C++ Programming

Lecture 12
Software Engineering Group

Philipp D. Schubert
Become a Teaching Assistant for SWTPra/SoPra in WS 2019
Revised & Updated: Scrum, up-to-date libraries and frameworks, ...

- Winter term 2019/2020
  - Tutor or programming tasks
  - 9,5h per week

- Tasks
  - Supervise a SWTPra/SoPra group
  - Participate in weekly tutor meetings

- Outcome
  - TAs become involved in our chair
  - Learn Scrum project management
  - Find topics for your thesis....

- Are you Interested? Contact
  - philipp.schubert@upb.de
Contents

1. Introduction to the final project
   1. Reading files
   2. Smith-Waterman algorithm
   3. Comparing the sequences / parallelization
   4. Post processing
2. Miscellaneous & advanced topics
3. What next?
Introduction to the project

- Compare genome sequences to each other
- DNA sequencing machines
  - Decode DNA molecules
  - Produce massive sequence (text) files
  - E.g. Ion torrent sequencer
    - $50,000
    - Small desktop machine

[Figure taken from http://gizmodo.com/5709604/got-50000-you-can-buy-yourself-a-personal-dna-sequencing-machine]
Introduction to the project

- Sequence alignment
  - ATTGACCTGA
  - ATCCTGA
  - How to find an optimal alignment?

- Smith-Waterman algorithm
  - Finds optimal alignment score (similarity)
  - Finds optimal alignment (according to the score)

[Figure taken from http://gizmodo.com/5709604/got-50000-you-can-buy-yourself-a-personal-dna-sequencing-machine]
Introduction to the project

- What is an alignment?
  - An alignment is a sequence of operations
    - Substitution
    - Copy
    - Deletion
    - Insertion
    - which transforms one sequence into another
Reading files

- Sequences files in fasta format
  - .fasta
  - .fas
  - .fa

> A fasta example header
ATAAGGTACGACACACT
AGATACACACATGAAAG
AACAGACTATATATTTT

- Sequence files can be huge
  - Reading line by line is usually too slow
  - Read file as one block
  - Runtime of the algorithm $O(|n| \cdot |m|)$
    - No need for memory mapped files

- Tasks:
  - Read files from disk
  - Remove the header line
  - Remove line breaks '\n'
  - Split these tasks into separate functions
Smith-Waterman algorithm

- Perform algorithm
  - ACGA
  - TCCG
- Weights
  - $\omega_{\text{mismatch}} = \omega_{\text{gap}} = -1$
  - $\omega_{\text{match}} = 2$
- Create matrix
  - Initialize first row to 0
  - Initialize first column to 0
  - Fill matrix according to recurrence
    - Highest matrix entry is the score
  - (Optimal alignment could be reconstructed from matrix)
- We are only interested in the score: Do we really need a matrix? $\rightarrow$ No!

$H(i, j) = \max \begin{cases} 
0 \\
H(i - 1, j - 1) + \omega(a_i, b_j) & \text{match/mismatch} \\
H(i - 1, j) + \omega_{\text{gap}} & \text{deletion} \\
H(i, j - 1) + \omega_{\text{gap}} & \text{insertion}
\end{cases}$

$\omega(a, b) = \begin{cases} 
\omega_{\text{match}}, & a = b \\
\omega_{\text{mismatch}}, & a \neq b
\end{cases}$
Parallelization, calling Smith-Waterman algorithm multiple times

- Compare each 50 character segment of sequence $n$ to each 50 character segment of sequence $m$ using Smith-Waterman

- Split into subtasks
  - Suppose usage of 2 threads
  - Split sequence $n$ into 2 parts
  - One thread compares every segment of first part to every segment of second sequence
  - Other thread compares segments of second part to every segment of second sequence
  - Caution at borders of parts
  - Caution for thread working on last part

- Both sequences are only read from
  - Make both sequences global variables!
How to model the tasks?

