Parallel HITORI Solver

with MPI and OpenMP

About us

Simone Brentan

239959

Matteo Costalonga

239960

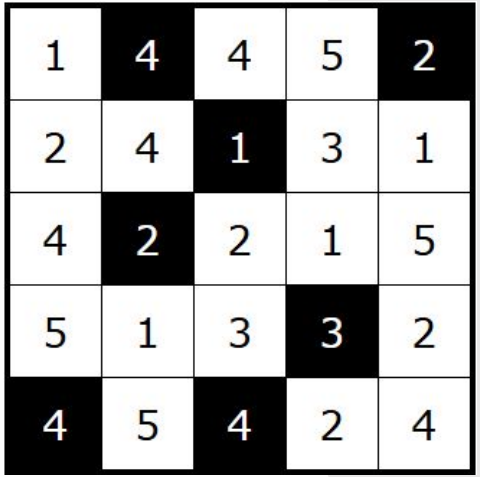
Alex Reichert

239961

HITORI

ひとりにしてくれ

From japanese: *"Hitori ni shite kure"*
Literally means: *"Leave me alone"*



Three approaches

MPI-based

Developed leveraging the *MPI* library for **multi-processing**

OpenMP-based

Developed leveraging the OpenMP library for **multi-threading**

Hybrid-based approach

Hybrid approach combining useful features from both ***MPI*** and ***OpenMP***

Algorithm analysis

Algorithm 1: Main function

```
1 board ← read_board(input_path);
2 pruned_solution ← cells_pruning(board);
3 spaces ← fragmentation(pruned, SOLUTION_SPACES);
4 solution ← solution_backtracking(spaces);
5 write_solution(solution)
```

Cells Pruning

To reduce and avoid useless computations

Solution Spaces Fragmentation

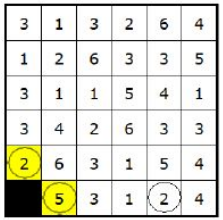
To normalize the computation time spent by the code

Solution Backtracking

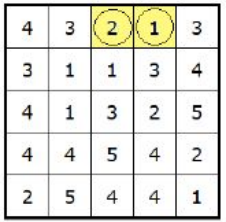
Recursively iterates the solution space to find the the solution

Pruning techniques

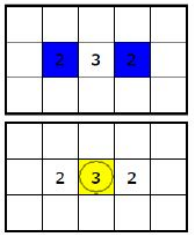
Must-paint



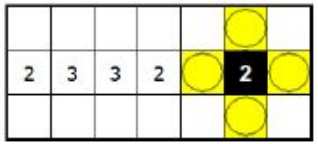
Uniqueness



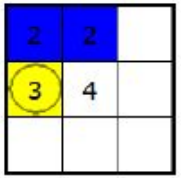
Sandwich



Isolation



Corner



Cell Values

X	O	?	O	X
O	O	O	X	O
O	X	O	O	O
O	O	O	O	X
X	O	X	O	O

4	3	2	4	2
1	4	3	3	2
4	1	5	1	3
3	2	4	5	2
1	5	1	3	1

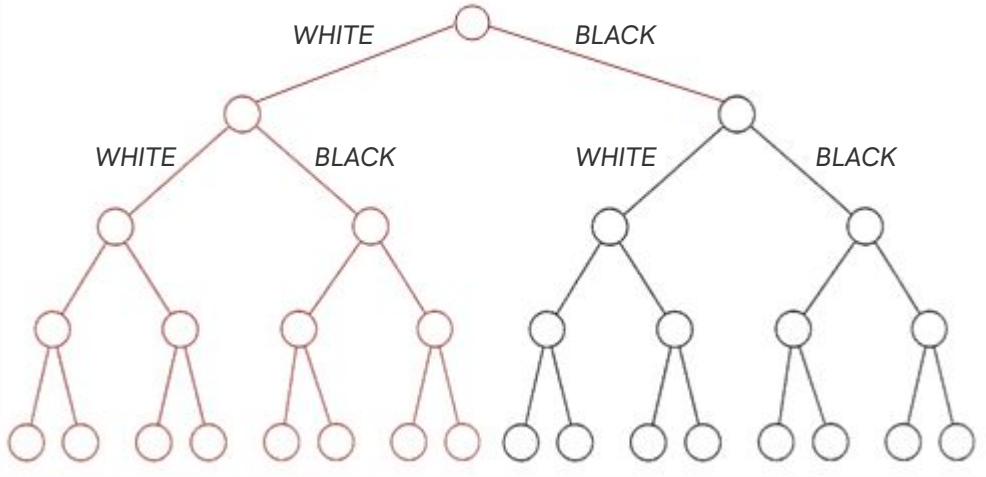
WHITE

BLACK

UNKNOWN

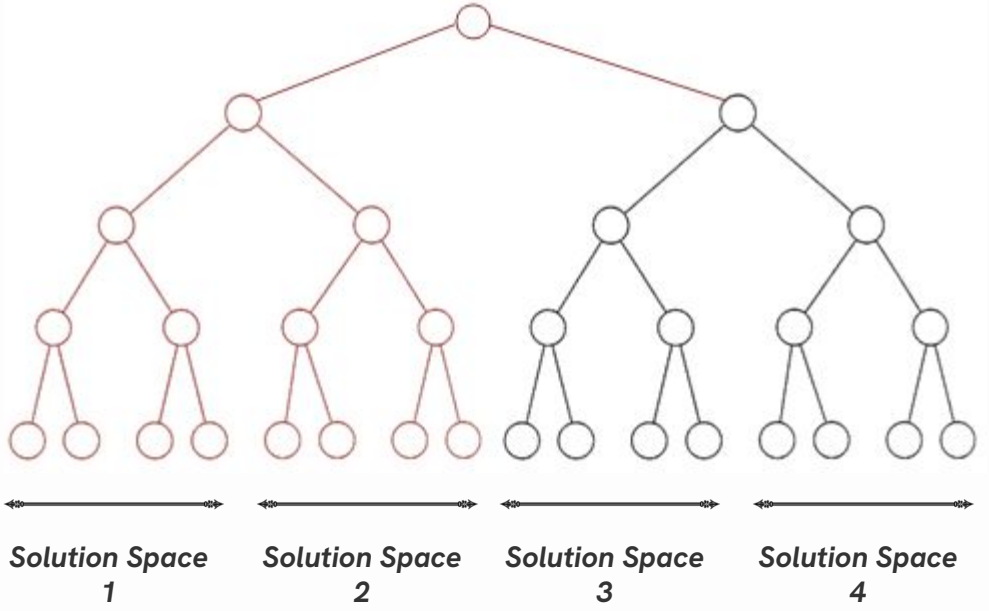
Solutions Space

Binary Tree



Solutions Space

Fragmentation



Solution Backtracking

BCB and Queue

Three core functions:

