国防科技大学
National University of Defense Technology

# Exploring ILP for VLIW architecture by Quantified Modeling and Dynamic Programming-based Instruction Scheduling

Author: Can Deng, Zhaoyun Chen, Yang Shi, Xichang Kong and Mei Wen*

Reporter: Can Deng

# Outline

- PART 1: **Introduction**
- PART 2: **Method**
- PART 3: **Results**
- PART 4: **Conclusion**

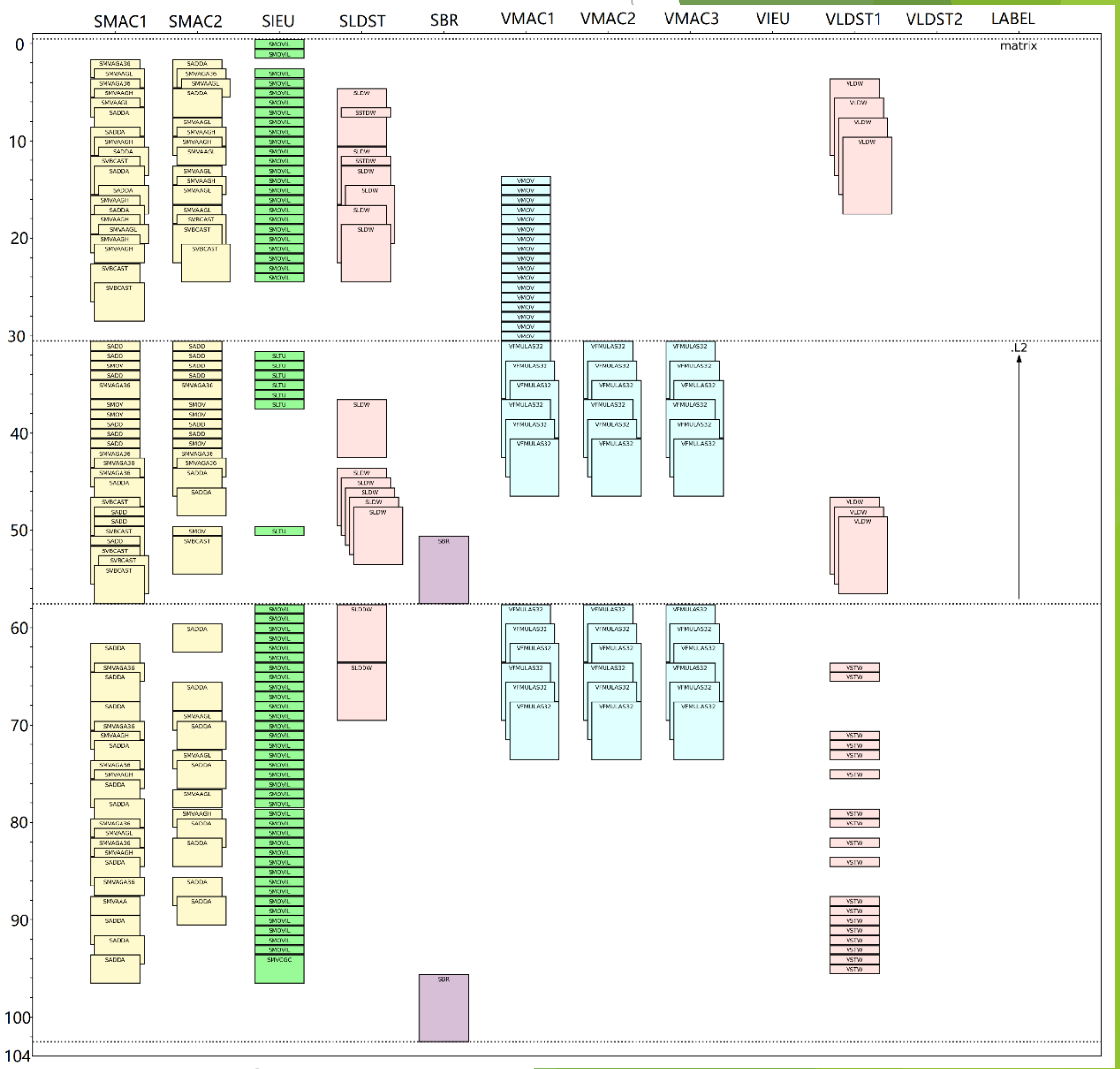
# PART 1

## Introduction

# 1.1 Background

- **VLIW** architecture is widely adopted in dedicated processors
- The performance of VLIW processors is getting higher and higher

