Efficient programs

myJoin

Mihnea Aleman - 12333410
Ivan Babiy - 12142160
Artur Ohanian - 12239164
Jakob Petermandl - 11776823
Marijana Petojevic - 12017529
Alex Polner - 12024704
Paul Spörker - 11912400

V0: myJoin Baseline Implementation

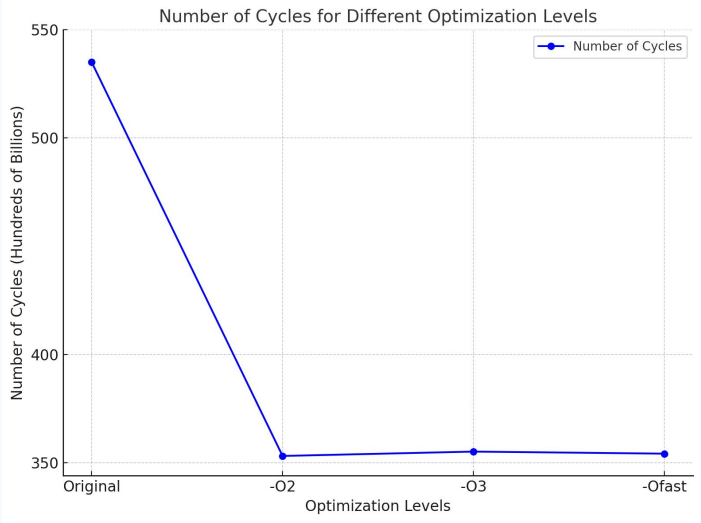
- C++ implementation
- Reading files line by line
- 4 nested for-loops
- Multi-stage join using `multimap`

```
multimap<string, string> map1(data1.begin(), data1.end());
multimap<string, string> map2(data2.begin(), data2.end());
multimap<string, string> map3(data3.begin(), data3.end());
multimap<string, string> map4(data4.begin(), data4.end());

for (const auto& [key1, value1] : map1) {
    auto range2 = map2.equal_range(key1);
    auto range3 = map3.equal_range(key1);

    for (auto it2 = range2.first; it2 != range2.second; ++it2) {
        for (auto it3 = range3.first; it3 != range3.second; ++it3) {
            auto range4 = map4.equal_range(it3->second);

            for (auto it4 = range4.first; it4 != range4.second; ++it4) {
                cout << it3->second << "," << key1 << "," << value1 << "," << it2->second << "," << it4->second << "\n";
            }
        }
    }
}
```

`Number of cycles:` 535.027.663.415

V1: Compiler flags

- Compiled with different flags
  - -O2
  - -O3
  - -Ofast



Number of cycles -O2: 353, 151, 865, 678
Number of cycles -O3: 355,163,021,185
Number of Cycles -Ofast: 354,211,527,370

V2: Improved line splitting

- Avoid creating a `stringstream`
- `find` and `substr`
- Use `emplace_back` instead of `push_b`

```cpp
// Split a string into tokens by a given delimiter
vector<string> split(const string &line, char delimiter) {
    vector<string> tokens;
    string token;
    size_t start = 0, end = 0;

    while ((end = line.find( c: delimiter,  pos: start)) != string::npos) {
        tokens.emplace_back(line.substr( pos: start,  n: end - start));
        start = end + 1;
    }
    tokens.emplace_back(line.substr( pos: start)); // Add the last token
    return tokens;
}
```

`Number of cycles:` 312 277 715 826

V3: Use std::move

- Use `std::move`
- Transfers resource from one object to another
- Avoids copying the string

```
auto tokens : vector<string>  = split(line,  delimiter: ',');
if (tokens.size() == 2) {
    data.emplace_back( t1: std::move(tokens[0]),  t2: std::move(tokens[1]));
}
```

`Number of cycles:` 308 833 085 225

V4: Use unordered_multimap

- Use `unordered_multimap` instead of `multimap`
- Insert and search in average O(1) instead of O(log(n))

```
unordered_multimap<string, string> map1;
unordered_multimap<string, string> map2;
unordered_multimap<string, string> map3;
unordered_multimap<string, string> map4;
```

