V Brazilian Symposium on Computing Systems Engineering

Applying Multi-Core Model Checking to Hardware-Software Partitioning in Embedded Systems

Alessandro Trindade**, Hussama Ismail**, and Lucas Cordeiro

Foz do Iguaçu / PR, 5th November of 2015

Motivation

- Embedded systems: parts in **HW** (↑ speed, ↑$$$) and other parts in **SW** (↓$, ↓ speed)
- Most critical step in 1st generation of HW/SW Co-design partitioning

- Model checking: describe the system behavior by a precise and not ambiguous (mathematical) model
  - Early detection of errors
  - Explore all states of a system in a automatic way

  (so instead of finding code violations, we can explore states until it solves the partitioning problem)

Motivation

- Embedded systems: parts in **HW** (↑ speed, ↑$$$) and other parts in **SW** (↓$, ↓ speed)
- Most critical step in 1st generation of HW/SW Co-design partitioning

- Model checking: describe the system behavior by a precise and not ambiguous (mathematical) model
  - Early detection of errors
  - Explore all states of a system in a automatic way

  (so instead of finding code violations, we can explore states until it solves the partitioning problem)

Objectives

Apply multi-core model checking based on satisfiability modulo theories (SMT) to solve the HW/SW partitioning

- Use OpenMP (Open Multi-Processing API) support to perform multi-core model checking
- Create and improve algorithms to implement the proposed technique
- Perform experimental evaluation over benchmarks
- Compare our approach with ILP (Integer Linear Programming) and GA (Genetic Algorithm) using MATLAB

Objectives

Apply multi-core model checking based on satisfiability modulo theories (SMT) to solve the HW/SW partitioning

- Use OpenMP (Open Multi-Processing API) support to perform multi-core model checking
- Create and improve algorithms to implement the proposed technique
- Perform experimental evaluation over benchmarks
- Compare our approach with ILP (Integer Linear Programming) and GA (Genetic Algorithm) using MATLAB

Objectives

Apply multi-core model checking based on satisfiability modulo theories (SMT) to solve the HW/SW partitioning

- Use OpenMP (Open Multi-Processing API) support to perform multi-core model checking
- Create and improve algorithms to implement the proposed technique
- Perform experimental evaluation over benchmarks
- Compare our approach with ILP (Integer Linear Programming) and GA (Genetic Algorithm) using MATLAB

Objectives

Apply multi-core model checking based on satisfiability modulo theories (SMT) to solve the HW/SW partitioning

- Use OpenMP (Open Multi-Processing API) support to perform multi-core model checking
- Create and improve algorithms to implement the proposed technique
- Perform experimental evaluation over benchmarks
- Compare our approach with ILP (Integer Linear Programming) and GA (Genetic Algorithm) using MATLAB

Objectives

Apply multi-core model checking based on satisfiability modulo theories (SMT) to solve the HW/SW partitioning

- Use OpenMP (Open Multi-Processing API) support to perform multi-core model checking
- Create and improve algorithms to implement the proposed technique
- Perform experimental evaluation over benchmarks
- Compare our approach with ILP (Integer Linear Programming) and GA (Genetic Algorithm) using MATLAB

Optimization

- Find the maximum or minimum value of a function
  - Minimize the effort and maximize the benefit
- There is not a unique method to solve all the problems
- Most popular technique: LP (Linear Programming)
  - Integer Linear Programming
  - Binary Linear Programming
- Heuristics Algorithms: GA (Genetic Algorithm) can solve more complex problems faster
  - Drawback: it may not find the global minimum/maximum (i.e., the optimal result)

Optimization

- Find the maximum or minimum value of a function
  - Minimize the effort and maximize the benefit
- There is not a unique method to solve all the problems
- Most popular technique: LP (Linear Programming)
  - Integer Linear Programming
  - Binary Linear Programming
- Heuristics Algorithms: GA (Genetic Algorithm) can solve more complex problems faster
  - Drawback: it may not find the global minimum/maximum (i.e., the optimal result)

Optimization

- Find the maximum or minimum value of a function
  - Minimize the effort and maximize the benefit
- There is not a unique method to solve all the problems
- Most popular technique: LP (Linear Programming)
  - Integer Linear Programming
  - Binary Linear Programming
- Heuristics Algorithms: GA (Genetic Algorithm) can solve more complex problems faster
  - Drawback: it may not find the global minimum/maximum (i.e., the optimal result)

Optimization

- Find the maximum or minimum value of a function
  - Minimize the effort and maximize the benefit
- There is not a unique method to solve all the problems
- Most popular technique: LP (Linear Programming)
  - Integer Linear Programming
  - Binary Linear Programming
- Heuristics Algorithms: GA (Genetic Algorithm) can solve more complex problems faster
  - Drawback: it may not find the global minimum/maximum (i.e., the optimal result)

Mathematical Modeling

Informal Model (Assumptions)

- There is only one software context and only one hardware context
  - Each component must be mapped into one of these two contexts.
- The software component implementation has a software cost associated (running time)
- The hardware component implementation has a hardware cost associated (area, heat dissipation or energy consumption)
- **Premisses**:
  - The hardware is significantly faster than software;
  - The running time of hardware is zero;
  - If two components are mapped to the same context, there is no overhead of communication between them.

Mathematical Modeling

Informal Model (Assumptions)

- There is only one software context and only one hardware context
  - Each component must be mapped into one of these two contexts.
- The software component implementation has a software cost associated (running time)
- The hardware component implementation has a hardware cost associated (area, heat dissipation or energy consumption)
- **Premisses**:
  - The hardware is significantly faster than software;
  - The running time of hardware is zero;
  - If two components are mapped to the same context, there is no overhead of communication between them.

Mathematical Modeling

Informal Model (Assumptions)

- There is only one software context and only one hardware context
  - Each component must be mapped into one of these two contexts.
- The software component implementation has a software cost associated (running time)
- The hardware component implementation has a hardware cost associated (area, heat dissipation or energy consumption)
- **Premisses**:
  - The hardware is significantly faster than software;
  - The running time of hardware is zero;
  - If two components are mapped to the same context, there is no overhead of communication between them.

