A Fast Heuristic Scheduling Algorithm for Periodic ConcurrenC Models

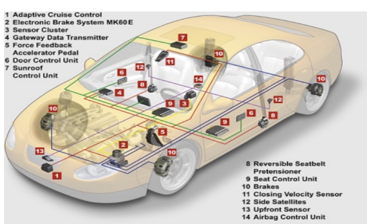
Weiwei Chen and Rainer Doemer
Center for Embedded Computer Systems
University of California, Irvine

Outline

- Introduction and Related Work
- ConcurrenC Model Simulation
- Periodic ConcurrenC Scheduling
  - Scheduling of SDF-like Models
  - Scheduling of Periodic ConcurrenC Models
- Experiment Results
- Conclusion and Future Work

Introduction

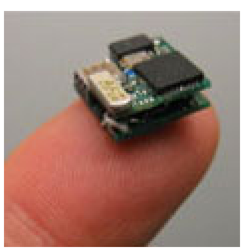
- Embedded systems are modeled and described at different levels of abstraction.
  - ***System-level Description Languages*** (SLDLs), like SystemC and SpecC are used for modeling and verification, and are simulated by **discrete event** (DE) simulation mechanism.
  - ***C-based SLDLs*** are well-defined, but ***Modeling*** is not.
    - "How to write a good model?"
    - Modeling guidelines are needed.
  - Simulation speed is not fast for ***DE-based*** single-thread kernels.



Source: www.aa1car.com

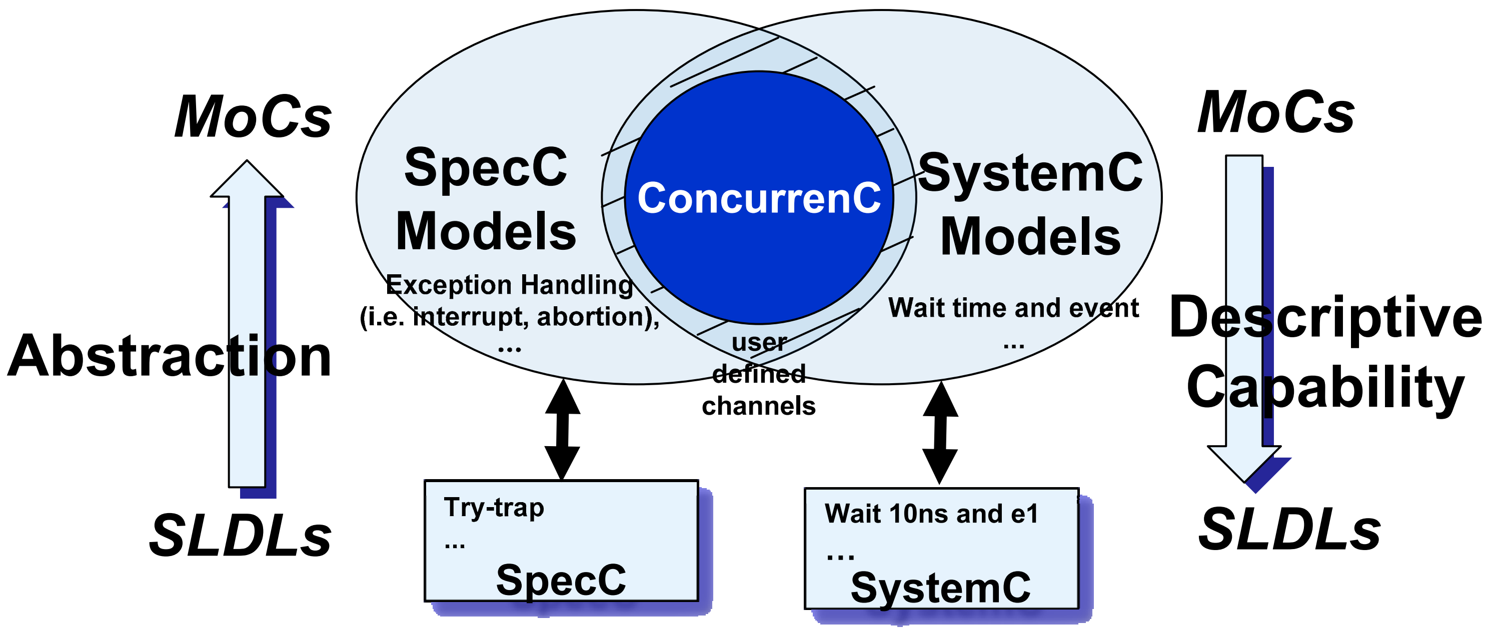
Source: www.nintendo.com

Source: www.apple.com


Source: P. Chou, UCI

ConcurrenC MoC

- ***ConcurrenC,*** a new model of computation (MoC) was proposed to emphasize the importance of the modeling.
  – A specific subset, called ***periodic ConcurrenC***, can be statically scheduled to improve the simulation speed.

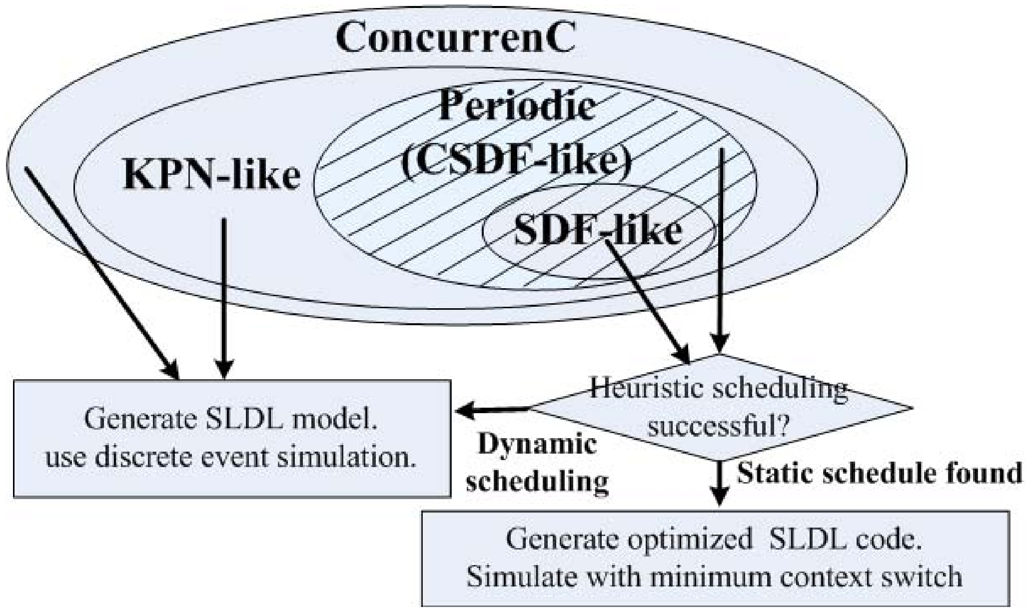


<u>Relationship between ConcurrenC and C-based SLDLs</u>

Related Work

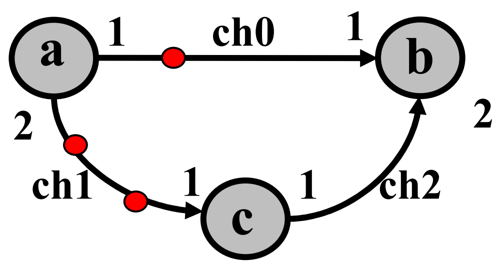
- System-level Description Languages
  – SystemC [T.Grotker 2002]
  – SpecC[D.D.Gajski 2000]
  – Discrete Event (DE) simulation [N.Savoiu 2002] [J.Zhu 2001]
- Model of Computation
  – Kahn Process Network (KPN) [G. Kahn 1974]
  – Synchronous Dataflow (SDF) [E.A.Lee 1987]
  – Cyclo-static Dataflow (CSDF) [G.Bilsen 1995]
  – ConcurrenC [W.Chen, R.Doemer, 2009]
- Static Scheduling
  – SDF [E.A.Lee 1987], CSDF [G.Bilsen 1994]
    - Builds an incidence matrix of the connected model graph
    - Solves a balance equation to get repetition vectors, and periodic admissible sequential schedule (PASS)
  – Model simulation in SLDL by modifying the simulation kernel [H.Patel 2004, 2005]
  – Analysis tool SDF3 for throughput, storage capacity, and buffer size minimization(NP-complete [S.S.Battacharyya 1996]) [S.Stuijk 2006, 2008]

