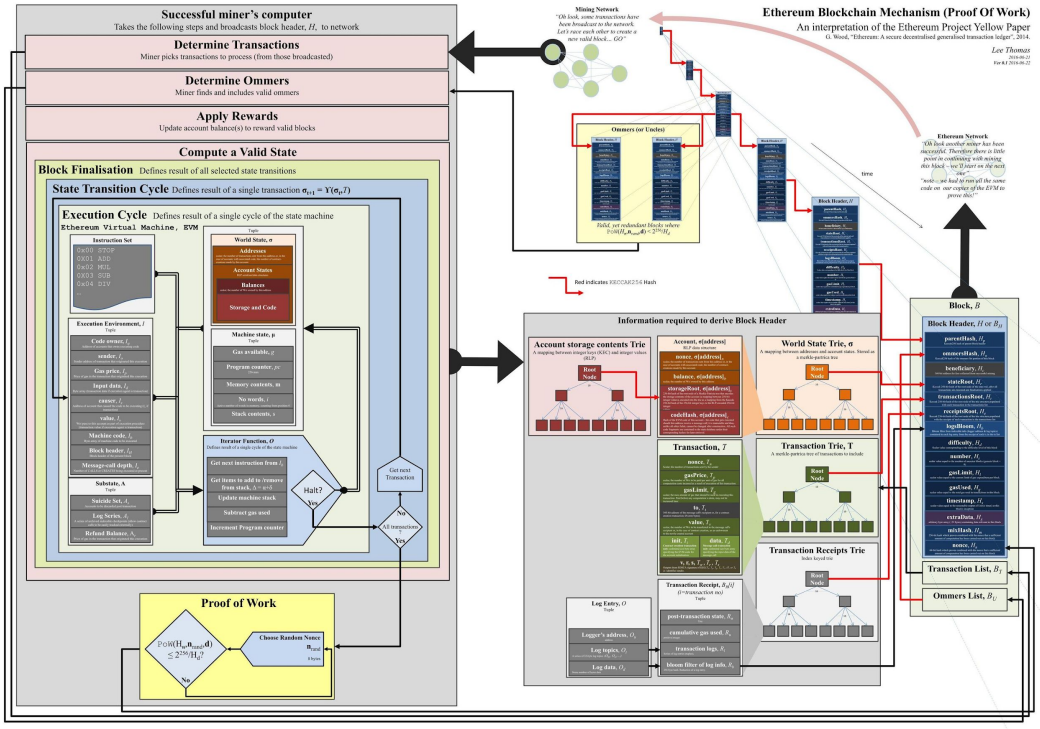
Profiling and Optimizing the Solidity Compiler

Killian Rutherford, Raphael Norwitz, Jiayang Li

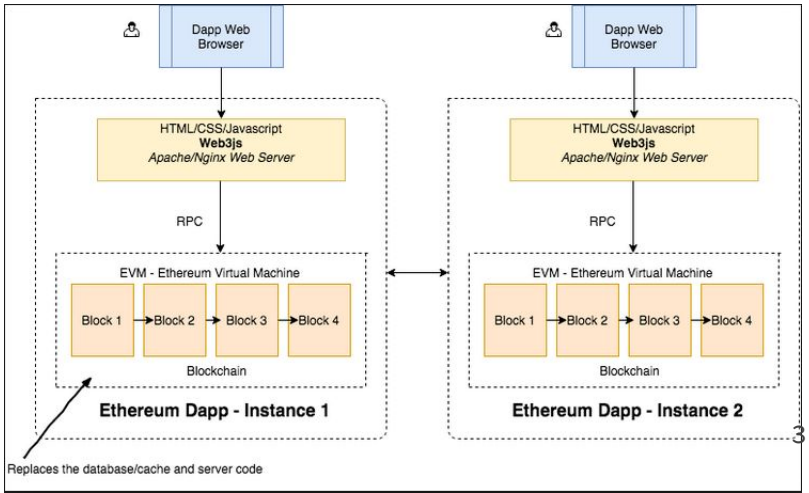
https://github.com/raphael-s-norwitz/solidity/tree/dev_tool_wrapper