Mathematical Modeling

Informal Model (Assumptions)

- There is only one software context and only one hardware context
  - Each component must be mapped into one of these two contexts.
- The software component implementation has a software cost associated (running time)
- The hardware component implementation has a hardware cost associated (area, heat dissipation or energy consumption)
- **Premisses**:
  - The hardware is significantly faster than software
  - The running time of hardware is zero
  - If two components are mapped to the same context, there is no overhead of communication between them

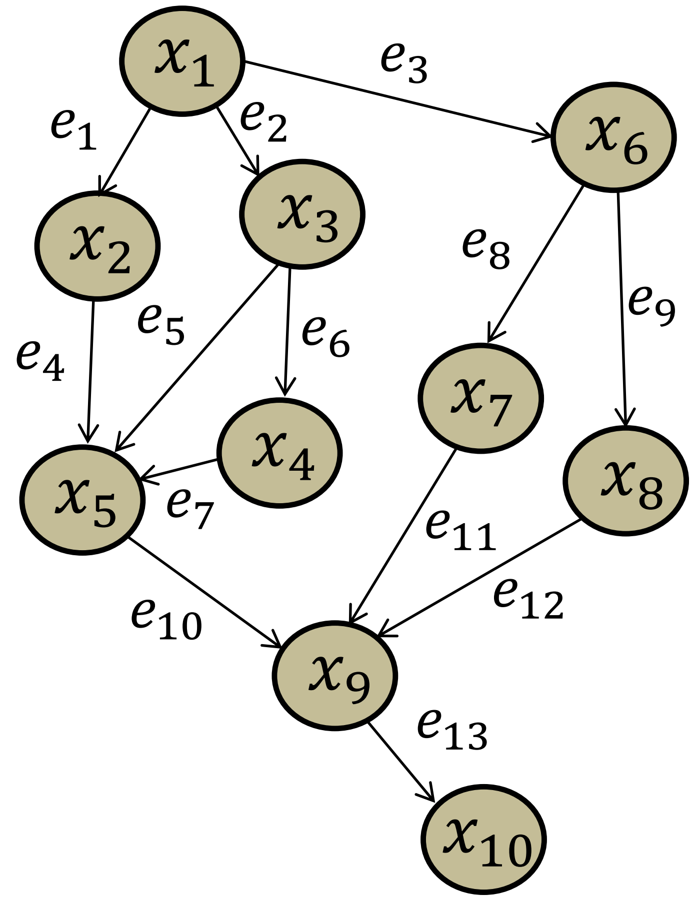
Mathematical Modeling

Formal Model

- Task graph G = (V, E)
- Vertices V = {x_1, x_2, …, x_n} : nodes are the components of the system to be partitioned (context)
- Each node x_i : has the hardware cost $h(x_i)$ and the software cost $s(x_i)$
  - $ HW (area, heat dissipation, energy consumption)
  - $ SW (execution time)
- Edges (E) represent communication between the components
- $c(x_i, x_j)$: represents the communication cost between x_i and x_j if they are in different contexts
- The HW-SW partitioning P has:

– $H_P = \sum h_i$ (hardware cost)

– $S_P = \sum s_i + \sum c(x_i, x_j)$ (software cost)



Example: 10 nodes & 13 edges

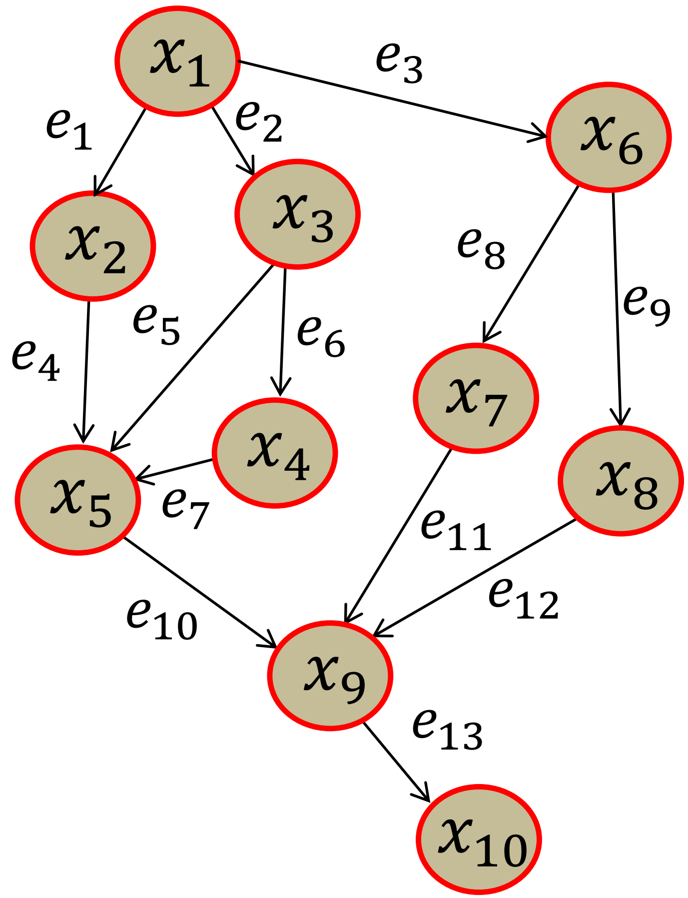
Mathematical Modeling

Formal Model

- Task graph G = (V, E)
- Vertices V = {x_1, x_2, …, x_n} : nodes are the components of the system to be partitioned (context)
- Each node x_i : has the hardware cost h(x_i) and the software cost s(x_i)
  - $ HW (area, heat dissipation, energy consumption)
  - $ SW (execution time)
- Edges (E) represent communication between the components
- c(x_i, x_j): represents the communication cost between x_i and x_j if they are in different contexts
- The HW-SW partitioning P has:

– $H_P = \sum h_i$ (hardware cost)

– $S_P = \sum s_i + \sum c(x_i, x_j)$ (software cost)



