OpenMP vs Rayon

Manoj Middepogu Yaswanth Kumar Orru Pranav Jangir

Problem Statement

- Compare two parallel programming paradigms OpenMP and Rayon across various parallel programming aspects.

Survey

- OpenMP
  - C, C++, Fortran
  - compiler directives, library routines
  - Fork Join Parallelism

- Rayon
  - Rust
  - Ownership, Borrowing Rules
  - Work Stealing

Features	OpenMp	Rust
Thread Configuration	M/A	M/A
Thread Scheduling	M/A	A
Thread Synchronisation	M/A	A
Race Condition Resolution	M/A	A
Nested Parallelism	✔	✔
Different tasks parallel Execution	✔	✔
Collapse	✔	✖
Type Checking	✖	✔
Nowait / taskwait	✔	✖

Experimental set-up

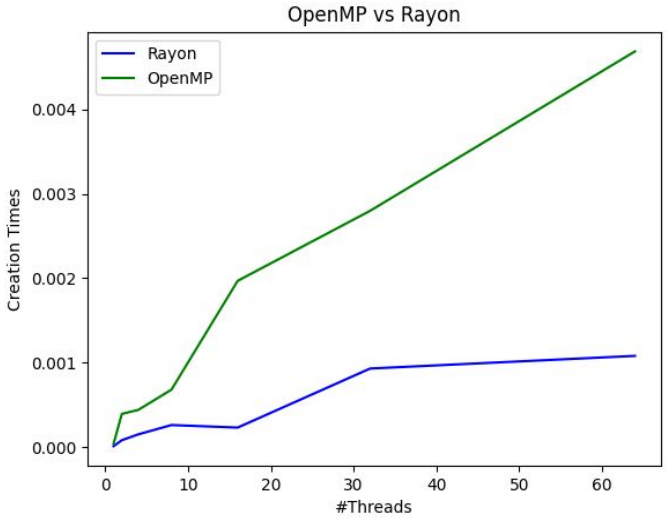
- Environment: crunchy1 | 256 GB | gcc = 11.2 | 64 cores

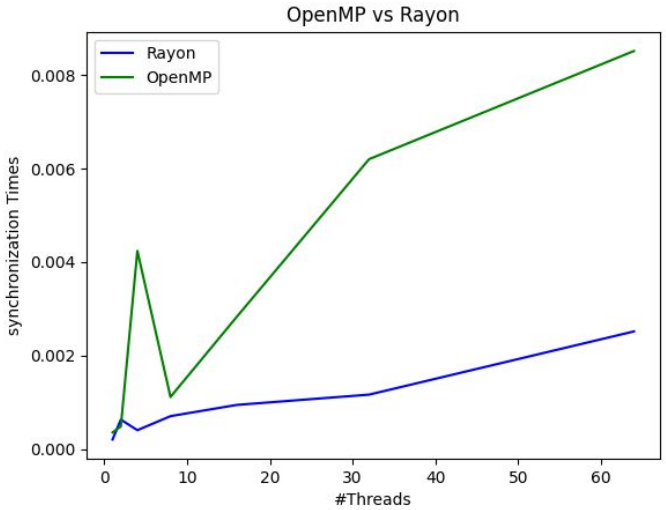
- Thread Creation and synchronisation times

- Matrix multiplication with OpenMP and Rayon
  - Full Parallel
  - Block-wise Parallel
  - Strassen's way
  - Linear Matrix Multiplication

