Variation-Aware Evaluation of MPSoC Task Allocation and Scheduling using Statistical Model Checking

Mingsong Chen[1*], Daian Yue[1], Xiaoke Qin[3], Xin Fu[2] and Prabhat Mishra[3]

[1]*Software Engineering Institute, East China Normal University, China*

[2]*ECE Department, University of Houston, USA*

[3]*CISE Department, University of Florida, USA*

***Presenter**

DATE 2015, Grenoble, France

Outline

- Introduction
- Preliminary Knowledge
  - Variation-aware Construction of NPTA
  - UPPAAL-SMC Based Evaluation
- Our Quantitative TAS Evaluation Approach
  - Model and Property Generation
  - SMC-Based TAS Evaluation
  - Parameter Tuning
- Experimental Results
- Conclusion

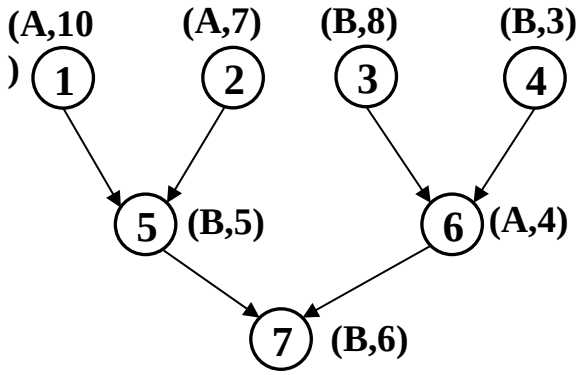
Outline

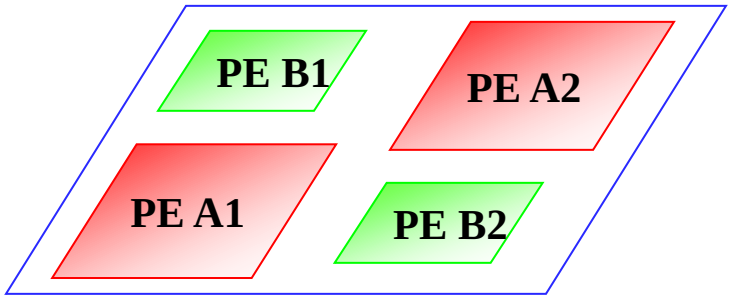
- Introduction
- Preliminary Knowledge
  - Variation-aware Construction of NPTA
  - UPPAAL-SMC Based Evaluation
- Our Quantitative TAS Evaluation Approach
  - Model and Property Generation
  - SMC-Based TAS Evaluation
  - Parameter Tuning
- Experimental Results
- Conclusion

Task Allocation & Scheduling in MPSoC

- Task allocation & scheduling is important for MPSoC Design
  - Maximize the utilization of available Processing Elements (PEs)
  - Satisfying various design constraints (e.g., response time, power)

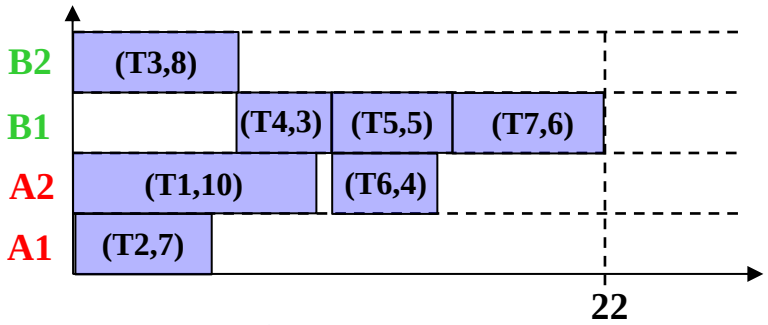
Given





Power(A)=10, Power(B)=15, MaxPower(Design)=35

Determine



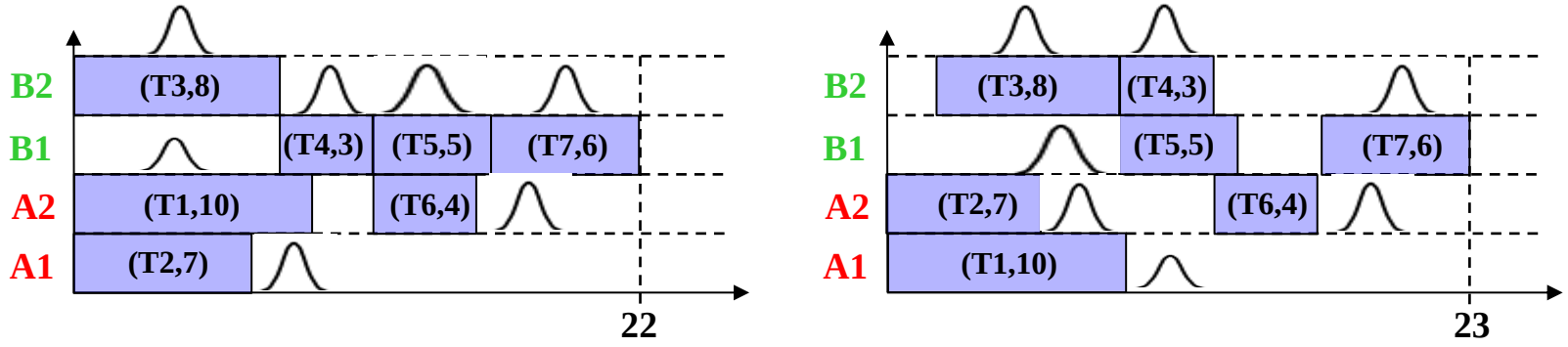
- TAS instance representation
  - (task scheduling sequence; task-resource binding info)
  - Example:(T3,T1,T2,T4,T5,T6,T7;B2,A2,A1,B1,B1,A2,B1)