Example: 10 nodes & 13 edges

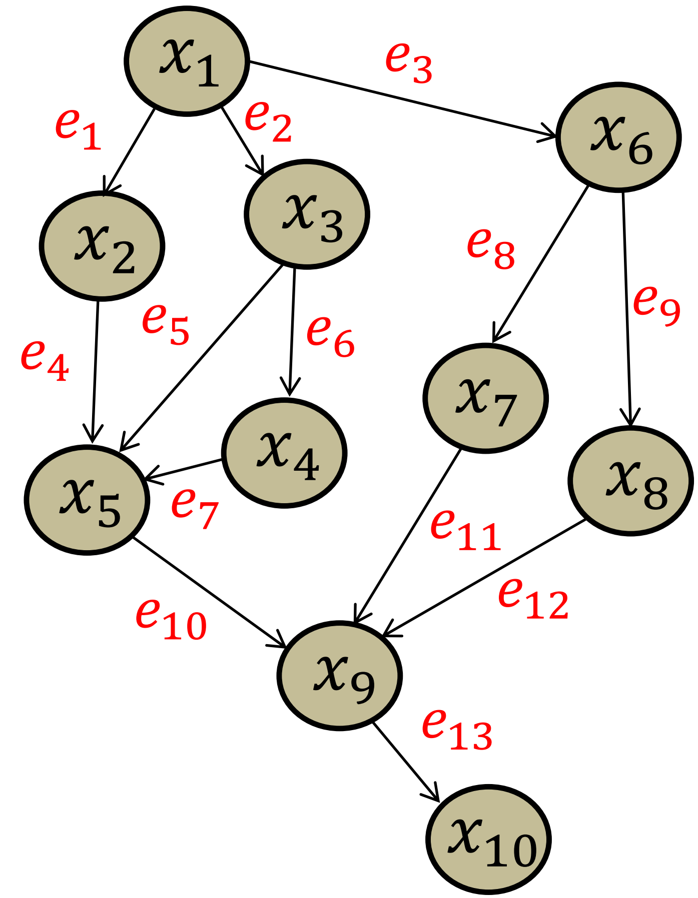
Mathematical Modeling

Formal Model

- Task graph G = (V, E)
- Vertices V = {x_1, x_2, …, x_n} : nodes are the components of the system to be partitioned (context)
- Each node x_i : has the hardware cost $h(x_i)$ and the software cost $s(x_i)$
  - $ HW (area, heat dissipation, energy consumption)
  - $ SW (execution time)
- Edges (E) represent communication between the components
- $c(x_i, x_j)$: represents the communication cost between x_i and x_j if they are in different contexts
- The HW-SW partitioning P has:

– $H_P = \sum h_i$ (hardware cost)

– $S_P = \sum s_i + \sum c(x_i, x_j)$ (software cost)



Example: 10 nodes & 13 edges

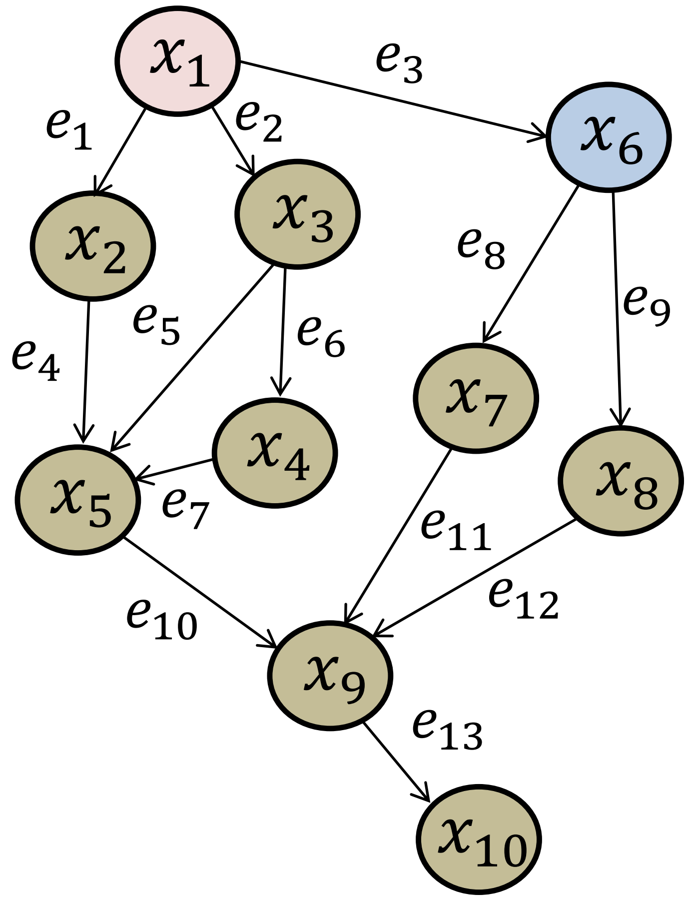
Mathematical Modeling

Formal Model

- Task graph G = (V, E)
- Vertices V = {x_1, x_2, …, x_n} : nodes are the components of the system to be partitioned (context)
- Each node x_i : has the hardware cost h(x_i) and the software cost s(x_i)
  - $ HW (area, heat dissipation, energy consumption)
  - $ SW (execution time)
- Edges (E) represent communication between the components
- c(x_i, x_j): represents the communication cost between x_i and x_j if they are in different contexts
- The HW-SW partitioning P has:

– $H_P = \sum h_i$ (hardware cost)

– $S_P = \sum s_i + \sum c(x_i, x_j)$ (software cost)