ConcurrenC Model Simulation



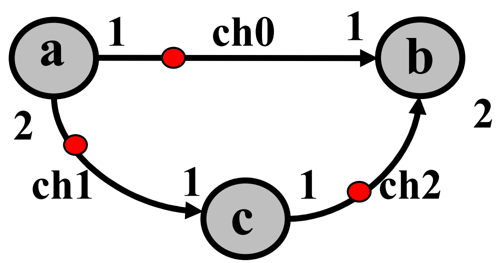
- **Simulation Strategy**
  - Periodic ConcurrenC Model
    - Not input dependent, KPN-like, Periodically Schedulable
    - static schedulable ➔ **optimized implementation**
    - Otherwise ➔**Generic SLDL implementation, DE simulated**
  - Non-Periodic ConcurrenC Model ➔**Generic SLDL implementation, DE simulated**

ConcurrenC Examples



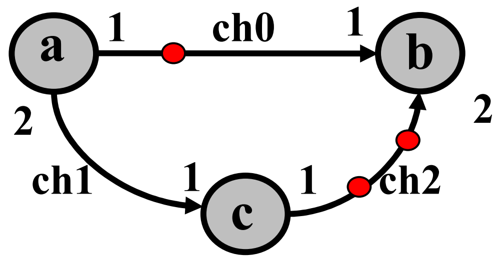
A SDF-like ConcurrenC example

ConcurrenC Examples



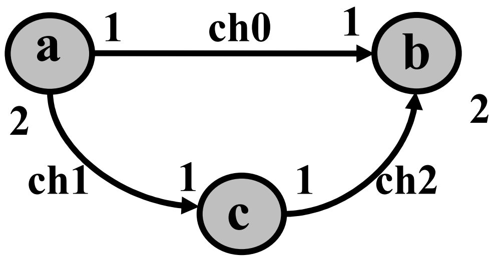
A SDF-like ConcurrenC example

ConcurrenC Examples



A SDF-like ConcurrenC example

ConcurrenC Examples



A SDF-like ConcurrenC example

The Periodic Admissible Sequential Schedule (PASS) is: a, c, c, b

Optimization for Fast Simulation

SpecC description for the SDF-like model

```
behavior a(i_int_sender ch0,
           i_int_sender ch1)
behavior b(i_int_receiver ch0,
           i_int_receiver ch2)

behavior Main(void)
{
   c_int_queue ch0(1ul), ch1(2ul), ch2(2ul);
   a b_a(ch0, ch1);
   b b_b(ch0, ch2);
   c b_c(ch1, ch2);

   int main(int argc, char** argv)
   {
      par
      {
               b_a.main();
               b_b.main();
               b_c.main();
      }
      return 0;
   }
};
```

thread 0 → int main(int argc, char** argv)
thread 1 → b_a.main();
thread 2 → b_b.main();
thread 3 → b_c.main();

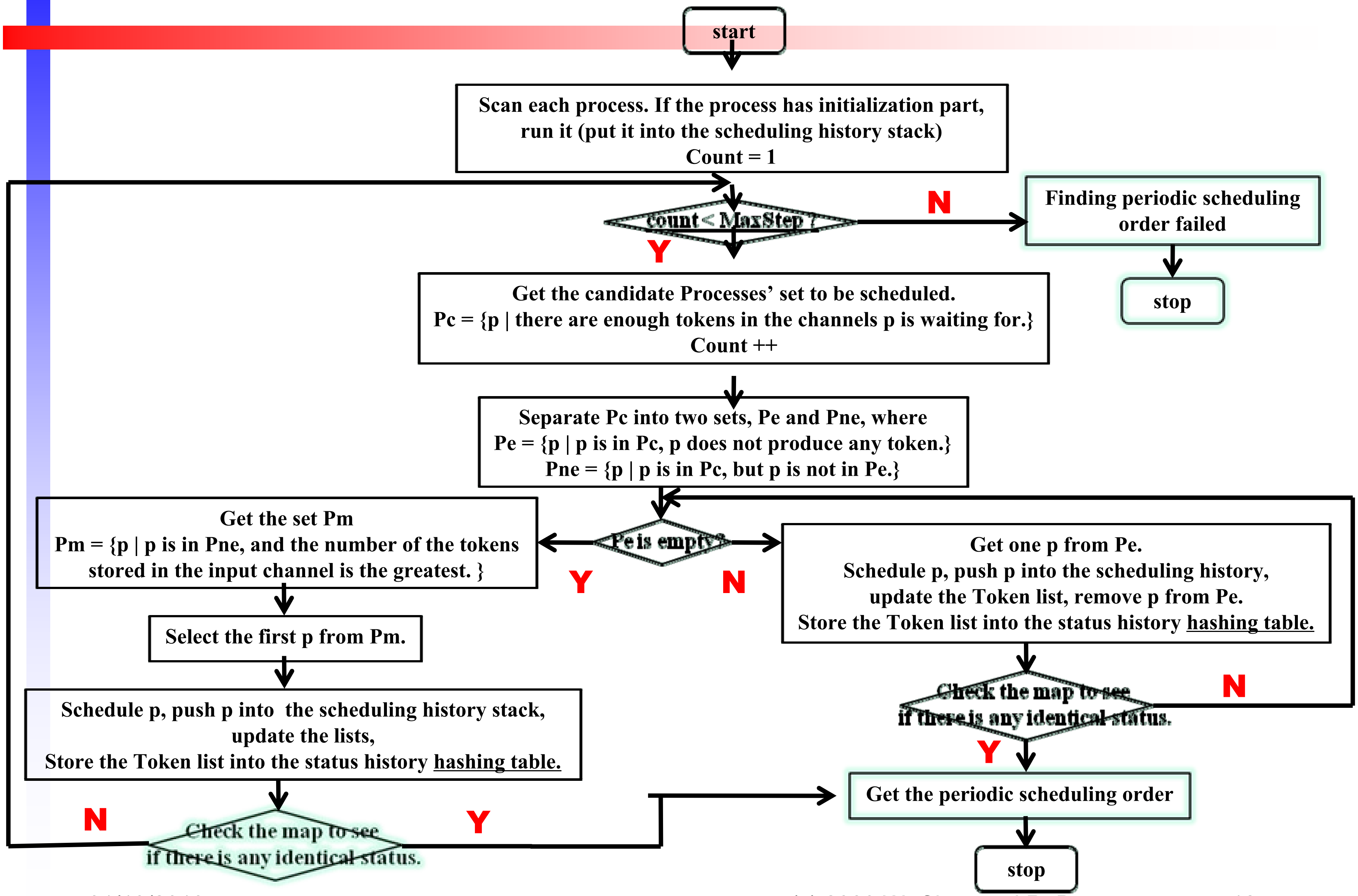
Optimized SpecC description for the SDF-like model

```
behavior a_seq(i_int_sender ch0,
               i_int_sender ch1)
behavior b_seq(i_int_receiver ch0,
               i_int_receiver ch2)

behavior Main(void)
{
   c_int_queue ch0(1ul), ch1(2ul), ch2(2ul);
   a_seq b_a(ch0, ch1);
   b_seq b_b(ch0, ch2);
   c_seq b_c(ch1, ch2);

   int main(int argc, char** argv)
   {
      while(1)
      {
               b_a.main();
               b_c.main();
               b_c.main();
               b_b.main();
      }
      return 0;
   }
};
```