- Build Leaf
- Next Leaf
- Check Hitori

Board Control Block

```
typedef struct BCB {
  CellState *solution;
  bool *solution_space_unknowns;
}
```

Algorithm 6: solution_backtracking(queue)

```
1 while queue is not empty do
2     block ← dequeue(queue);
3     new_leaf ← next_leaf(block);
4     if check_hitori(new_leaf) then
5         return new_leaf
6     else
7         enqueue(queue, block);
8 return false
```

Solution Backtracking

Core Functions

Build Leaf

- **Input**: a *CellValue* matrix with some *UNKNOWN* cells
- **Task**: recursively filling unknown cells with BLACK or WHITE
- **Output**: ***True*** *if a leaf was correctly built or* ***False***
- **Top-down** exploration

Next Leaf

- **Input**: a *CellValue* matrix with all the cells either *BLACK* or *WHITE*
- **Task:** Get next possible solution leaf
- **Output**: a *CellValue* matrix which is a possible solution matrix
- **Bottom-up** exploration

Check Hitori

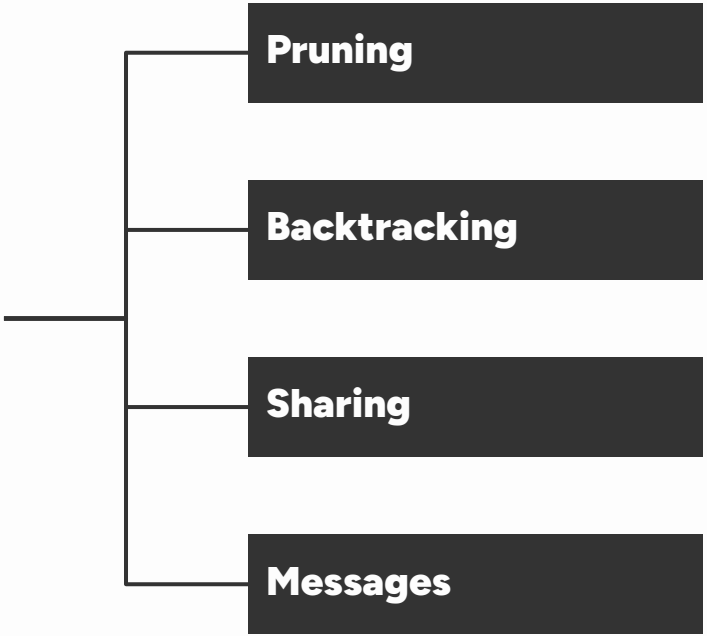
- **Input**: a CellValue matrix which is a possible solution
- **Output**: **True** if the matrix is the *HITORI* solution, **False** otherwise
- **DFS** exploration

1

MPI-based

Hitori solver using *multi-processing*

MPI Parallelization



Pruning

Splitting matrix in rows and columns and process them using different processes

Backtracking

A dynamic workload balancing system assigns one or more **Solution Spaces** to each process

Sharing

A **Solution Space Sharing** technique allows for processes to work together

Messages

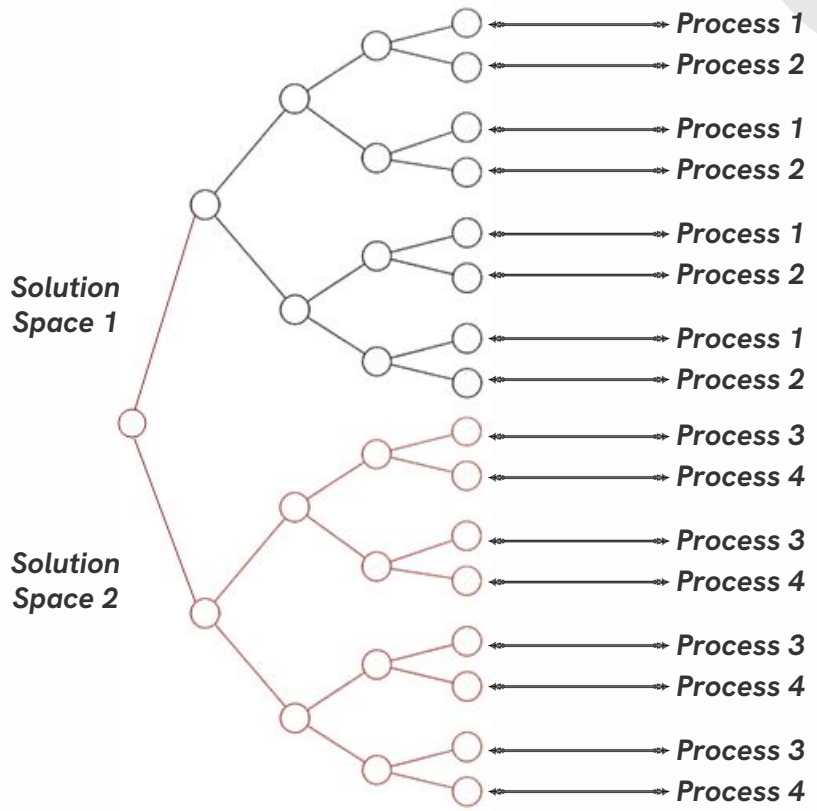
A **Star Topology** message passing architecture is employed