Variations in TAS

- The control the manufacturing process in deep-submicron technologies is becoming harder
  - Increasing performance and power consumption variations across identically designed PEs and Chips
  - Up to 25% performance variation and power variation within an experimental Intel CPU [P. Gupta *et al.*, TCAD, 2013]
- WCET methods can lead to overly pessimistic evaluation
- Variation-aware performance analysis is becoming more and more important in system-level task allocation.
  - *Performance yield* is proposed to define the probability of an assigned TAS instance meeting required MPSoC design constraints under variation [Wang *et al.*, ICCAD, 2007]
  - Various heuristic strategies have been proposed to maximize performance yields

Limitations & Challenges in TAS under Variations

- Due to variations, it's hard to determine which strategy can generate TAS instances with better performance yield. E.g.,



- Limitations of previous work
  - Inaccurate modeling of parallel task execution (e.g., ILP method)
  - Constraint solving based approaches can only answer yes or no, but cannot answer why the performance yield is bad

Challenges:

i) How to accurately model parallel task executions under various kinds of variations?

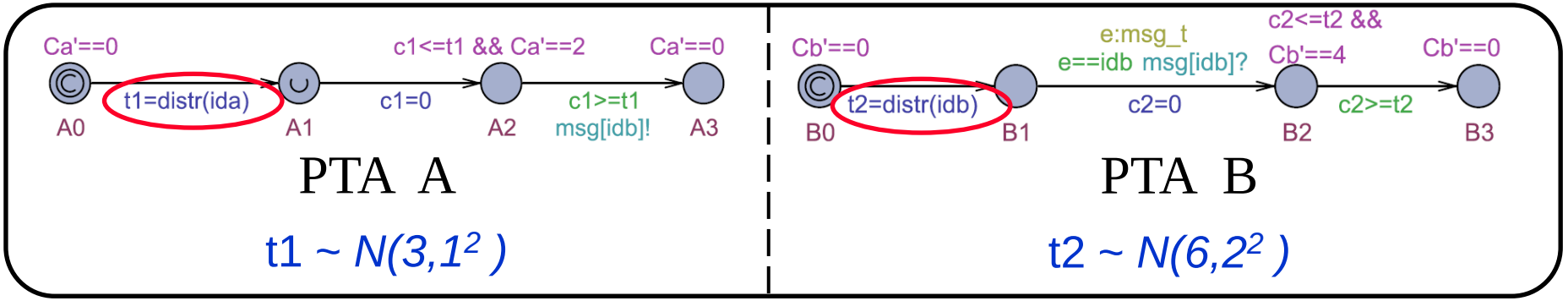
ii) How to enable the quantitative reasoning of performance yield to achieve better designs automatically?

Outline

- Introduction
- Preliminary Knowledge
  - Variation-aware Construction of NPTA
  - UPPAAL-SMC Based Evaluation
- Our Quantitative TAS Evaluation Approach
  - Model and Property Generation
  - SMC-Based TAS Evaluation
  - Parameter Tuning
- Experimental Results
- Conclusion

Variation-Aware Construction of NPTA

- NPTA - Network of Priced Timed Automata
- An NPTA instance, (A | B)



Time of reaching ($A3$, $B3$) ~ $N(9,1^2+2^2)$.

- A possible behavior of the NPTA (A|B)

$$((A_0,B_0),[c_1=0,c_2=0,C_a=0,C_b=0]) \xrightarrow{0}$$

$$((A_1,B_1),[c_1=0,c_2=0,C_a=0,C_b=0]) \xrightarrow{0}$$

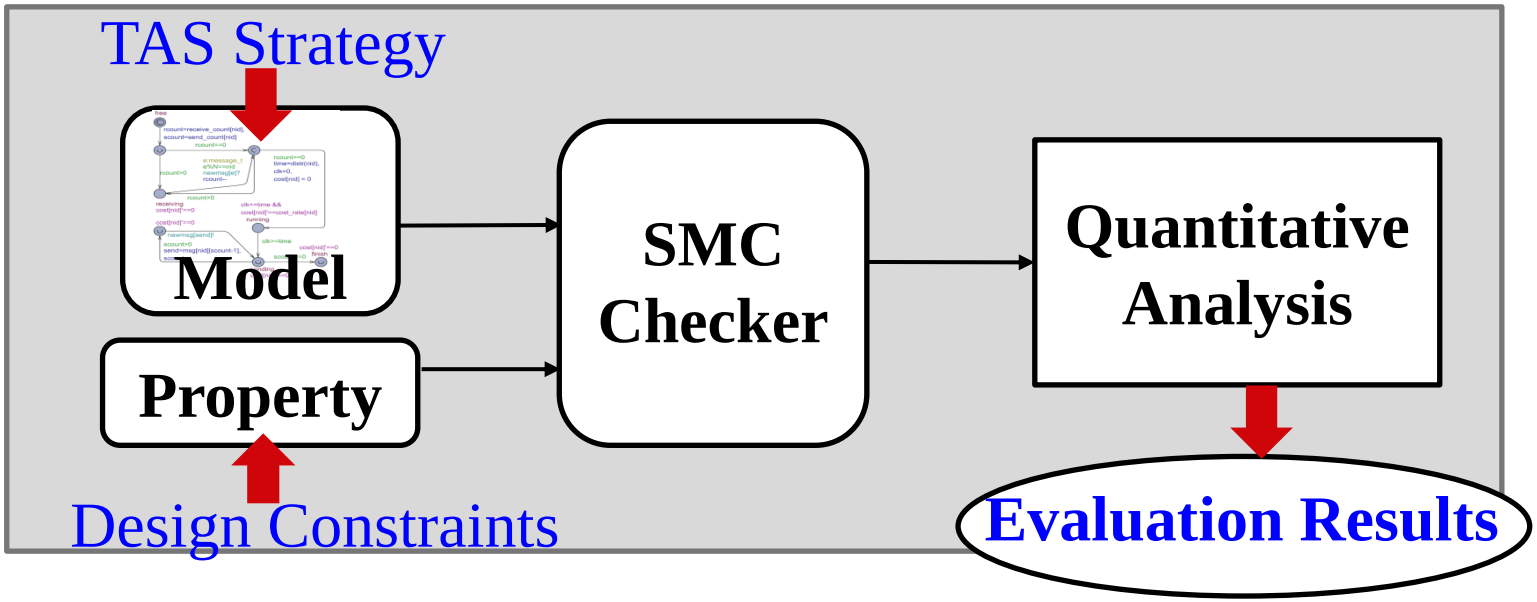
$$((A_2,B_1),[c_1=0,c_2=0,C_a=0,C_b=0]) \xrightarrow{2.5} \xrightarrow{msg[idb]!}$$