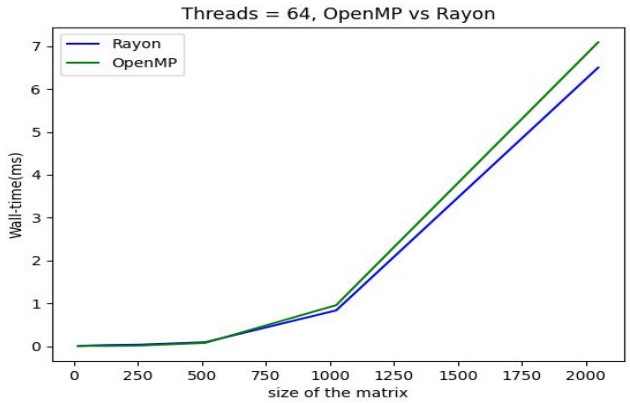
- Tracking
  - Wall-time
    - vs #Threads for fixed n*n size(n=1024)
    - vs n: size of the matrix at fixed threads(=64, =8)
  - Speedup, Efficiency
  - Page faults, Cache hits/misses

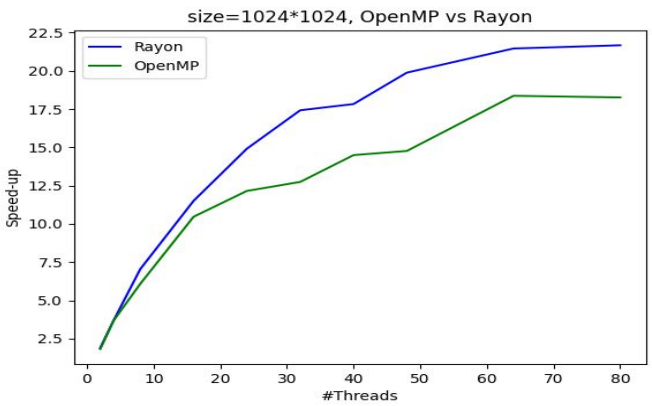
Analysis - RunTime Overhead

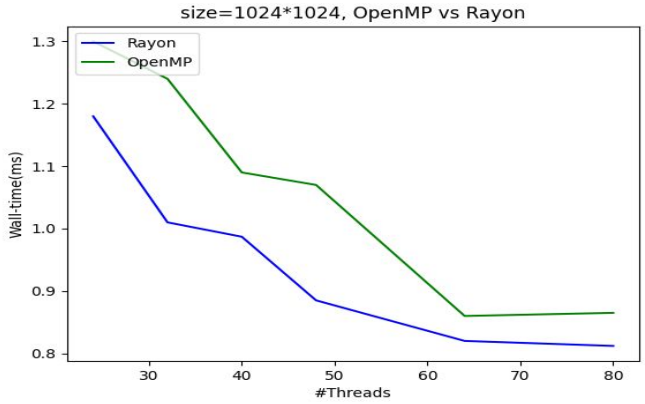


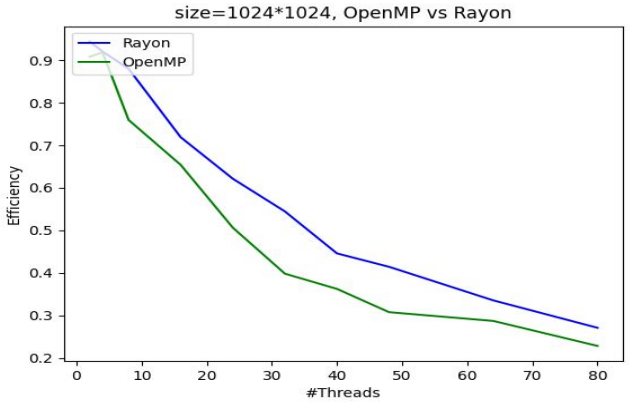


Analysis - Fully Parallel