Example: 10 nodes & 13 edges

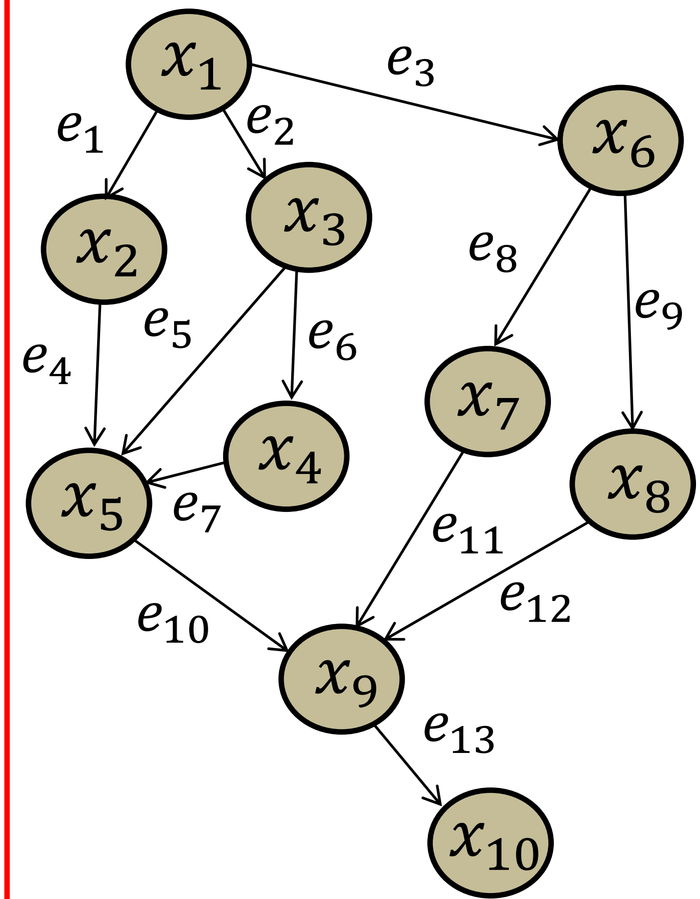
Mathematical Modeling

Formal Model

- Task graph G = (V, E)
- Vertices V = {x_1, x_2, …, x_n} : nodes are the components of the system to be partitioned (context)
- Each node x_i : has the hardware cost $h(x_i)$ and the software cost $s(x_i)$
  - $ HW (area, heat dissipation, energy consumption)
  - $ SW (execution time)
- Edges (E) represent communication between the components
- $c(x_i, x_j)$: represents the communication cost between x_i and x_j if they are in different contexts
- The HW-SW partitioning P has:

– $H_P = \sum h_i$ (hardware cost)

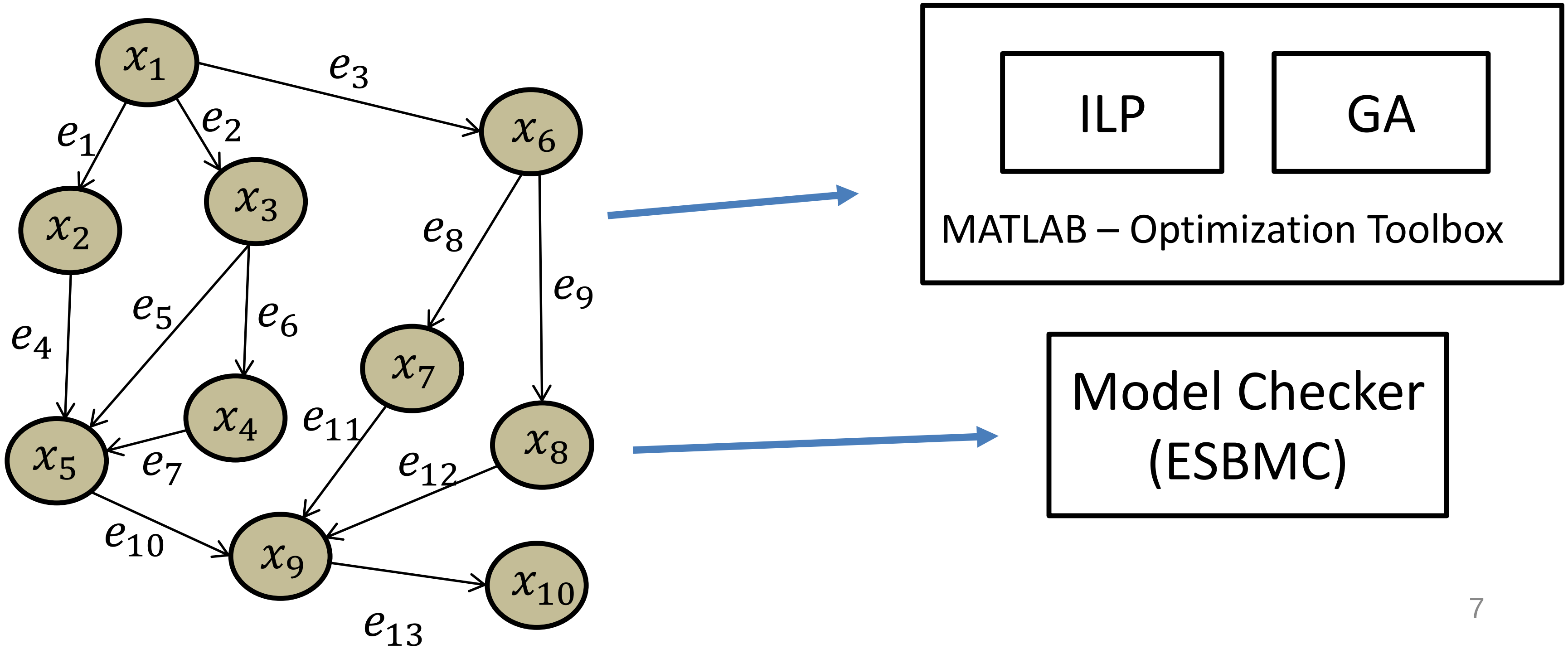
– $S_P = \sum s_i + \sum c(x_i, x_j)$ (software cost)



