Symbiotic Job Scheduling on the IBM POWER8

J. Feliu[1] S. Eyerman[2] J. Sahuquillo[1] S. Petit[1]

[1]Department of Computing Engineering (DISCA)
Universitat Politècnica de València
jofepre@gap.upv.es, {jsahuqui,spetit}@disca.upv.es

[2]Intel Belgium
stijn.eyerman@intel.com

March 16th, 2016

[2] *This work was done while Stijn Eyerman was at Ghent University*

Outline

Introduction

- Scheduling is important for manycore / manythread systems
  - Combinatorial amount of ways to schedule applications with different performance
- Scheduling for CMPs of SMT cores is challenging
  - Different levels of resource sharing
  - SMT performance very sensitive to co-runners
- Selecting the optimal schedule is an NP-hard problem
- Predicting the performance of a schedule is not trivial because of the high amount of resource sharing in SMTs

Introduction

Previous work on symbiotic scheduling

- Uses sampling to explore the space of possible schedules *(Snavely et al., ASPLOS'00)*
- Relies on novel hardware *(Eyerman et al, ASPLOS'10)*
- Performs an offline analysis with μ*benchmarks* to predict the interference between applications *(Zhang et al., MICRO'14)*

Introduction

Previous work on symbiotic scheduling

- Uses sampling to explore the space of possible schedules *(Snavely et al., ASPLOS'00)*
- Relies on novel hardware *(Eyerman et al, ASPLOS'10)*
- Performs an offline analysis with μ*benchmarks* to predict the interference between applications *(Zhang et al., MICRO'14)*

Our symbiotic job scheduler

- Online model-based scheduling
- Without sampling schedules
- On a recent commercial processor

Introduction
Main contributions

- **Interference model**
  - Predicts the interference among threads on a SMT core
  - Based on CPI stacks
  - Considers contention in all the shared resources

Introduction
Main contributions

- **Interference model**
  - Predicts the interference among threads on a SMT core
  - Based on CPI stacks
  - Considers contention in all the shared resources
- **Online scheduler**
  - Quickly explore the schedule space to select the optimal one
  - Quickly adapt to phase behavior

Introduction
Main contributions

- **Interference model**
  - Predicts the interference among threads on a SMT core
  - Based on CPI stacks
  - Considers contention in all the shared resources
- **Online scheduler**
  - Quickly explore the schedule space to select the optimal one
  - Quickly adapt to phase behavior
- **Implemented and evaluated on the IBM POWER8**
  - Average system throughput increase by 10.3% over a random scheduler and 4.7% over Linux

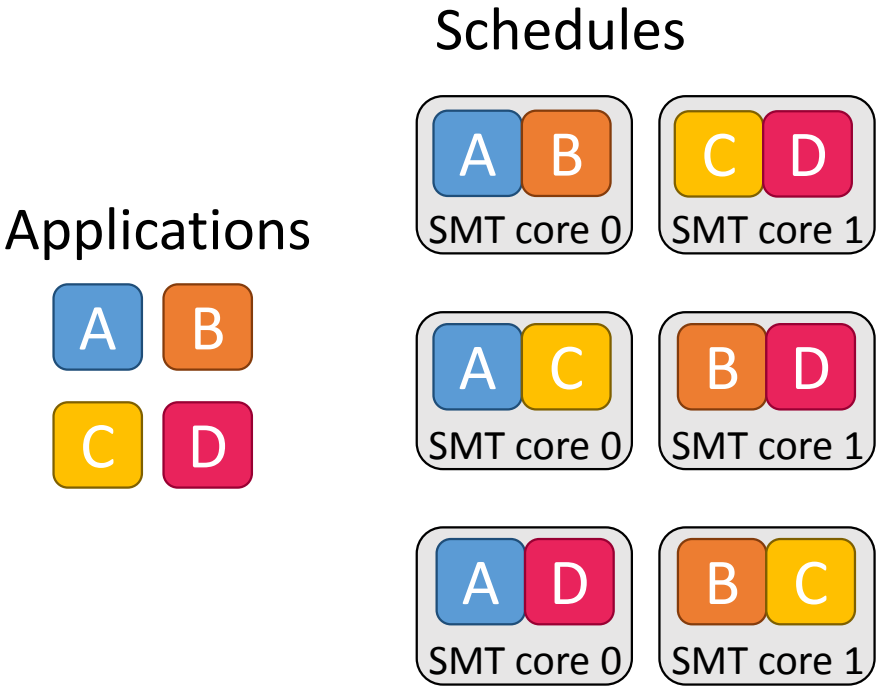
Outline

Predicting Job Symbiosis

- Symbiotic scheduler: based on a model that estimates job symbiosis
  - Predicts the slowdown of the application on a schedule
  - It is fast, allowing us to select the optimal schedule

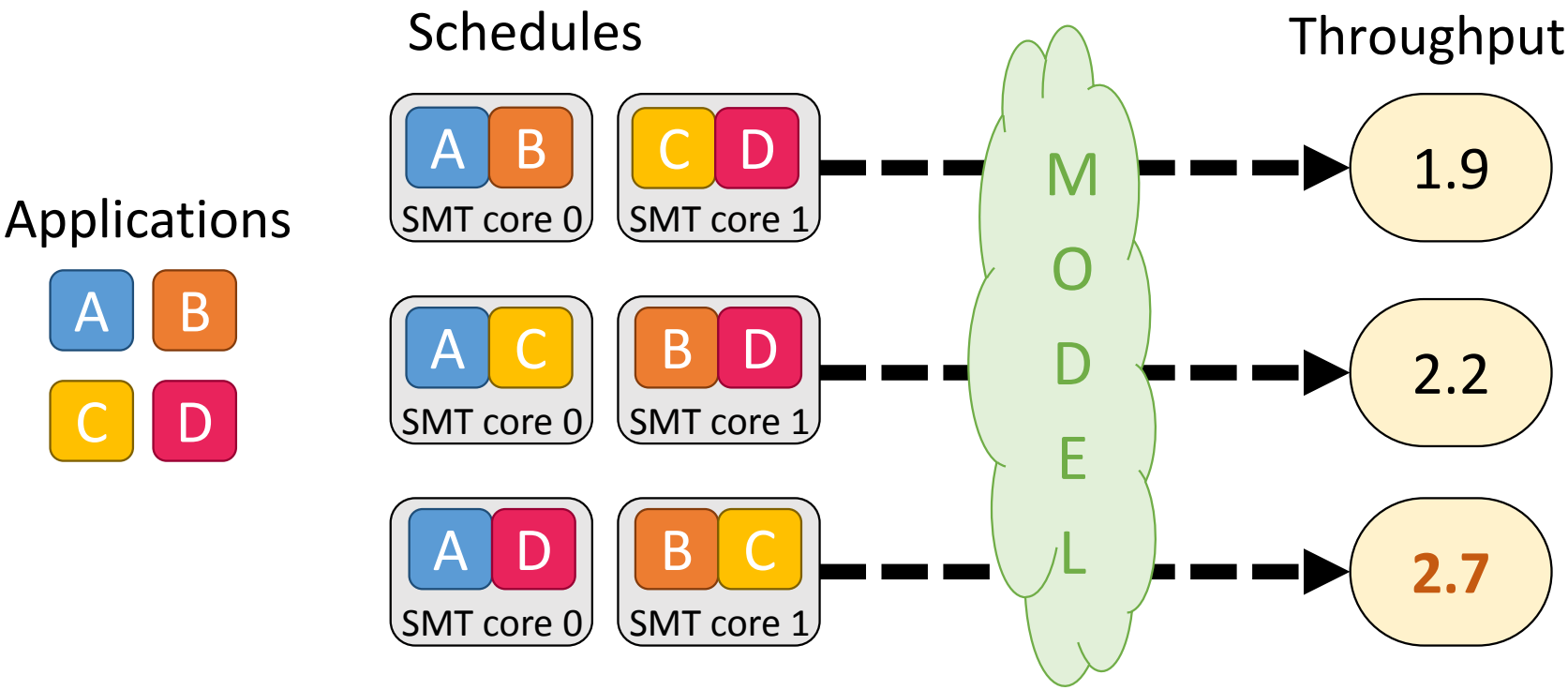
Predicting Job Symbiosis

- Symbiotic scheduler: based on a model that estimates job symbiosis
  - Predicts the slowdown of the application on a schedule
  - It is fast, allowing us to select the optimal schedule



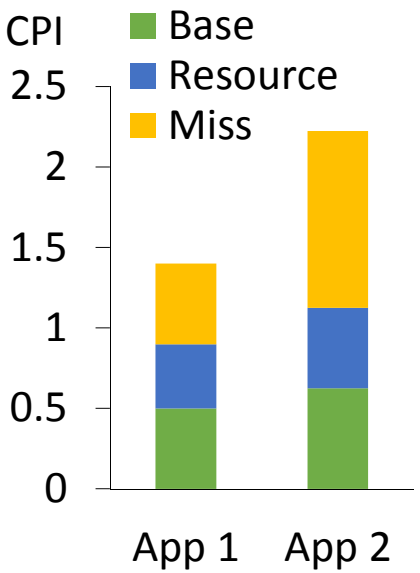
Predicting Job Symbiosis

- Symbiotic scheduler: based on a model that estimates job symbiosis
  - Predicts the slowdown of the application on a schedule
  - It is fast, allowing us to select the optimal schedule



Interference model

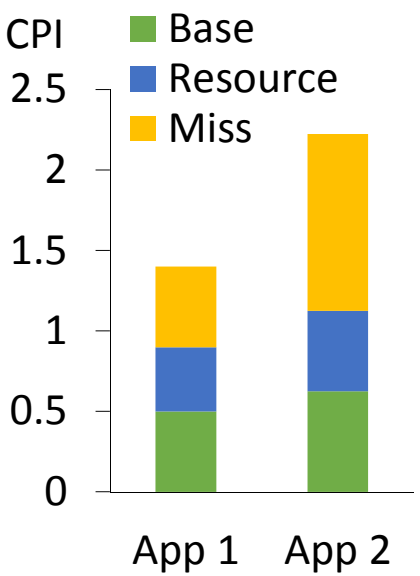
- The proposed model leverages CPI stacks to predict job symbiosis



Measured single-threaded CPI stacks

Interference model

- The proposed model leverages CPI stacks to predict job symbiosis



Measured single-threaded CPI stacks

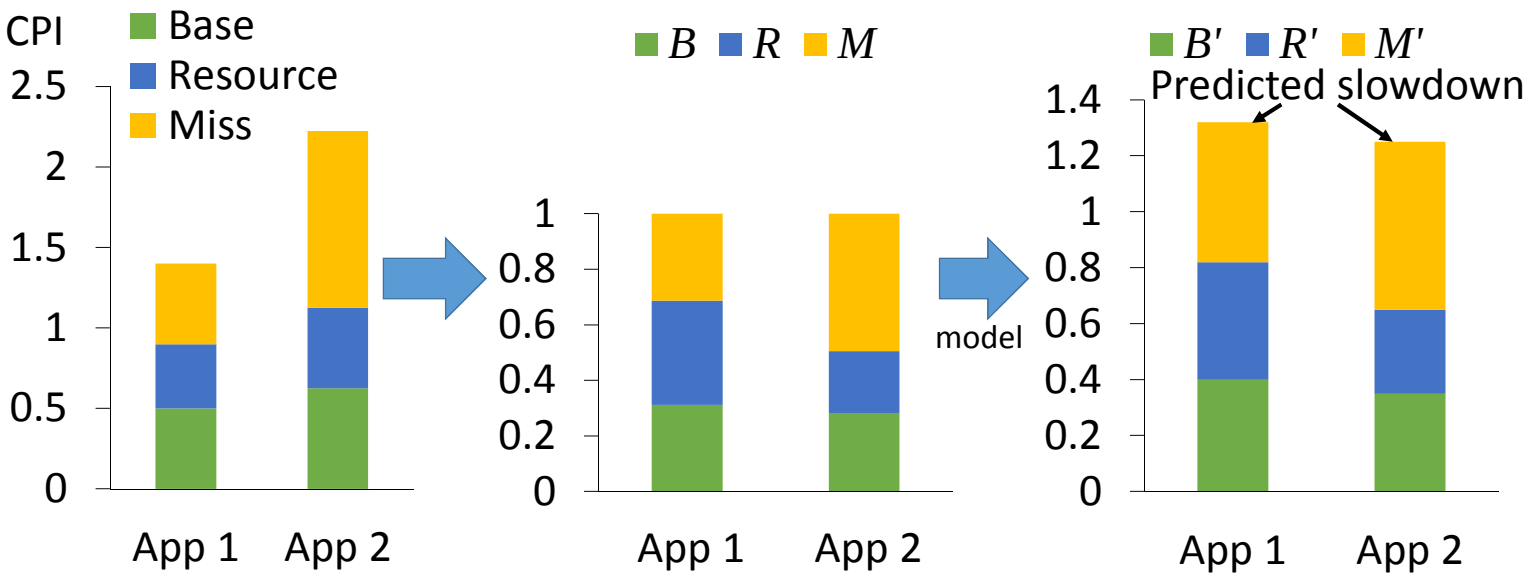
CPI Stacks

Divide the execution cycles into various components:

- Base: cycles where instructions are completed
- Resource: no instruction completed due to resource stall
- Miss: no instruction completed due to miss event

Interference model

- The proposed model leverages CPI stacks to predict job symbiosis
- Model: estimates the slowdown
  - Interprets the normalized CPI components as probabilities
  - Calculates the probability of interference



Model equation

Each component is modeled with the equation:

$$C'_j = \alpha_C + \beta_C C_j + \gamma_C \sum_{k \neq j} C_k + \delta_C C_j \sum_{k \neq j} C_k \quad (1)$$

Model equation

Each component is modeled with the equation:

$$C'_j = \alpha_C + \beta_C C_j + \gamma_C \sum_{k \neq j} C_k + \delta_C C_j \sum_{k \neq j} C_k \quad (1)$$

Components

- C_j represents thread j own component in ST mode

Model equation

Each component is modeled with the equation:

$$C'_j = \alpha_C + \beta_C C_j + \gamma_C \sum_{k \neq j} C_k + \delta_C C_j \sum_{k \neq j} C_k \quad (1)$$

Components

- C_j represents thread j own component in ST mode
- C_k represents the ST component of the other threads in the schedule

Model equation

Each component is modeled with the equation:

$$C'_j = \alpha_C + \beta_C C_j + \gamma_C \sum_{k \neq j} C_k + \delta_C C_j \sum_{k \neq j} C_k \quad (1)$$

Components

- C_j represents thread j own component in ST mode
- C_k represents the ST component of the other threads in the schedule
- C_j' identifies the SMT component of thread j

Model equation

Each component is modeled with the equation:

$$C'_j = \alpha_C + \beta_C C_j + \gamma_C \sum_{k \neq j} C_k + \delta_C C_j \sum_{k \neq j} C_k \quad (1)$$

Parameters

- α_C reflects a constant increase in SMT over ST

Model equation

Each component is modeled with the equation:

$$C'_j = \alpha_C + \beta_C C_j + \gamma_C \sum_{k \neq j} C_k + \delta_C C_j \sum_{k \neq j} C_k \quad (1)$$

Parameters

- α_C reflects a constant increase in SMT over ST
- β_C reflects the fraction or relative increase of the original ST component appears in SMT execution

Model equation

Each component is modeled with the equation:

$$C'_j = \alpha_C + \beta_C C_j + \gamma_C \sum_{k \neq j} C_k + \delta_C C_j \sum_{k \neq j} C_k \quad (1)$$

Parameters

- α_C reflects a constant increase in SMT over ST
- β_C reflects the fraction or relative increase of the original ST component appears in SMT execution
- γ_C models the impact of the sum of the ST components of the other co-scheduled threads

Model equation

Each component is modeled with the equation:

$$C'_j = \alpha_C + \beta_C C_j + \gamma_C \sum_{k \neq j} C_k + \delta_C C_j \sum_{k \neq j} C_k \quad (1)$$

Parameters

- α_C reflects a constant increase in SMT over ST
- β_C reflects the fraction or relative increase of the original ST component appears in SMT execution
- γ_C models the impact of the sum of the ST components of the other co-scheduled threads
- δ_C models extra interactions that may occur between threads

Model equation

Each component is modeled with the equation:

$$C'_j = \alpha_C + \beta_C C_j + \gamma_C \sum_{k \neq j} C_k + \delta_C C_j \sum_{k \neq j} C_k \quad (1)$$

Parameters

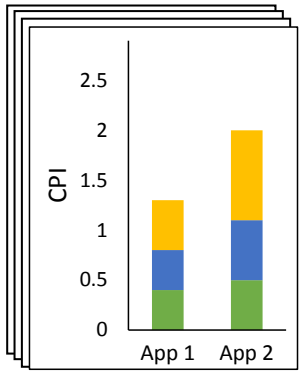
- α_C reflects a constant increase in SMT over ST
- β_C reflects the fraction or relative increase of the original ST component appears in SMT execution
- γ_C models the impact of the sum of the ST components of the other co-scheduled threads
- δ_C models extra interactions that may occur between threads
- The meaningful parameters are determined using regression

Model construction and slowdown estimation

- Model parameters determined by linear regression
  - One-time offline training
    - Based on experimental data
    - Not tied to applications, no need to retrain, no overfit

Model construction and slowdown estimation

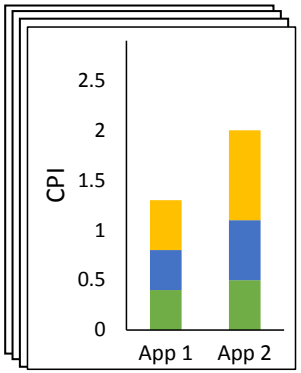
- Model parameters determined by linear regression
  - One-time offline training
    - Based on experimental data
    - Not tied to applications, no need to retrain, no overfit



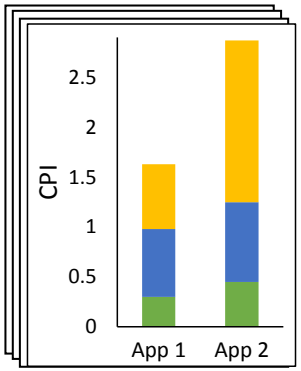
Measured ST CPI stacks (Alone)

Model construction and slowdown estimation

- Model parameters determined by linear regression
  - One-time offline training
    - Based on experimental data
    - Not tied to applications, no need to retrain, no overfit



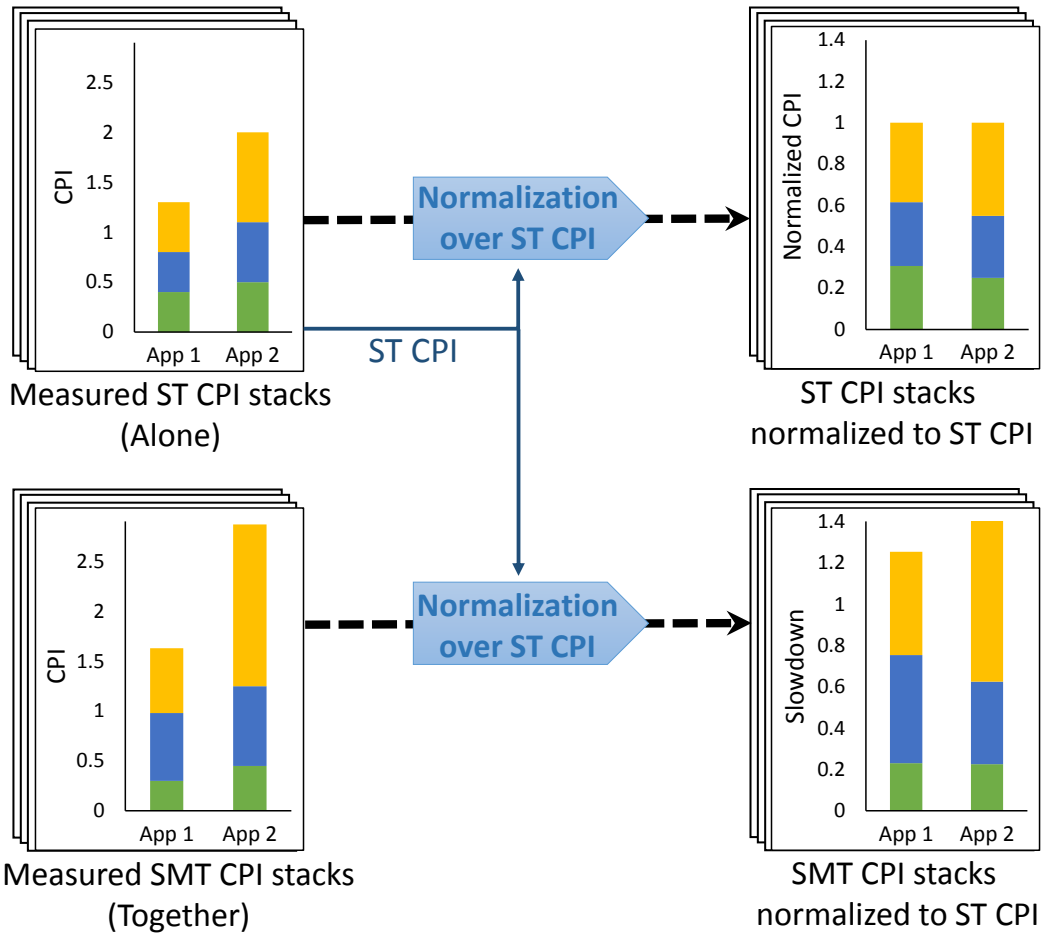
Measured ST CPI stacks (Alone)



Measured SMT CPI stacks (Together)

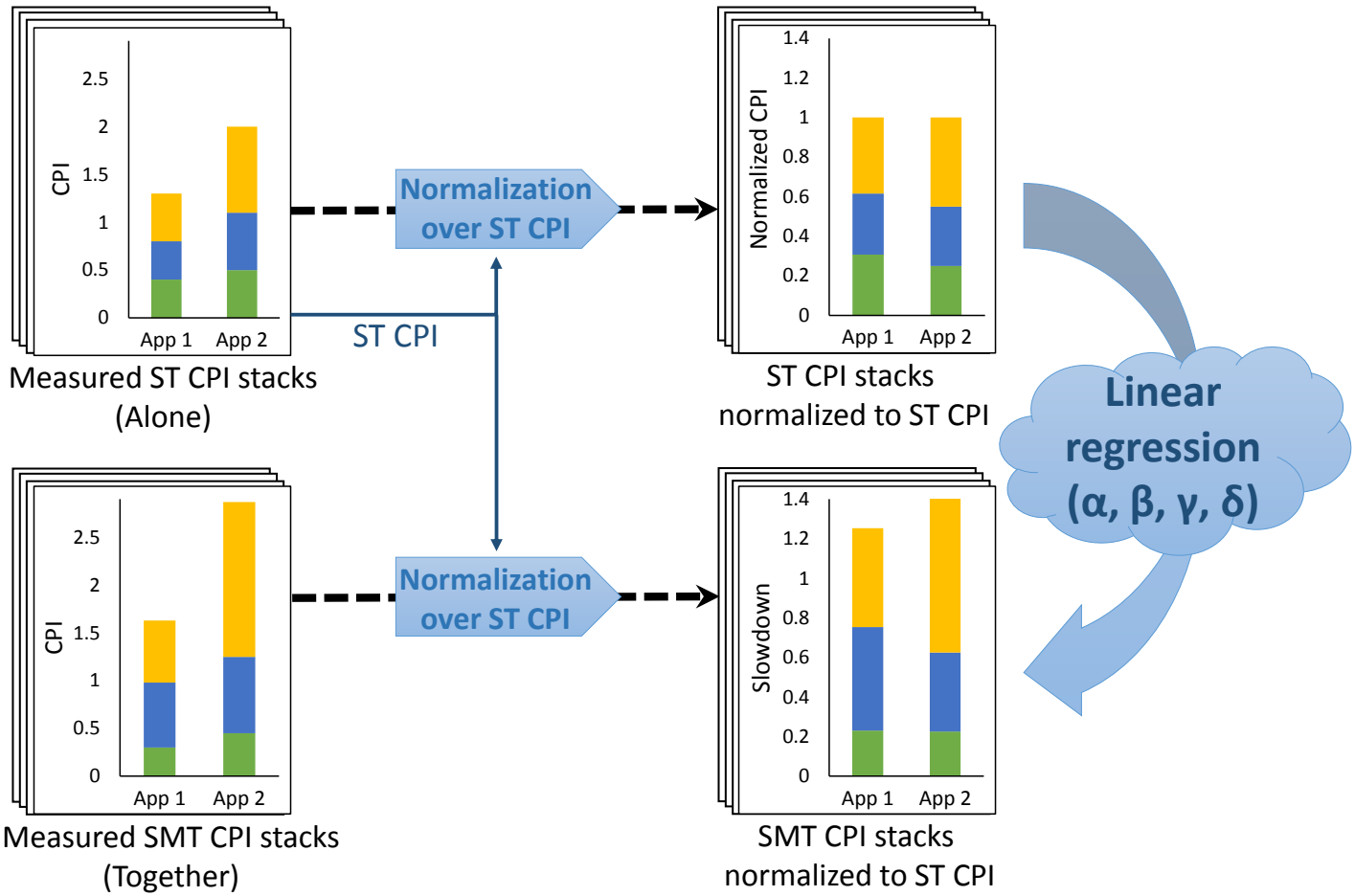
Model construction and slowdown estimation

- Model parameters determined by linear regression
  - One-time offline training
    - Based on experimental data
    - Not tied to applications, no need to retrain, no overfit