$$((A_3,B_2),[c_1=2.5,c_2=0,C_a=5,C_b=0]) \xrightarrow{5.1}$$

$$((A_3,B_3),[c_1=7.6,c_2=5.1,C_a=5,C_b=20.4]) \xrightarrow{\cdots} \ldots$$

Statistical Model Checking (SMC)

- Our TAS evaluation is based on SMC, which is effective for checking large stochastic systems

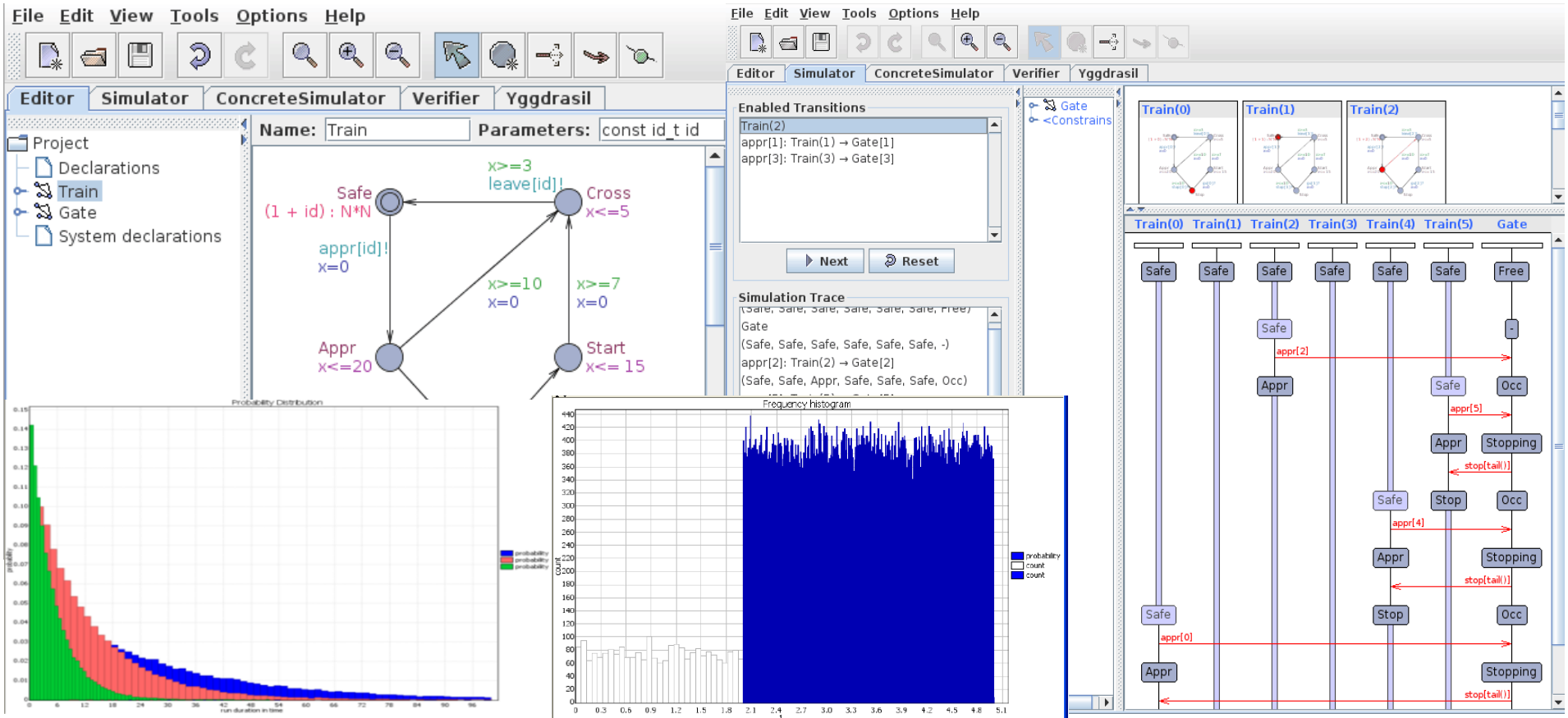


- UPPAAL-SMC supported queries
  - Qualitative check: ***Pr [time <= bound] (<> expr) >= p***
  - Quantitative check: ***Pr [time <= bound] (<> expr)***
  - Probability comparison:

  Pr [time1 <= bound1] (<> expr1) >= Pr [time2 <= bound2] (<> expr2)

UPPAAL-SMC

- UPPAAL-SMC versus formal model checking
  - Based on simulation, thus requiring far less memory and time
  - Allow high scalable validation approximation
  - Support quantitative performance analysis
- Application domains: Real-time systems, Smart building, Biology, …

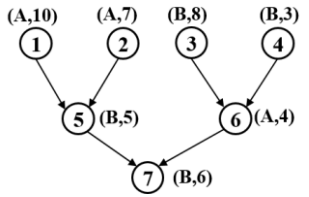


Outline

- Introduction
- Preliminary Knowledge
  - Variation-aware Construction of NPTA
  - UPPAAL-SMC Based Evaluation
- Our Quantitative TAS Evaluation Approach
  - Model and Property Generation
  - SMC-Based TAS Evaluation
  - Parameter Tuning
- Experimental Results
- Conclusion

Problem Definition

Task Graph DAG *G = (V,E)*



Resources with Variation Info

- PEs +type infor
- PE variation info

Design Constraints

- Maximum power
- Response time
- ……

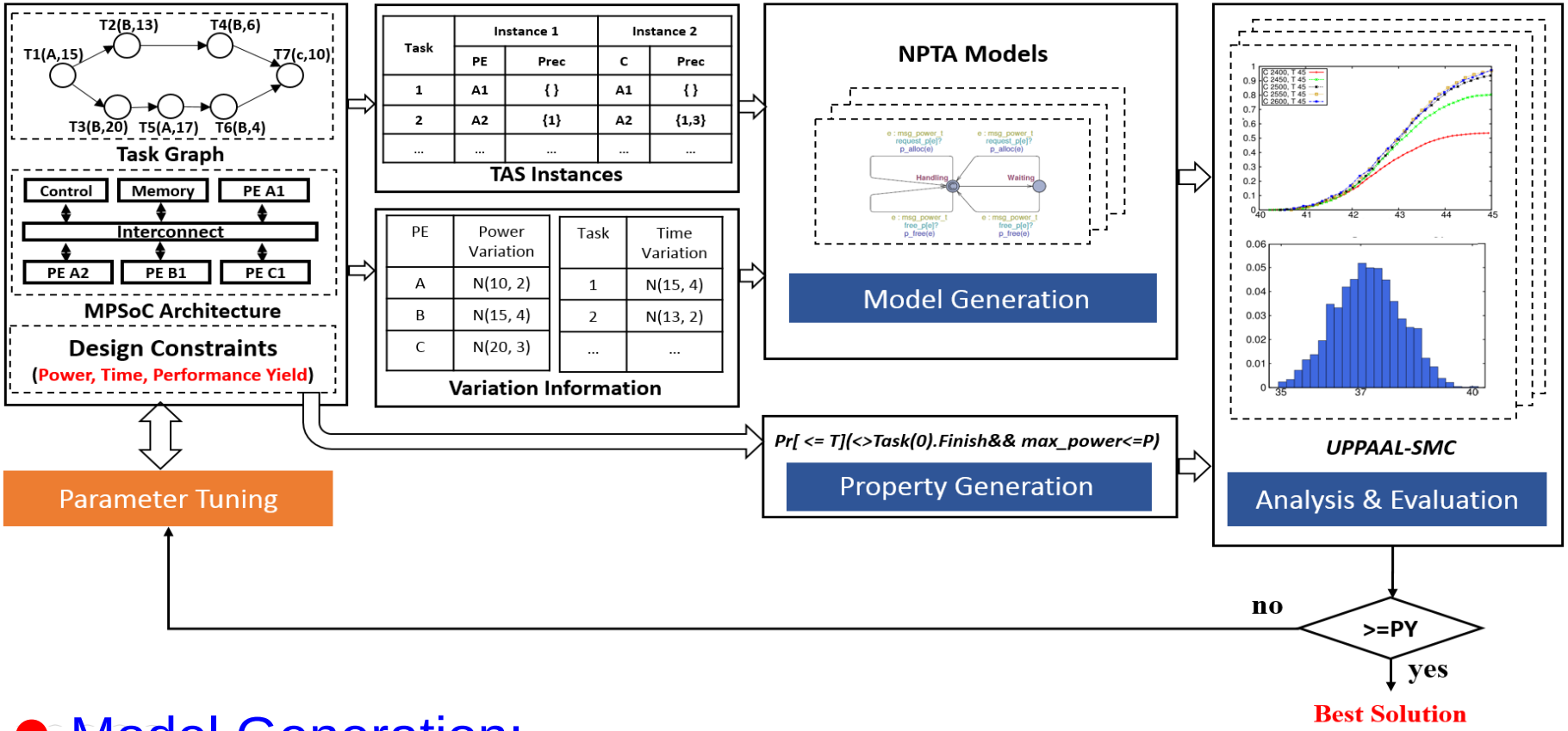
TAS Strategies

- Build in
- User specified
- ……

Quantitative Evaluation Framework