MPI Parallelization

Solution Space Sharing

When processes share a solution space:

- Introduction of decrementing ***solutions_to_skip*** variable in recursive loop
- **Check Hitori** is called when *solutions_to_skip=0*
- After calling **Check Hitori**, solutions_to_skip is restored to the number of sharing processes



MPI Parallelization

Message Passing Architecture

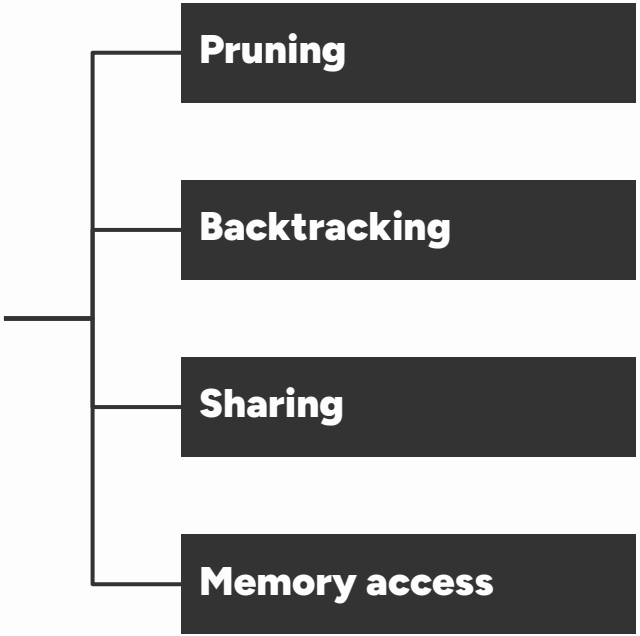
- **MASTER-WORKER** process relationship
- Different types of messages:
  - *TERMINATE*
  - *ASK_FOR_WORK*
  - *STATUS_UPDATE*
  - *...*
- *Different channels of communication:*
  - ***M2W_MESSAGE*** *(Master to worker)*
  - ***W2M_MESSAGE*** *(Worker to master)*
  - ***W2W_MESSAGE*** *(Worker to worker - dedicated)*

2

OpenMP-based

Hitori solver using *multi-threading*

OpenMP Parallelization



Pruning

Task-based pruning techniques are assigned to the threads

Backtracking

Each thread operates on its assigned solution spaces with a **local queue**

Sharing

A static workload balancing system assigns shared **Solution Spaces** to each thread

Memory access

Threads communicate using the shared memory

MPI Parallelization

Tasks

- An **asynchronous** piece of code that can be executed in a parallel region
- Spawned by a master thread, which then joins the workers
- Employed for both **pruning** and **backtracking**
- Reduce synchronization overhead and memory contention issues

OpenMP Parallelization

System to share solution spaces computation:

1. In place of *ASK_FOR_WORK* message of MPI
2. **Solution spaces** and *solutions_to_skip* are assigned to each thread since the beginning
3. **No communication** is required except the TERMINATION signal.
4. More stable during the beginning phase and reduced overhead

3

Hybrid-based

Hitori solver using both MPI and OpenMP

MPI vs OpenMP

Pros and Cons

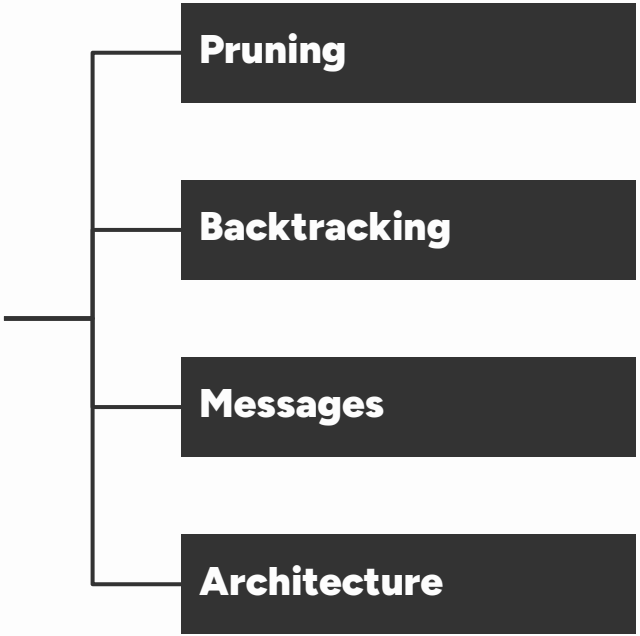
MPI

- **PROS**:
  - Enables **horizontal** scalability
  - Efficient **Star Topology** Architecture
- **CONS**:
  - May introduce **uncontrollable errors**
  - High code complexity
  - Too many messages cause additional overhead

OpenMP

- **PROS**:
  - Enables efficient **vertical** scalability
  - More stable workload balancing
- **CONS**:
  - Memory contention issues
  - **False Sharing**
  - Has a limited scalability