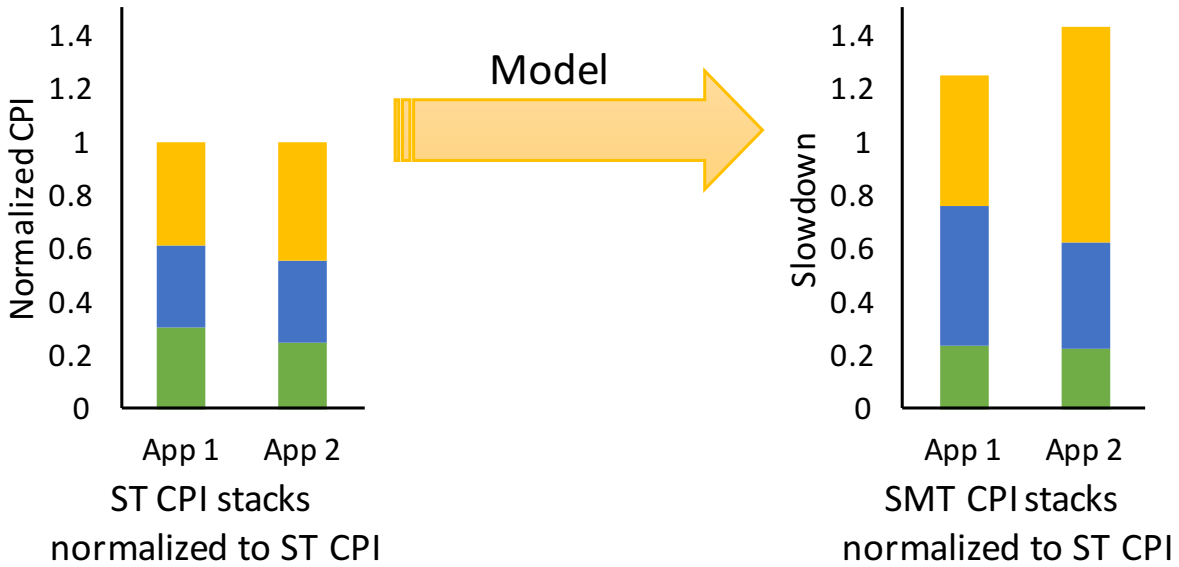


Model construction and slowdown estimation

- Model parameters determined by linear regression
  - One-time offline training
    - Based on experimental data
    - Not tied to applications, no need to retrain, no overfit

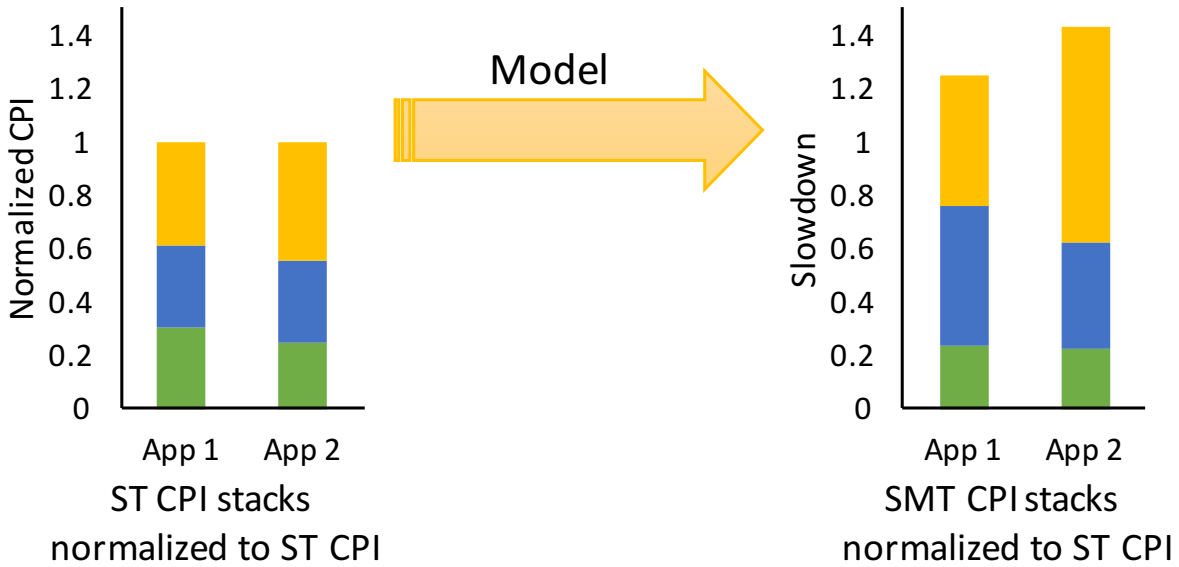


Obtaining ST CPI stacks in SMT mode



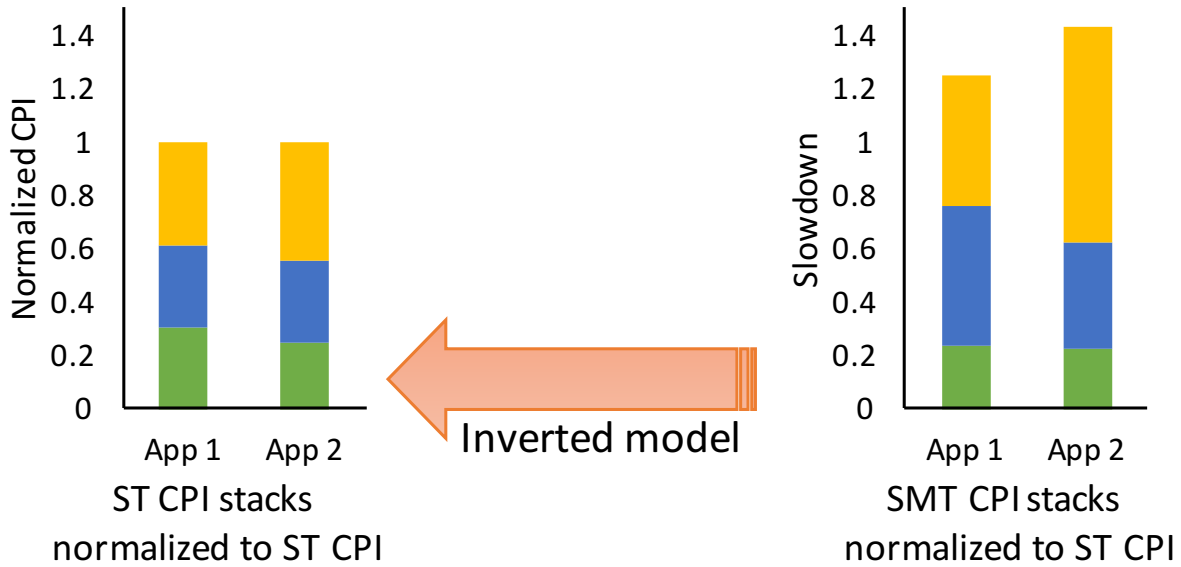
Obtaining ST CPI stacks in SMT mode

- **Obtaining the ST CPI stacks is not a trivial issue**
  - Offline profiling of CPI stacks (Impractical) ✗
  - Sampling CPI stacks at runtime (Overhead) ✗
  - Specific hardware to collect ST CPI stacks online (Unavailable) ✗



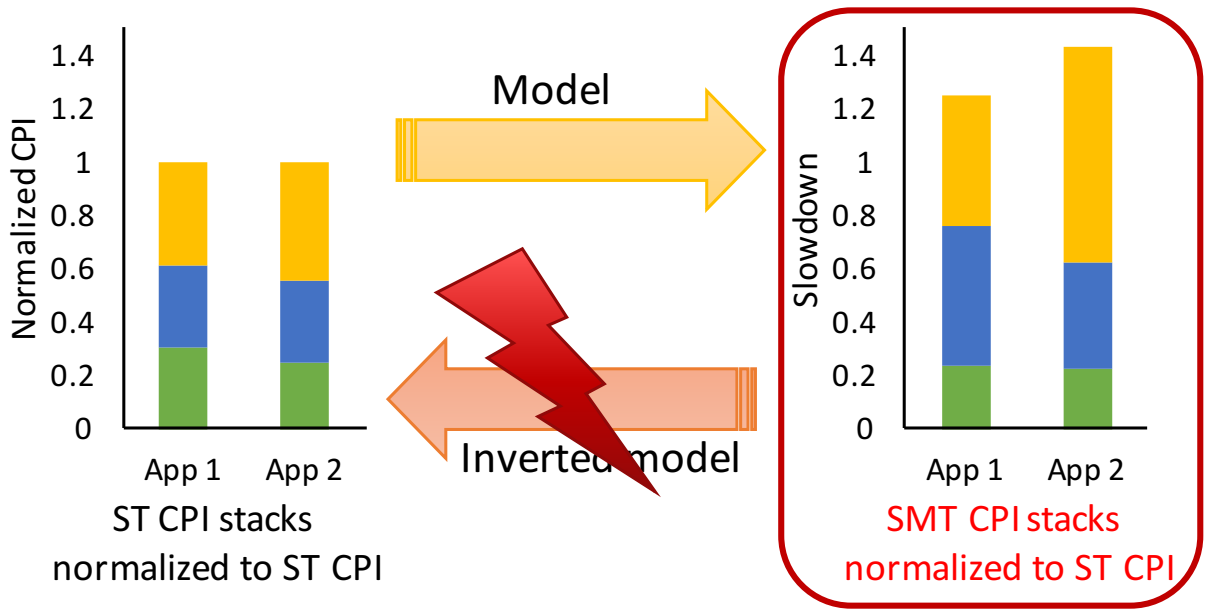
Obtaining ST CPI stacks in SMT mode

- **Obtaining the ST CPI stacks is not a trivial issue**
  - Offline profiling of CPI stacks (Impractical) ✗
  - Sampling CPI stacks at runtime (Overhead) ✗
  - Specific hardware to collect ST CPI stacks online (Unavailable) ✗
- Measure the SMT CPI stacks and **invert the model** to obtain ST CPI stacks



Obtaining ST CPI stacks in SMT mode

- **Obtaining the ST CPI stacks is not a trivial issue**
  - Offline profiling of CPI stacks (Impractical) ✗
  - Sampling CPI stacks at runtime (Overhead) ✗
  - Specific hardware to collect ST CPI stacks online (Unavailable) ✗
- Measure the SMT CPI stacks and **invert the model** to obtain ST CPI stacks
  - Not trivial: ST CPI not available in SMT execution
  - Solved with an approximate approach



Outline

Reduction of the cycle stack components

- 45 events form the full CPI stack of the the IBM POWER8
- 6 thread-level counters are implemented (4 programmable)
  - Structural conflicts on some events that cannot be measured together
  - 19 time slices required to build the full CPI stack
- **Unacceptable for scheduling**
  - Obtaining an updated CPI stack is not possible
- Fortunately, the CPI stack model is build hierarchically
  - Top level with 5 components
  - The model accuracy is reduced, but it has lower complexity and use updated CPI stacks

Correction factor

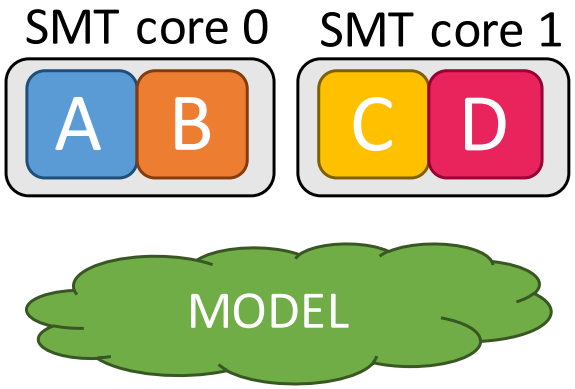
- The model is relatively accurate in general, but more inaccurate for particular schedules
  - Interference in the inter-core shared resources not directly modeled (e.g. LLC interference)

Correction factor

- The model is relatively accurate in general, but more inaccurate for particular schedules
  - Interference in the inter-core shared resources not directly modeled (e.g. LLC interference)
- **Correction factor**
  - $Cf = \frac{Measured\ \ slowdown}{Model\ \ slowdown}$