Blockchain and Ethereum



Blockchain and Ethereum

- Blockchain ≈ Linked-list <batches (blocks) transactions>
  - transactions can be function call trace of programs → smart contract
- Ethereum Virtual Machine executes EVM bytecode
- Decentralization → all machines verify computations
- Applications:
  - Digital Currency
  - Anti-censorship
  - public records
- Tons of machines as witnesses
- Implemented by Merkle tree



Solidity

- Javascript-like semantics compiles down to EVM
  - Statically typed
  - Inheritance
  - Libraries
  - user-defined types
- Deployed contracts create distributed applications (dapps)
- Compiling test sets and libraries is slow
- Compiler doesn’t have any profiling interface
  - gcc -Q/--ftime-report

FTime Report - Goals

- Inspired by GCC’s -ftime-report option(Alex Samuel)
- Maintain a stack data structure
- Record time information in different stages of the compiling process
- Show users the stages examined and the elapsed time
- Identify stages that take long time

Examples and Usage

```
contract Coin {
    address public minter;
    mapping (address => uint) public balances;

    event Sent(address from, address to, uint amount);

    function Coin() public {
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        if (msg.sender != minter) return;
        balances[receiver] += amount;
    }

    function send(address receiver, uint amount) public {
        if (balances[msg.sender] < amount) return;
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

Coin.sol

Specify the --ftime-report flag for the solidity compiler

Example: solc --ftime-report Coin.sol

Examples and Usage

namespace/function name	unix begin time(μs)	time elapsed(μs)
CommandLineInterface::processInput()	358	3384
_CLI::readInputFilesAndConfigureRemappings	363	257
_CompilerStack::setRemappings	631	2
_CompilerStack::setEVMVersion	680	0
_CompilerStack::compile	684	3016
_CompilerStack::parse	684	175
_CompilerStack::analyze	860	1573
_CompilerStack::compileContract: Coin	2438	1254
_CompilerStack::createMetadata	2450	266
_Compiler::compileContract	2856	661
_ContractCompiler::compileContract	2861	303
_ContractCompiler::initializeContext	2869	26
_ContractCompiler::appendFunctionSelector	2896	94
_ContractCompiler::appendMissingFunctions	2994	168
_ContractCompiler::initializeContext	3175	7
_ContractCompiler::appendMissingFunctions	3200	0
_ContractCompiler::appendMissingFunctions	3201	0
_CompilerContext::optimse	3203	311
_ContractCompiler::initializeContext	3603	6
_ContractCompiler::appendMissingFunctions	3641	0
CommandLineInterface::actOnInput	3757	172

```
Void
CompilerStack::setEVMVersion(EVMVersion
_version)
{
    TimeNodeWrapper profile(t_stack,
"CompilerStack::setEVMVersion");
        solAssert(m_stackState <
State::ParsingSuccessful, "Set EVM
version after parsing.");
    m_evmVersion = _version;
}
```

Design: TimeNode (Chrono and Standard Library)

```
class TimeNodeStack
{
public:
     TimeNodeStack();
     ~TimeNodeStack();
     void push(std::string name);
     std::string pop();
     std::string printString(bool tree);
     void print();
     void print_recursive());
     bool print_flag = false;
     bool tree = true;
private:
     std::vector<TimeNode> stack;
     std::vector<TimeNode> print_stack;
     time_point start;
};
```

```
class TimeNodeWrapper
{
public:
     TimeNodeWrapper(TimeNodeStack& t_stack,
std::string given_name); // call stack.push()
     void pop(); // users may want to manually
pop to measure specific code
     ~TimeNodeWrapper(); // call stack.pop();

private:
     std::string name;
     bool popped;
     TimeNodeStack& stack;
};
```

BOOST test case