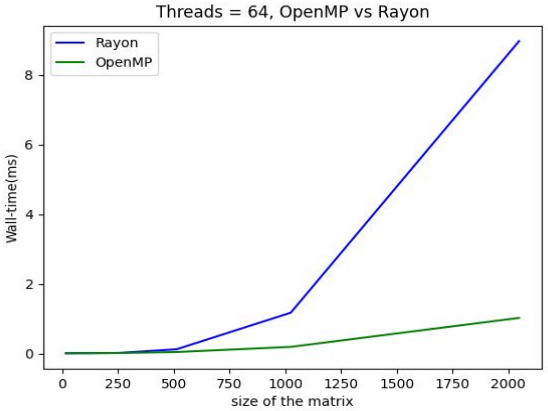


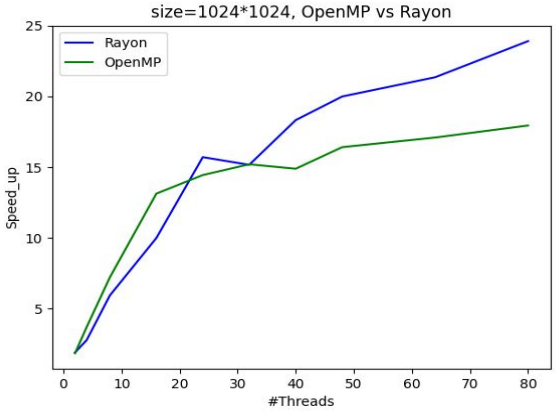


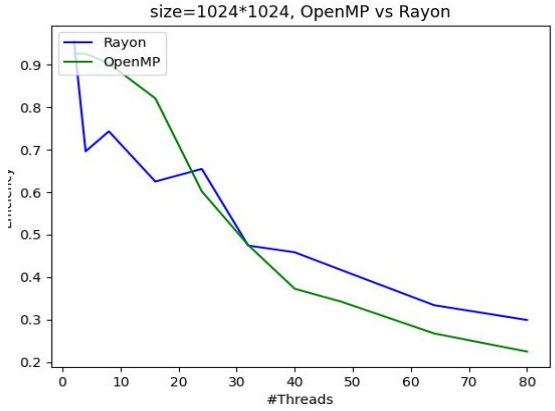


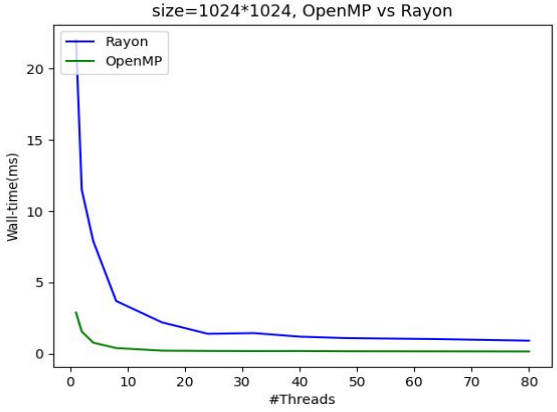


Analysis - Block Parallel









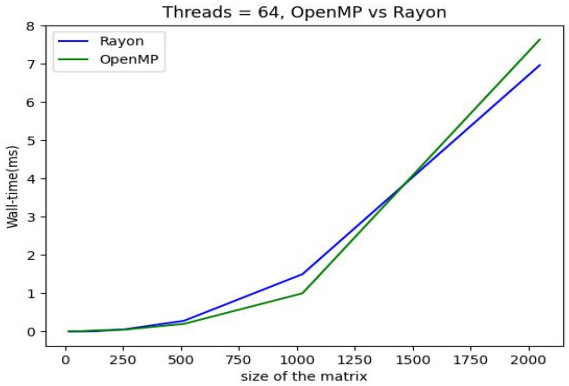
Rayon

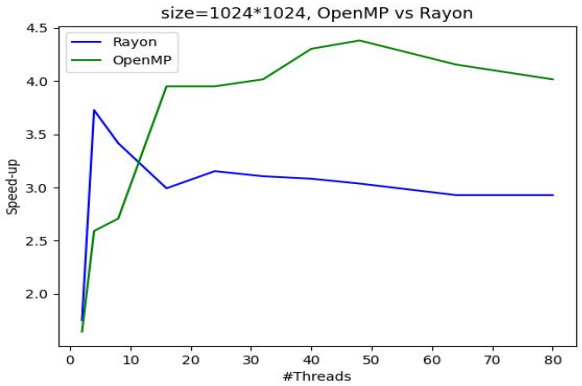
```
     6,670      page-faults
 4,982,688      cache-misses              #    2.240 % of all cache refs
222,419,491     cache-references
```