`Number of cycles:` 121 187 831 303

V5: Mem reservations and String scope

- Based on myjoin4
- Reserve memory for `std::vector` in `read_file`
- Reduce scope of `std::string`
- Remove now unused `split` method

```
vector<pair<string, string>> read_file(const string &filename) {
    vector<pair<string, string>> data;
    ifstream file(filename);
    string line;

    file.seekg(0, ios::end); // Estimate file size for pre-allocat
    size_t estimated_lines = file.tellg() / 50; // Rough guess: 50
    file.seekg(0, ios::beg);

    data.reserve(estimated_lines); // Reserve space

    while (getline(file, line)) {
        //reduce scope of std::string
        size_t delim = line.find(',');
        if (delim != string::npos) {
            data.emplace_back(
                line.substr(0, delim),
                line.substr(delim + 1)
            );
        }
    }
```

`Number of cycles:` 109 042 856 561

V6: Loop unrolling

- Based on myjoin5
- Unroll the two `innermost` loops by `two`
- Reduction of the branch overhead as well as the overall iterations
- Use a buffer for the output and flush it in one go for a more efficient output

```
std::ostringstream buffer; // Use a buffer for efficient output

for (const auto &[key1, value1] : map1) {
    auto range2 = map2.equal_range(key1);
    auto range3 = map3.equal_range(key1);
    if (range2.first == range2.second || range3.first == range3.second) continue; //

    for (auto it2 = range2.first; it2 != range2.second; ++it2) {
        auto it3 = range3.first;
        while (it3 != range3.second) {
            // Unroll by 2 for it3 loop
            if (it3 != range3.second) {
                auto range4 = map4.equal_range(it3->second);
                auto it4 = range4.first;
                while (it4 != range4.second) {
                    // Unroll by 2 for it4 loop
                    if (it4 != range4.second) {
                        buffer << it3->second << "," << key1 << "," << value1 << ","
                               << it2->second << "," << it4->second << "\n";
                        ++it4;
                    }
                    if (it4 != range4.second) {
                        buffer << it3->second << "," << key1 << "," << value1 << ","
                               << it2->second << "," << it4->second << "\n";
                        ++it4;
                    }
                }
                ++it3;
            }
```

`Number of cycles:` 105 060 222 941

V7: Parallelize csv reading

- Based on myjoin6
- Use `async-threads` to parallelize IO
- Utilizing available resources effectively
- Reducing the backend overhead
- Only decreased the time not cycles

```
void my_join(const string &file1, const string &file2, const string &file3,
    // read files in parallel
    auto future_data1 = std::async(std::launch::async, read_file, file1);
    auto future_data2 = std::async(std::launch::async, read_file, file2);
    auto future_data3 = std::async(std::launch::async, read_file, file3);
    auto future_data4 = std::async(std::launch::async, read_file, file4);

    // wait for files to be read
    auto data1 = future_data1.get();
    auto data2 = future_data2.get();
    auto data3 = future_data3.get();
    auto data4 = future_data4.get();

    unordered_multimap<string, string> map1, map2, map3, map4;
```

`Number of cycles:` 113 064 693 638

V8: Initialize unordered_multimap w/ reserved space

- Based on myjoin6
- Removed creation of `map1` - map not necessary for file1
- Initialized `unordered_multimap` with reserved space

```
void my_join(const string &file1, const string &file2, const string &file3, c
    auto data1 = read_file(file1);
    auto data2 = read_file(file2);
    auto data3 = read_file(file3);
    auto data4 = read_file(file4);

    // Initialize unordered_multimap with reserved space to reduce rehashing
    unordered_multimap<string, string> map2(data2.begin(), data2.end());
    unordered_multimap<string, string> map3(data3.begin(), data3.end());
    unordered_multimap<string, string> map4(data4.begin(), data4.end());
```

`Number of cycles:` 90 858 019 300

V9: Parallelize join process

- Use `std::thread`
- Process different chunks of `data1` in parallel
- Merge the results into output buffer