```
BOOST_AUTO_TEST_CASE(check_scoped_destructor){
     TimeNodeStack new_stack; TimeNodeWrapper test1(new_stack,"hello");
     {TimeNodeWrapper test2(new_stack, "world"); TimeNodeWrapper test3(new_stack,"!!"); }
     test1.pop();  std::string result = new_stack.printString(true)
     std::regex reg("(namespace/function name)[\\s]+(unix begin time)"
          "(.*)[\\s]+(.*)\n(-*)\n(hello)(\\s+)(\\d+)(\\s+)(\\d+)(.*)\n"
          "( \\\\_world)(\\s+)(\\d+)(\\s+)(\\d+)(.*)\n"
          "(   \\\\_!!)(\\s+)(\\d+)(\\s+)(\\d+)(.*)\n");
     BOOST_REQUIRE(std::regex_match(result, reg)); };
```

Results/Optimization

For large solidity contracts

- CompilerStack::analyze() and CompilerStack::optimise() take relatively long time (~60% compile time)

Study the functions called in these 2 functions further

- SMTChecker()
- applyMethods()

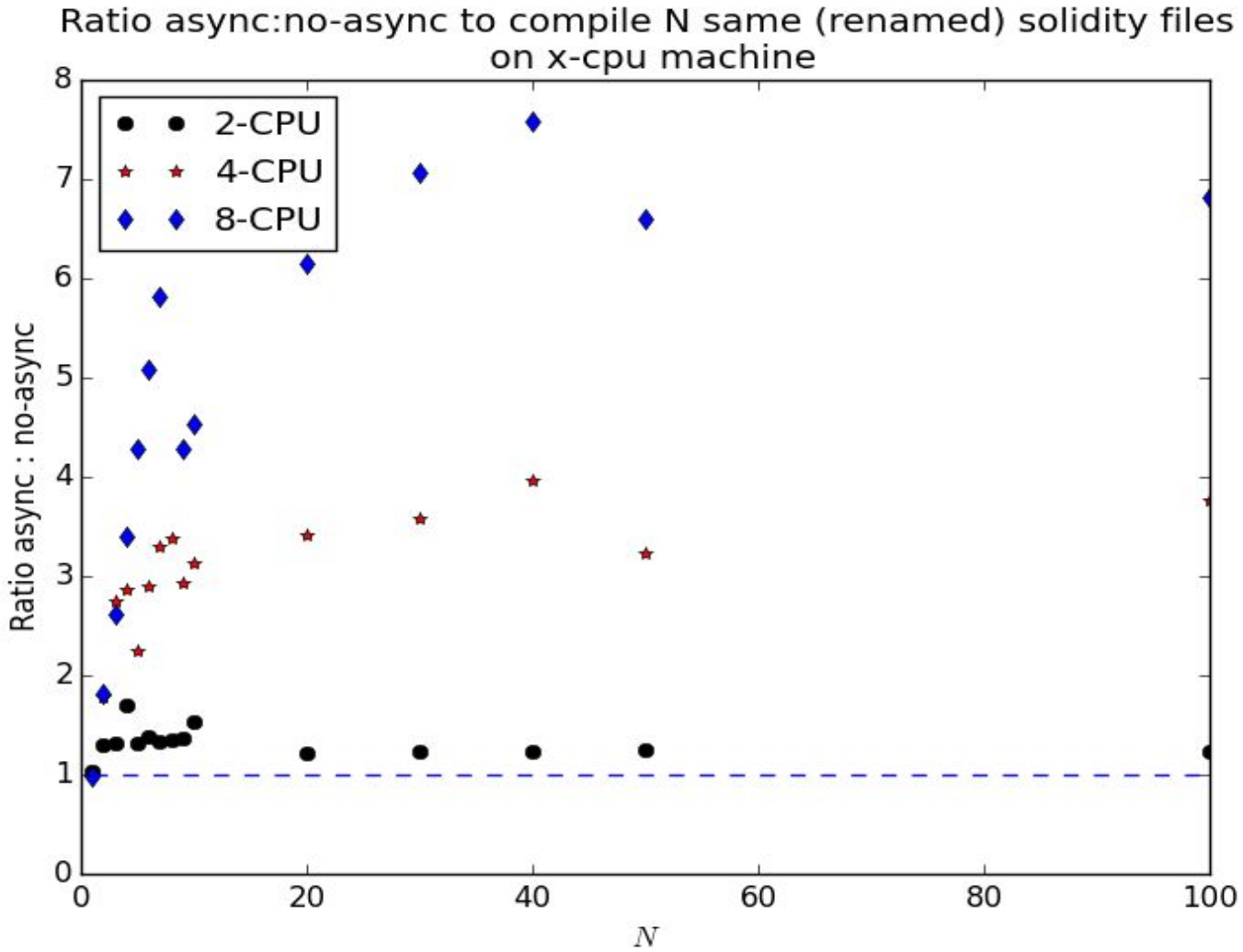
```
template <typename Method, typename... OtherMethods>
void applyMethods(OptimiserState& _state, Method, OtherMethods... _other)
{
        if (!Method::apply(_state))
                applyMethods(_state, _other...);
}