LS solution



# 1.1 Background

- **Disadvantage:**
  - LS algorithms make a decision from the feasible solutions in a local view
  - The efficiency of the final solution is unpredictable

| | LS (list scheduling) | DP (dynamic programming) |
|---|---|---|
| Searching space | Loccal view | Global view |
| Goal | Feasible solution | Optimal solution |
| Time overhead | Low | low |
| Space Complexity | Low | High |

# 1.2 Motivation

- Propose a dynamic programming based strategy (DPS) to make a trade-off
- Achieve a high efficiency scheduling solution within acceptable time overhead
- Construct a quantifiable model for the instruction scheduling problem and get a theoretical upper bound of efficiency

# PART 2

# Method

# 2.1 ● Objective:

$$min(T) \ = \ min(max(\sum_{q=0}^{T_0-1}(1-\sum_{t=0}^{q} X_{i,t}^{f}) + C_i))$$

## ●Constraints:

$$\sum_{f=0}^{m-1}\sum_{t=0}^{T_0-1} X_{i,t}^{f} = 1\ (\forall I_i \in \boldsymbol{I}) \qquad (1)$$

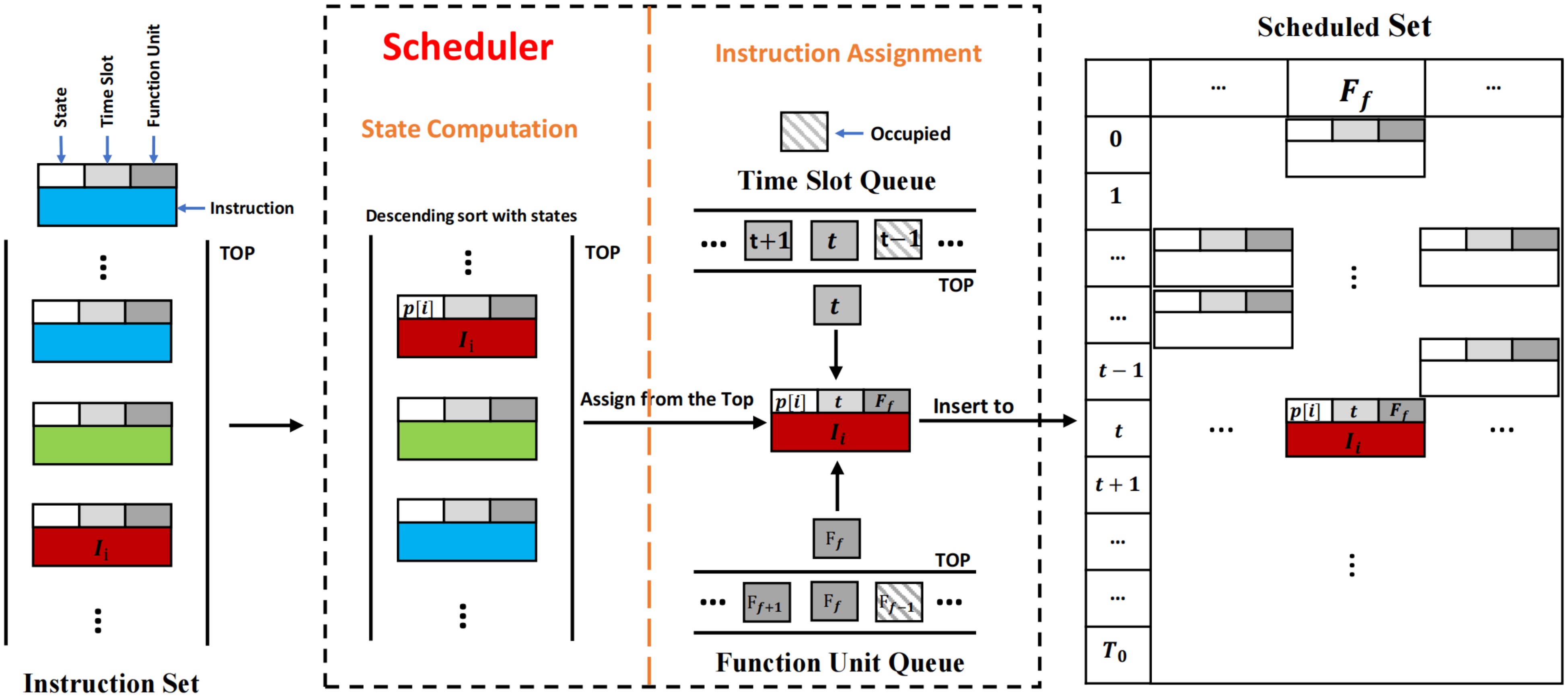
$$\sum_{f=0}^{m-1}\sum_{t=0}^{T_0-1} Y_{i,f} \cdot X_{i,t}^{f} = 1\ (\forall I_i \in \boldsymbol{I}) \qquad (2)$$