Correction factor

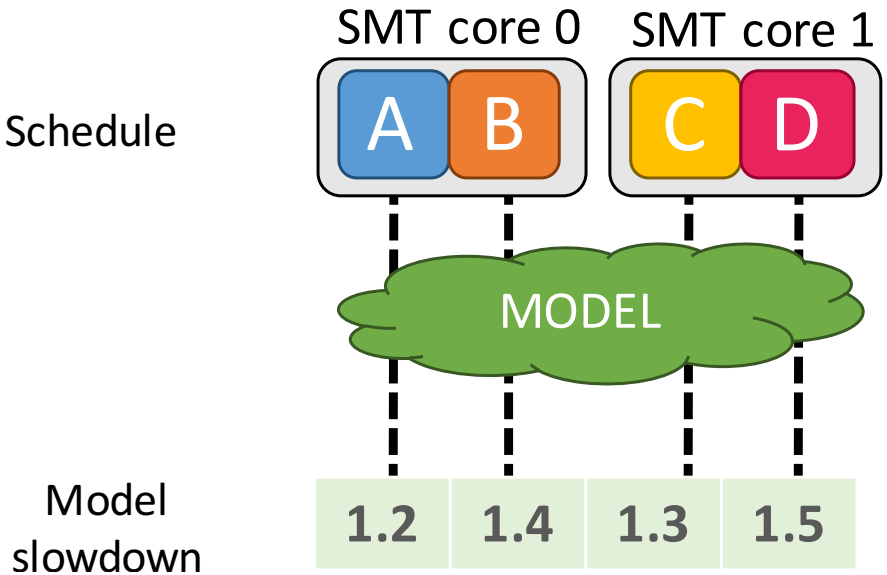
Schedule



Application \ Co-runner	A	B	C	D
A	-	1.0	1.0	1.0
B	1.0	-	1.0	1.0
C	1.0	1.0	-	1.0
D	1.0	1.0	1.0	-

$$Cf = \frac{Measured\ slowdown}{Model\ slowdown}$$

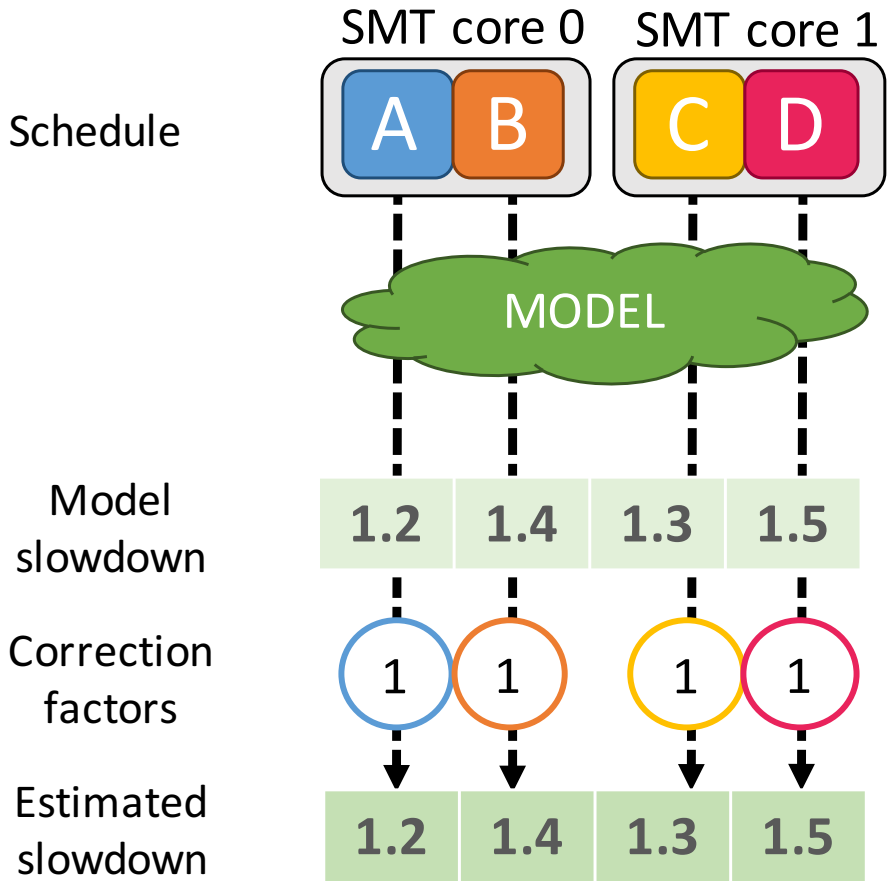
Correction factor



Application \ Co-runner	A	B	C	D
A	-	1.0	1.0	1.0
B	1.0	-	1.0	1.0
C	1.0	1.0	-	1.0
D	1.0	1.0	1.0	-

$$Cf = \frac{Measured\ slowdown}{Model\ slowdown}$$

Correction factor

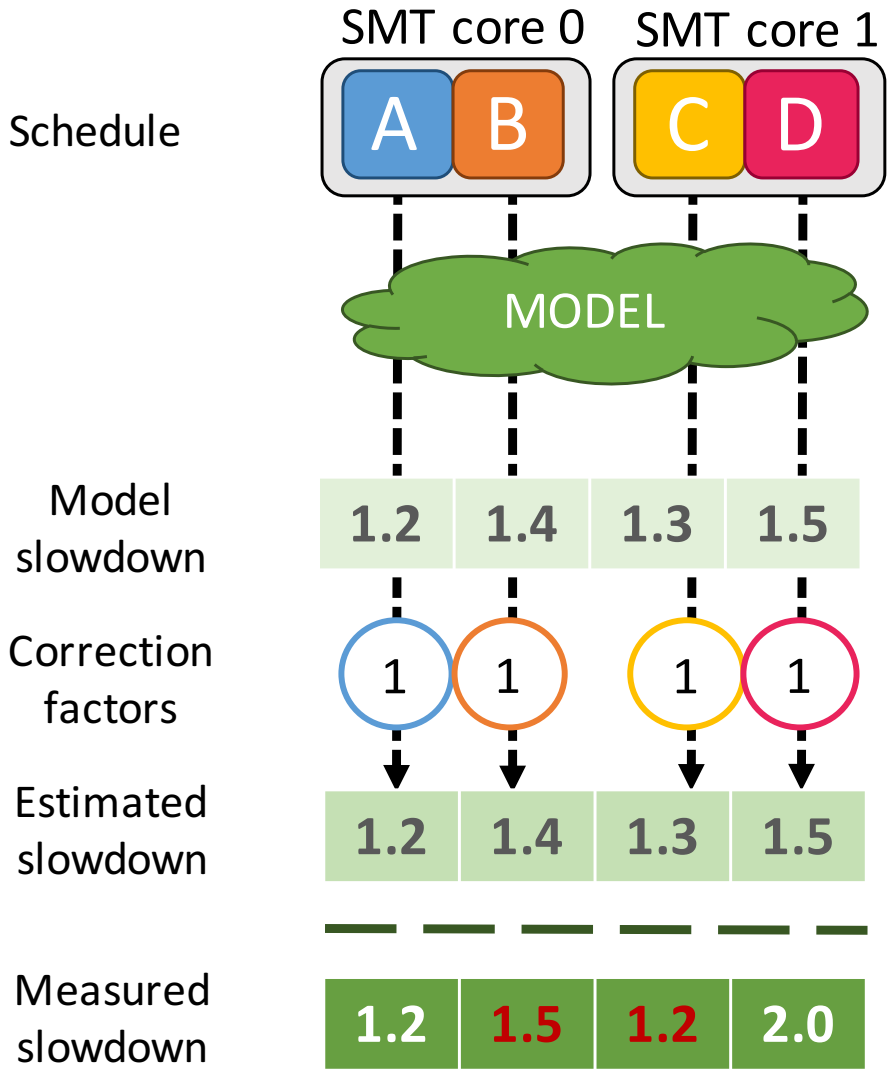


Co-runner

Application	A	B	C	D
A	-	1.0	1.0	1.0
B	1.0	-	1.0	1.0
C	1.0	1.0	-	1.0
D	1.0	1.0	1.0	-

$$Cf = \frac{Measured\ slowdown}{Model\ slowdown}$$

Correction factor

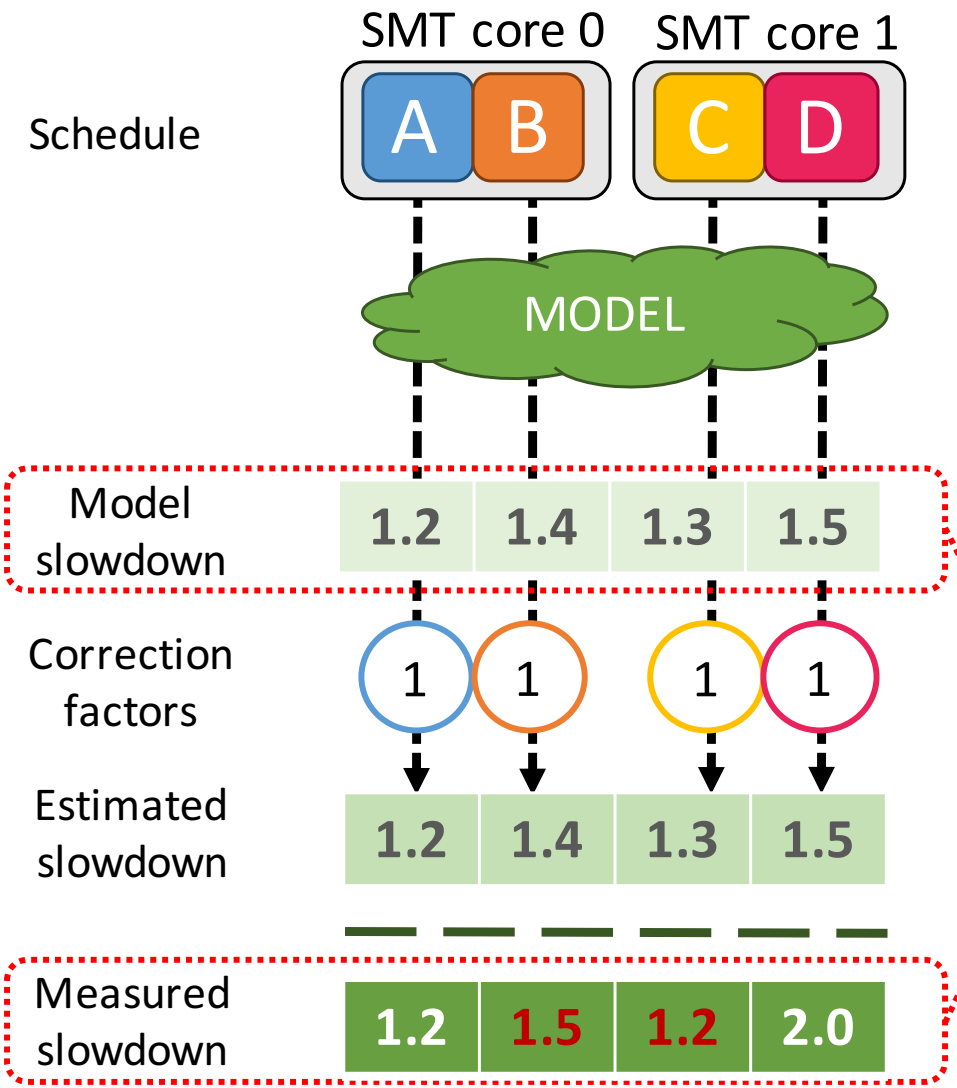


Co-runner

Application	A	B	C	D
A	-	1.0	1.0	1.0
B	1.0	-	1.0	1.0
C	1.0	1.0	-	1.0
D	1.0	1.0	1.0	-

$$Cf = \frac{Measured\ slowdown}{Model\ slowdown}$$

Correction factor

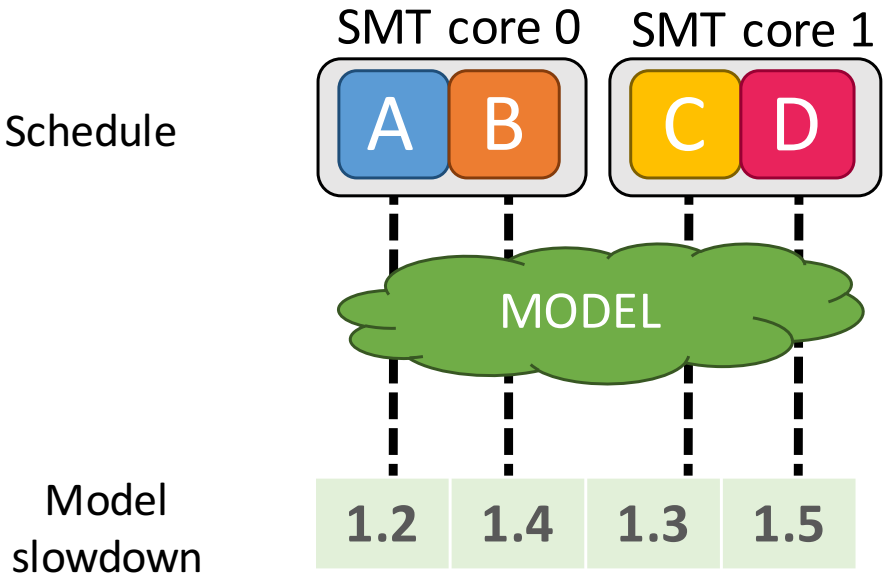


Co-runner

Application	A	B	C	D
A	-	1.0	1.0	1.0
B	1.1	-	1.0	1.0
C	1.0	1.0	-	0.9
D	1.0	1.0	1.0	-