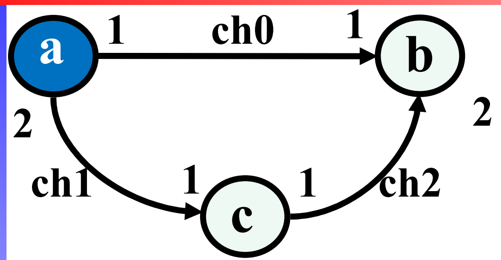
thread 0 → int main(int argc, char** argv)

PASS is (a, c, c, b)

Scheduling Algorithm for SDF-like Models



SDF-like Model Scheduling Example



STEP	

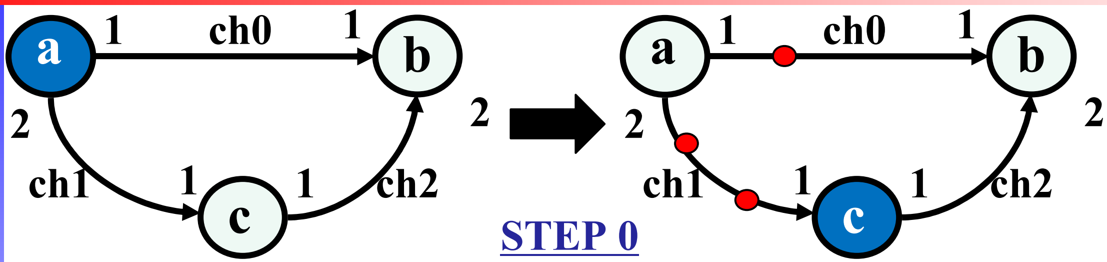
List of the number of tokens in each channel (Token[ch])

ch0	0
ch1	0
ch2	0

List of the channels that each process is waiting for (the number of tokens required) chWait[proc][ch]

a	-
b	ch0(1)&ch2(2)
c	ch1(1)

SDF-like Model Scheduling Example



STEP		0

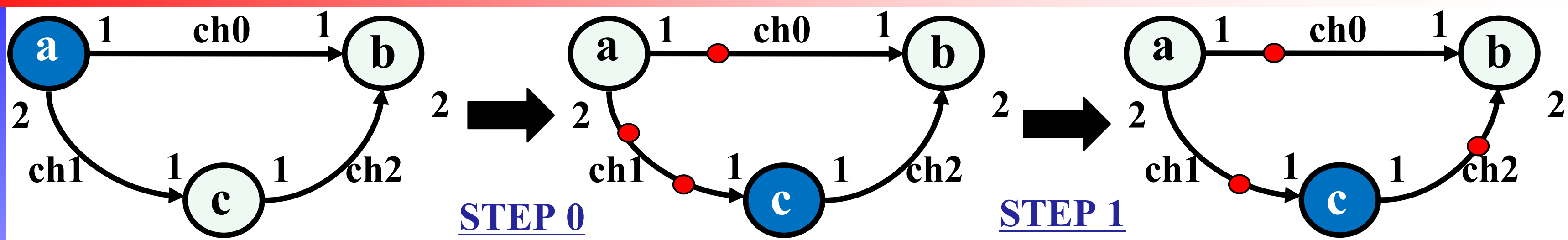
List of the number of tokens in each channel (Token[ch])

		a
ch0	0	1
ch1	0	2
ch2	0	0

List of the channels that each process is waiting for (the number of tokens required) chWait[proc][ch]

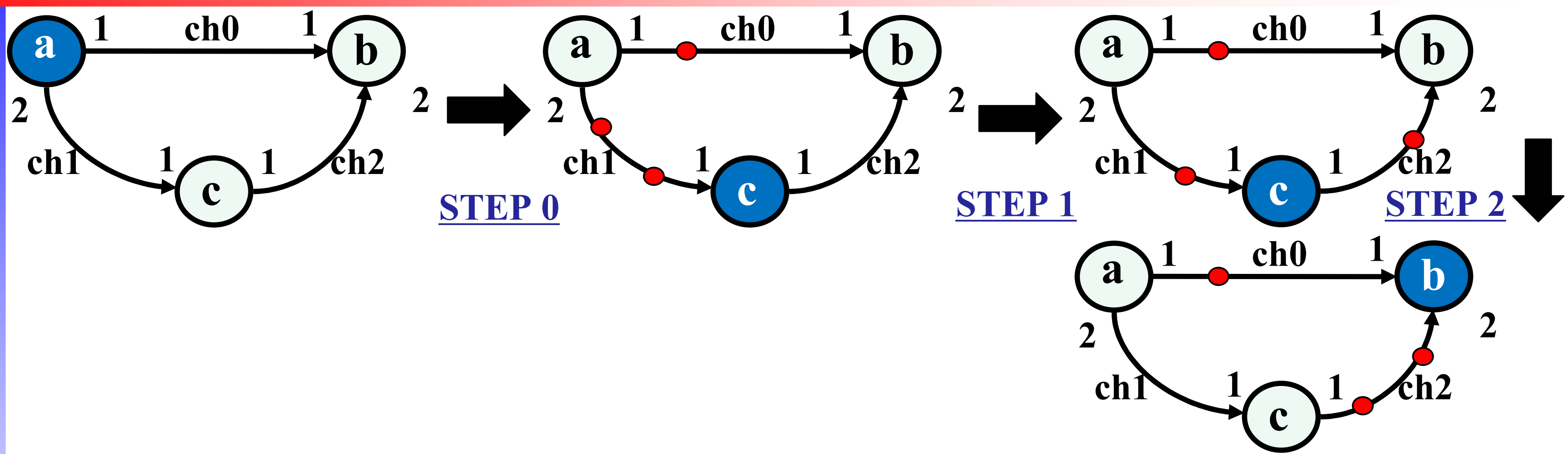
a	-	-
b	ch0(1)&ch2(2)	ch0(1)&ch2(2)
c	ch1(1)	ch1(1)

SDF-like Model Scheduling Example



STEP		0	1
List of the number of tokens in each channel (Token[ch])			
		a	c
ch0	0	1	1
ch1	0	2	1
ch2	0	0	1
List of the channels that each process is waiting for (the number of tokens required) chWait[proc][ch]			
a	-	-	-
b	ch0(1)&ch2(2)	ch0(1)&ch2(2)	ch0(1)&ch2(2)
c	ch1(1)	ch1(1)	ch1(1)

SDF-like Model Scheduling Example



STEP		0	1	2

List of the number of tokens in each channel (Token[ch])