$$\sum_{i=0}^{n-1} X_{i,t}^{f} \le 1(\forall t \in \boldsymbol{T}, \forall F_f \in \boldsymbol{F}) \qquad (3)$$

$$S_i + Edge_{i,l} \le S_l\ (I_i, I_l \in \boldsymbol{I}) \qquad (4)$$

| | |
|---|---|
| $\boldsymbol{I} = \{I_0 \ldots I_i \ldots I_{(n-1)}\}$ | Sequence of Instruction |
| $\boldsymbol{F} = \{F_0 \ldots F_f \ldots F_{(m-1)}\}$ | Sequence of function unit |
| $\boldsymbol{T} = \{0 \ldots t \ldots (T_0 -)\}$ | Sequence of cycle |
| $X_{i,t}^{f}$ | Binary variable {0,1} |
| $Y_{i,f}$ | Binary variable {0,1} |

# 2.2 Dynamic Programming-Based Strategy(DPS)



# 2.2.1 State Computation

**Algorithm 1:** State Computation

**Input:** Instructions
**Output:** States

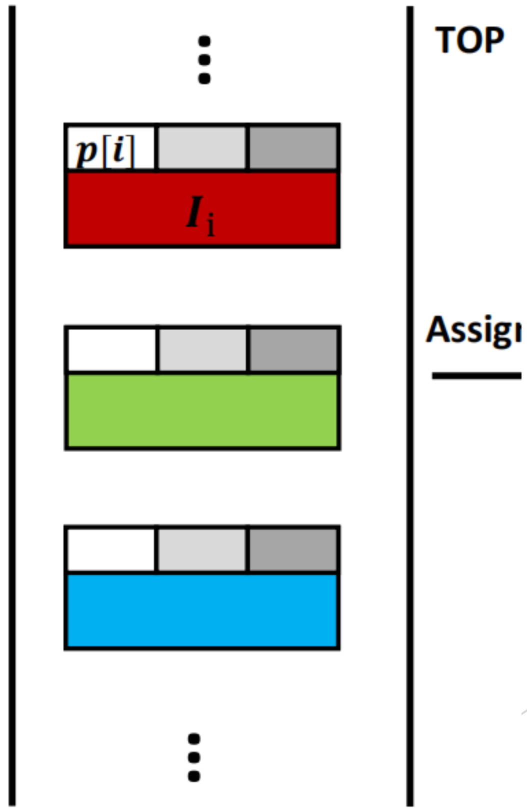
**for** $i \leftarrow 0$ *to* $(n-1)$ **do**
  //$n$ is the number of instructions;
  **if** $chl = 0$ **then**
    //$chl$ is the number of children;
    $p[i] \leftarrow C_i$
  **else**
    **for** $j \leftarrow 0$ *to* $(chl-1)$ **do**
      $p[i] \leftarrow \max(p[j] + Edge_{i,j},\ C_i)$;