$$Cf = \frac{Measured\ slowdown}{Model\ slowdown}$$

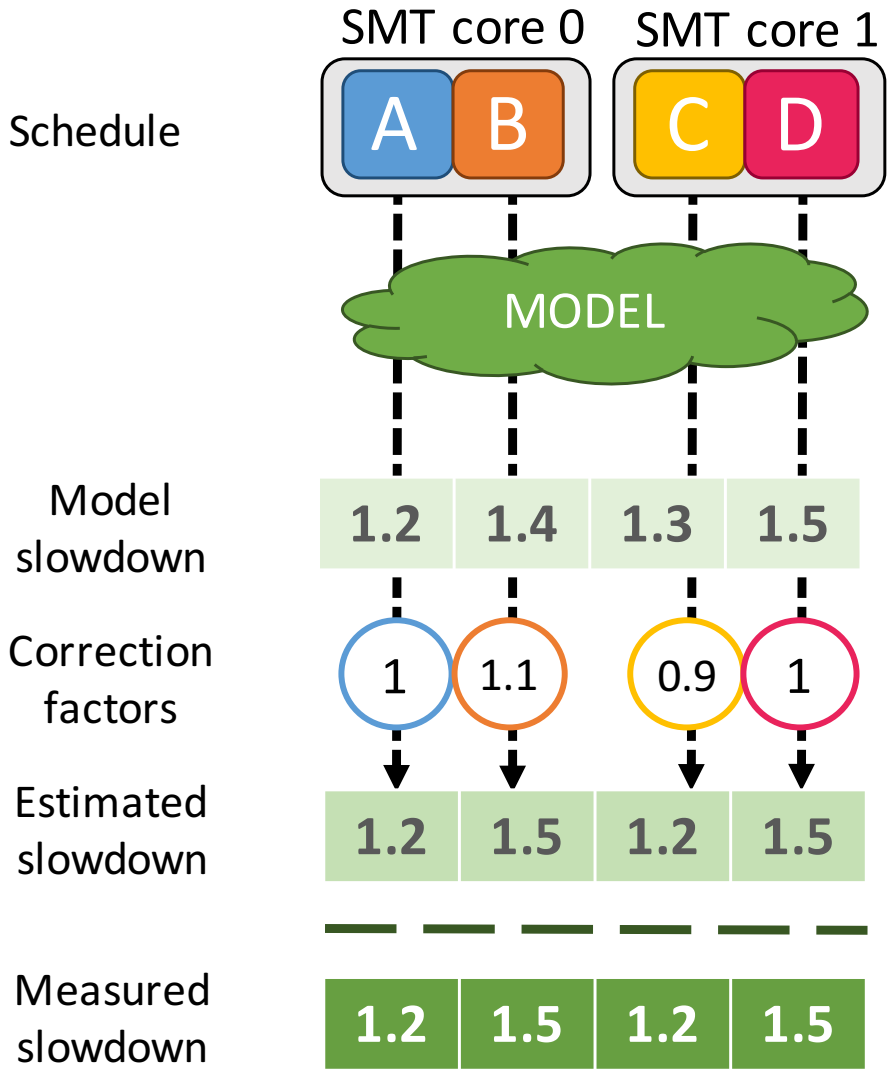
Correction factor



Application \ Co-runner	A	B	C	D
A	-	1.0	1.0	1.0
B	1.1	-	1.0	1.0
C	1.0	1.0	-	0.9
D	1.0	1.0	1.0	-

$$Cf = \frac{Measured\ slowdown}{Model\ slowdown}$$

Correction factor



Co-runner

Application	A	B	C	D
A	-	1.0	1.0	1.0
B	1.1	-	1.0	1.0
C	1.0	1.0	-	0.9
D	1.0	1.0	1.0	-

$$Cf = \frac{Measured\ slowdown}{Model\ slowdown}$$

Correction factor

- The model is relatively accurate in general, but more inaccurate for particular schedules
  - Interference in the inter-core shared resources not directly modeled (e.g. LLC interference)
- **Correction factor**
  - $Cf = \frac{Measured\ slowdown}{Model\ slowdown}$
  - Updated using an exponential moving average, to smooth out sudden changes

Correction factor

- The model is relatively accurate in general, but more inaccurate for particular schedules
  - Interference in the inter-core shared resources not directly modeled (e.g. LLC interference)
- **Correction factor**
  - $Cf = \frac{Measured\ slowdown}{Model\ slowdown}$
  - Updated using an exponential moving average, to smooth out sudden changes
- Requires knowledge of the isolated performance
  - Very sparsely run the applications in ST mode, incurring 0.2% overhead

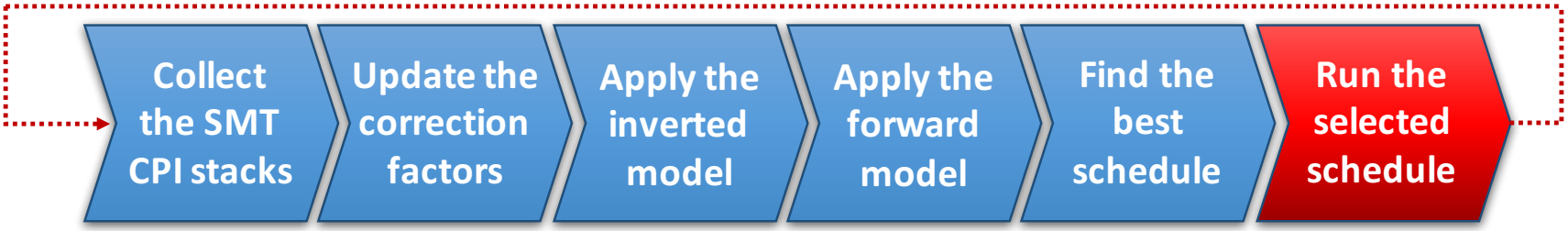
Selection of the optimal schedule

- Too large number of different schedules
  - $\frac{n!}{c!\left(\frac{n}{c}!\right)^c}$ n applications onto c cores
  - More than 2M schedules for 16 applications in 8 cores!

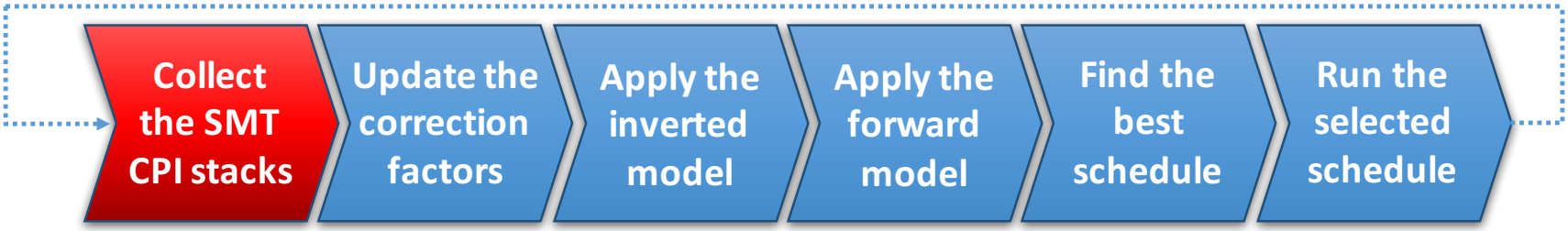
Selection of the optimal schedule

- Too large number of different schedules
  - $\frac{n!}{c!\left(\frac{n}{c}!\right)^c}$ n applications onto c cores
    - More than 2M schedules for 16 applications in 8 cores!
- **Modeled as a minimum-weight perfect matching problem**, that can be solved in polynomial time using the blossom algorithm