Hybrid Design Choices



Single process task-based pruning techniques similar to OpenMP

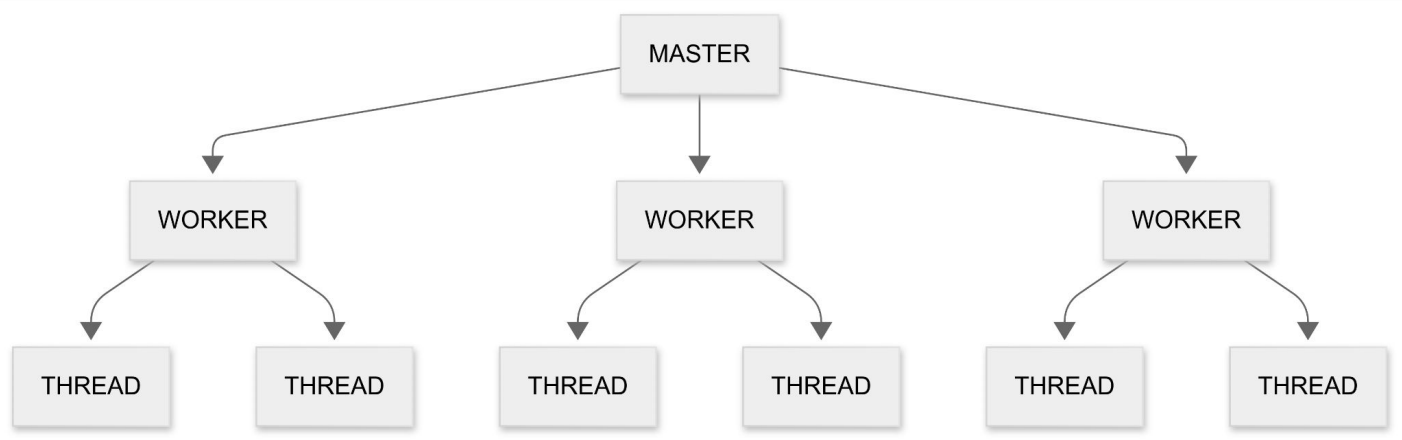
Combination of MPI solutions space sharing and OpenMP **workload balancing system**

Simplified message types: only *TERMINATION* signal is used

An **Extended Star Topology** of processes and threads is used

Message Passing Architecture

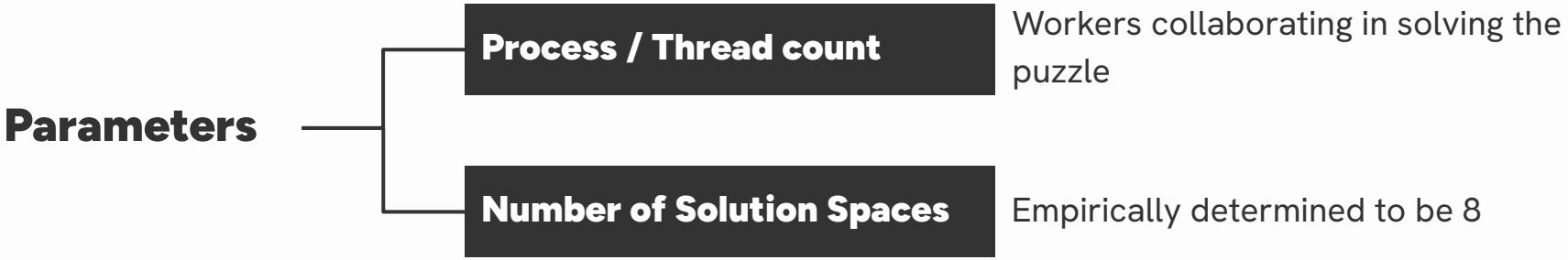
Extended Star Topology



Evaluation

Speedup, Efficiency and Scalability

Algorithm Parameters



Parameters setup

MPI

```
#!/bin/bash
#PBS -l select=$PROCESSES:ncpus=1:mem=4gb
#PBS -l place=scatter:excl
#PBS -l walltime=0:02:00
#PBS -q short_cpuQ
#PBS -o ./output/
#PBS -e ./output/

cd $PBS_O_WORKDIR
module load mpich-3.2
mpirun.actual -n $PROCESSES ./build/main.out $INPUT
```

Parameters setup

OpenMP

```
#!/bin/bash
#PBS -l select=1:ncpus=$THREADS:mem=4gb
#PBS -l place=pack:excl
#PBS -l walltime=0:02:00
#PBS -q short_cpuQ
#PBS -o ./output/
#PBS -e ./output/

cd $PBS_O_WORKDIR
export OMP_NUM_THREADS = $THREADS
./build/main.out $INPUT
```