Example: 10 nodes & 13 edges

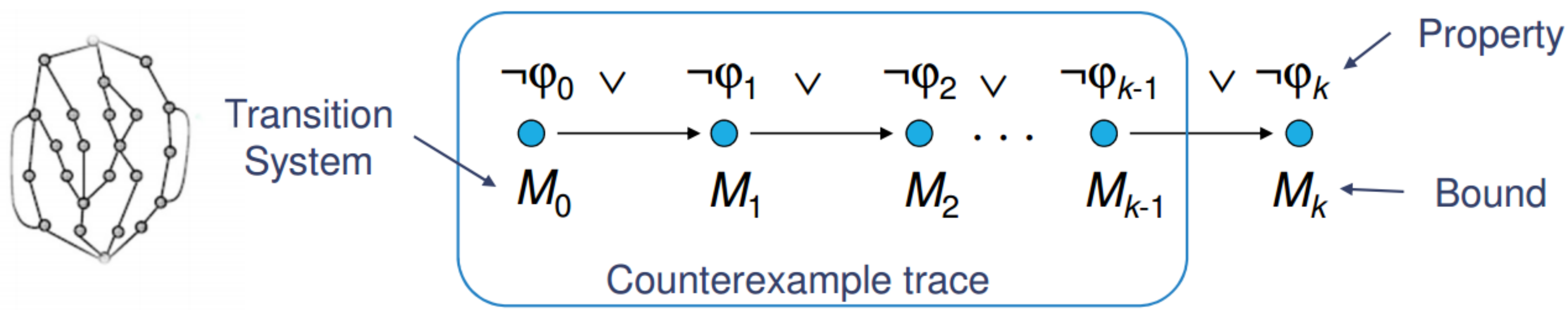
Mathematical Modeling

- This paper focus on the case where the initial software cost is given (S_0)
- We want $S_P < S_0$ and the minimal necessary hardware cost to resolve the problem (The complexity is **NP-Hard**)



Bounded Model Checking

- Basic Idea: given a transition system M, check negation of a given property φ up to given depth k

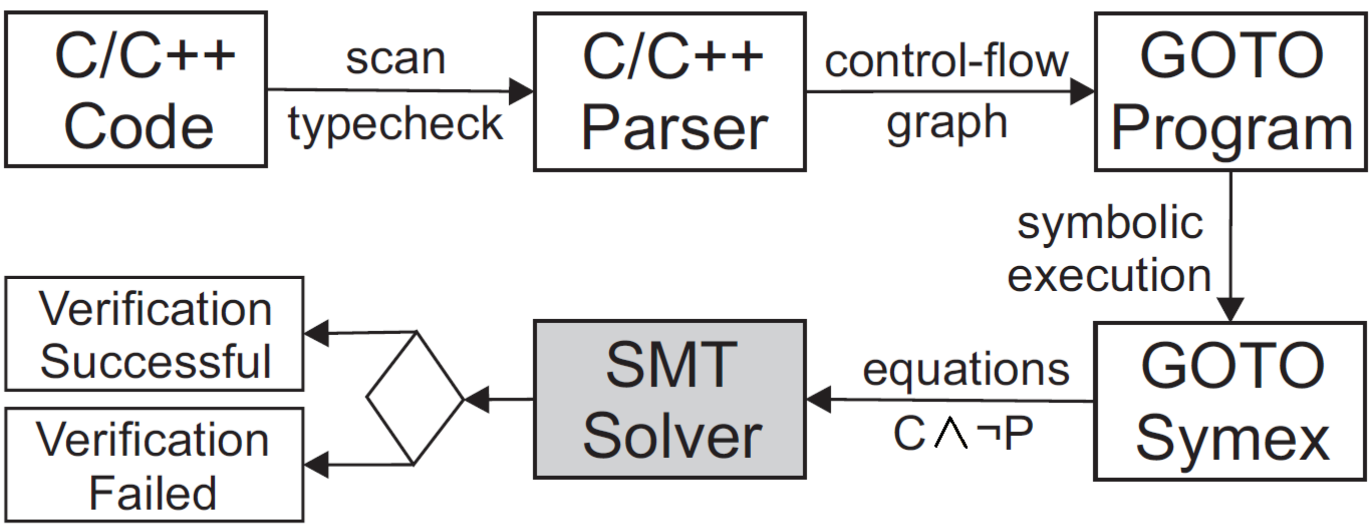


- Translated into a VC ψ such that: **ψ is satisfiable iff φ has counterexample (steps until the violation) of max. depth k**
- BMC has been applied successfully to verify (embedded) software since early 2000's. In 2014, Alessandro used BMC perform HW-SW partitioning.

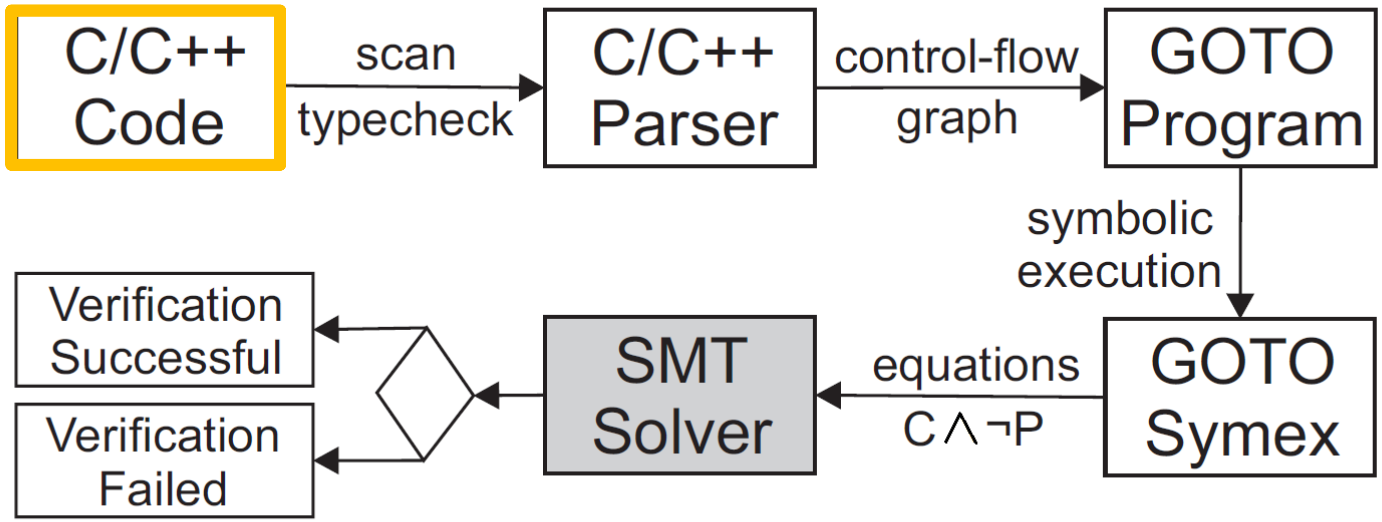
ESBMC (Model Checker)