$$p[i] = \max(p[j] + Edge_{i,j}, C_i)$$

## State Computation

Highest State First

**Descending sort with states**

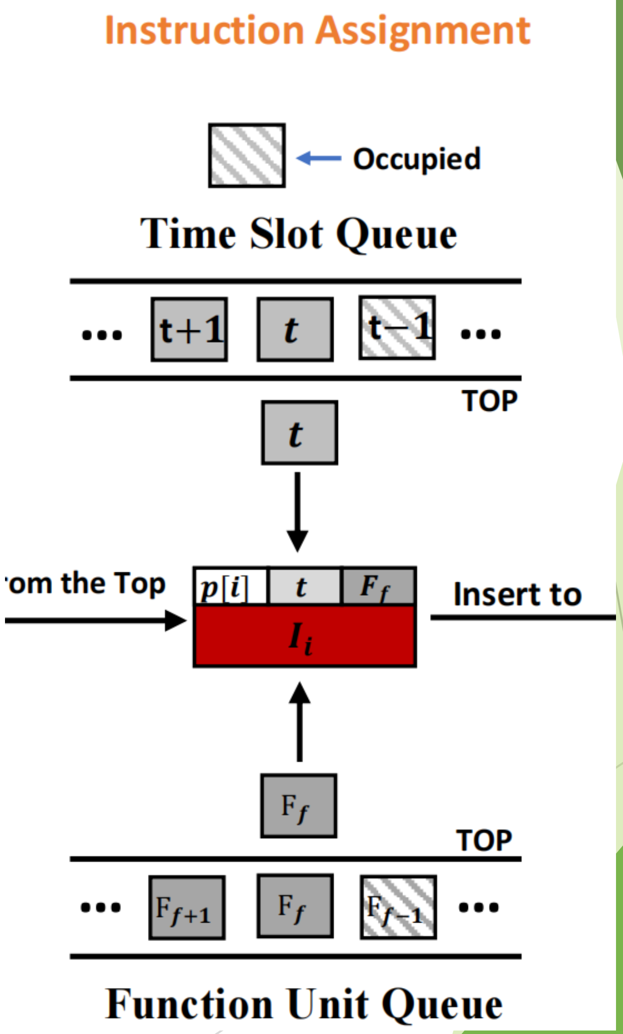


# 2.2.2 Instruction Assignment

**Algorithm 2:** Pesudocode for DPS Method

**Input:** A block
**Output:** A scheduled block

Call State Computation //Call for Algorithm 1;
**for** $i \leftarrow 0$ *to* $(n-1)$ **do**
    Pop the highest state instruction $I_i$;
    **for** $t \leftarrow ES_i$ *to* $T_0 - 1$ **do**
        **for** $k \leftarrow 0$ *to* $(m-1)$ **do**
            **if** *function unit available* **then**
                Allocate function unit $F_f$ and time slot $t$ for instruction $I_i$;
            **else**
                Shift to the next time slot;
    Update $ES$ values for all the children;