Performance Yield Results

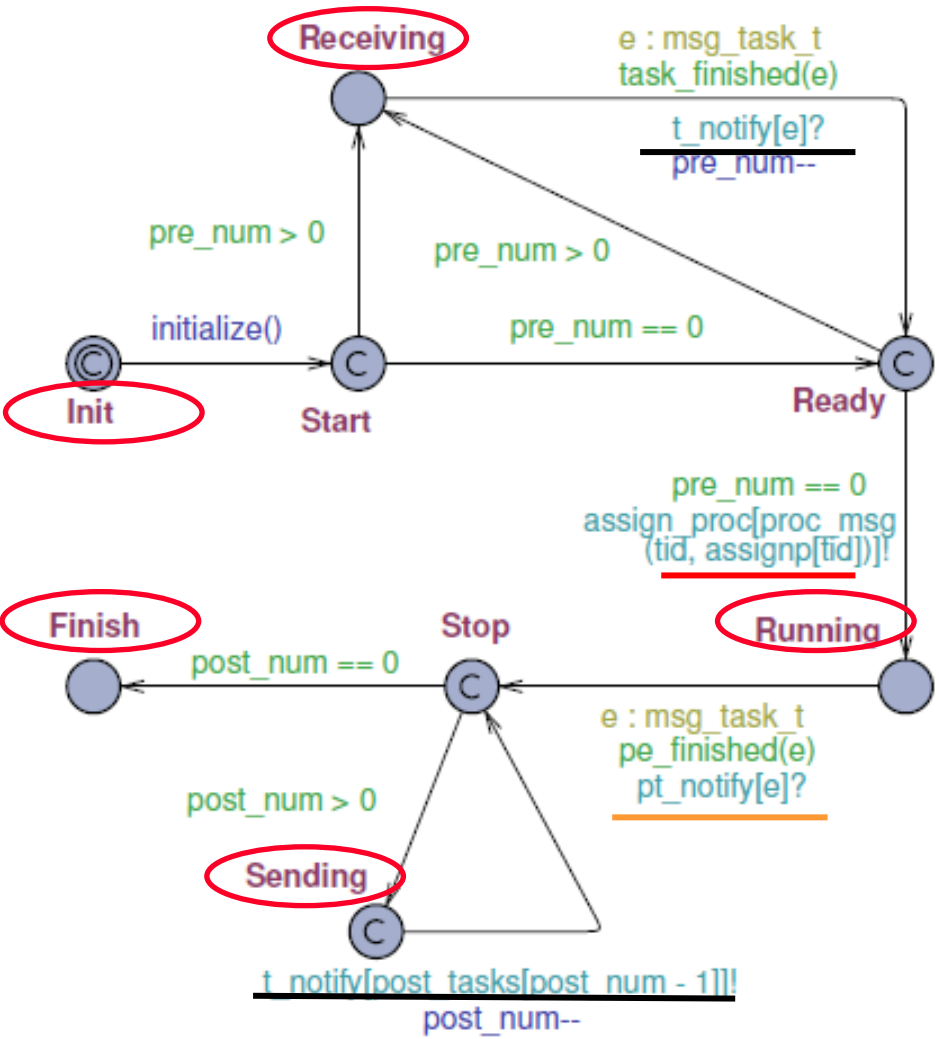
Our Framework



- Model Generation:
  - TAS instances and variation information are translated into NPTA model
- Property Generation
  - Design constraints are converted into propoerties to enable queires
- Analysis & Evaluation
  - Conduct the automated quantitative analysis using UPPAAL-SMCs

NPTA Model Generation

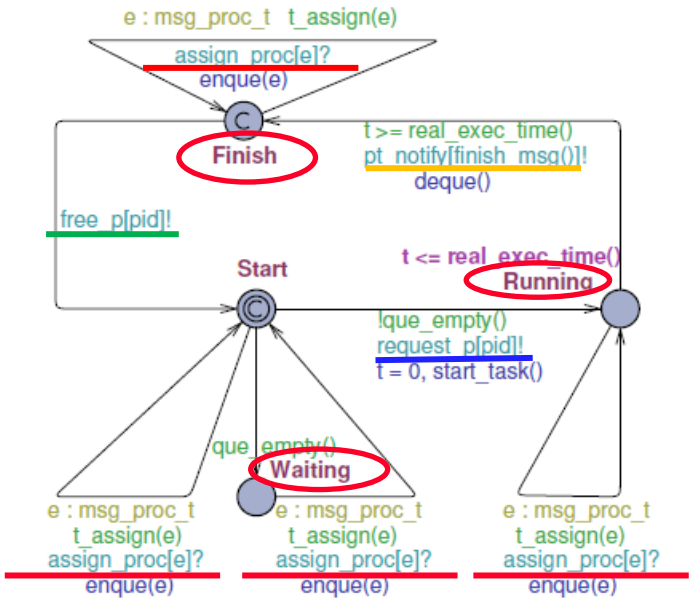
Front-end Model for Tasks



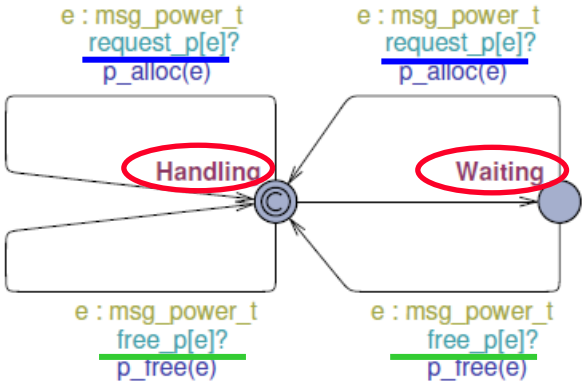
- Task.Initial state
  - The beginning of a task
- Task.Receiving state
  - Tries to obtain notification messages from all the predecessors
- Task.Running state
  - All predecessors finished
  - Current task is executing
- Task.Sending state
  - Notify all successive tasks about its completion
- Task.Finish state
  - The completion of a task

NPTA Model Generation

Front-end Model for PEs



Front-end Model for Power Monitor



Each PE has a queue containing ready tasks.