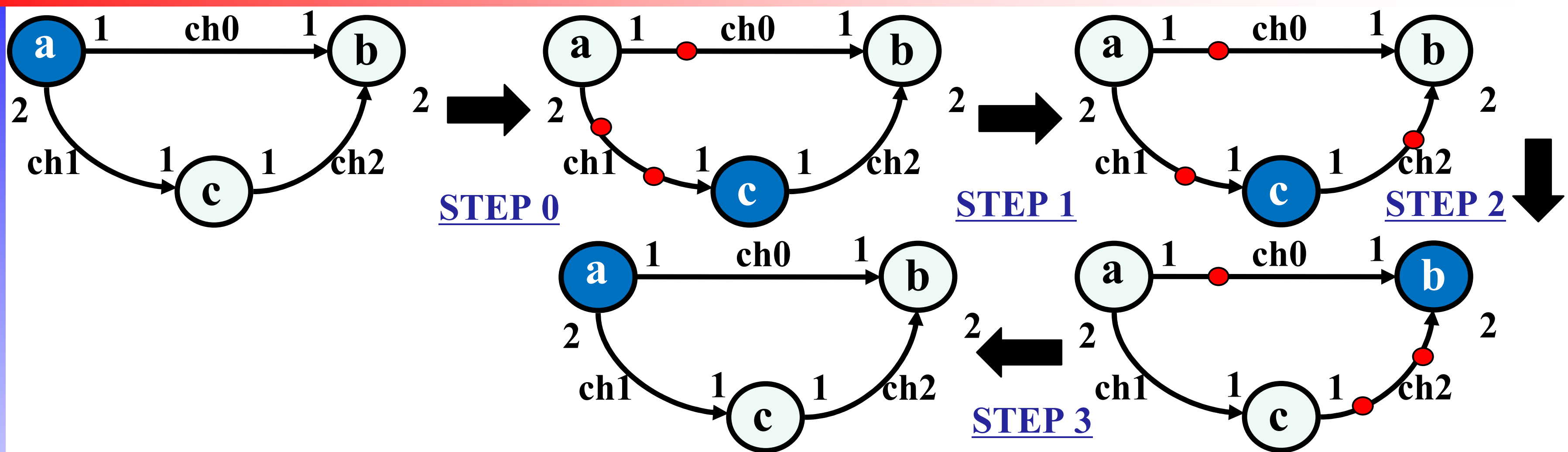
		a	c	c
ch0	0	1	1	1
ch1	0	2	1	0
ch2	0	0	1	2

List of the channels that each process is waiting for (the number of tokens required)

chWait[proc][ch]

a	-	-		-
b	ch0(1)&ch2(2)	ch0(1)&ch2(2)	ch0(1)&ch2(2)	ch0(1)&ch2(2)
c	ch1(1)	ch1(1)	ch1(1)	ch1(1)

SDF-like Model Scheduling Example



STEP		0	1	2	3

List of the number of tokens in each channel (Token[ch])

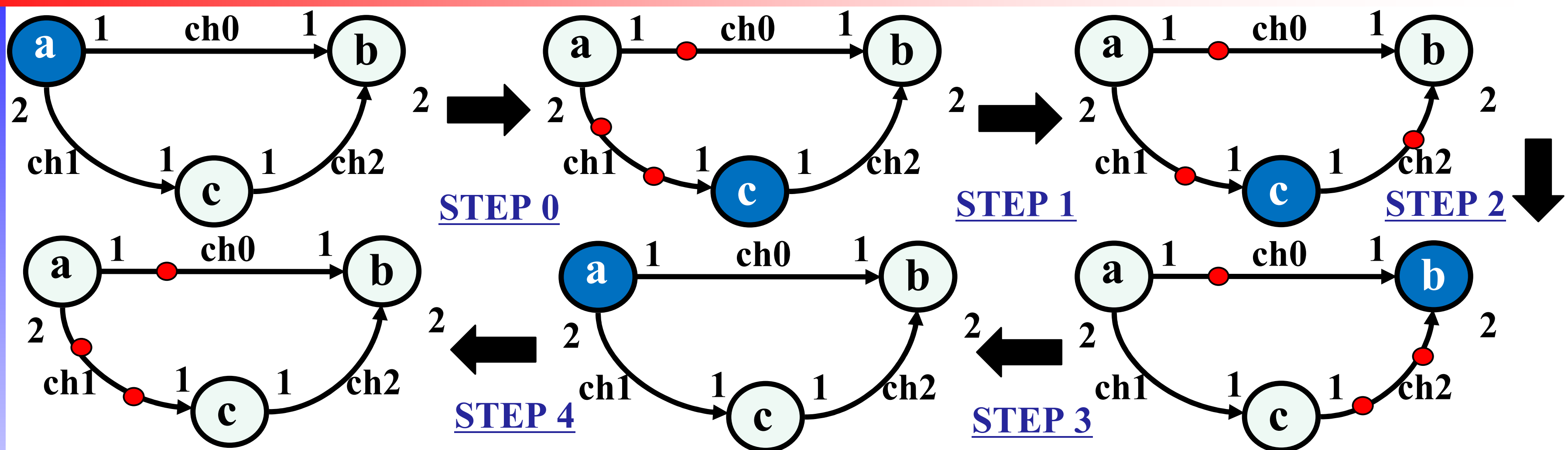
		a	c	c	b
ch0	0	1	1	1	0
ch1	0	2	1	0	0
ch2	0	0	1	2	0

List of the channels that each process is waiting for (the number of tokens required)

chWait[proc][ch]

a	-	-		-	-
b	ch0(1)&ch2(2)	ch0(1)&ch2(2)	ch0(1)&ch2(2)	ch0(1)&ch2(2)	ch0(1)&ch2(2)
c	ch1(1)	ch1(1)	ch1(1)	ch1(1)	ch1(1)

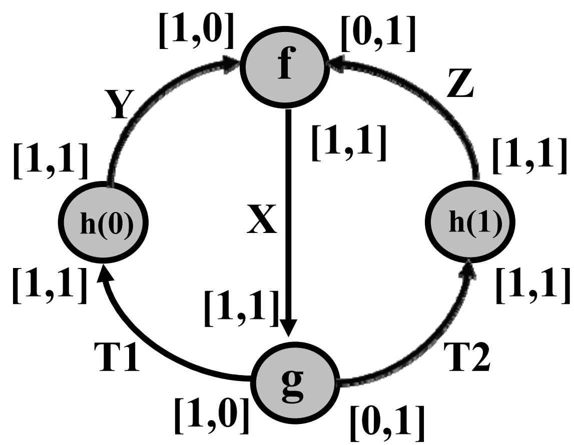
SDF-like Model Scheduling Example



STEP		0	1	2	3	4
List of the number of tokens in each channel (Token[ch])						
		a	c	c	b	a
ch0	0	1	1	1	0	1
ch1	0	2	1	0	0	2
ch2	0	0	1	2	0	0
List of th[illegible]at each process is waiting for (the number of tokens required)						
a		-	-	-	-	-
b	ch0(1)&ch2(2)	ch0(1)&ch2(2)	ch0(1)&ch2(2)	ch0(1)&ch2(2)	ch0(1)&ch2(2)	ch0(1)&ch2(2)
c	ch1(1)	ch1(1)	ch1(1)	ch1(1)	ch1(1)	ch1(1)