- Cycles increased, time decreased

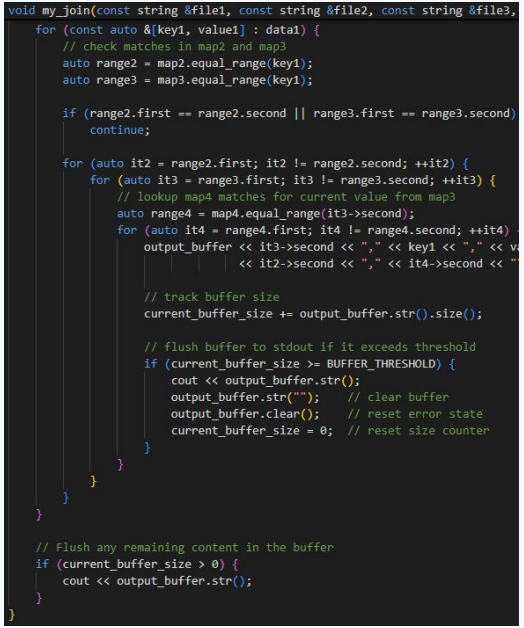
```cpp
void my_join(const string &file1, const string &file2, const string &file3, co
    auto data1 = read_file(file1);
    auto data2 = read_file(file2);
    auto data3 = read_file(file3);
    auto data4 = read_file(file4);

    // Initialize unordered_multimap with reserved space to reduce rehashing
    unordered_multimap<string, string> map2(data2.begin(), data2.end());
    unordered_multimap<string, string> map3(data3.begin(), data3.end());
    unordered_multimap<string, string> map4(data4.begin(), data4.end());

    // prepare multithreading
    size_t num_threads = thread::hardware_concurrency();
    if (num_threads == 0) num_threads = 4; // default 4 threads
    size_t chunk_size = data1.size() / num_threads;

    vector<thread> threads;
    vector<vector<pair<string, string>>> chunks;

    // split data1 into chunks
    for (size_t i = 0; i < num_threads; ++i) {
        auto start = data1.begin() + i * chunk_size;
        auto end = (i == num_threads - 1) ? data1.end() : start + chunk_size;
        chunks.emplace_back(start, end);
    }

    ostringstream output_buffer;
    mutex output_mutex;

    // launch threads
    for (size_t i = 0; i < num_threads; ++i) {
        threads.emplace_back(process_chunk, ref(chunks[i]), ref(map2), ref(map
                             ref(output_buffer), ref(output_mutex));
    }

    // wait for threads to complete
    for (auto &t : threads) {
        t.join();
    }

    // Flush the buffer to standard output
    cout << output_buffer.str();
}
```

`Number of cycles:` 99 157 079 419

V10: Better output handling

- Based on myjoin8
- Write output in batches - instead of storing results in memory
- Flush buffer to `std::cout` when threshold is reached
- Reduces memory usage

- Cycles increased, time unchanged



```
void my_join(const string &file1, const string &file2, const string &file3,
    for (const auto &[key1, value1] : data1) {
        // check matches in map2 and map3
        auto range2 = map2.equal_range(key1);
        auto range3 = map3.equal_range(key1);

        if (range2.first == range2.second || range3.first == range3.second)
            continue;

        for (auto it2 = range2.first; it2 != range2.second; ++it2) {
            for (auto it3 = range3.first; it3 != range3.second; ++it3) {
                // lookup map4 matches for current value from map3
                auto range4 = map4.equal_range(it3->second);
                for (auto it4 = range4.first; it4 != range4.second; ++it4)
                    output_buffer << it3->second << "," << key1 << "," << va
                                  << it2->second << "," << it4->second << "

                    // track buffer size
                    current_buffer_size += output_buffer.str().size();

                    // flush buffer to stdout if it exceeds threshold
                    if (current_buffer_size >= BUFFER_THRESHOLD) {
                        cout << output_buffer.str();
                        output_buffer.str("");    // clear buffer
                        output_buffer.clear();    // reset error state
                        current_buffer_size = 0;  // reset size counter
                    }
                }
            }
        }
    }

    // Flush any remaining content in the buffer
    if (current_buffer_size > 0) {
        cout << output_buffer.str();
    }
}
```

`Number of cycles:` 95 875 908 602

V11: In memory sorting

- Based on myjoin8
- Sort `data2`, `data3`, and `data4` by their keys
- Allows binary search instead of hash based lookups
- Reduces random memory access

