)
```

  - 'toupper too heavy

12.5 General Tuning

- Bloom filter operations
  - 정규화 과정 최적화

```
#define quick_toupper( c ) (0x60 < (c) ? (c) - 0x20 : (c))↵
```

  - 분기가 없는 **더 빠른 toupper**

```
c &= 0xdf; // quicker toupper↵
```

12.6 Parallelization (for khmer optimization)

- Thread 기반 병렬화
  - 자료가 RAM에 있는 경우에 이익이 있음
  - 저장 매체의 대역폭이 작을 경우
    - Multi thread간의 대기시간 증가
    - ==➔ 병렬화의 이익이 별로 없음
  - Data pump & parser의 최적화를 통해 미리 읽어와서 buffer에 저장
    - ==➔ thread 기반 병렬화의 이익을 볼 수 있음

12.6 Parallelization (for khmer optimization)

- Thread-safety and Threading
  - Critical section, racing and dead lock
  - Thread 프로그램이 항상 thread-safety 한 것은 아님
  - API 를 통해서 해결
    - API가 thread 상태의 객체를 관리하도록 progamming
    - Thread는 함수 진입시에 thread 식별 번호를 조회해야 하는 부담이 있음
    - 하지만 기존의 single thread 방식의 program의 interface를 그대로 사용 가능함

12.6 Parallelization (for khmer optimization)

- Data Pump & Parser Operations
  - NUMA (non uniform memory access)
    - Multi core or multi processor System
    - Memory와 core간의 거리로 인한 latency
  - 해결
    - Buffer를 실행 thread 수와 같은 수로 나눔
    - 각 thread가 buffer를 연결시켜서 core와 memory의 접근성 증가

12.6 Parallelization (for khmer optimization)

- Bloom filter operations
  - Thread들이 해싱 테이블 공유
    - 경합 (data racing) 문제 발생
    - ==➔ atomic 으로 문제 최소화
    - ==➔ GNU compiler에서 atomic 지원
  - 미해결 문제
    - K-mer counting 후에 해시테이블을 저장소에 기록할 때 생기는 병목

12.6 Parallelization (for khmer optimization)

- Scaling (규모 확장성)
  - 1.9 x (good !)

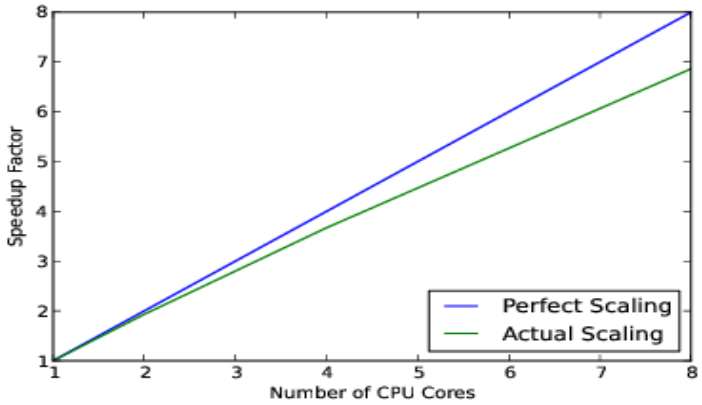


Figure 12.4 - Speedup factor from 1 to 8 CPU cores

  - Amdahl's law
    - 병렬화를 통한 성능 증가는 serial 구간에 의해 결정

12.7 Conclusion

- Project의 목적
  - 정확성 (# 1 !!!)
  - 정확성 & 유용성
  - 성능 & 규모 가변성 (performance and scalability) =➔ the 2nd consideration.
- 하지만 이 최적화를 통해
  - single thread의 성능 증가,
  - multi thread의 scalability,
  - 유지보수, 확장성 등의 이익을 얻을 수 있었다.

12.7 Future Direction

- API 확장
- Use case 문서화
- 잘 특성화된 (well-characterized) component 제공
- +
  - Low memory data structure theory 활용
  - 분산알고리즘 연구
  - Rolling hash function 추가