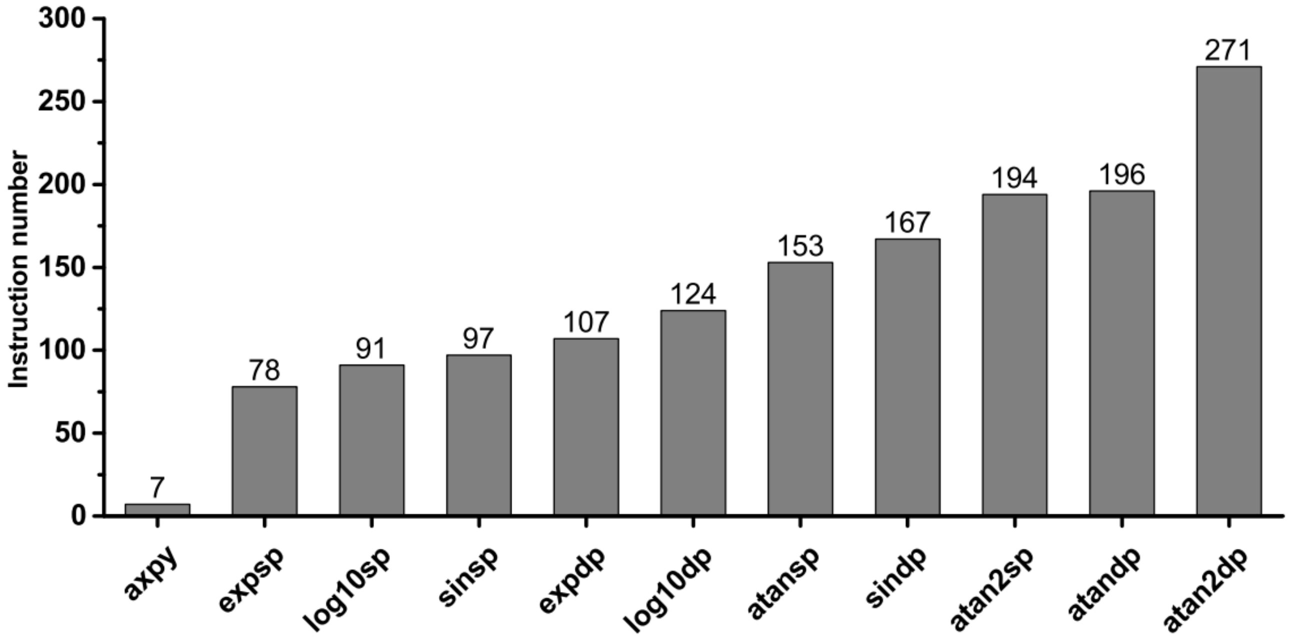
## 2.3 Experiments

- **Platform**: FT-Matrix DSP
- **Benchmark**: Transcendental Functions
- **BaseLines**: Heterogeneous Earliest Finish Time (HEFT), Critical-Path-Node-Dominant (CPND), and Longest Job First (LJF)