PASS

Periodic ConcurrenC Example

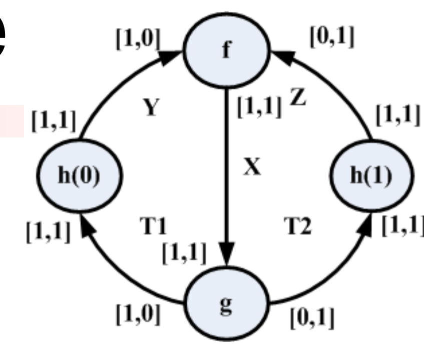


KPN-like ConcurrenC example that qualifies as CSDF-like

Scheduling of Periodic ConcurrenC Models

- Additional Data Structure (for channel prediction)
  - Internal state of each process (local control variables). ➔acquired dynamically when scheduling
  - Waiting channel list for each process and the number of input tokens for each process for each iteration. ➔ acquired dynamically when scheduling
- Code Slicing
  - **REPLACE: replace channel send / receive with Token[P] list** updating (in _t()) and **chWait[N][P] predicting (in _ch()).**
  - **REMOVE: (a) the loop statement, e.g. while(1); (b) statements** dealing with the data variables.
  - **KEEP: statements dealing with the state variables (in _t())** and make these variables *static to the functional block.*

Code Slicing Example




```
behavior f(i_int_receiver Y,  i_int_receiver Z,
               i_int_sender W)
{
   int b = 1;
   void main()
   {
      while(1)
      {
         if(b)
            Y.receive(&i);
         else
            Z.receive(&i);
         X.send(i);
         b = (b + 1) % 2;
      }
   }
};
```

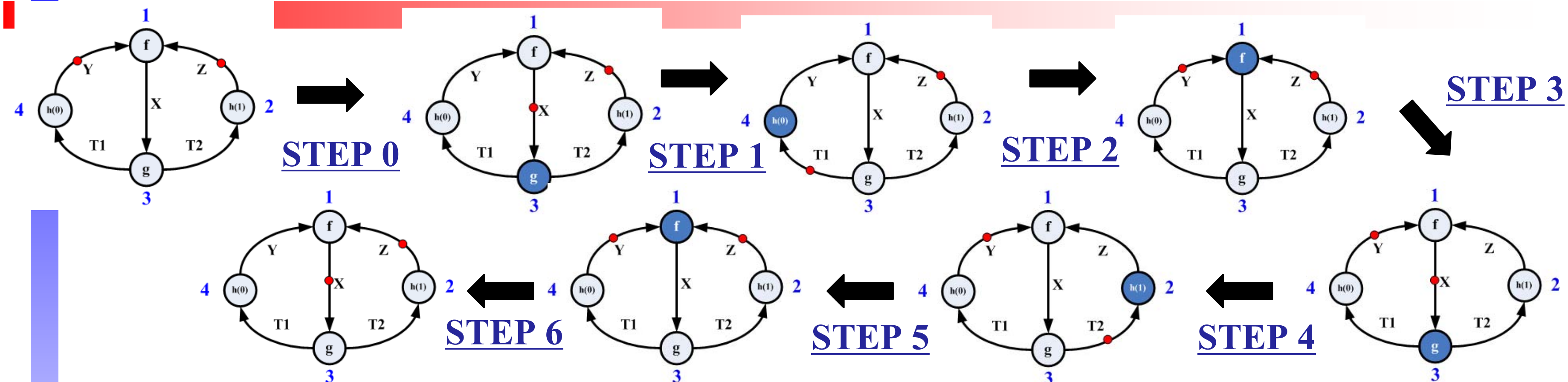
SpecC description for the KPN-like model

Sliced description code for the KPN-like model

```
void  f_slice::f_t()
// update Token[]
{
   //while(1)
   //{
      if(b)
         Token[Y] --;
      else
         Token[Z] --;
      Token[X] ++;
      b = (b + 1) % 2;
   //}
}
```

```
void  f_slice::f_ch()
// update chWait[][]
{
   //while(1)
   //{
      if(b){
         chWait[f_proc][Y] = 1;
         chWait[f_proc][Z] = 0;}
      else{
         chWait[f_proc][Y] = 0;
         chWait[f_proc][Z] = 1;}
   //}
}
```

Periodic ConcurrenC Model Scheduling Example



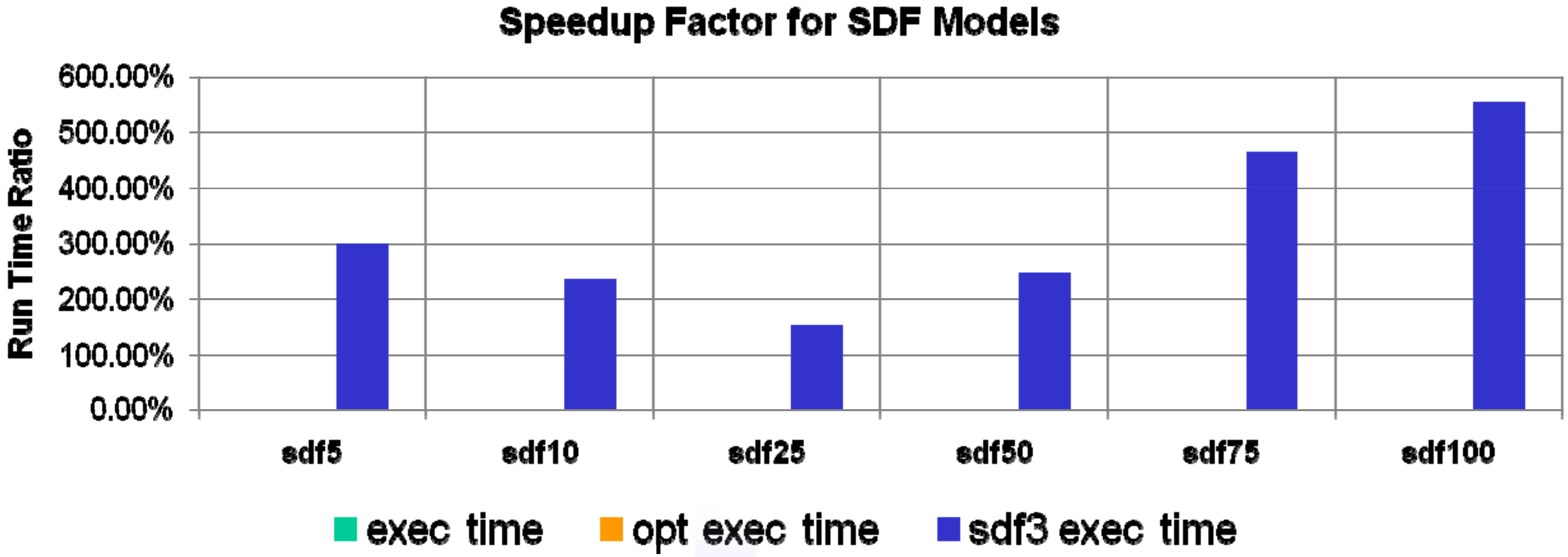
STEP		0	1	2	3	4	5	6
List of the number of tokens in each channel. Token[ch]								
		f	g	h(0)	f	g	h(1)	f
X	0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1
Y	1	0	0	1	1	1	1	0
Z	1	1	1	1	0	0	1	1
T1	[illegible]	[illegible]	1	0	0	0	0	0
T2	[illegible]	[illegible]	0	0	0	1	0	0
List of the channels that each process is waiting for. chWait[proc][ch] (the number of tokens required for each iteration)								
h(0)	T1(1)	T1(1)	T1(1)	T1(1)	T1(1)	T1(1)	T1(1)	T1(1)
h(1)	T2(1)	T2(1)	T2(1)	T2(1)	T2(1)	T2(1)	T2(1)	T2(1)
f	Y(1) & Z(0)	Y(0) & Z(1)	Y(0) & Z(1)	Y(0) & Z(1)	Y(1) & Z(0)	Y(1) & Z(0)	Y(1) & Z(0)	Y(0) & Z(1)
g	X(1)	X(1)	X(1)	X(1)	X(1)	X(1)	X(1)	X(1)
Internal condition variables								
f.b	1	0	0	0	1	1	1	0
g.b	1	1	0	0	0	1	1	1