bool PeepholeOptimiser::optimise()
{
        OptimiserState state {m_items, 0, std::back_inserter(m_optimisedItems)};
        while (state.i < m_items.size())
                applyMethods(state, PushPop(), OpPop(), DoublePush(), DoubleSwap(), JumpToNext(),
                                UnreachableCode(), TagConjunctions(), Identity());
        if (m_optimisedItems.size() < m_items.size() || (
                m_optimisedItems.size() == m_items.size() && (
                        eth::bytesRequired(m_optimisedItems, 3) < eth::bytesRequired(m_items, 3) ||
                        numberOfPops(m_optimisedItems) > numberOfPops(m_items)
                )
        ))
```

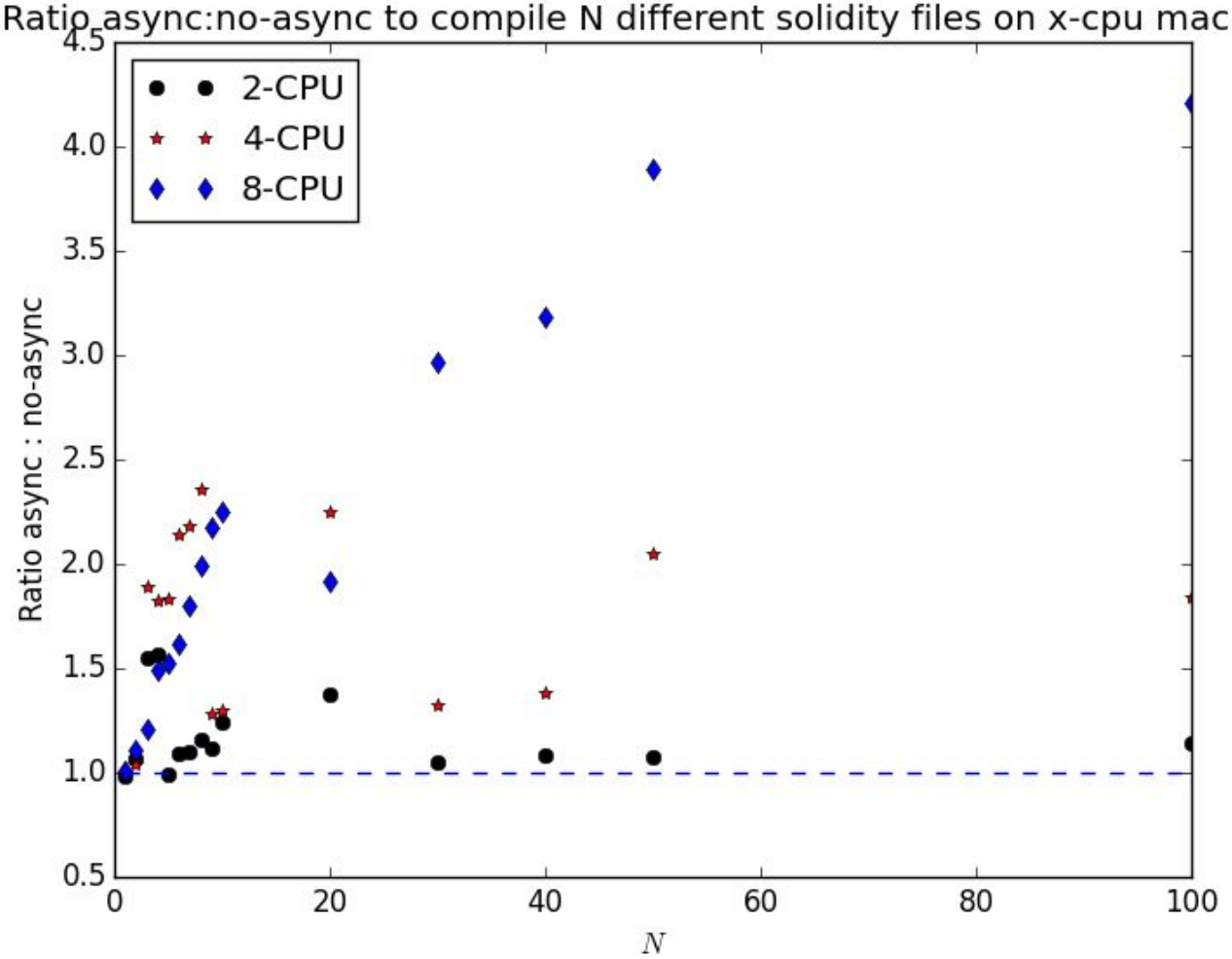
Parallelisation

- Solidity compiler can be run with multiple filenames after one another