- Model as a class
- Provide member variables for all information required
  - Start of its corresponding part in sequence $n$
  - End of its corresponding part in sequence $n$
  - ...
- Provide a constructor to correctly initialize members
- Implement the call operator to start the actual computations

Example

```cpp
/* this is not complete */
class SWDTask {
private:
    size_t start_sp;
    size_t end_sp;
    int smith_waterman_distance(...);

public:
    SWDTask(size_t ssp, size_t esp);
    void operator() ()
};
```
Avoid unnecessary copies of `std::string`

- Copying just blocks your processor
  ```cpp
  int smith_waterman_distance(string a, string b);
  
  for (/* sensible loop */) {  
    smith_waterman_distance(/* ... */ , /* ... */);
  }
  
  A. Have the sequences as global variables and just pass start and end positions
    ```cpp
    string n = /* ... */;
    string n = /* ... */;
    
    int smith_waterman_distance(int start, int end);
    
    for (/* sensible loop */) {  
      smith_waterman_distance(/* ... */ , /* ... */);
    }
  ```

- Runtimes may vary from several seconds up to one hour!
  ```cpp
  B. Or use C++ 17 `std::string_view`
  ```
Post processing

- For each starting position in one sequence
  - Find the starting position in the other sequence with the highest score
  - Add this highest-score-triple to your post-processed final results

<table>
<thead>
<tr>
<th>Start in SOX3</th>
<th>Start in SRY</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>80</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>85</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>81</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>90</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
<td>96</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>72</td>
</tr>
</tbody>
</table>

These results are made-up
Writing results to file

- Write the post-processed results back to a file
- Use a csv (comma separated values) file format
  
  `SOX3,SRY,Score
  10,20,74
  14,25,80
  123,243,96
  214,501,81`

  These results are made-up

- Plot the results by using the python script
- Or plot the results by using a spread-sheet software like MS Excel or Libre Office

- Send me your solution as an email
  - The entire source code and compile command (e.g. Makefile), (plots are optionally)
  - Include your complete name (first name, middle name, last name), field of study and faculty
Results before preprocessing
Results after preprocessing
Questions to the project?

This is not a group project: plagiarism will not be tolerated.
There is still more!
Optimize optimized things

- “Writing fast code”, Andrei Alexandrescu
  
  - Part I  
    https://www.youtube.com/watch?v=vrfYLI8X8k&t=1201s
  
  - Part II  
    https://www.youtube.com/watch?v=9tvbz8CSl8M&t=208s

- Example

```c
size_t count_digits(size_t number){
    size_t digits = 0;
    do {
        ++digits;
        number /= 10;
    } while (number);
    return digits;
}
```

- An (micro-)optimized example

```c
size_t count_digits(size_t number){
    size_t digits = 1;
    for (;;) {
        if (number < 10) return digits;
        if (number < 100) return digits + 1;
        if (number < 1000) return digits + 2;
        if (number < 10000) return digits + 3;
        number /= 10000;
        digits += 4;
    }
}
```

- Why is the second version faster?

  - Division is a more expensive operation
  - Comparison and addition is much cheaper
Miscellaneous

- Very incomplete list of names to know
  - Bjarne Stroustrup
  - Andrei Alexandrescu
  - Chandler Carruth
  - Sean Parent
  - Herb Sutter
  - Scott Meyers
  - ... many more

- C++ on youtube
  - CppCon
  - code::dive
  - ... many more
Allocators for container types

- C++ concept: Allocator

```cpp
#include <iostream>
#include <memory>
using namespace std;
int main() {
    // usually
    int* i = new int(42);
    int* array = new int[10];
    delete i;
    delete[] array;
    // one level deeper
    allocator<int> a;
    int* other = a.allocate(10);
    for (int i = 0; i < 10; ++i)
        other[i] = 2;
    a.deallocate(other, 10);
    return 0;
}
```

- Every STL/BOOST container can be parameterized by an allocator!
- Allocator defines an allocation strategy
  - When to allocate memory?
  - When to deallocate memory?
Allocators for container types

- Calls to `new` and `delete` are bottle-necks in HPC.
- Calls go to the operating system, everything else has to wait.
- Imagine some iterative algorithm:

```cpp
matrix a = // some matrix;
matrix b = // some matrix;

// some iterative algorithm
while (some condition) {
    matrix c = a * b;
    a = update(a, c);
    b = update(b, c);
}
// use matrix a, b, c
```

- Suppose `matrix` allocate its elements on the heap.
- `new` and `delete` are called many times!
  - If `operator*` and `update()` are optimized, `new` and `delete` are a bottle-neck.
  - A custom allocator can help!
Allocators for container types

- Allocators allow to define your own allocation strategy
- For example (most game consoles do this)
  1. Call `new` only once at program start
  - Allocate everything you need up-front
  2. At runtime your allocator takes care
  3. Call `delete` only once at the end of your program
- BOOST provides some allocators already
- Caution
  - Objects allocated with different allocators cannot be used together!