- Cycles and time increased (a lot)

```cpp
void my_join(const string &file1, const string &file2, const string &file3, const
    auto data1 = read_file(file1);
    auto data2 = read_file(file2);
    auto data3 = read_file(file3);
    auto data4 = read_file(file4);

    // sort data2, data3, data4 by key
    sort(data2.begin(), data2.end(), [](const auto &a, const auto &b) {
        return a.first < b.first;
    });

    sort(data3.begin(), data3.end(), [](const auto &a, const auto &b) {
        return a.first < b.first;
    });

    sort(data4.begin(), data4.end(), [](const auto &a, const auto &b) {
        return a.first < b.first;
    });

    ostringstream output_buffer;

    for (const auto &[key1, value1] : data1) {
        // binary search for matches in data2 and data3
        auto matches2 = find_all_matches(data2, key1);
        auto matches3 = find_all_matches(data3, key1);

        if (matches2.empty() || matches3.empty())
            continue;

        for (const auto &[key2, value2] : matches2) {
            for (const auto &[key3, value3] : matches3) {
                // binary search for matches in data4
                auto matches4 = find_all_matches(data4, value3);

                for (const auto &[key4, value4] : matches4) {
                    output_buffer << value3 << "," << key1 << "," << value1 << ","
                                  << value2 << "," << value4 << "\n";
                }
            }
        }
    }

    // Flush the buffer to standard output
    cout << output_buffer.str();
}
```

`Number of cycles:` 280 943 563 629

V12: Minimize string operations

- Based on myjoin8
- Assign unique `int` IDs to strings (i.e.: “intern” strings)
- Lower memory overhead during joins

- Cycles and time increased

```cpp
// interning helper: maps strings to unique int IDs
class StringInterner {
private:
    unordered_map<string, int> string_to_id;
    vector<string> id_to_string;
    int next_id = 0;

public:
    // get or assign ID to string
    int intern(const string &s) {
        auto it = string_to_id.find(s);
        if (it != string_to_id.end()) {
            return it->second;
        }
        string_to_id[s] = next_id;
        id_to_string.push_back(s);
        return next_id++;
    }

    // get original string from ID
    const string &resolve(int id) const {
        return id_to_string[id];
    }
};
```

`Number of cycles:` 120 568 790 850

V13: Parallelize output writing

- Based on myjoin8
- Offload output writing to separate thread
- Decouples output writing (I/O) from join process

- Cycles and time increased

```
void my_join(const string &file1, const string &file2, const string &file3, const string &
    // start writer thread
    thread writer(writer_thread_func);

    for (const auto &[key1, value1] : data1) {
        auto range2 = map2.equal_range(key1);
        auto range3 = map3.equal_range(key1);

        if (range2.first == range2.second || range3.first == range3.second)
            continue;

        for (auto it2 = range2.first; it2 != range2.second; ++it2) {
            for (auto it3 = range3.first; it3 != range3.second; ++it3) {
                auto range4 = map4.equal_range(it3->second);
                for (auto it4 = range4.first; it4 != range4.second; ++it4) {
                    // enqueue the output line
                    {
                        lock_guard<mutex> lock(queue_mutex);
                        output_queue.push(it3->second + "," + key1 + "," + value1 + "," +
                                          it2->second + "," + it4->second + "\n");
                    }
                    cv.notify_one();
                }
            }
        }
    }

    // signal writer thread to finish
    {
        lock_guard<mutex> lock(queue_mutex);
        finished = true;
    }
    cv.notify_one();
    writer.join();
}
```

`Number of cycles:` 123 938 714 878

V14: Use memory-mapped files

- Based on myjoin8
- Used `mmap` for reading `.csv` files
- Reduces overhead of file I/O
- OS handles paging more efficiently

- Cycles and time remained the same

```
vector<pair<string, string>> read_file(const string &filename) {
    size_t start = 0;
    for (size_t i = 0; i < file_size; ++i) {
        if (file_in_memory[i] == '\n') {
            size_t line_length = i - start;
            string line(file_in_memory + start, line_length);
            size_t delim = line.find(',');
            if (delim != string::npos) {
                data.emplace_back(
                    line.substr(0, delim),
                    line.substr(delim + 1)
                );
            }
            start = i + 1;
        }
    }

    // handle last line (doesn't end with newline)
    if (start < file_size) {
        string line(file_in_memory + start, file_size - start);
        size_t delim = line.find(',');
        if (delim != string::npos) {
            data.emplace_back(
                line.substr(0, delim),
                line.substr(delim + 1)
            );
        }
    }

    munmap(file_in_memory, file_size);
    close(fd);
    return data;
}
```