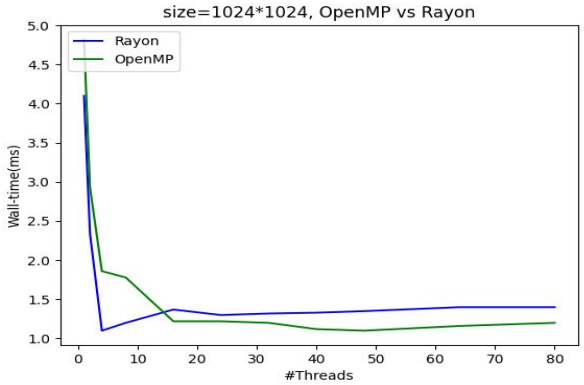
OpenMP

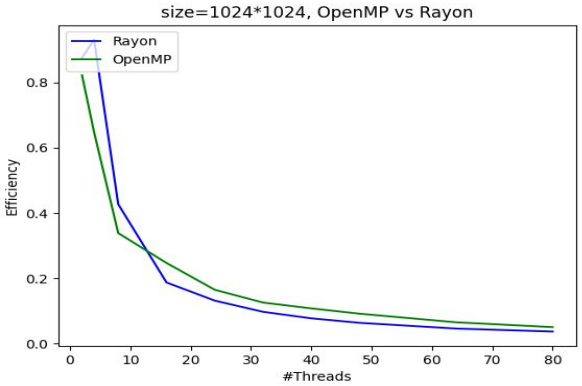
```
Performance counter stats for 'cargo run --release -- 1 1024 1':

        18,286      page-faults
     9,041,154      cache-misses              #    0.456 % of all cache refs
 1,982,333,557      cache-references
```

Analysis - Strassen's Algorithm









Analysis - Linear Way

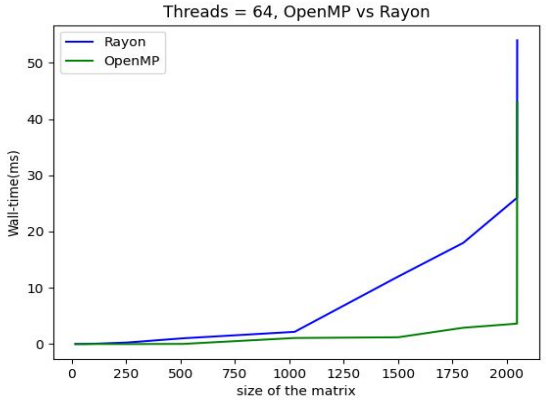
OpenMp

```
 Performance counter stats for './matrix_multiply':

          47,259      page-faults
     8,319,514,100      cache-misses              #   37.996 % of all cache refs
    21,895,618,240      cache-references
```

N = 2048

```
Performance counter stats for './matrix_multiply':

          22,243      page-faults
     525,573,690      cache-misses              #    3.736 % of all cache refs
  14,067,011,849      cache-references
```

N = 2047



Rayon

```
Performance counter stats for 'cargo run --release -- 64 2048 3':

          85,537      page-faults
     8,382,095,551      cache-misses              #   79.841 % of all cache refs
    10,498,439,290      cache-references
```

N = 2048

```
Performance counter stats for 'cargo run --release -- 64 2047 3':

          35,947      page-faults
     1,180,516,468      cache-misses              #    6.886 % of all cache refs
    17,144,218,283      cache-references
```

N = 2047

Takeaways

- Rust's high-level abstractions are so efficient that the overhead of thread communication is even lower than that of C++ code.
- Despite Rust's dynamic scheduling advantages, in certain cases, OpenMP code, optimized for cache utilization, proves lower run-time.
- The choice between OpenMp and Rayon depends on the specific characteristics of the problem at hand.