- Higher level: Compiling a library by splitting files up across processors. Expected to pay off with large number of files since compiler has to be instantiated on each thread
- Lower level: To deal with dependency issues between files which higher level cannot handle

Parallelisation

- Benchmarking datasets:
  - Identical Files (but renamed each time and function names changed to avoid as much automatic compiler optimization as possible) - used BlindAction.sol
  - Random subset of files from github with all solidity files (to give “real world” potential improvement)
- For real life files, tested out of 10,000 files from github repo - ended with 100 files approximately 10,000 lines of solidity
- Using async to parallelise
- Tested on google cloud instances, n1 Standard vCPU

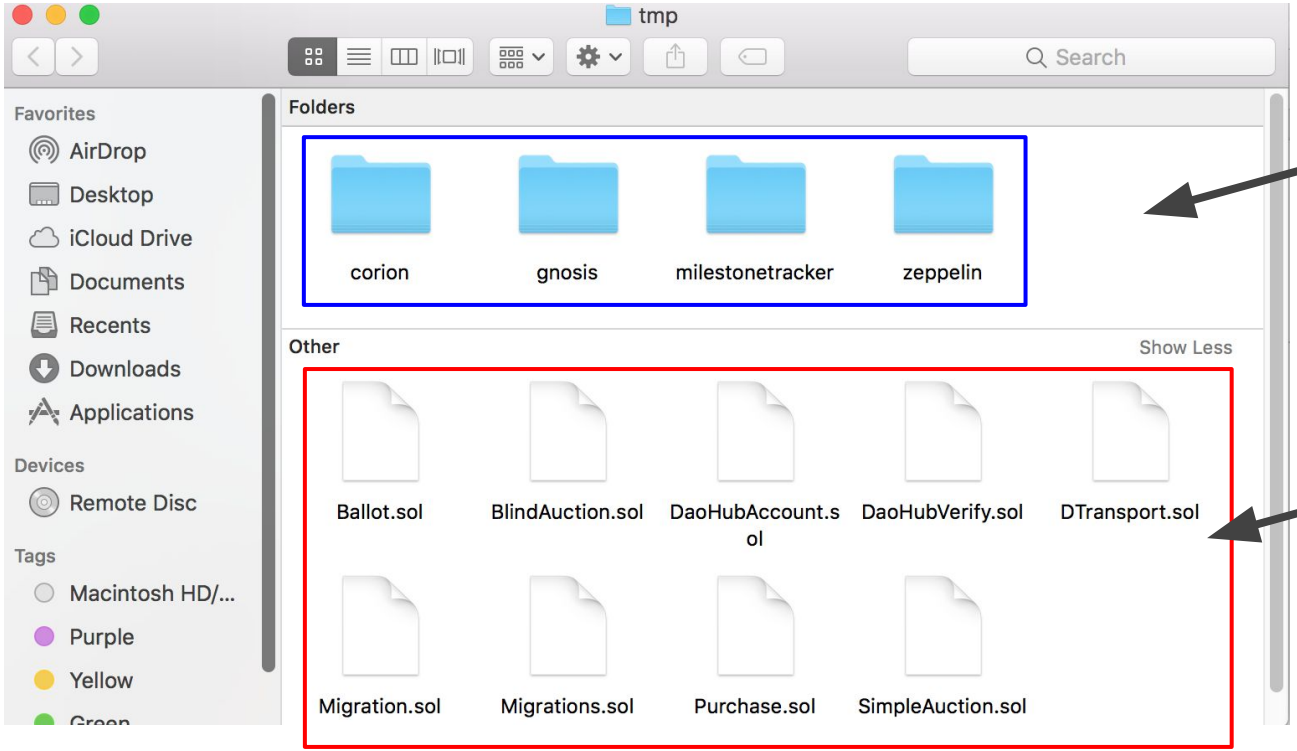

Ratio async:no-async to compile N same (renamed) solidity files
on x-cpu machine
2-CPU
4-CPU
8-CPU