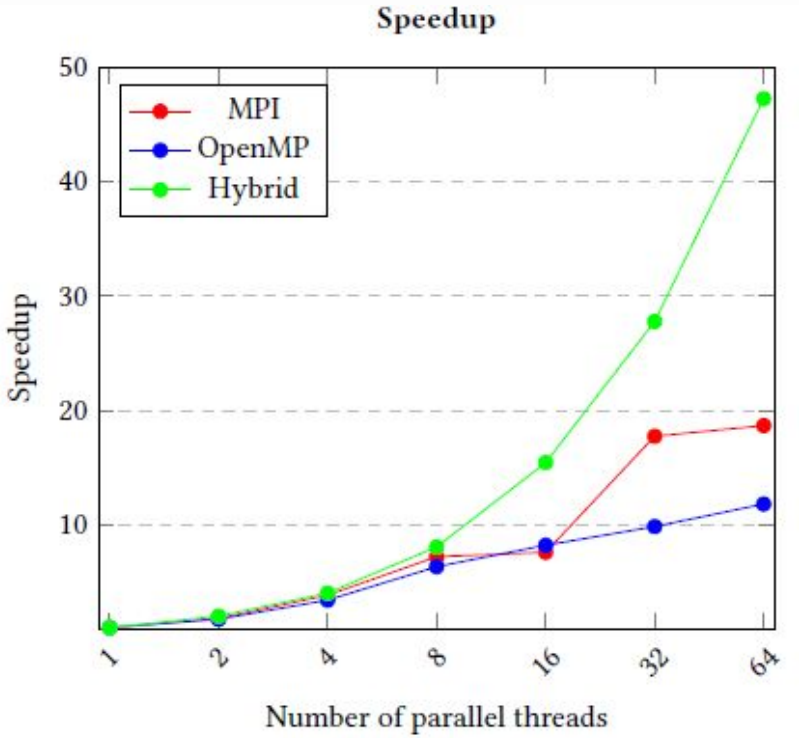
Parameters setup

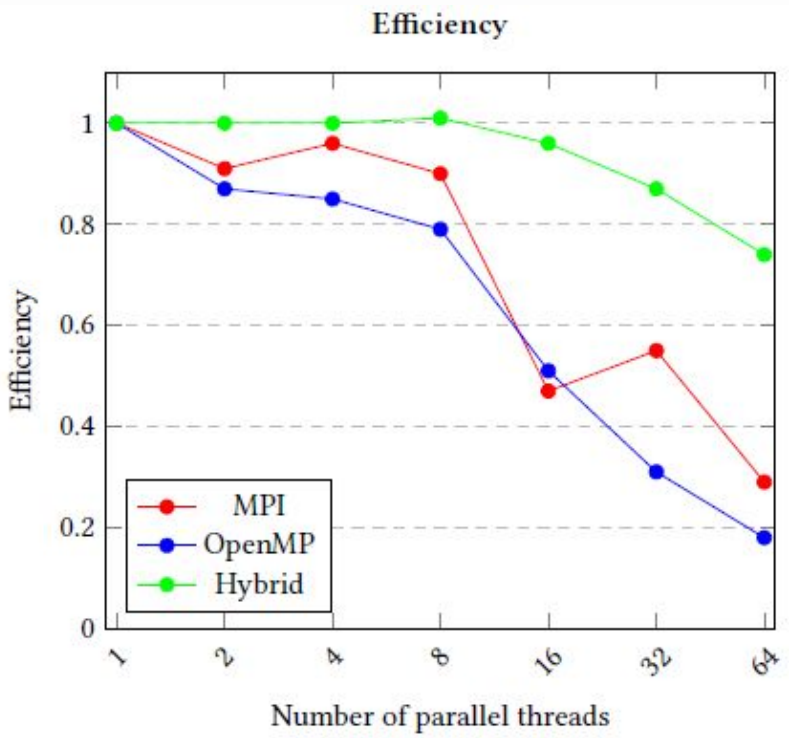
Hybrid

```
#!/bin/bash
#PBS -l select=$PROCESSES:ncpus=$THREADS:mem=4gb
#PBS -l place=scatter:excl
#PBS -l walltime=00:05:00
#PBS -q short_cpuQ
#PBS -o ./output/
#PBS -e ./output/

cd $PBS_O_WORKDIR
module load mpich-3.2
export OMP_NUM_THREADS = $THREADS
mpirun.actual -n $PROCESSES ./build/main.out $INPUT
```