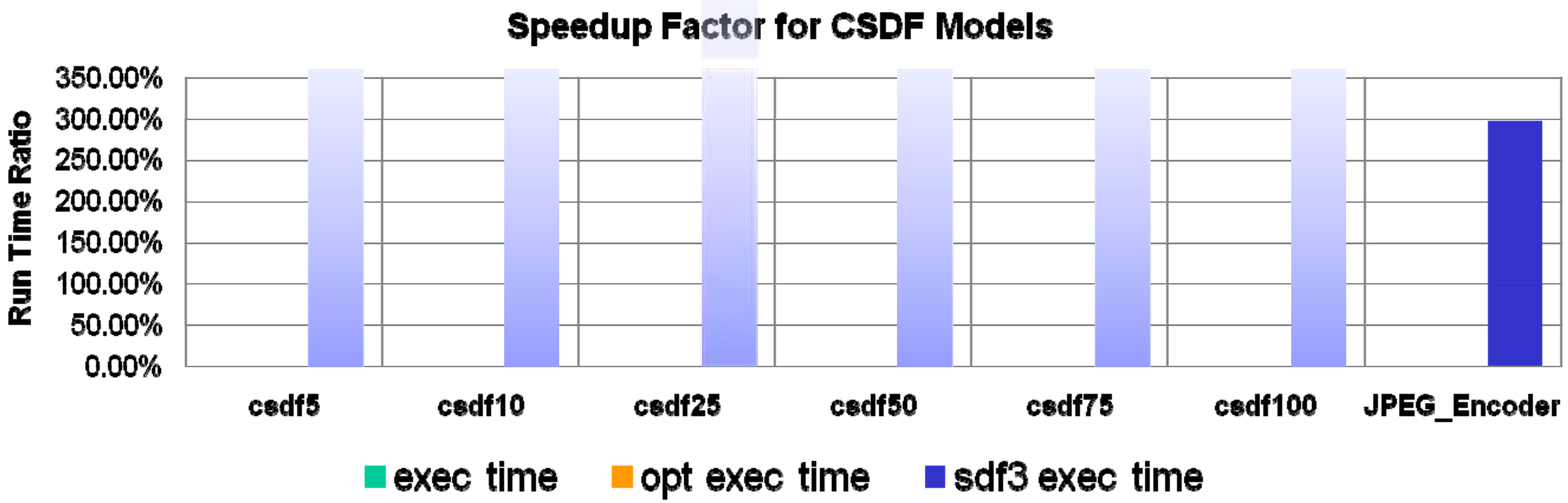
PASS

Experiment Results

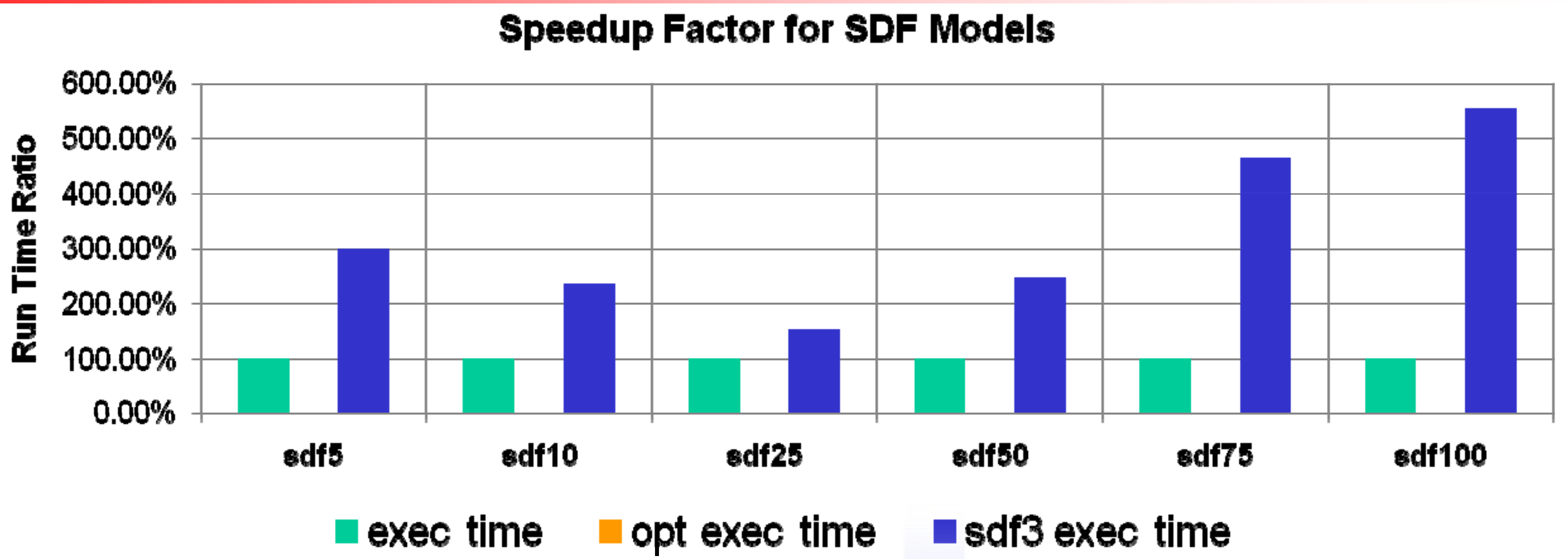
- Static Scheduling Algorithm Performance
  - Heuristic static scheduling algorithm
  - Optimized heuristic static scheduling algorithm (status comparisons are reduced)
  - ***sdf3*** scheduling algorithm
- Simulation speed of the models
  - comparison is made between generic DE implementation and static scheduled implementation.