Ratio async : no-async
N
0
1
2
3
4
5
6
7
8
0
20
40
60
80
100


Ratio async:no-async to compile N different solidity files on x-cpu machine
2-CPU
4-CPU
8-CPU
Ratio async : no-async
N
0
20
40
60
80
100
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5

Parallelisation Tool

- Experimented high-level parallelisation of the solidity compiling process
- Assumed a directory structure to tackle the dependency issues
- Developed from the code for parallelisation test
- Wrapped the compiling process and parallelised them with the help of C++’s async() and standard library

Parallelisation Tool



Files that depend on each other and need to be compiled together will live in the same subdirectory.

Files that are independent and can be compiled in any order/ together or separately.
Put them in the top directory.

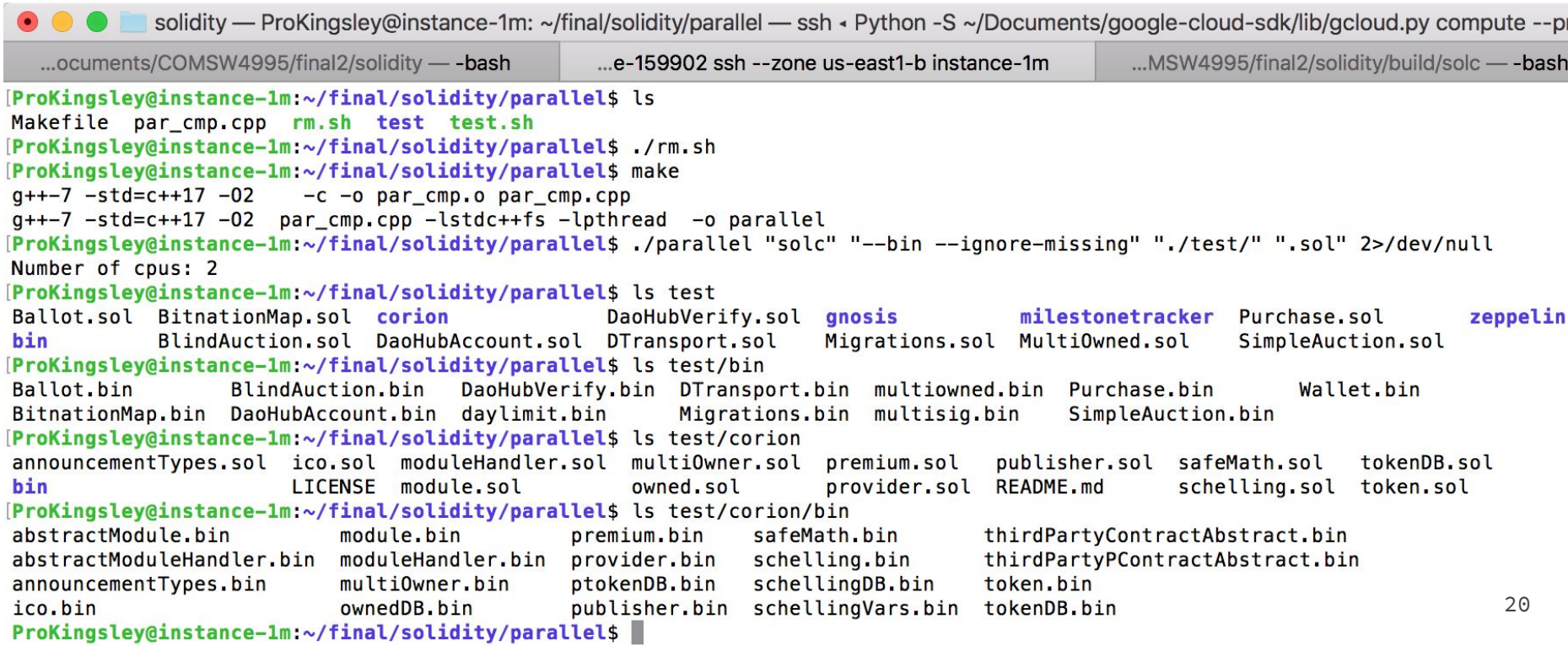
Parallelisation Tool