`Number of cycles:` 92 337 135 105

V15: Further optimize data structure

- Based on myjoin8
- Replace `unordered_multimap` with `robin_hood::unordered_map`
- `robin_hood` hashing offer faster hash maps with reduced memory overhead

```
void my_join(const string &file1, const string &file2, const
    robin_hood::unordered_map<string, vector<string>> map2;
    robin_hood::unordered_map<string, vector<string>> map3;
    robin_hood::unordered_map<string, vector<string>> map4;

    map2.reserve(data2_size);
    map3.reserve(data3_size);
    map4.reserve(data4_size);

    // Populate maps
    for (const auto& [key, value] : data2) {
        map2[key].push_back(value);
    }
    for (const auto& [key, value] : data3) {
        map3[key].push_back(value);
    }
    for (const auto& [key, value] : data4) {
        map4[key].push_back(value);
    }
```

`Number of cycles:` 85 025 197 043

V16: String view

- Based on myjoin15
- Use `std::string_view` instead of `std::string` - to avoid unnecessary memory allocations
- Read the entire file into memory at once - instead of reading line by line - to reduce I/O operations

```
vector<pair<string_view, string_view>> read_file(const string &filename, string
    ifstream file(filename, ios::ate | ios::binary);
    if (!file) {
        cerr << "Error: Unable to open file " << filename << "\n";
        exit(1);
    }

    // Read the entire file into memory
    size_t file_size = file.tellg();
    file.seekg(0, ios::beg);
    file_content.resize(file_size);
    file.read(file_content.data(), file_size);

    // Parse the file content into key-value pairs
    vector<pair<string_view, string_view>> data;
    size_t estimated_lines = file_size / 50; // estimate
    data.reserve(estimated_lines);

    size_t start = 0;
    size_t end = 0;
    while ((end = file_content.find('\n', start)) != string::npos) {
        size_t delim = file_content.find(',', start);
        if (delim != string::npos && delim < end) {
            data.emplace_back(
                string_view(file_content.data() + start, delim - start),
                string_view(file_content.data() + delim + 1, end - delim - 1));
        }
        start = end + 1;
    }

    return data;
}
```

`Number of cycles:` 46 191 998 137

V17:

- Based on myjoin16
- Removed `ostringsstream`, it can be less efficient than string
- Output in chunks of 64KB

```
if (output_buffer.size() >= 65536) {
    cout << output_buffer;
    output_buffer.clear();
    output_buffer.reserve(65536);
}
```

`Number of cycles:` 40 973 671 158

V18: V17 with parallel file reading

- Parallel file reading with std::launch
- Decreased time but increased time number of iterations