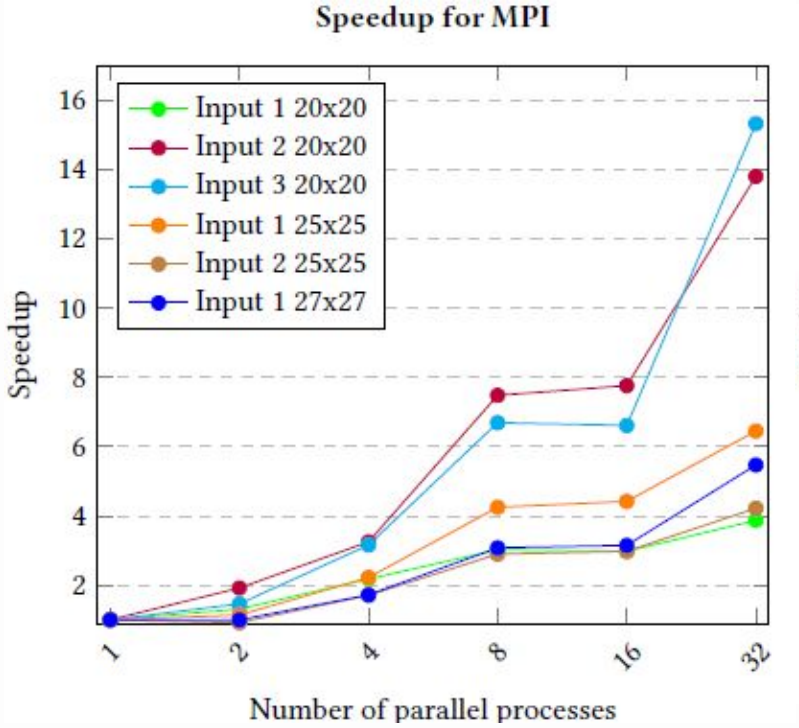
Experimental Evaluation

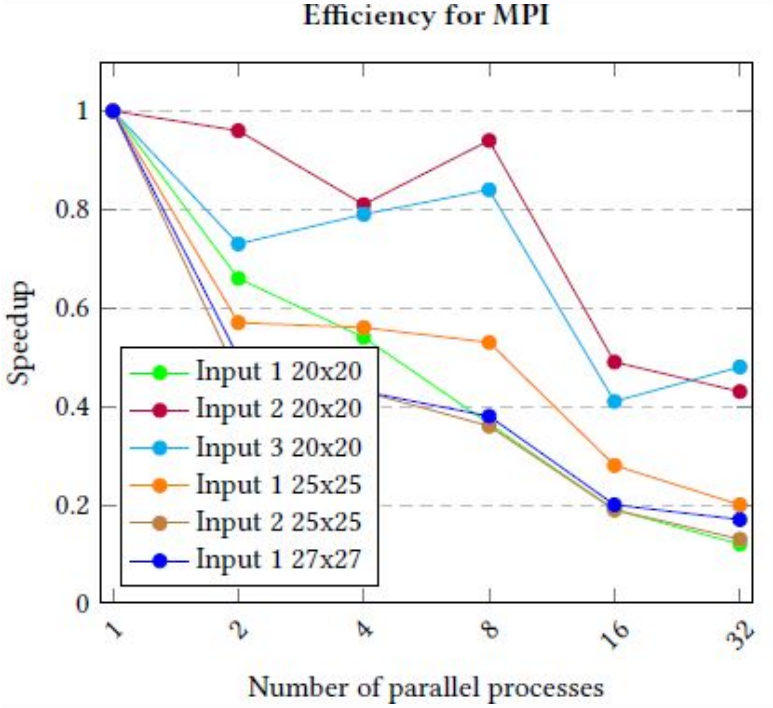




Performance across approaches

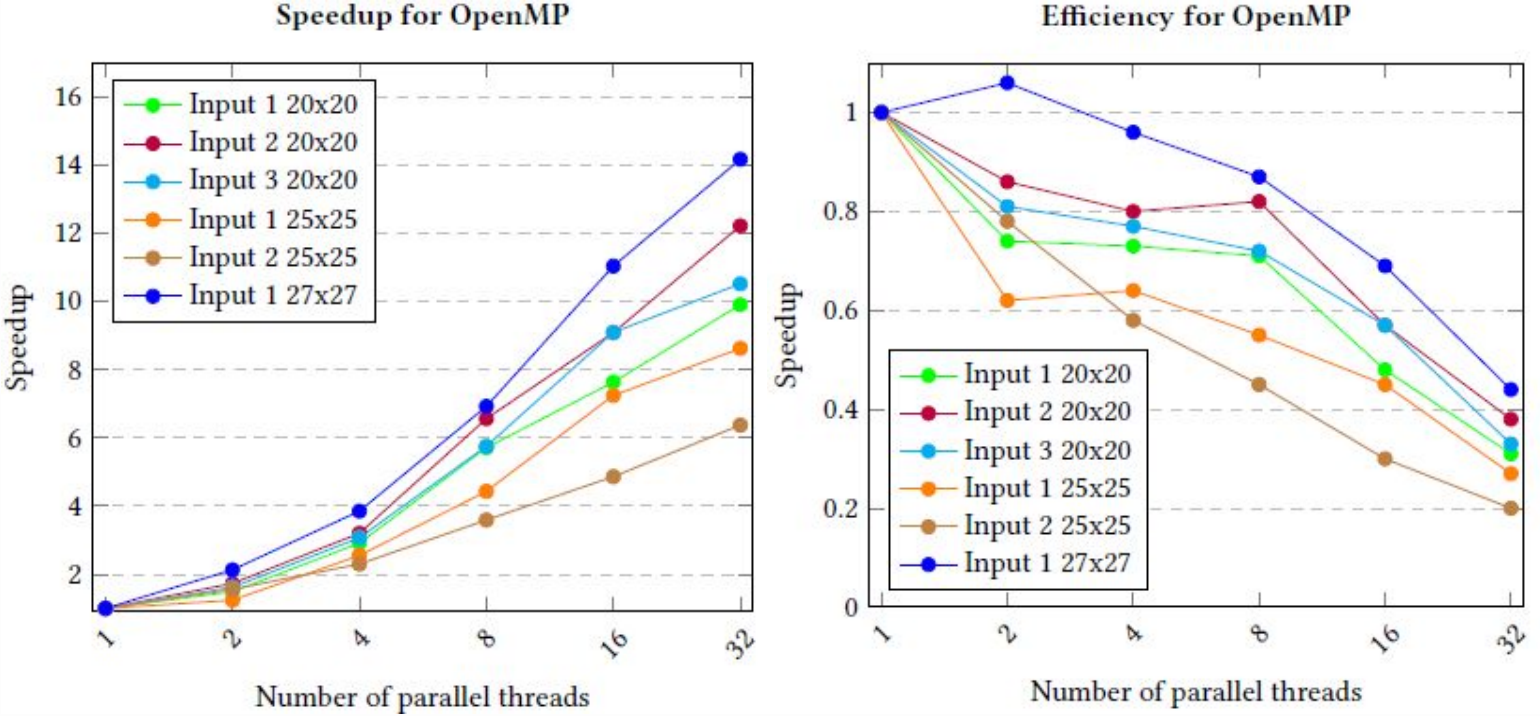
MPI