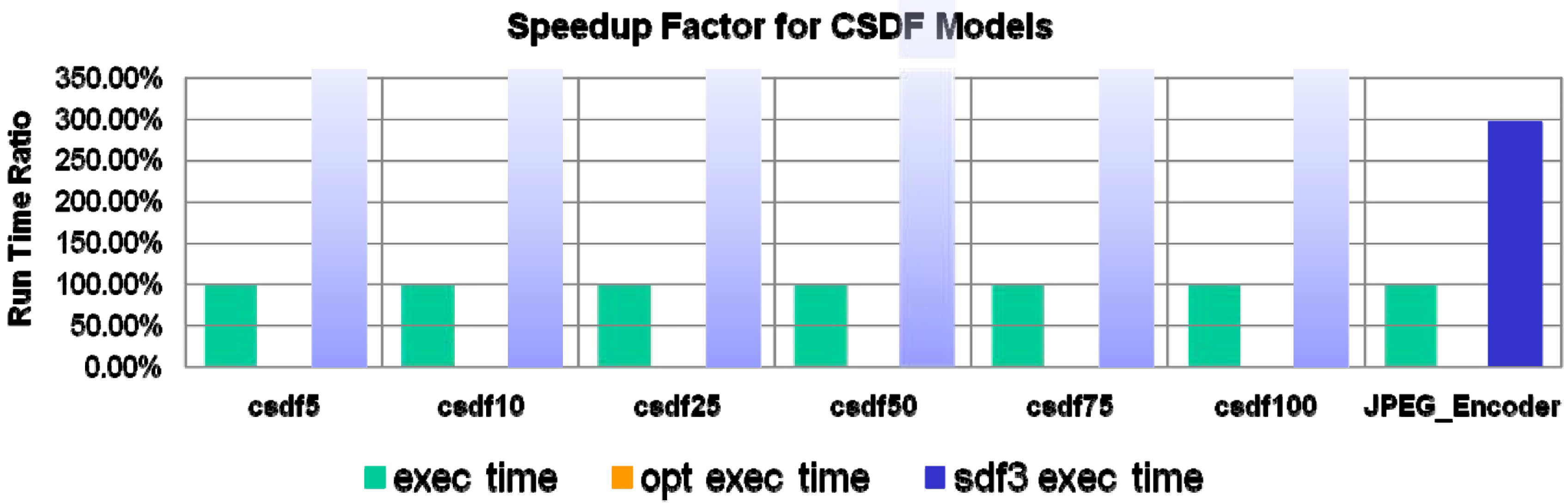
Experiment Results



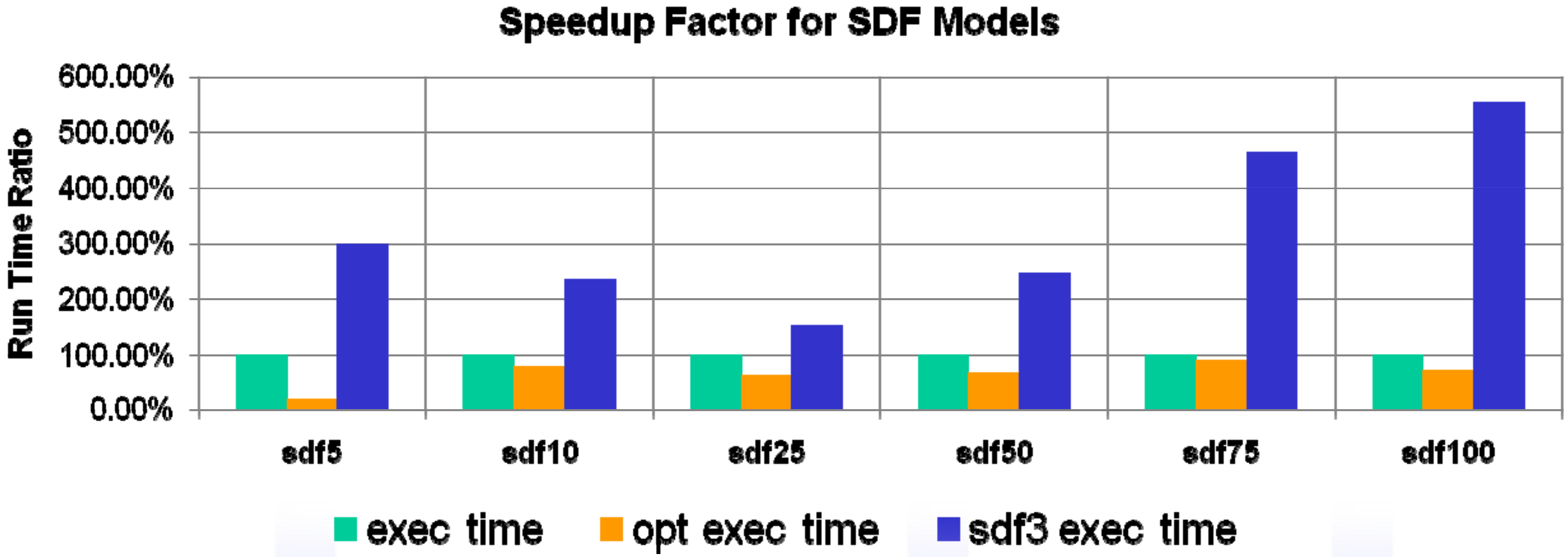


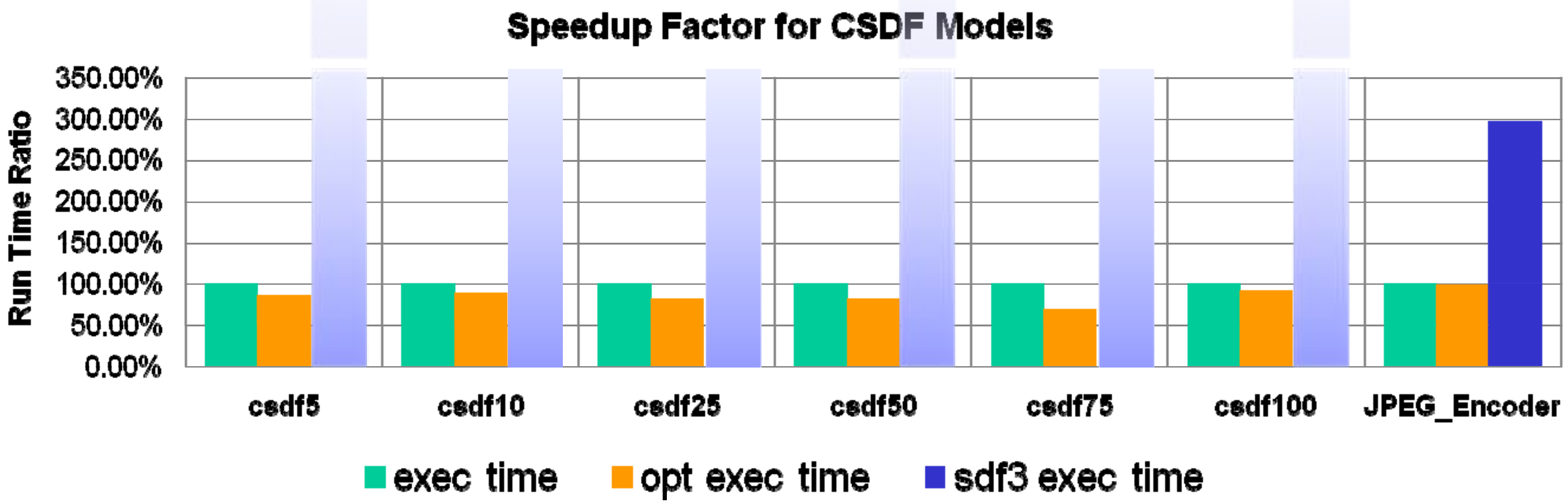
Experiment Results



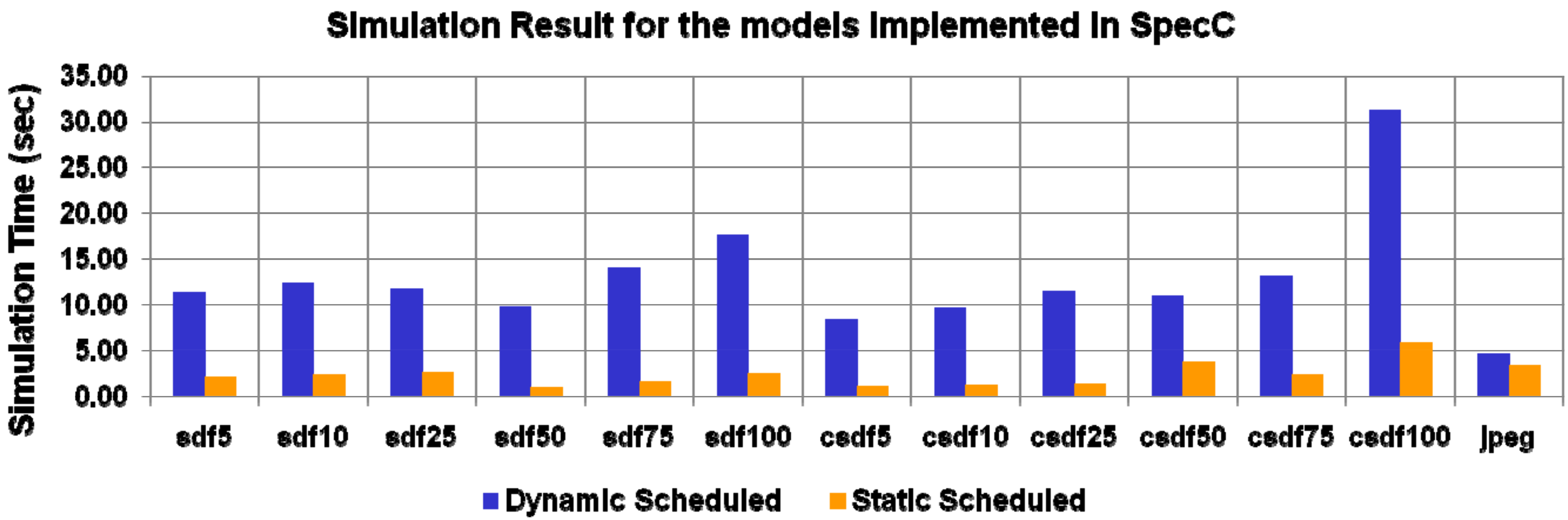


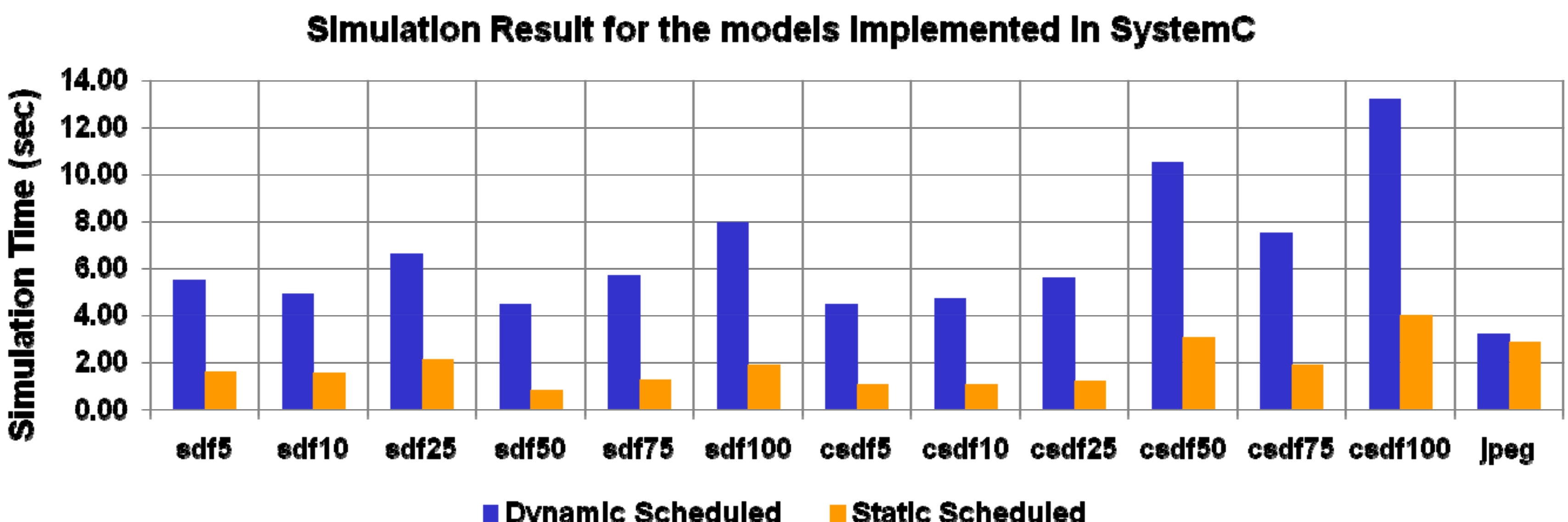
Experiment Results





Experiment Results





Conclusion and Future Work

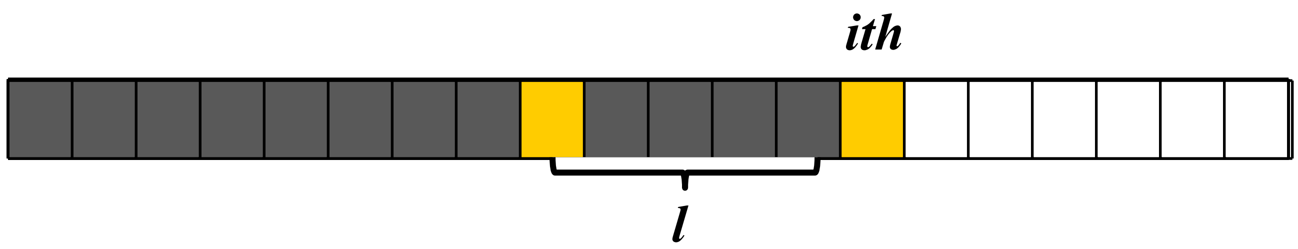
- Conclusion
  - A new heuristic scheduling algorithm of periodic ConcurrenC models are discussed.
  - The algorithm achieves speedup than classic matrix-based algorithms and can handle large models which cannot be handled by the previous algorithm in reasonable time.
  - Static Scheduling helps to improve the simulation speed of periodic ConcurrenC models.
- Future work
  - Support for timing information in ConcurrenC blocks.