- ESBMC (Efficient SMT-Based Context-Bounded Model Checker) is a model checker for ANSI-C and C++ source code
  - Check overflows, pointer safety, memory leaks, arrays bounds, atomicity, etc.
- Uses Satisfiability Modulo Theories (SMT) (addition to Boolean Satisfiability)
- SMT Solvers as back-end to decrease software complexity

Architecture:

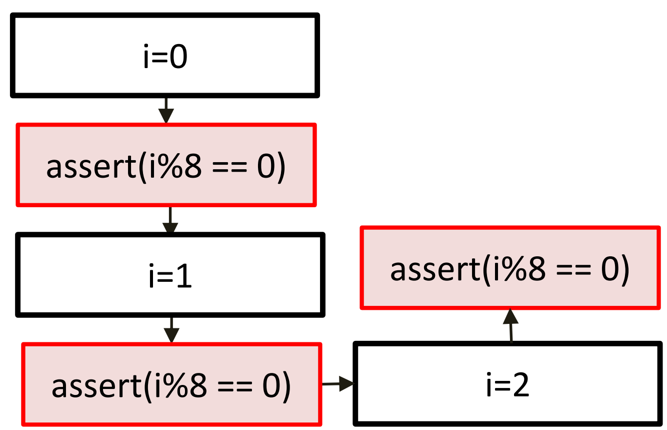


ESBMC Architecture




```
#include<stdio.h>
#include<assert.h>

int main(){
   int i = 0;
   for (i=0; i<50;i++){
      assert(i%8==0);
   }
}
```

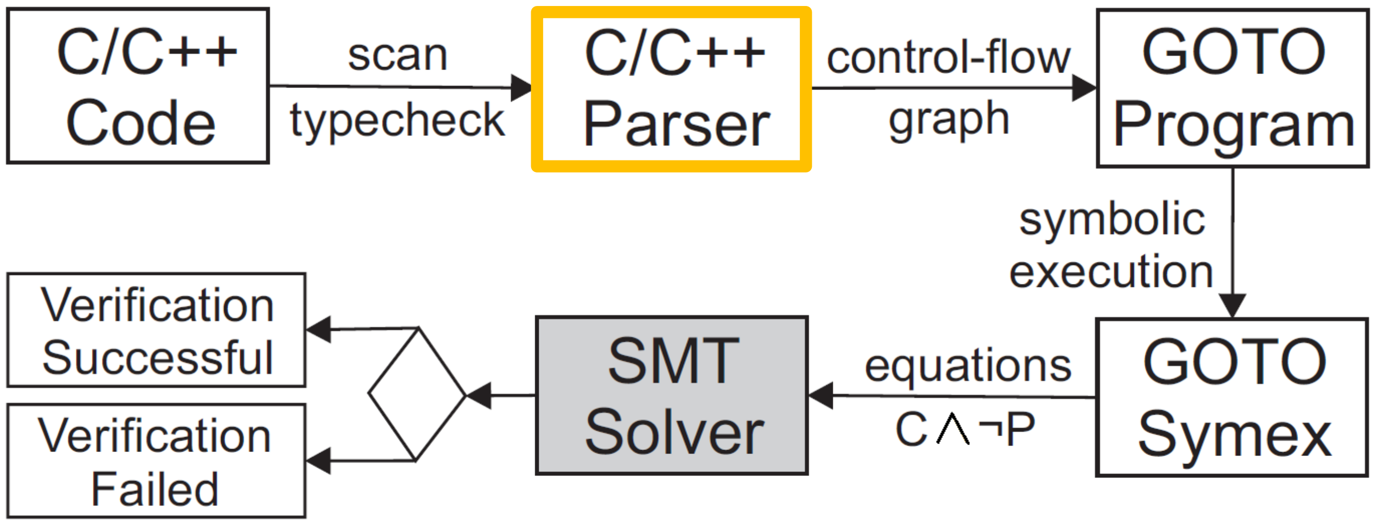
k=3 (bound)



$$C := \begin{bmatrix} i0 := 0 \wedge \\ i1 := 1 \wedge \\ i2 := 2 \end{bmatrix}$$

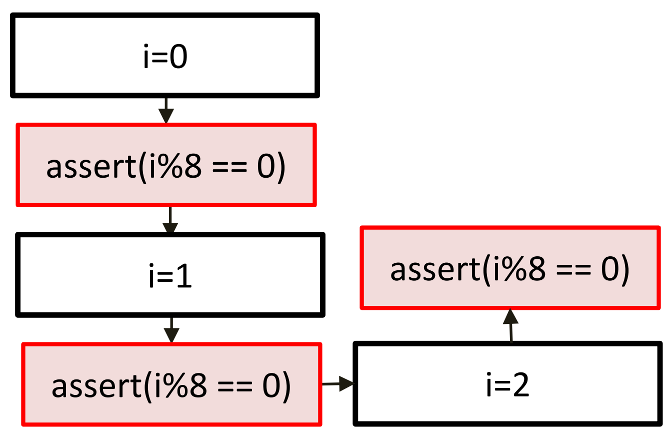
$$P := \begin{bmatrix} i0\%8 == 0 \wedge \\ i1\%8 == 0 \wedge \\ i2\%8 == 0 \end{bmatrix}$$

ESBMC Architecture




```
#include<stdio.h>
#include<assert.h>

int main(){
    int i = 0;
    for (i=0; i<50;i++){
        assert(i%8==0);
    }
}
```