```
vector<tuple<std::experimental::filesystem::path, vector<string>>> pool;
distribute_tasks(path, extension, pool, true); // Populating pool with names of files to compile
for (auto & p : pool) {
  auto[path, sources] = p;
  if (path == root_path) { // Compile files in top directory in parallel
    parallel_compile(compiler, flags, sources, vec_res);
  } else {
    string tasks; // Assign a new thread for compiling each batch of dependent files
    for (auto & task : sources) {tasks += task + " ";}
    vec_res.push_back(async(std::launch::async, compile, compiler, flags, tasks, path));
  }
} // C++ algorithm
for_each(vec_res.begin(), vec_res.end(), [](future<int> &res){ res.get(); });
```

Parallelisation Tool

- Compile it with gcc-7
- Example command:
  - ./parallel “solc” “--bin --ignore-missing” “./test/” “.sol”
- Find the generated binary files in the “bin” directory generated during the process
- Each directory/ subdirectory has one “bin” directory

Parallelisation Tool



References and Acknowledgements

_ _ _

Options for Debugging Your Program or GCC - Red Hat

https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/4/html/Using_the_GNU_Compiler_Collection/debugging-options.html

timvar.c - Alex Samuel: https://github.com/gcc-mirror/gcc/blob/master/gcc/timevar.c

Ethereum: https://www.ethereum.org/

Solidity: http://solidity.readthedocs.io/en/latest/

@chriseth: Thanks for providing useful feedback

Prof. Stroustrup and TA's for course, guidance and feedback

Pictures:
-https://ethereum.stackexchange.com/questions/2286/what-diagrams-exist-to-illustrate-the-ethereum-blockchain-creation-process

-Silicon Valley/Parks and Recreation TV show

Questions?

Future direction:

-Report memory usage and space on blockchain

-Modify tool to support concurrency

More Info:

github.com/raphael-s-norwitz/solidity