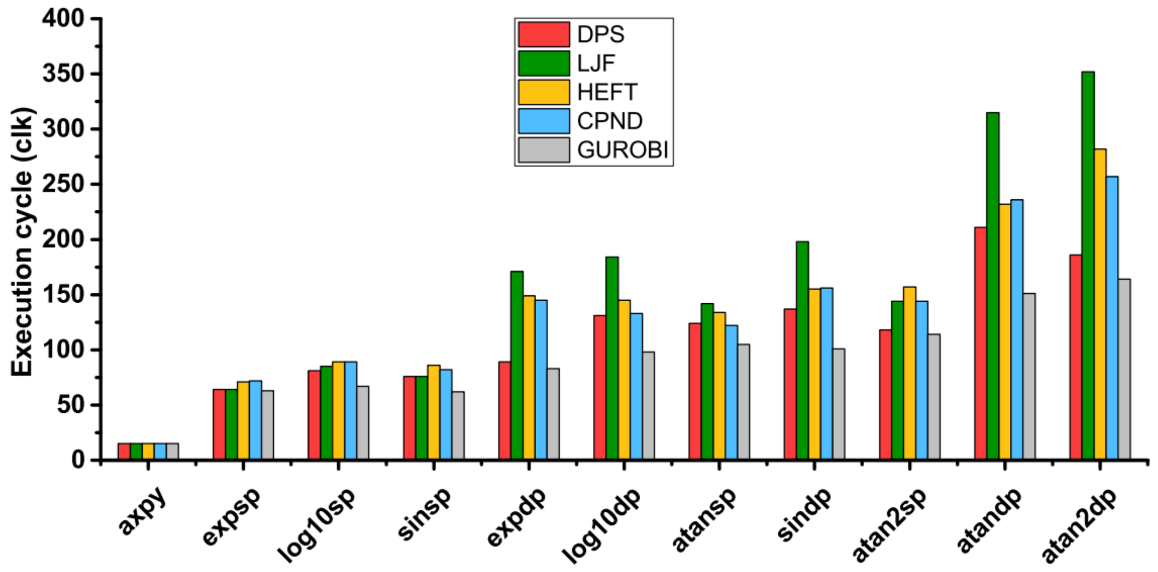
# PART 3

# Results

# 3.1 Execution cycle of solutions



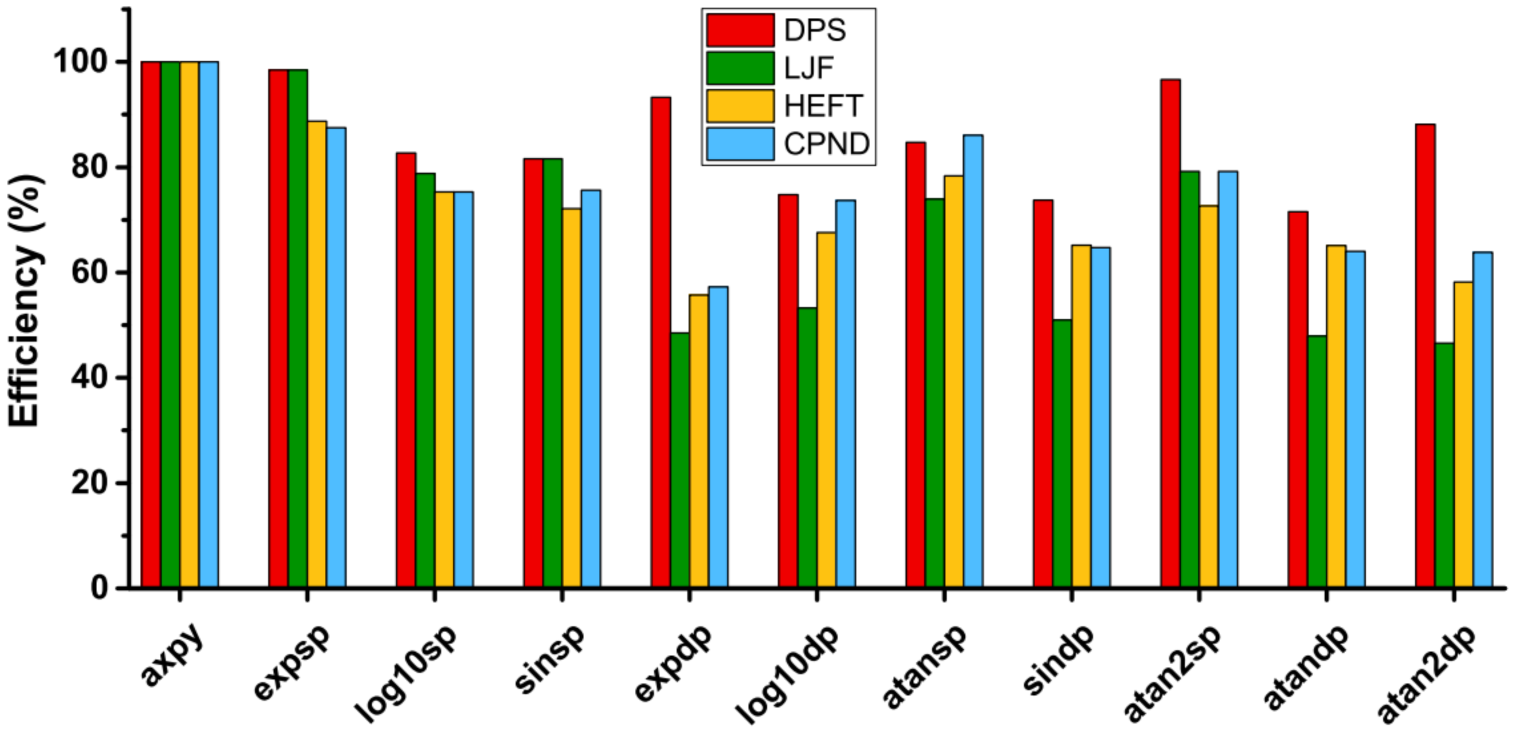

Instruction number of benchmarks



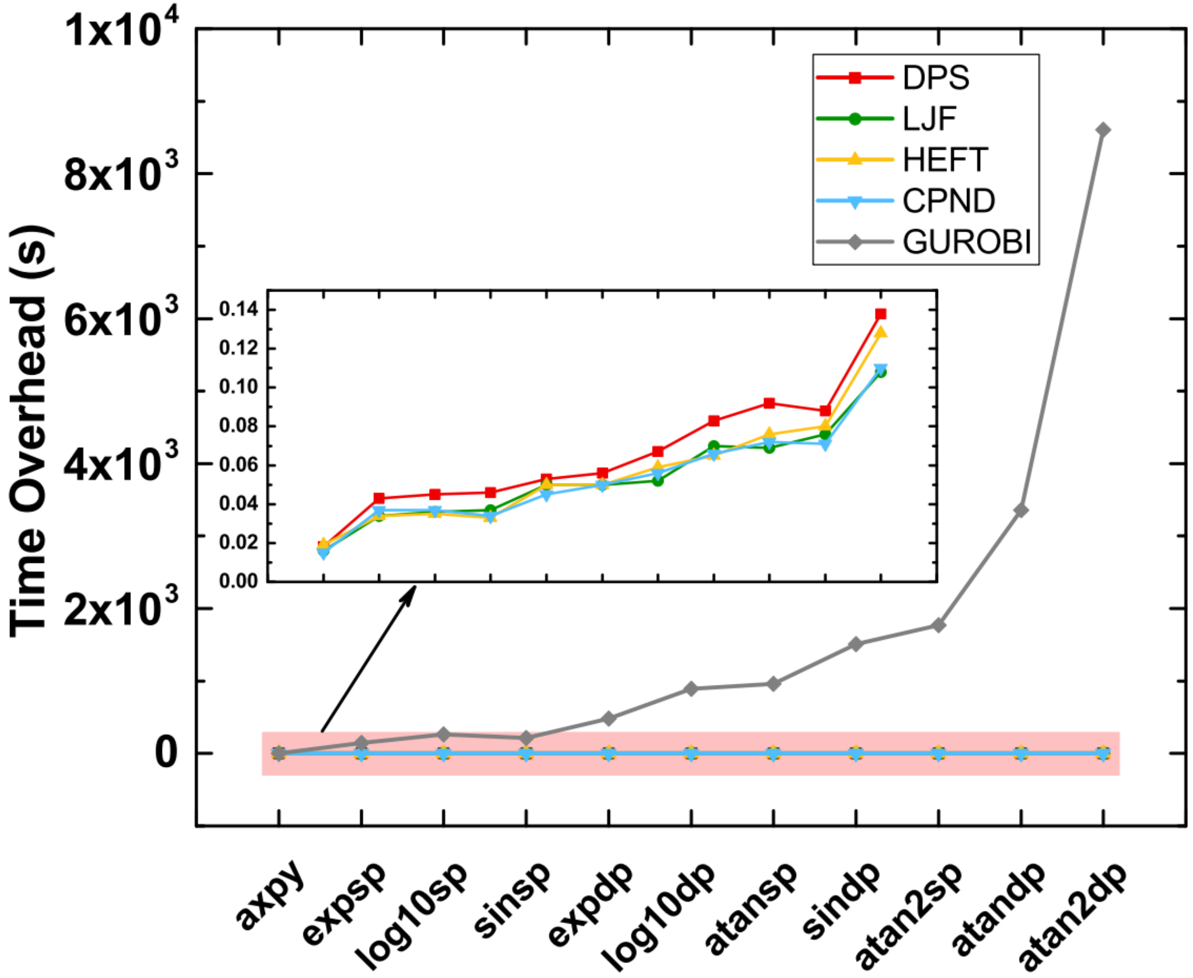

Execution cycle of the benchmarks

# 3.2 Efficiency

$$Efficiency = \frac{Execution\ cycle\ of\ GUROBI}{Execution\ cycle\ of\ the\ methods}$$



# 3.2 Time Overhead



# PART 4

## Conclusion

# 4. Conclusion

- The DPS proposed in this work achieves a trade-off between execution and time overhead
- Compared with the three LS algorithms, DPS shows a good scalability and efficiency improvement of up to 44% within acceptable time overhead
- One future work is to explore the optimization space toward the optimal solutions.

Thank you!