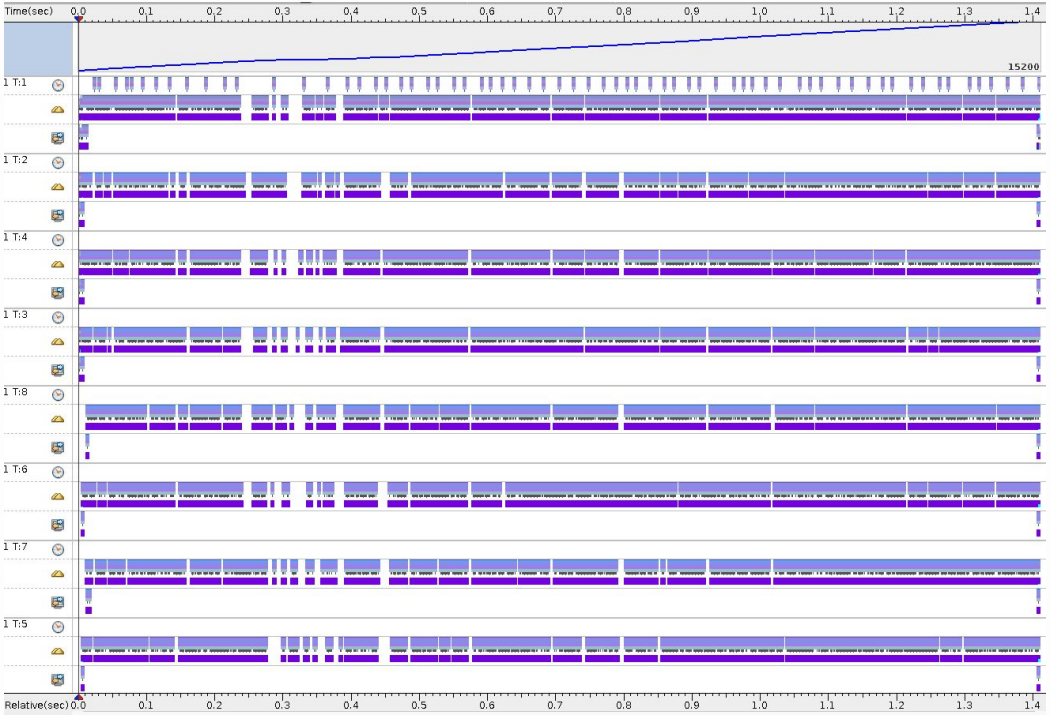
Performance across approaches

OpenMP



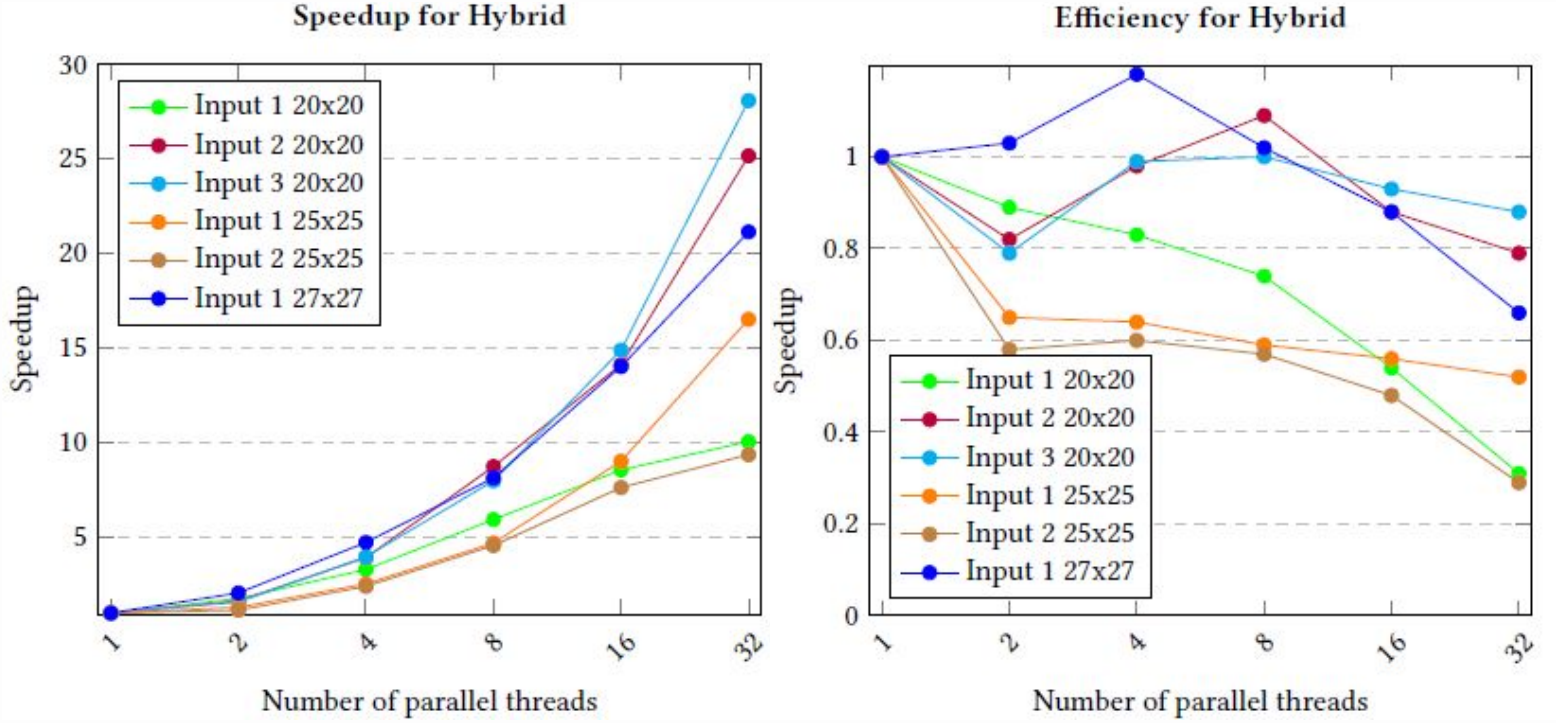
Performance across approaches

OpenMP

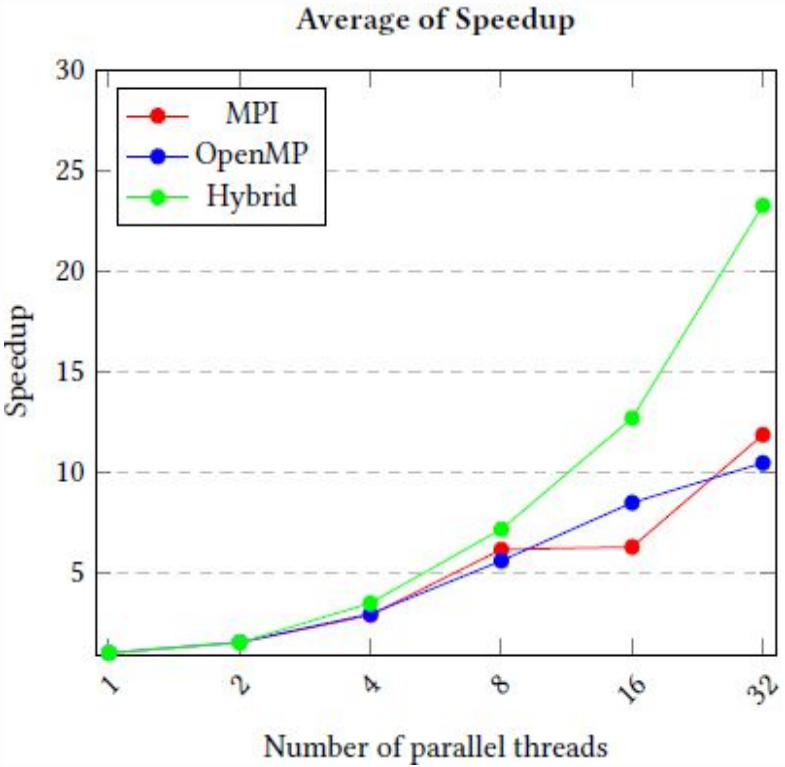


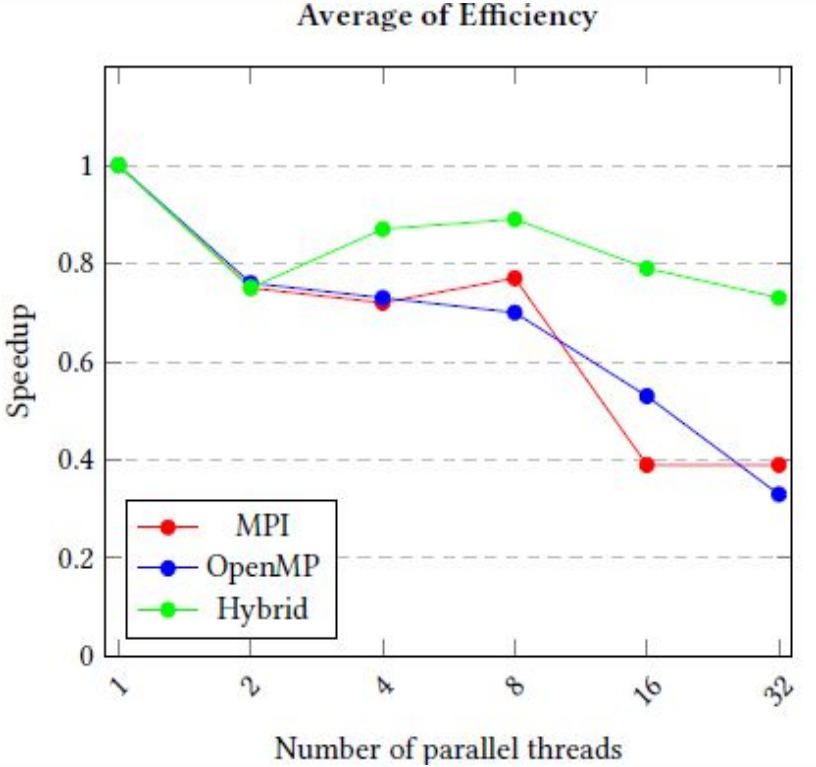