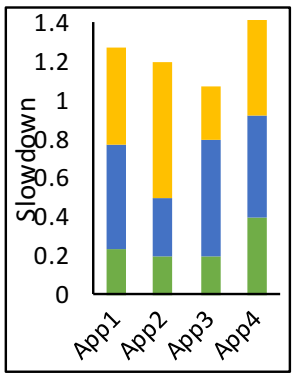
Scheduling steps



Scheduling steps



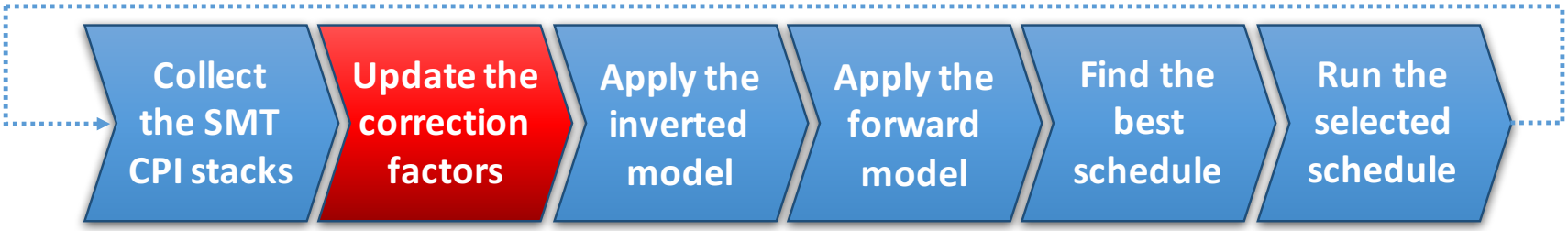
Measured SMT CPI stacks



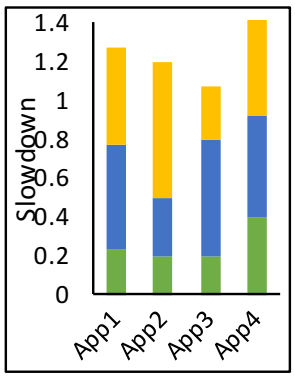
Normalized SMT CPI

1.3 1.2 1.1 1.4

Scheduling steps



Measured SMT CPI stacks

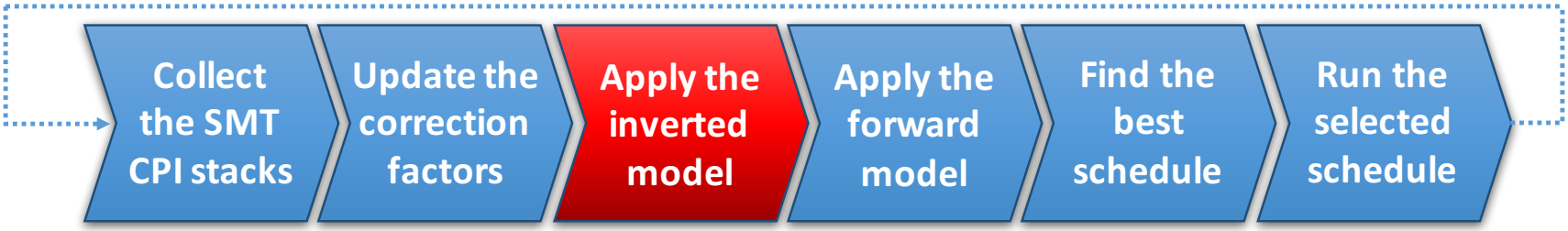


Update correction factors

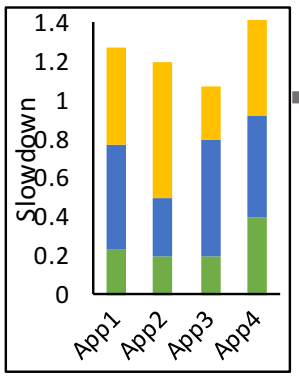
Normalized SMT CPI

	A	B	C	D
A	-	1.0	1.0	1.0
B	0.9	-	1.0	1.0
C	1.0	1.0	-	0.8
D	1.0	1.0	1.0	-

Scheduling steps

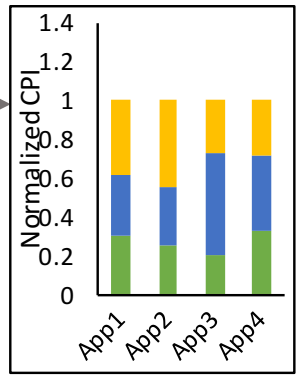


Measured SMT CPI stacks



Inverted model

Estimated ST CPI stacks

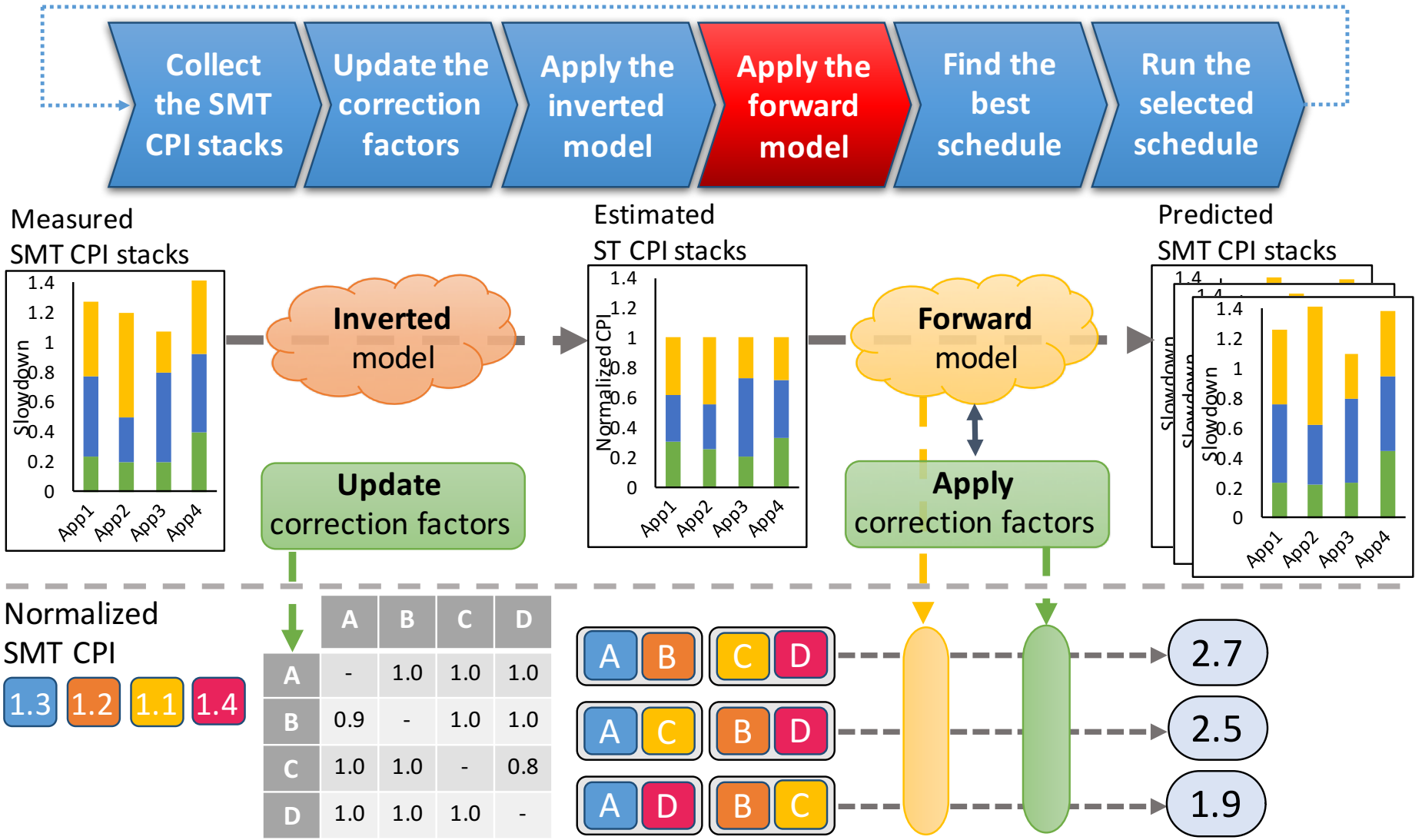


Update correction factors

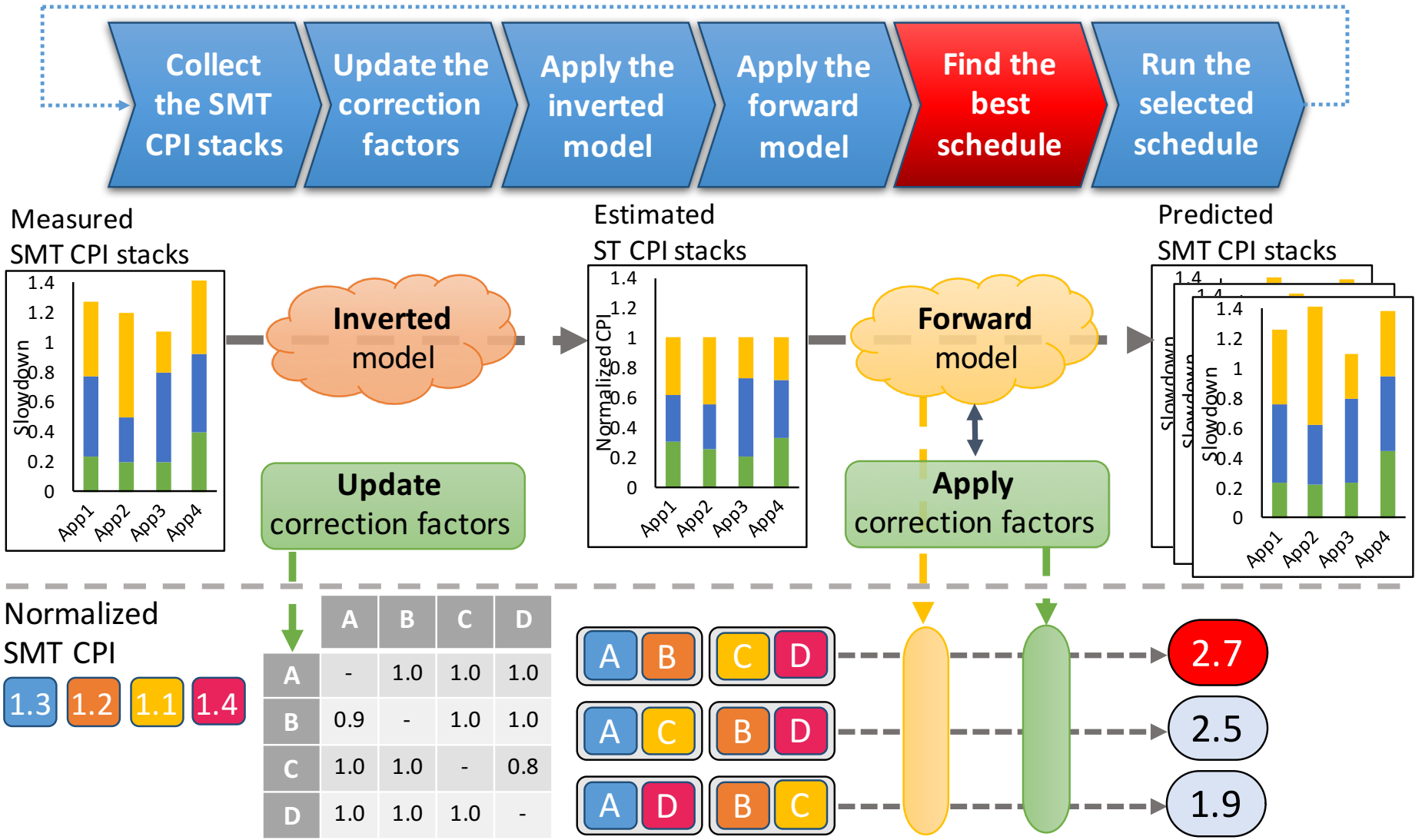
Normalized SMT CPI

	A	B	C	D
A	-	1.0	1.0	1.0
B	0.9	-	1.0	1.0
C	1.0	1.0	-	0.8
D	1.0	1.0	1.0	-

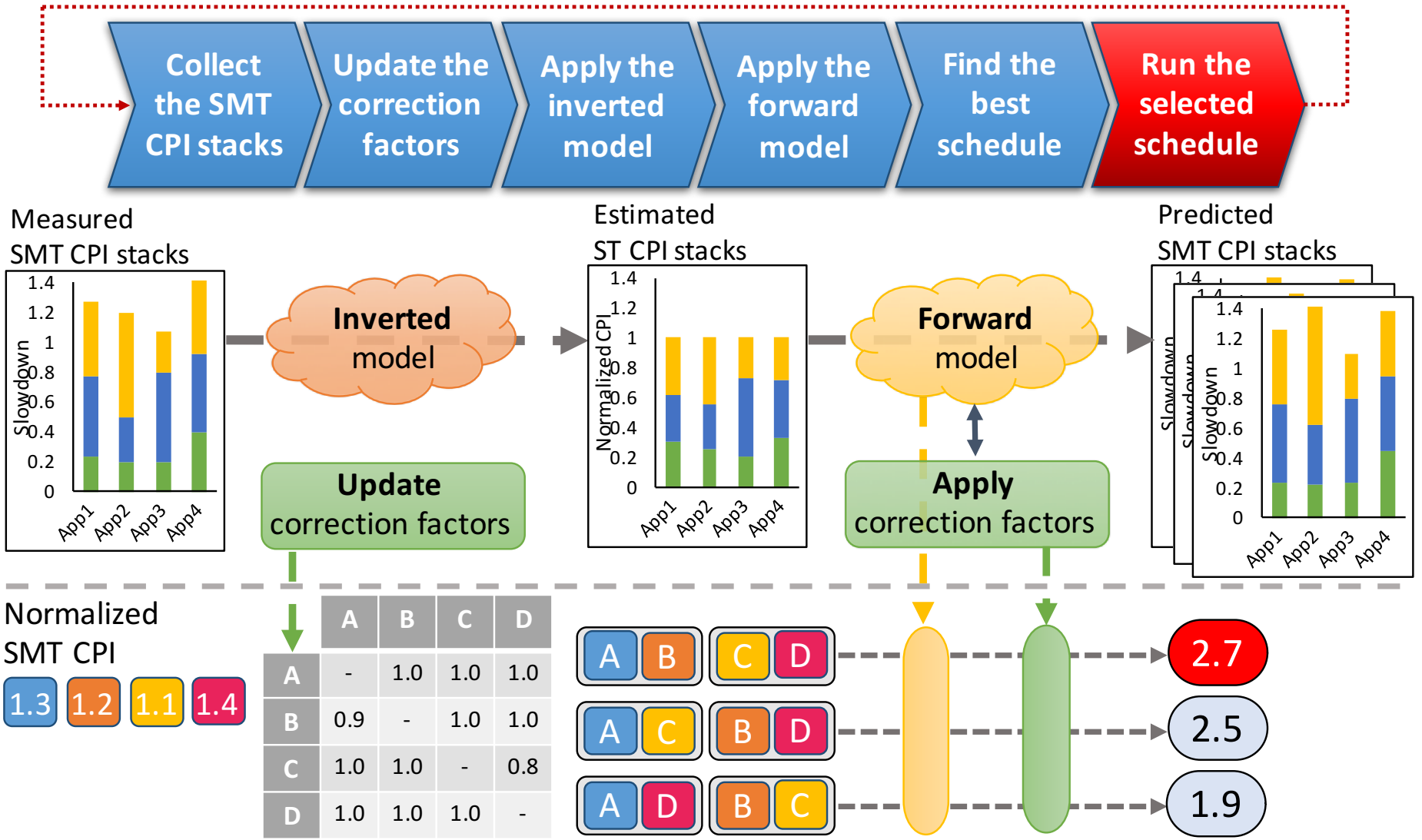
Scheduling steps



Scheduling steps



Scheduling steps



Outline

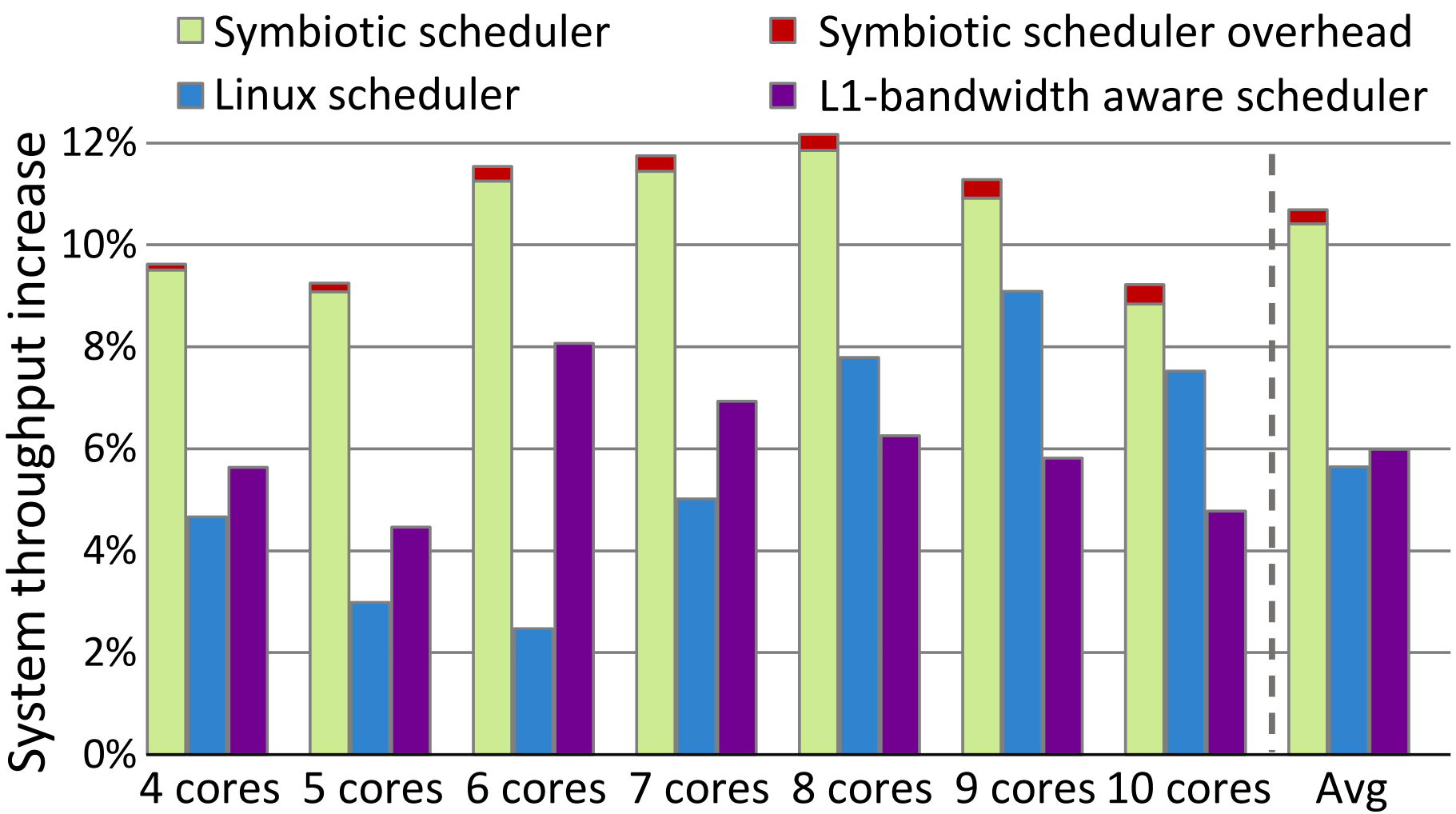
Experimental setup

- 10-core IBM POWER8
- SPEC CPU2006 benchmarks (reference input set)
- 105 multiprogram workloads
  - From 8-application combinations on 4 cores to 20-application combinations on 10 cores
- Metrics:
  - System throughput (STP), by means of the weighted speedup metric
  - System fairness, $Unfairness = \frac{Max\ Slowdown_i}{Min\ Slowdown_j}\ \forall\{i,j\} \in \{1,N\}$