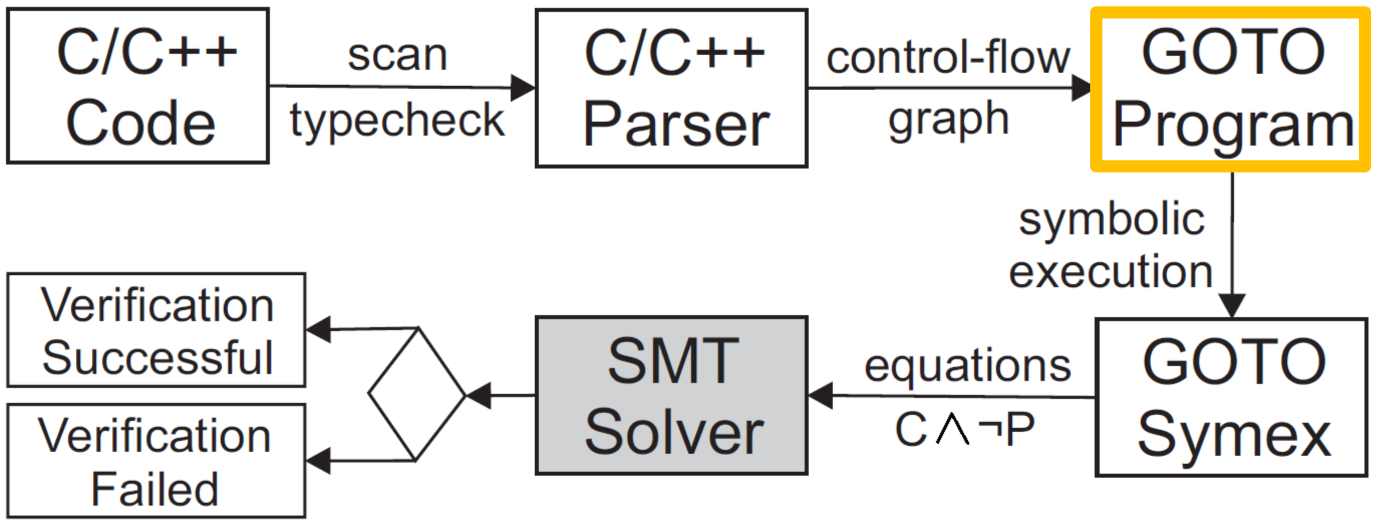
k=3 (bound)



$$C := \begin{bmatrix} i0 := 0 \wedge \\ i1 := 1 \wedge \\ i2 := 2 \end{bmatrix}$$

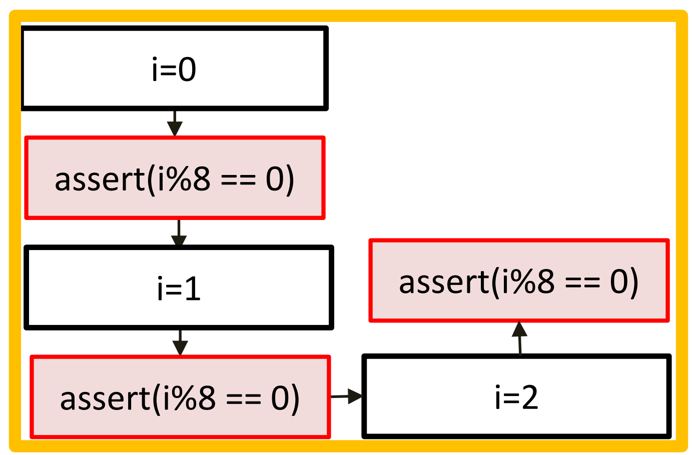
$$P := \begin{bmatrix} i0\%8 == 0 \wedge \\ i1\%8 == 0 \wedge \\ i2\%8 == 0 \end{bmatrix}$$

ESBMC Architecture




```c
#include<stdio.h>
#include<assert.h>

int main(){
    int i = 0;
    for (i=0; i<50;i++){
        assert(i%8==0);
    }
}
```

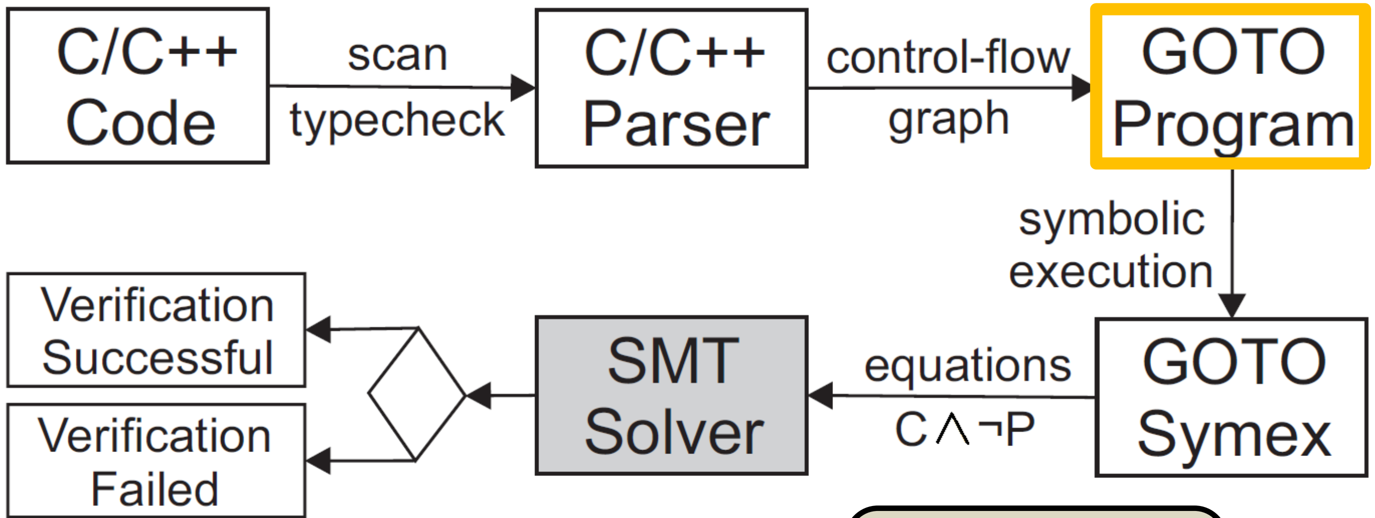
k=3 (bound)