Performance across approaches

Hybrid

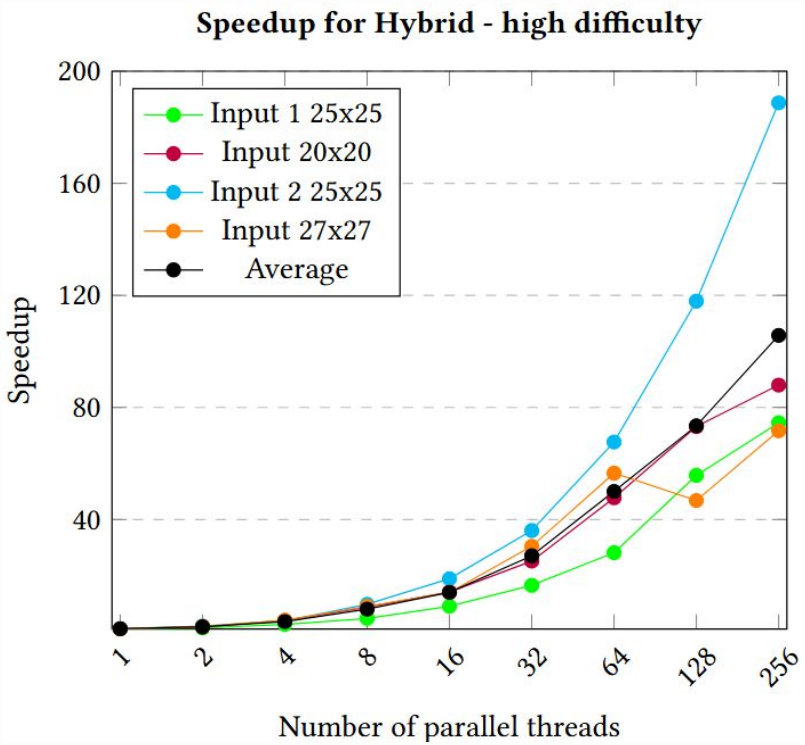


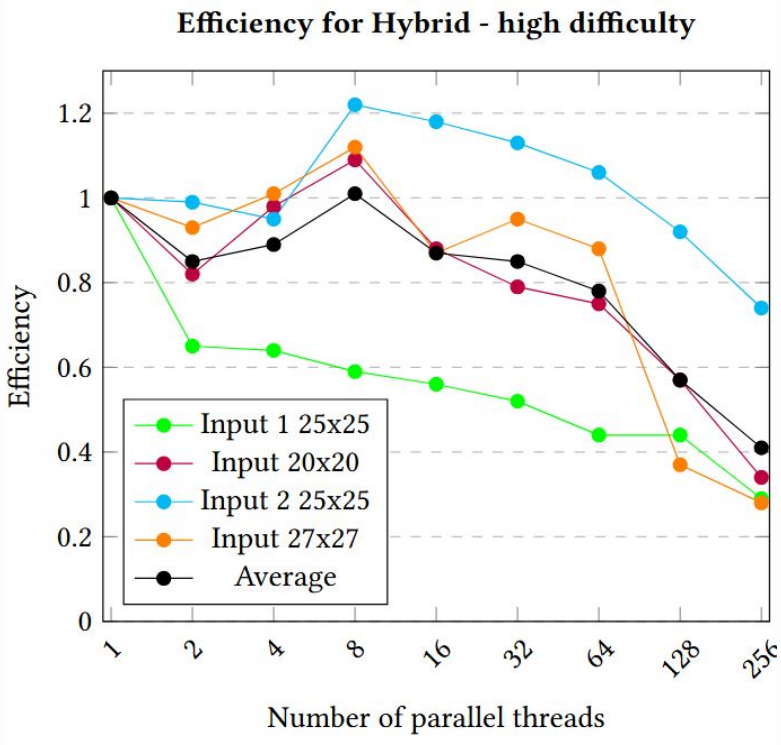
Results for non-trivial





Limit testing Hybrid approach





Thanks!