  - Extend this scheduling strategy into a distributed simulation environment.

backups

Algorithm Performance Optimization

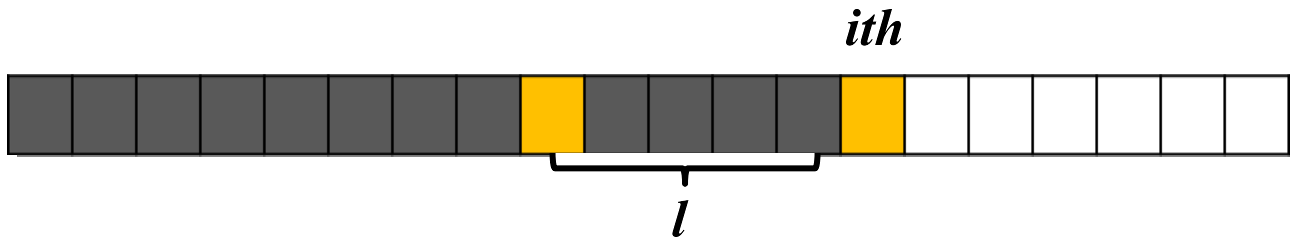
- Main idea: Reduce the time of status comparison.
- length of PASS is ***l***, PASS is found at ***ith*** scheduling step

– Number of comparisons without optimization: ***i(i-1)/2***



Algorithm Performance Optimization

- Main idea: Reduce the time of status comparison.
- length of PASS is ***l***, PASS is found at ***ith*** scheduling step
– Number of comparisons without optimization: $i(i-1)/2$



- Compare every N steps, PASS is found at ***tNth*** scheduling step ($tN >= i$), the length of PASS is still ***l*** (proved in the paper).
  - Number of comparisons without optimization: $t(t-1)N/2$
  - Speedup for comparison is almost N.