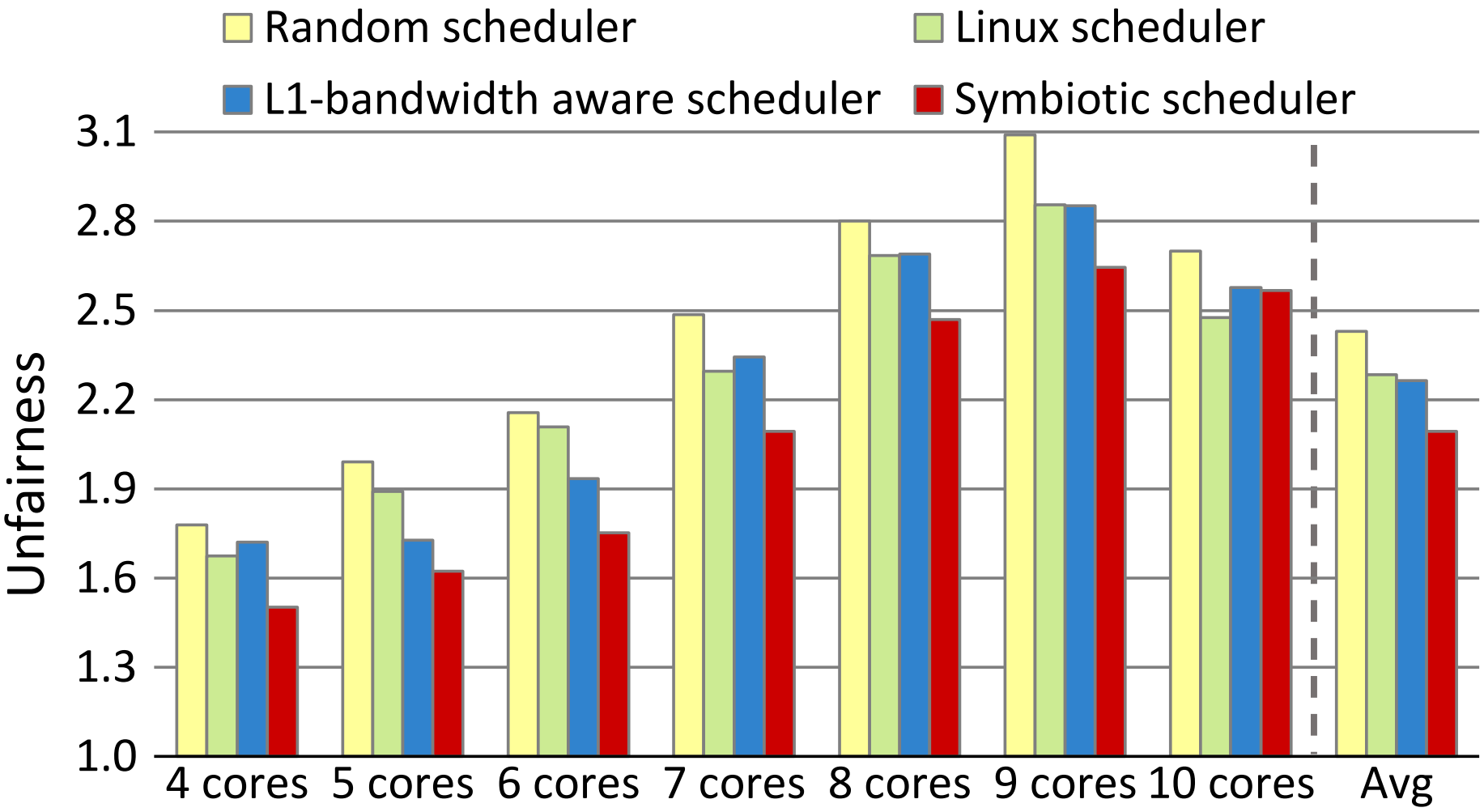
Experimental setup

- 10-core IBM POWER8
- SPEC CPU2006 benchmarks (reference input set)
- 105 multiprogram workloads
  - From 8-application combinations on 4 cores to 20-application combinations on 10 cores
- Metrics:
  - System throughput (STP), by means of the weighted speedup metric
  - System fairness, $Unfairness = \frac{Max\ Slowdown_i}{Min\ Slowdown_j}\ \forall\{i,j\} \in \{1,N\}$
- **Four schedulers are compared:**
  - Random
  - Linux, default Completely Fair Scheduler (CFS)
  - L1-bandwidth aware scheduler, which balances the L1 bandwidth utilization among cores. *Feliu et al., PACT'13*
  - Symbiotic scheduler

System throughput increase

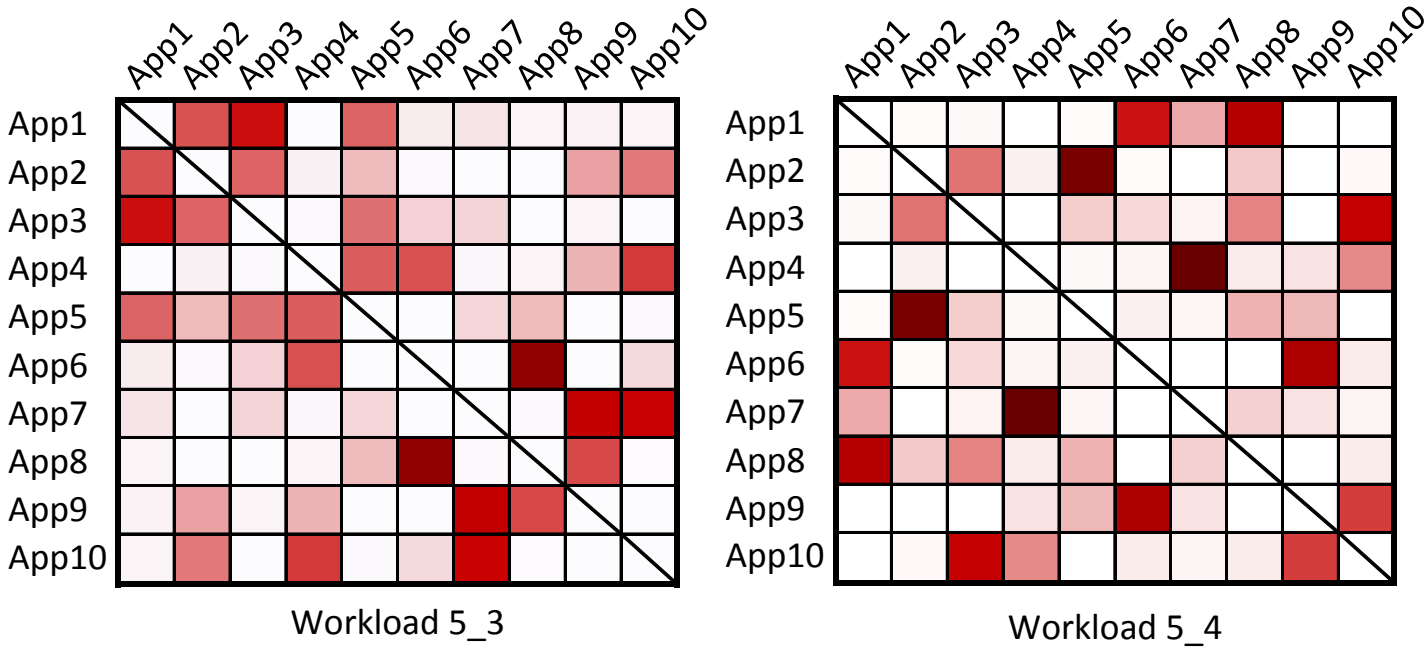


System unfairness



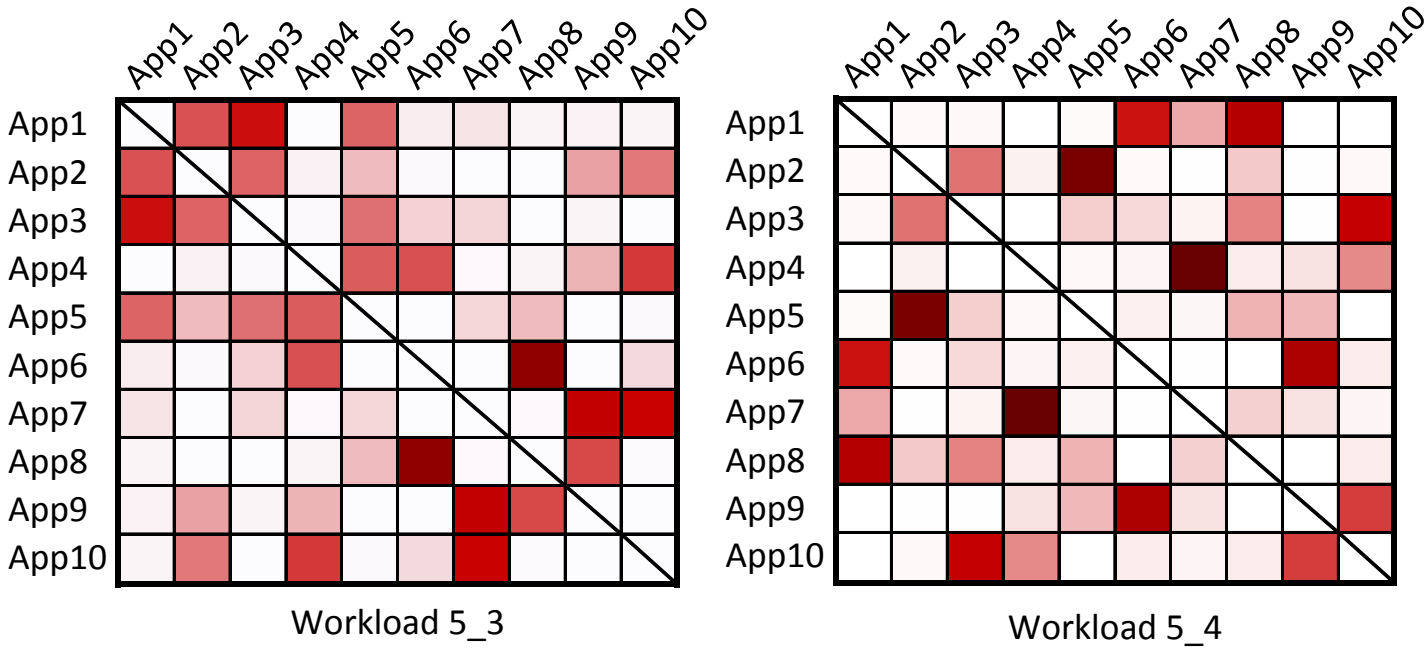
Unfairness is a lower is better metric

Symbiosis patterns



- Frequency matrices of the job co-schedules
  - The darker the color the more frequently the couple is scheduled together on a SMT core

Symbiosis patterns



- Frequency matrices of the job co-schedules
  - The darker the color the more frequently the couple is scheduled together on a SMT core
  - In workload 5_4 two couples are scheduled very frequently ($> 65\%$) $\Rightarrow$ High symbiosis
  - In workload 5_3 there is not that predominant couple $\Rightarrow$ High phase behavior

Outline

Conclusions

- Scheduling has considerable impact on the performance of SMT multicores
- **Novel symbiotic job scheduler for SMT multicores**
  - Quick estimation of the performance of schedules to select the optimal one
  - Using CPI stacks can quickly adapt to phase behavior
  - No need of additional hardware nor sampling schedules
- Improve the system throughput of the random and Linux schedulers, on average, by 10.3% and 4.7%

Symbiotic Job Scheduling on the IBM POWER8

J. Feliu[1] S. Eyerman[2] J. Sahuquillo[1] S. Petit[1]

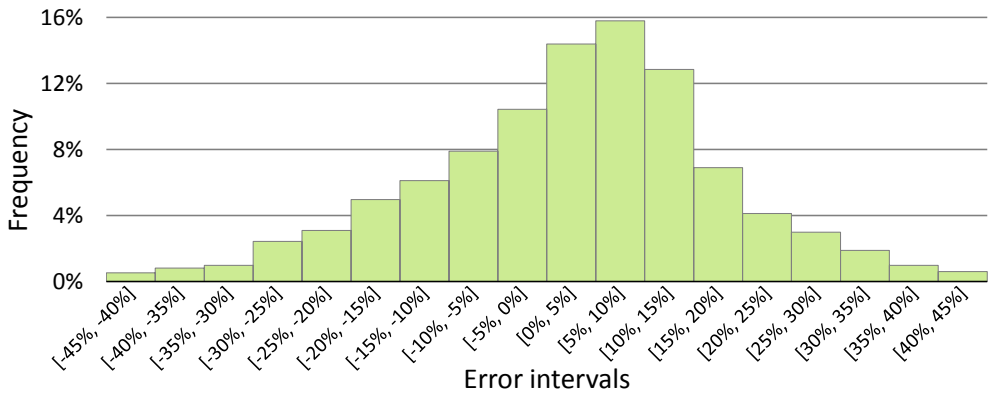
[1]Department of Computing Engineering (DISCA)
Universitat Politècnica de València
jofepre@gap.upv.es, {jsahuqui,spetit}@disca.upv.es