- The minimal allocator

```cpp
#include <cstddef>

template <class T>
struct SimpleAllocator {
    typedef T value_type;

    SimpleAllocator(/*ctor args*/);

    template <class U>
    SimpleAllocator(const SimpleAllocator<U>& other);

    T* allocate(std::size_t n);
    void deallocate(T* p, std::size_t n);
};

template <class T, class U>
bool operator==(const SimpleAllocator<T>&, const SimpleAllocator<U>&);

template <class T, class U>
bool operator!=(const SimpleAllocator<T>&, const SimpleAllocator<U>&);
```
Separate allocation from initialization: \textit{new} and \textit{delete} revisited

- Allocating a type dynamically is a two step process
  - Allocate memory on the heap
  - Initialize the memory using the constructor

- Can we re-use the allocated heap memory?
  - Yes!

```c
struct S {
    int x;
    int y;
    S(int x, int y) : x(x), y(y) {}
}

int main() {
    S *s = new S(1, 2);
    s->x = 13;
    s->y = 13;
    delete s;
    return 0;
}
```
Separate allocation from initialization

- Use (default) \texttt{placement new}

```cpp
#include <iostream>
#include <cstdlib>
#include <memory>
using namespace std;

struct S {
    int x;
    int y;
    S(int x, int y) : x(x), y(y) {}  
    void print() {
        cout << "x: " << x << "y: " << y << 'n';
    }
};
```

- You can also define your own operator \texttt{new} and \texttt{delete}

```
int main() {
    // using the heap
    S *s = new S(1, 2);
    s->x = 13;
    s->print();
    // call dtor but do not free
    s->~S();
    // construct and place in 's'
    S *t = new(s) S(42, 1024);
    t->print();
    // call dtor and free
    delete t;
    // using the stack
    unsigned char buffer[100];
    // construct and place in 'buffer'
    S *u = new(buffer) S(11, 22);
    u->print();
    // is on stack, so call dtor
    u->~S();
    return 0;
}
```

Debug your code: gdb and lldb

- If the code is too complex to be executed in your head …
  let a debugger execute it for you!
- gdb GNU debugger
- lldb LLVM debugger
- Command-line debugging tools
- What is debugging:
  - Inspect your code and your variables, registers, … by executing it line by line
  - Set break points and halt your program at interesting points
  - Painful (but practical) to use in the command-line
  - Better use it within some IDE like VS Code
How to debug your code?

- Set break-points right before the code of interest
  - Multiple break-points can be set
- ‘Watch’ variables of interest
- Step through the code
- Detect where it goes wrong
- Fix the bug
- Check the fix

Six Stages of Debugging
1. That can’t happen.
2. That doesn’t happen on my machine.
3. That shouldn’t happen.
4. Why does that happen?
5. Oh, I see.
6. How did that ever work

[Figure taken from https://www.linkedin.com/pulse/debug-your-code-easy-way-sanette-tanaka-1]
How to debug your code?

- Compile your code with \texttt{-g}

\texttt{-g} Produce debugging information in the operating system's native format (stabs, COFF, XCOFF, or DWARF 2). GDB can work with this debugging information.

[...]

GCC allows you to use \texttt{-g} with \texttt{-O}. The shortcuts taken by optimized code may occasionally produce surprising results: some variables you declared may not exist at all; flow of control may briefly move where you did not expect it; some statements may not be executed because they compute constant results or their values are already at hand; some statements may execute in different places because they have been moved out of loops.

Nevertheless it proves possible to debug optimized output. This makes it reasonable to use the optimizer for programs that might have bugs.

[...]
How to debug your code using VS Code?

```makefile
all:
    clang++ -std=c++14 -Wall -Wextra -g -O0 main.cpp -o main

clean:
    rm -f main
```
// Verwendet IntelliSense zum Ermitteln möglicher Attribute.
// Zeigen Sie auf vorhandene Attribute, um die zugehörigen Beschreibungen anzuzeigen.
// Weitere Informationen finden Sie unter https://go.microsoft.com/fwlink/?linkid=830387
"version": "0.2.0",
"configurations": [

  {
    "name": "(gdb) Launch",
    "type": "launch",
    "request": "launch",
    "program": "${workspaceFolder}/main",
    "args": [],
    "stopAtEntry": false,
    "cwd": "${workspaceFolder}",
    "environment": [],
    "externalConsole": true,
    "MIMode": "gdb",
    "setupCommands": [
      {
        "description": "Enable pretty-printing for gdb",
        "text": "-enable-pretty-printing",
        "ignoreFailures": true
      }
    ]
  }
]
```cpp
#include <iostream>
#include <vector>
#include <string>
using namespace std;

int add(int i, int j) {
    return i + j;
}

int main() {
    int a = 4;
    int b = 0;
    int c = add(a, b);
    vector<int> v(c, 12);
    for (auto &i : v) {
        i = 8;
    }
    set<string> s;
    s.insert("Hello");
    s.insert("World");
    s.insert("!");
    return 0;
}
```

Stopped due to shared library event (no libraries added or removed)
loaded ".libs/lib-x86-64.so.6". Symbols loaded.