- PE.Waiting state
  - Waiting for ready tasks assigned to this PE
- PE.Running state
  - Executing the task at the head of the queue with a specified time
- PE.Finish state
  - The completion of a task execution
- Power.Handling state
  - Registration of a task running on some PE with a specified power
- Power.Finish state
  - Release the task on the PE together with its power requirement

NPTA Model Generation

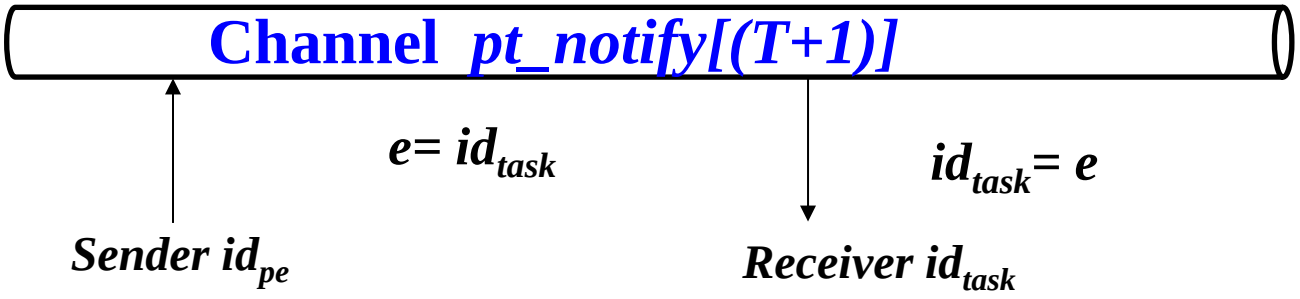
Back-end Configuration describes both concurrent behaviors of tasks and variation information.

- **Task precedence configuration**
  - To use a matrix *PM* to indicate task precedence relation
  - *PM[i][j]=1* → the i^{th} tasks finishes before j^{th} task starts
- **Variation configuration**
  - Describe the time and power distributions of services
  - *tvar[N+1][2]* denotes the time variation
  - *pvar[N+1][2]* denotes the power variation
- **Different TAS instance will have different above configurations**

NPTA Model Generation

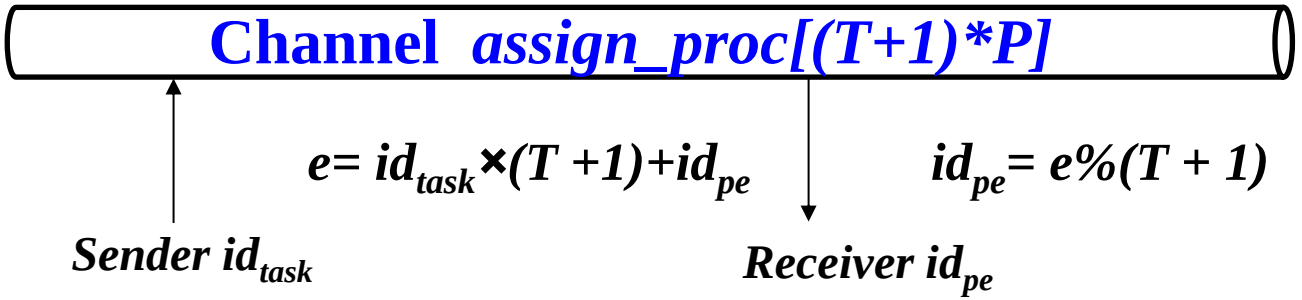
Our approach uses broadcast channels to synchronize PTAs. Assume that there are *T* tasks and *P* PEs.

- **Communication from PEs to tasks (many to 1)**



Private channel from all the Pes to task with ID id_{task}

- **Communication from tasks to PEs (many to many)**



Property Generation

"What is the performance yield that the task graph can be completed under the time constraint of x and power constraint of y?"

Pr [<= x] (<> Task(0).finish && max_power<= y)

- ***[<= x]*** **indicates the time constraint of t**
- ***<>p checks*** **whether customer requirement p can be fulfilled eventually.**
- ***Task(0).finish*** **indicates the completion of all the tasks**
- ***max_power<= y*** **denotes that the maximum power of the task execution cannot be larger than *y***

TAS Instance Generation

Our framework incorporates two built-in power-constrained time-minimization TAS methods. Other TAS approaches can be easily integrated in our framework.

- **List Scheduling**
  - **Can quickly achieve near-optimal TAS instances**
- **BULB Approach**
  - **Can explore all the search space to get optimal TAS instances in a branch-and-bound manner**

By tuning the operation enumeration order, we can obtain different TAS instances using the same above TAS approach.

Parameter Tuning

When the generated TAS instances cannot meet the specified performance yield, our framework allows the tuning of design constraint or architecture parameters to explore better instances.