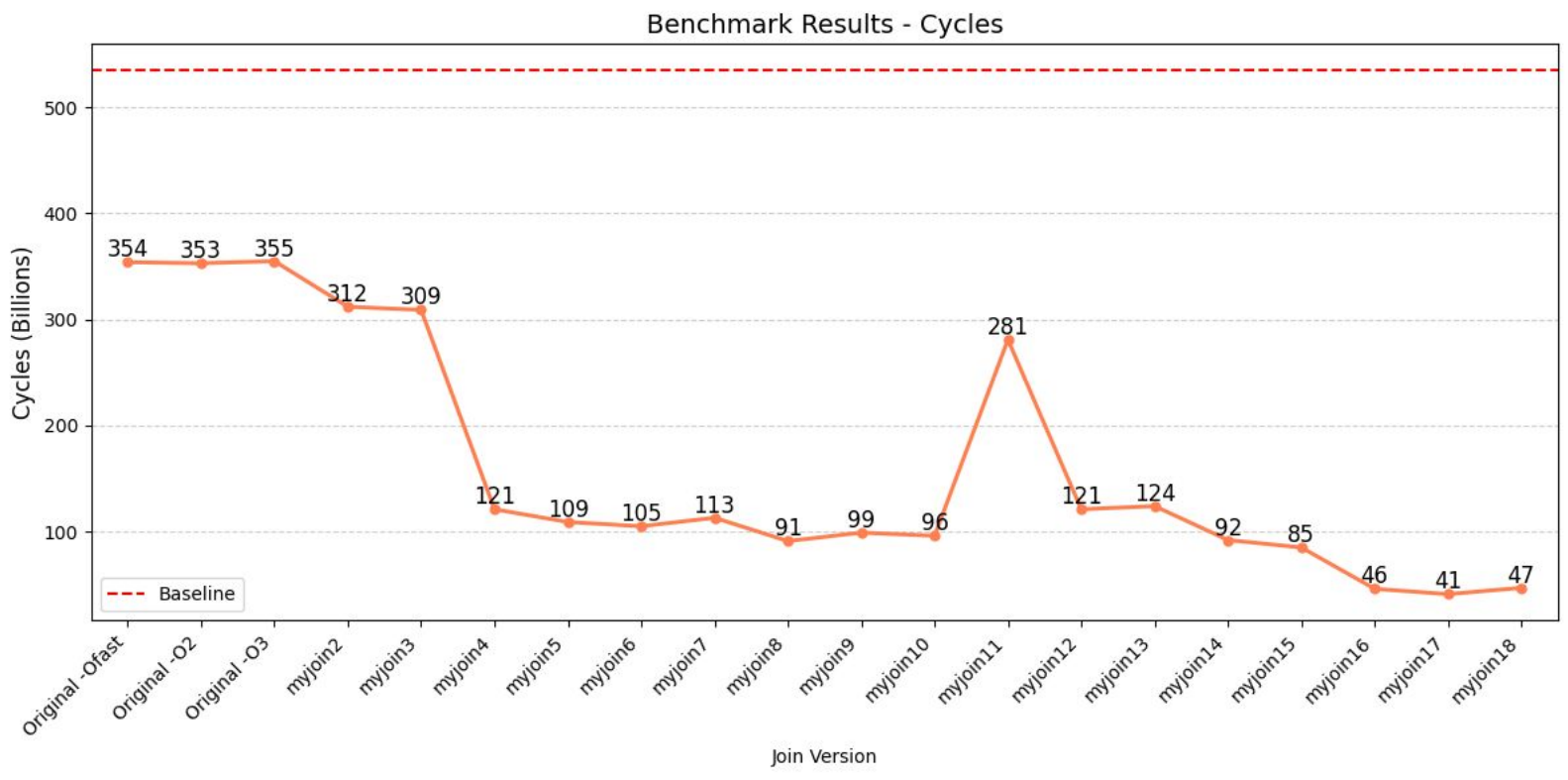
```
void my_join(const string& file1, const string& file2,
             const string& file3, const string& file4) {
    // Parallel file reading
    auto future1 = async(launch::async, read_file, file1);
    auto future2 = async(launch::async, read_file, file2);
    auto future3 = async(launch::async, read_file, file3);
    auto future4 = async(launch::async, read_file, file4);

    auto f1 = future1.get();
    auto f2 = future2.get();
    auto f3 = future3.get();
    auto f4 = future4.get();
```