Breakpoint 1, main () at main.cpp:11
11    int main() {

Breakpoint 2, main () at main.cpp:12
12    int a = 4;
Loaded ".libs/lib-x86_64-linux-gnu/libstdc++.so.6". Symbols loaded.
Loaded ".libs/lib-x86_64-linux-gnu/libc.so.6". Symbols loaded.
Loaded ".libs/lib-x86_64-linux-gnu/libm.so.6". Symbols loaded.

Execute debugger commands using ".exec <command>", for example ".exec info registers" will list registers in use (when GDB is the debugger)
```cpp
#include <iostream>
#include <vector>
#include <set>
#include <string>

using namespace std;

int add(int i, int j) {
    return i + j;
}

int main() {
    int a = 4;
    int b = 6;
    int c = add(a, b);
    vector<int> v(c, 12);
    for (auto &i : v) {
        i = 8;
    }
    set<string> s;
    s.insert("Hello");
    s.insert("World");
    s.insert("!");
    return 0;
}
```
What next?

- Use C++ in your projects
- Get more experience
- Be curious
- Make mistakes
- Take your time
- C++ is huge
  - So specialize in what you like
- Reads books, blog articles, programming forums
- Learn the tools of professional software development
  - Build tools e.g. make, cmake, ...
  - Debuggers e.g. gdb, lldb
  - Tools from the compiler tool chain e.g. nm
  - Version control systems e.g. git (https://git.cs.upb.de)
Designing code analysis for large-scale software systems (DECA)

- Winter semester 2019/2020
- Lead by Eric Bodden

Contents

- Intra-procedural data-flow analysis
- Call-graph construction algorithms
- Context-insensitive inter-procedural data-flow analysis
- Context-sensitivity using the call-strings approach
- Value-based context
- Context-sensitivity using the functional approach
- Efficiently solving distributed problems in the IFDS and IDE frameworks
- Current challenges in inter-procedural static program analysis
- Applications to software security
Overview on Data-Flow Analysis

- SEPL Goethe University, Frankfurt am Main
- Lecture “Foundations of Programming Languages”, Christoph Reichenbach
- Have a look at the following youtube playlist (one lecture unit)
  - https://www.youtube.com/watch?v=sxiFwiCgoVo&list=PLgJZZQPiH1mHIZAyIF1baZbMplzxXn90o
- Optimizations and static analysis
  - What?
  - Why?
  - How?
#include <iostream>
using namespace std;
enum _multiplication { times, multiply, mult, $cool$ };
// tiny int wrapper to trick the type system
struct _int {
  int i;
  // implicit conversion int -> _int
  _int(int i) : i(i) {}  
  // implicit conversion operator _int -> int
  operator int() const { return i; }
};

_int operator/(int i, _multiplication m) { return _int(i); }

int operator/(_int j, int k) { return j * k; }

int main() {
  int a = 12 /times/ 2;
  int b = 144 /$cool$/ 3;
  int c = 4 /multiply/ 2 /mult/ 3;
  cout << "a= " << a << " b= " << b << " c= " << c << "\n";
  return 0;
}
Be critical

- Sadly not everything about C++ is great!
- A nice talk that sums up some of the biggest oddities of C++
- C++ bashing: “A case against C++”, Felix von Leitner
  - “Why C++ is bad for the environment, causes global warming and kills puppies”
  - https://media.ccc.de/v/cccamp07-en-1951-A_Case_Against_C++
- However, at the end I hope you find C++ somewhat useful!
Job offer

- Topic
  - Static analysis
  - C++ programming
  - LLVM compiler framework

- Benefits
  - Money ;-) 
  - Fun 
  - Learn a lot 
  - Invitations to our professional and social events 
  - Opportunity for bachelor / master thesis 
  - Lots of career options 
  - Working on an important topic 

- Just send me an email
Thank you for your attention

Questions?