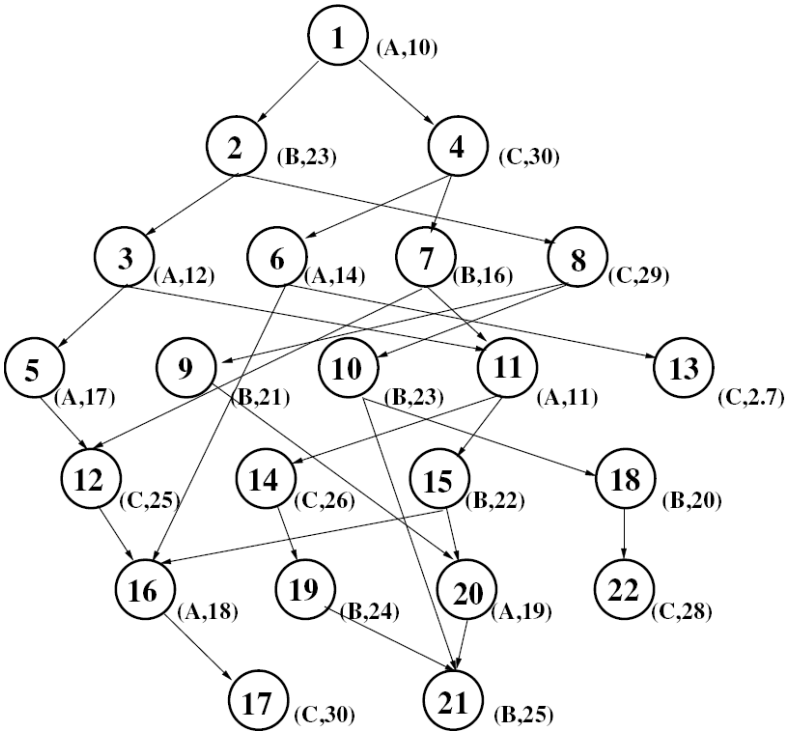
- **Power constraint tuning**
  - Iteratively reduce the power constraint until
    - Either achieve a satisfying performance yield,
    - or the time constraint is violated
- **Architectural configuration exploration**
  - Explore and evaluate all possible MPSoC designs with different PE combinations to achieve a required performance yield

Outline

- Introduction
- Preliminary Knowledge
  - Variation-aware Construction of NPTA
  - UPPAAL-SMC Based Evaluation
- Our Quantitative TAS Evaluation Approach
  - Model and Property Generation
  - SMC-Based TAS Evaluation
  - Parameter Tuning
- Experimental Results
- Conclusion

Case Study

- We use TGFF to generate a 22-node synthetic task graph (max input-degree equals 3 and max output-degree equals 2).
- We adopt UPPAAL-SMC as the engine of our framework with parameters ε=0.05 , α= 0.05.
- All the experimental results were obtained on a Ubuntu Linux desktop with 4.0 GHz AMD CPU and 8GB RAM.



Each task is assigned with a the information of the target PE type and mean execution time.

Case Study

- There are three types of PEs in the MPSoC design and the task execution time and power follow the Gaussian distribution on these PEs [Sarangi *et al.*, IEEE TSM, 2008].

Table 1: Power Variation for MPSoC PEs

Type	Power Variation	Exec. Time Variation
A	$N(10, 1.00^2)$	$N(x, (0.05x)^2)$
B	$N(15, 2.25^2)$	$N(x, (0.10x)^2)$
C	$N(20, 2.60^2)$	$N(x, (0.07)^2)$

- In table 1, *x* represents the mean execution time. For example, if the mean execution time of a task is 10, its execution time on PE A follows the Gaussian distribution $N(10,0.5^2)$.
- In the experiment, we apply four TAS strategies: two list scheduling variants (i.e., *List 1* and *List 2*), and two BULB variants (i.e., *BULB 1* and *BULB 2*).

MPSoC Design and Constraint

- Assume that the MPSoC design consists of 8 PEs (including 3 PEs of type A, 3 PEs of type B, and 2 PEs of type C) and the design is fixed before TAS.
- **Design constraint:** The performance yield cannot be smaller than 90% within 190 time units and 65 power units.

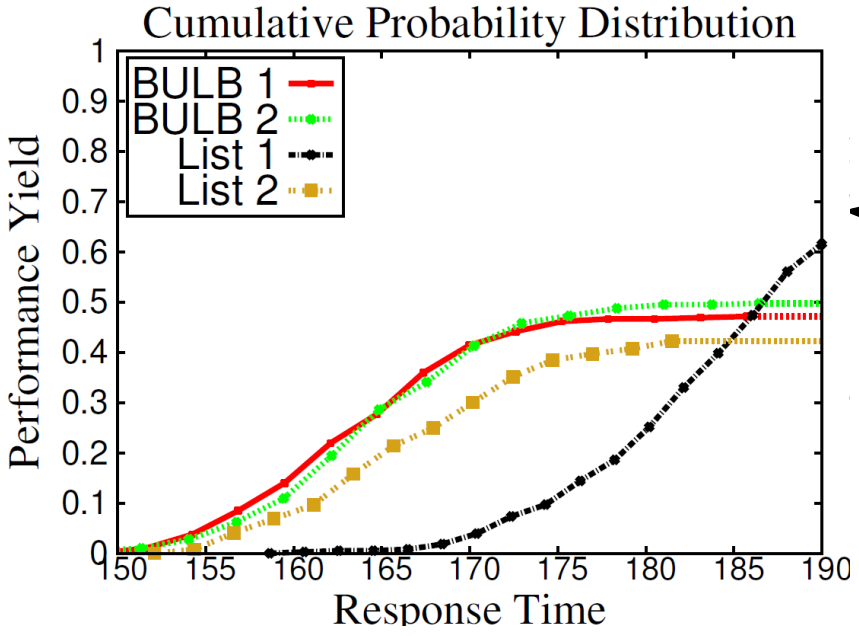


Fig 1. Power Constraint = 65

Although BULB is optimal without considering variations, List 1 achieves the best result.

None of the generated TAS instances is selected!

The reason of low performance yield is mainly due to the power variation, because the increase of response time cannot improve the performance yield after time 175.