Number of cycles: 47 498 081 175

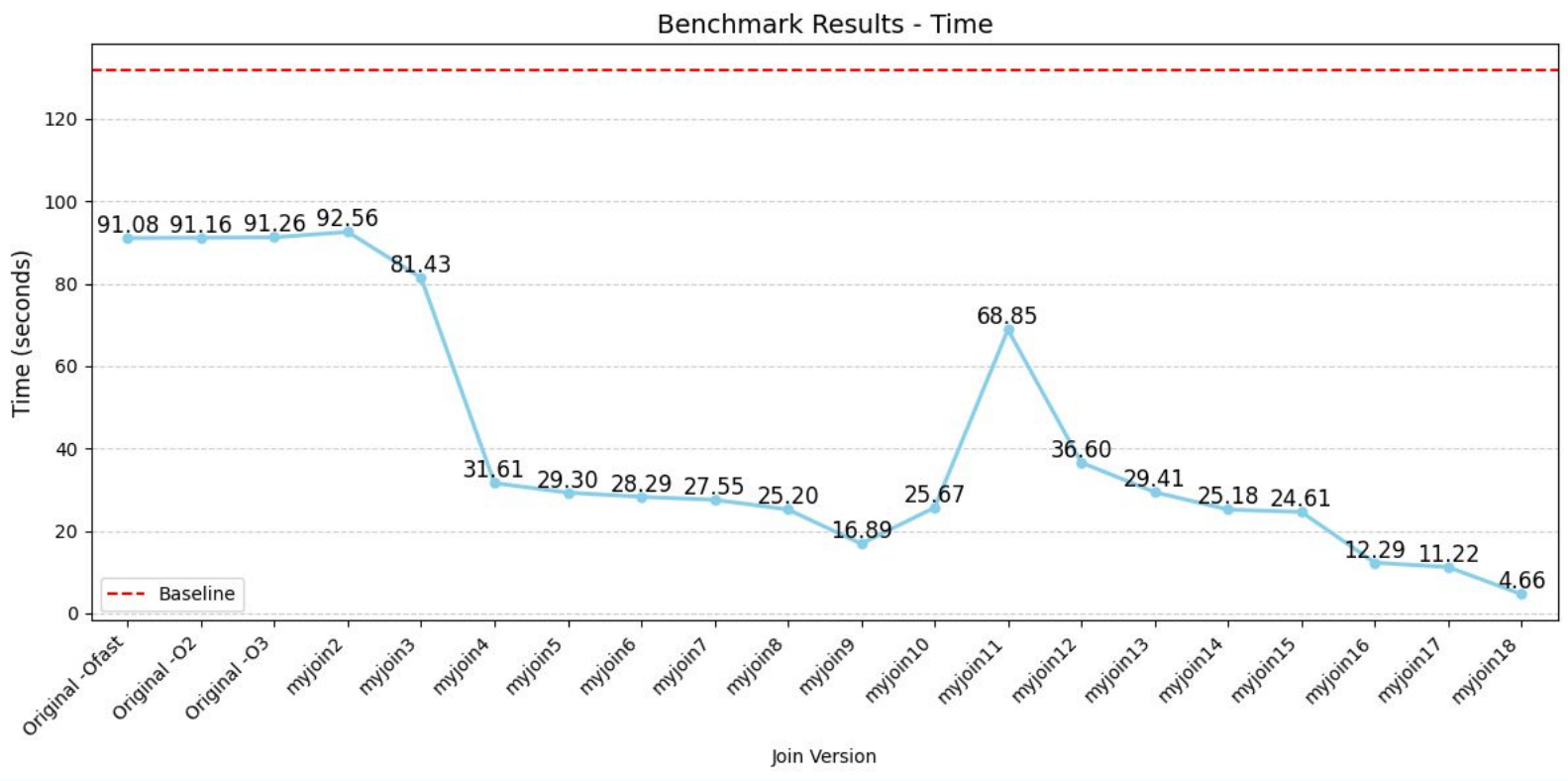
V18 vs. V17

	V17	V18 (3 CPUs)
Task-clock (s)	11.22	13.98
Time-elapsed (s)	11.22	4.65
Cycles (B)	40.97	47.48