[2]Intel Belgium
stijn.eyerman@intel.com

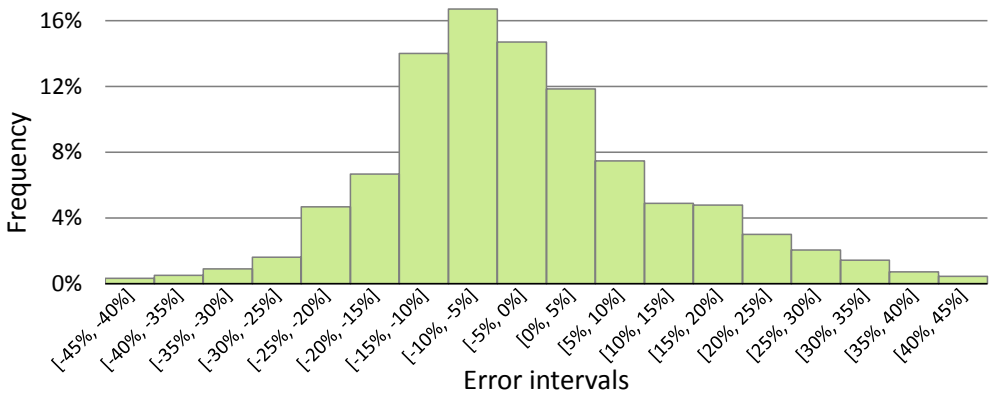
March 16th, 2016

[2] *This work was done while Stijn Eyerman was at Ghent University*

Model accuracy



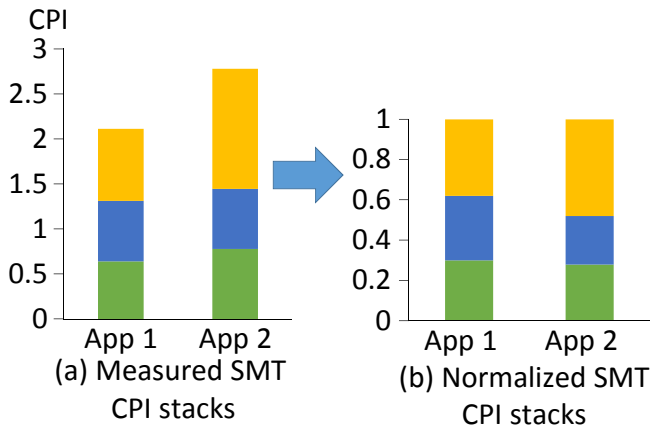
- Distribution of error from ST CPI stacks alone to SMT CPI stacks together
- Average absolute error: 12.3%

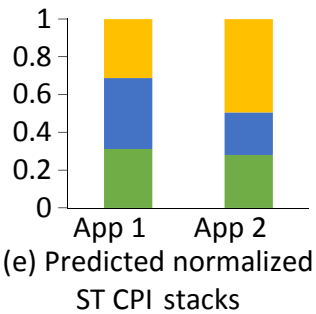


- Distribution of error from SMT CPI stacks together to ST CPI stacks alone
- Average absolute error: 13.4%

Obtaining ST CPI stacks in SMT mode

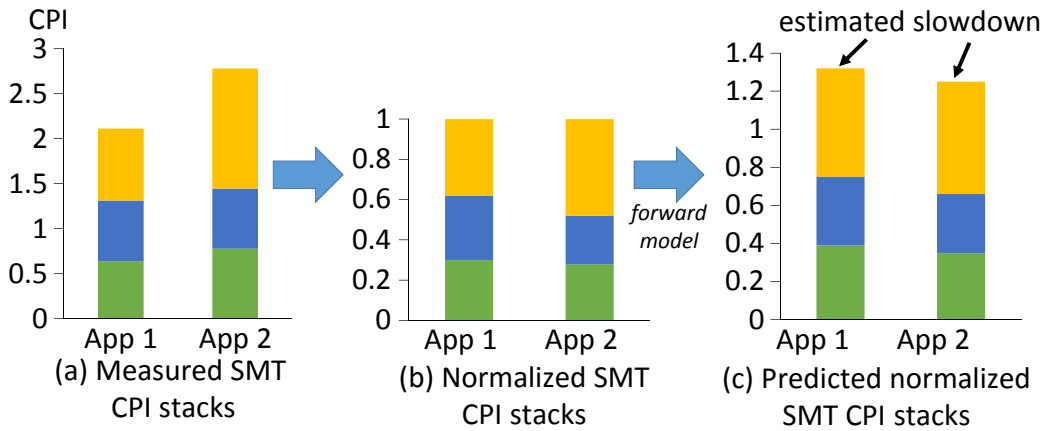
- Approximate approach
  - SMT components normalized to SMT CPI $\approx$ ST components normalized to ST CPI
    - Both add to one
    - If the relative increase of the components is the same, then both stacks are equal
  - However, it is not accurate enough





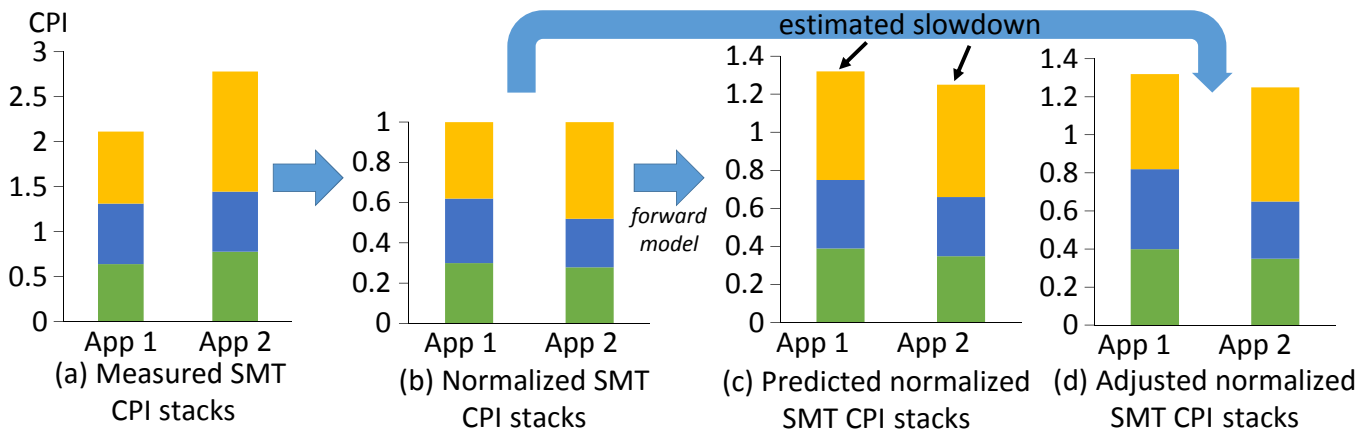
Obtaining ST CPI stacks in SMT mode

- Approximate approach
  - SMT components normalized to SMT CPI $\approx$ ST components normalized to ST CPI
  - Estimate the slowdown applying the model to the estimated normalized ST CPI



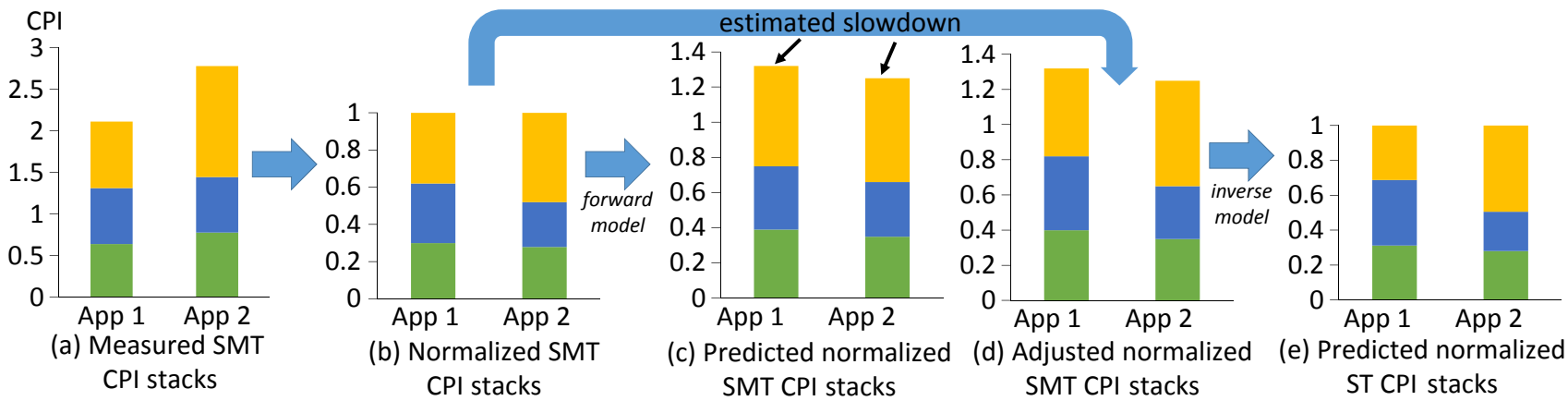
Obtaining ST CPI stacks in SMT mode

- Approximate approach
  - SMT components normalized to SMT CPI $\approx$ ST components normalized to ST CPI
  - Estimate the slowdown applying the model to the estimated normalized ST CPI
  - Renormalize the measured SMT CPI stacks using the estimated slowdown

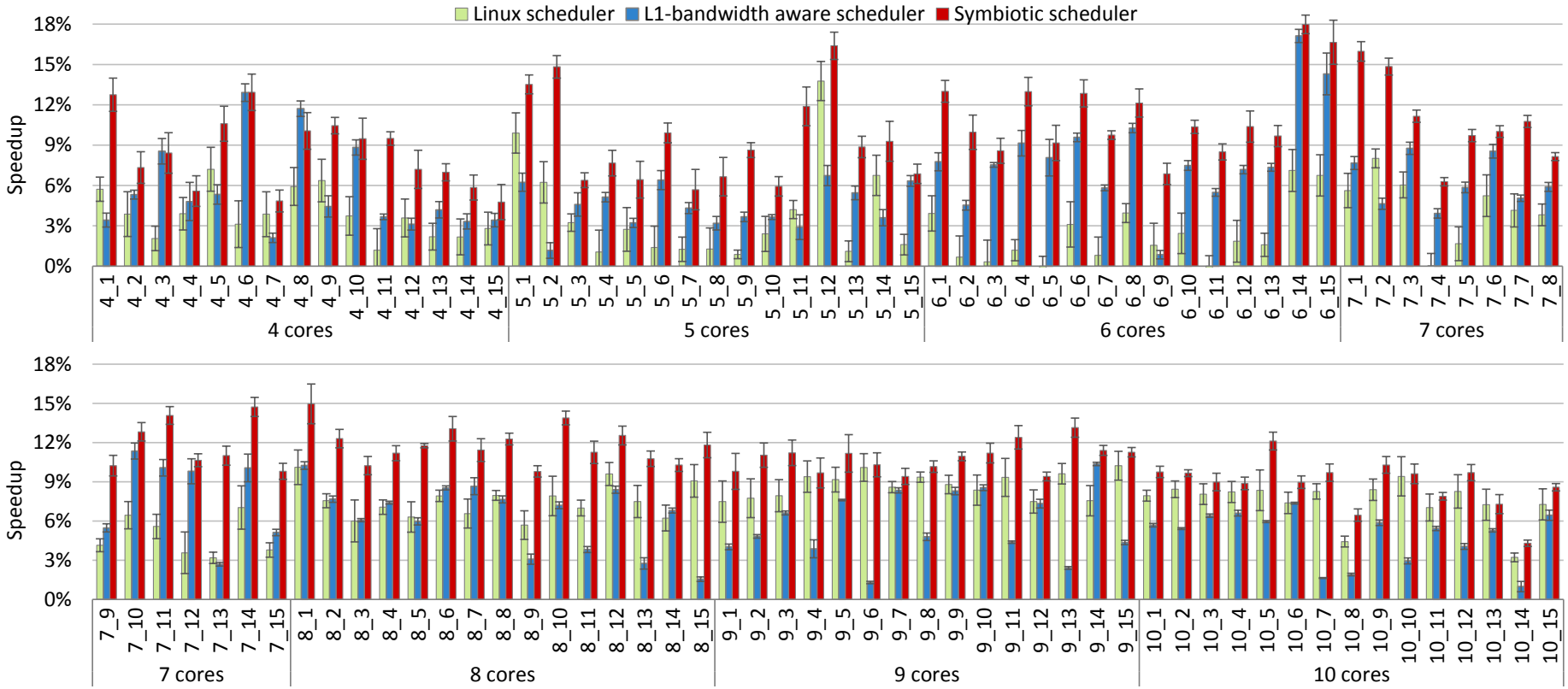


Obtaining ST CPI stacks in SMT mode

- Approximate approach
  - SMT components normalized to SMT CPI $\approx$ ST components normalized to ST CPI
  - Estimate the slowdown applying the model to the estimated normalized ST CPI
  - Renormalize the measured SMT CPI stacks using the estimated slowdown
  - Apply the inverse model to obtain new estimates for the ST CPI stacks



System throughput performance



Core asymmetry