Tuning of Power Constraint

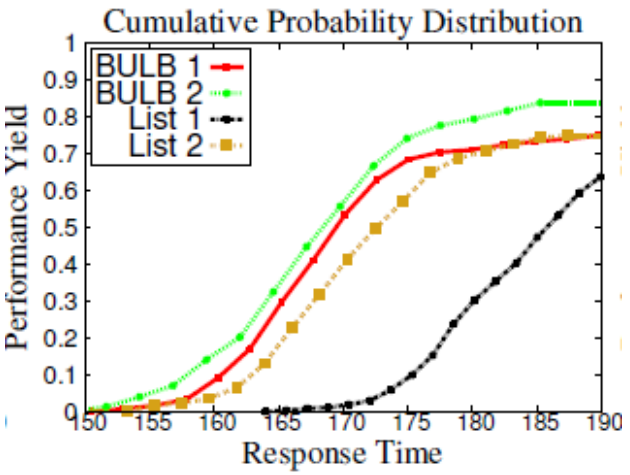


Fig 2. Power Constraint = 60

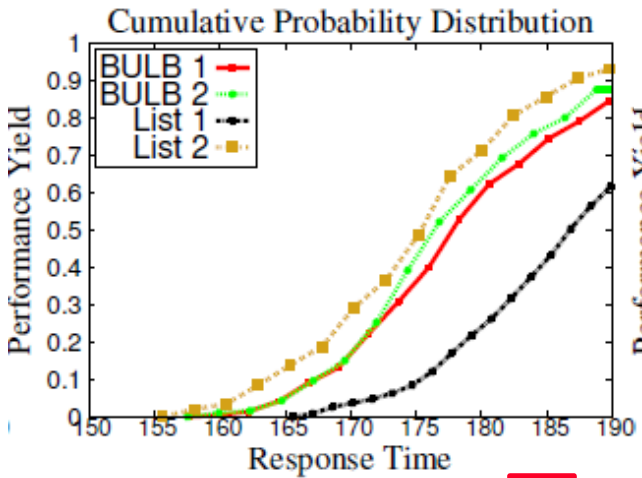


Fig 3. Power Constraint = 55

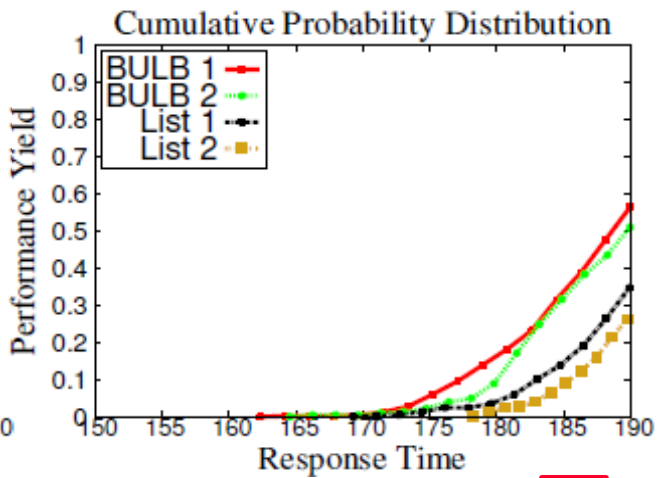
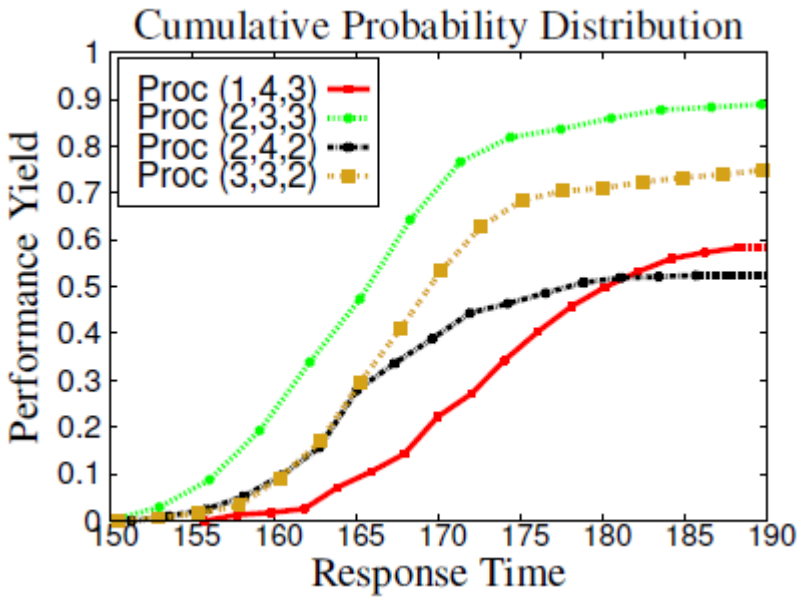


Fig 4. Power Constraint = 50

- Tune the power constraint with an interval of 5 units
  - Significant increase of performance yield in Fig. 2 and Fig. 3.
  - The reduction of the power constraint in TAS generation will result in large response time, thus affecting the overall performance yield (see Fig. 4).

Tuning of Architectural Configurations

Pr[<= 190](<> Task(0).Finish && max_power<=60)



Proc(x,y,z) **means that the design of *x* PEs of type A, *y* PEs of type B, and *z* PEs of type C.**

Fig. 5 Evaluation of different architectural configurations

- Assume that the MPSoC design can be customized with at most 8 PEs of different types.
- By using *BULB 1*, *Proc (2,3,3)* achieves the best performance yield as well as the best average response time.

Conclusion

- Automated variation-aware evaluation of TAS strategies is important in MPSoC design flow
  - Reduce the decision making efforts of MPSoC designers
  - Improve the overall performance yields
- We propose an UPPAAL-SMC-based TAS strategy evaluation framework
  - Support complex performance yield queries under power and time variations
  - Enable the tuning of design constraint parameters to explore TAS instances with better performance yields
- Comprehensive experimental results demonstrates the efficacy of our approach

Thank you !