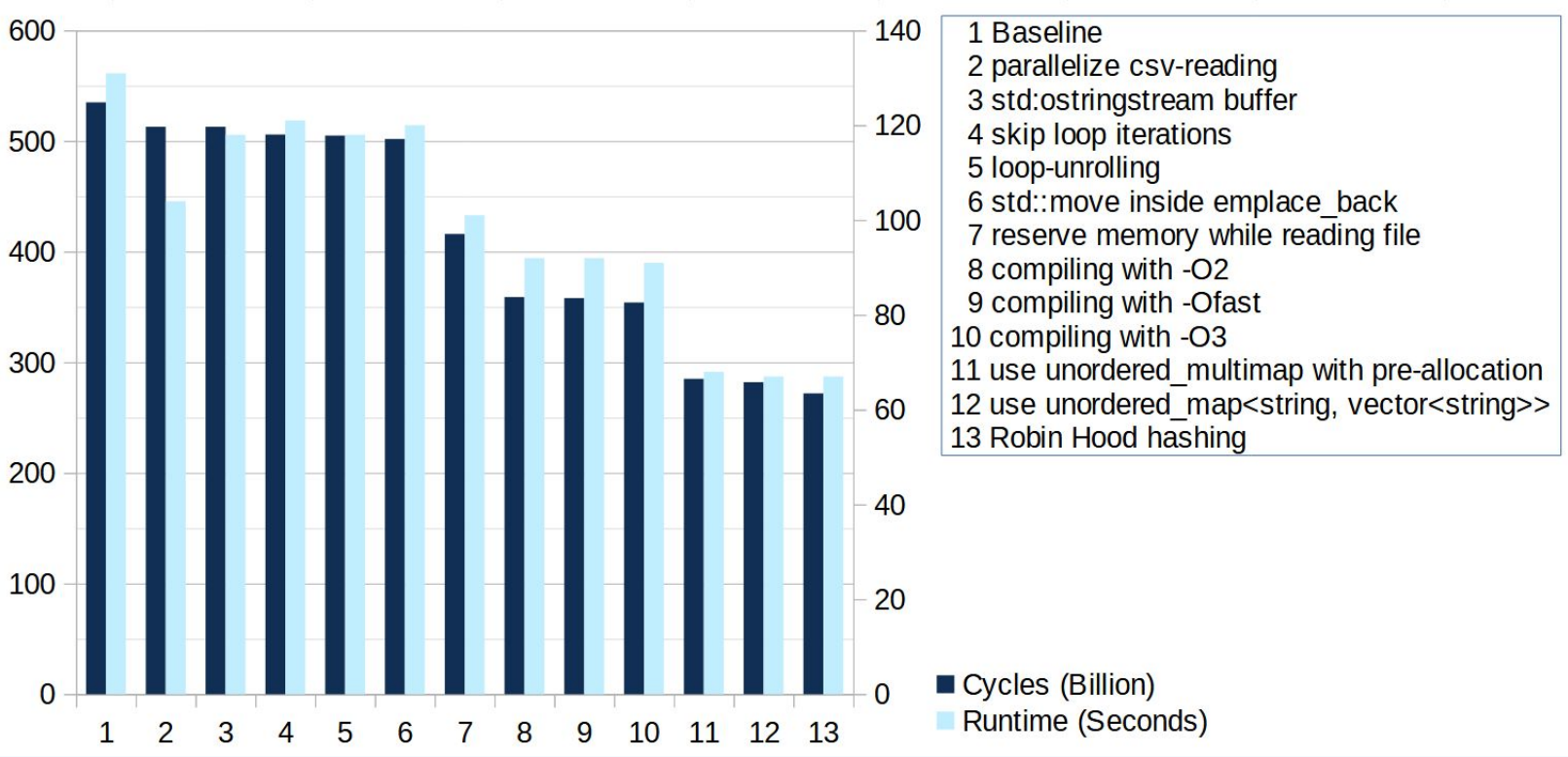
Results


Benchmark Results - Cycles
Cycles (Billions)
500
400
300
200
100
354
353
355
312
309
121
109
105
113
91
99
96
281
121
124
92
85
46
41
47
Baseline
Original -Ofast
Original -O2
Original -O3
myjoin2
myjoin3
myjoin4
myjoin5
myjoin6
myjoin7
myjoin8
myjoin9
myjoin10
myjoin11
myjoin12
myjoin13
myjoin14
myjoin15
myjoin16
myjoin17
myjoin18
Join Version

Results


Benchmark Results - Time
Time (seconds)
Join Version
Baseline
91.08
91.16
91.26
92.56
81.43
31.61
29.30
28.29
27.55
25.20
16.89
25.67
68.85
36.60
29.41
25.18
24.61
12.29
11.22
4.66
Original -Ofast
Original -O2
Original -O3
myjoin2
myjoin3
myjoin4
myjoin5
myjoin6
myjoin7
myjoin8
myjoin9
myjoin10
myjoin11
myjoin12
myjoin13
myjoin14
myjoin15
myjoin16
myjoin17
myjoin18

Single Optimizations



Thanks for your attention!

Questions?