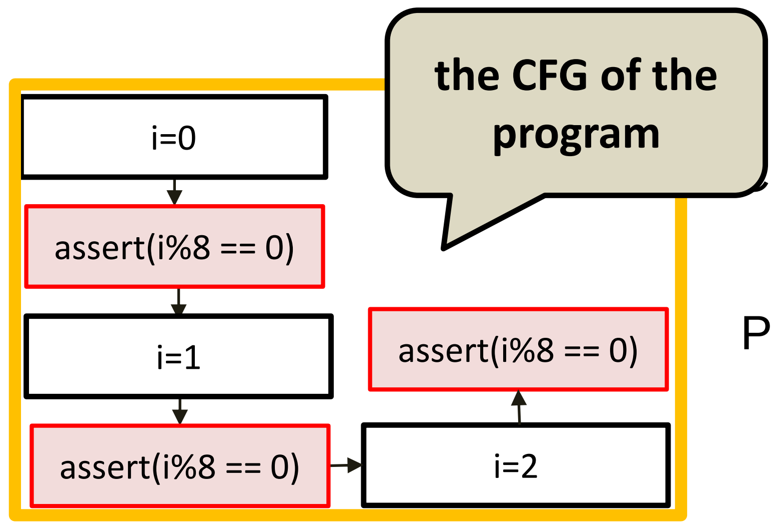
$$C := \left[\begin{array}{l} i0 := 0 \wedge \\ i1 := 1 \wedge \\ i2 := 2 \end{array}\right]$$

$$P := \left[\begin{array}{l} i0\%8 == 0 \wedge \\ i1\%8 == 0 \wedge \\ i2\%8 == 0 \end{array}\right]$$

ESBMC Architecture




```
#include<stdio.h>
#include<assert.h>

int main(){
   int i = 0;
   for (i=0; i<50;i++){
      assert(i%8==0);
   }
}
```

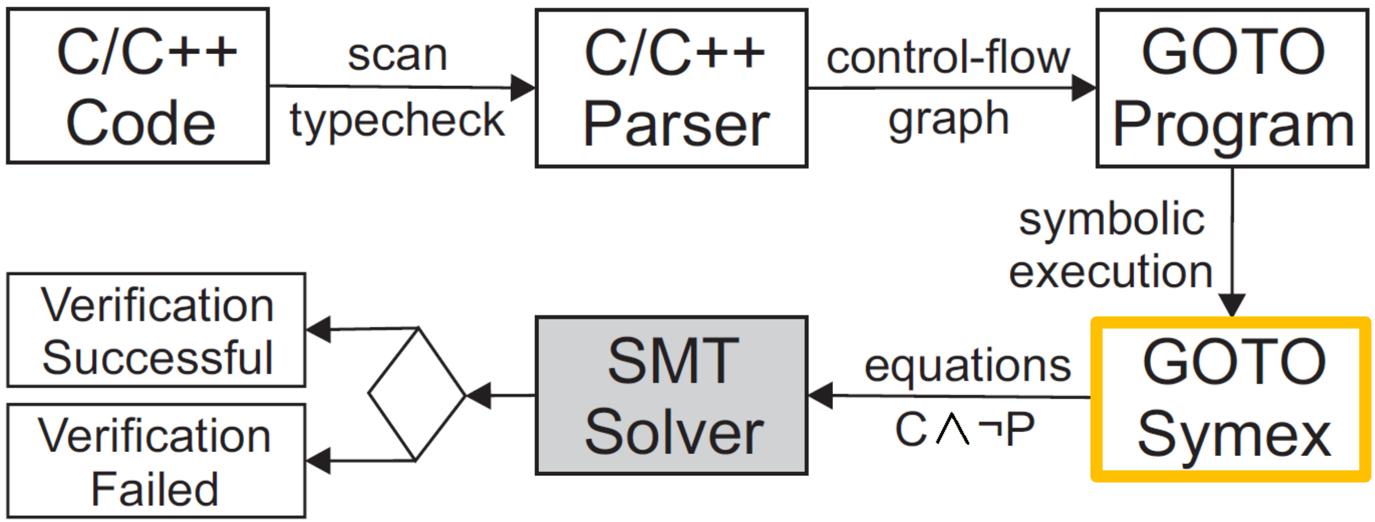


$$:= \begin{bmatrix} i0 := 0 \wedge \\ i1 := 1 \wedge \\ i2 := 2 \end{bmatrix}$$

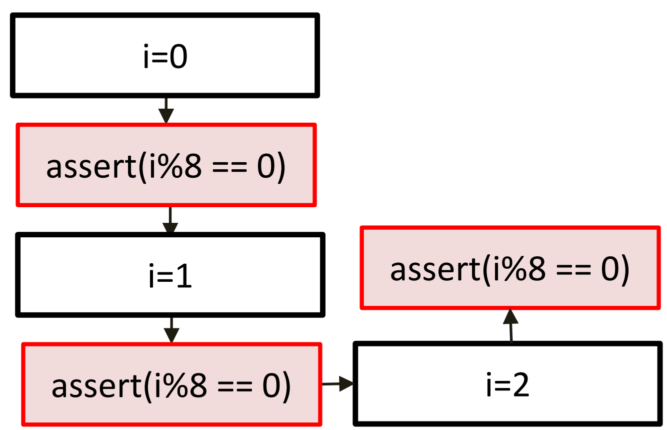
$$P := \begin{bmatrix} i0\%8 == 0 \wedge \\ i1\%8 == 0 \wedge \\ i2\%8 == 0 \end{bmatrix}$$

k=3 (bound)

ESBMC Architecture



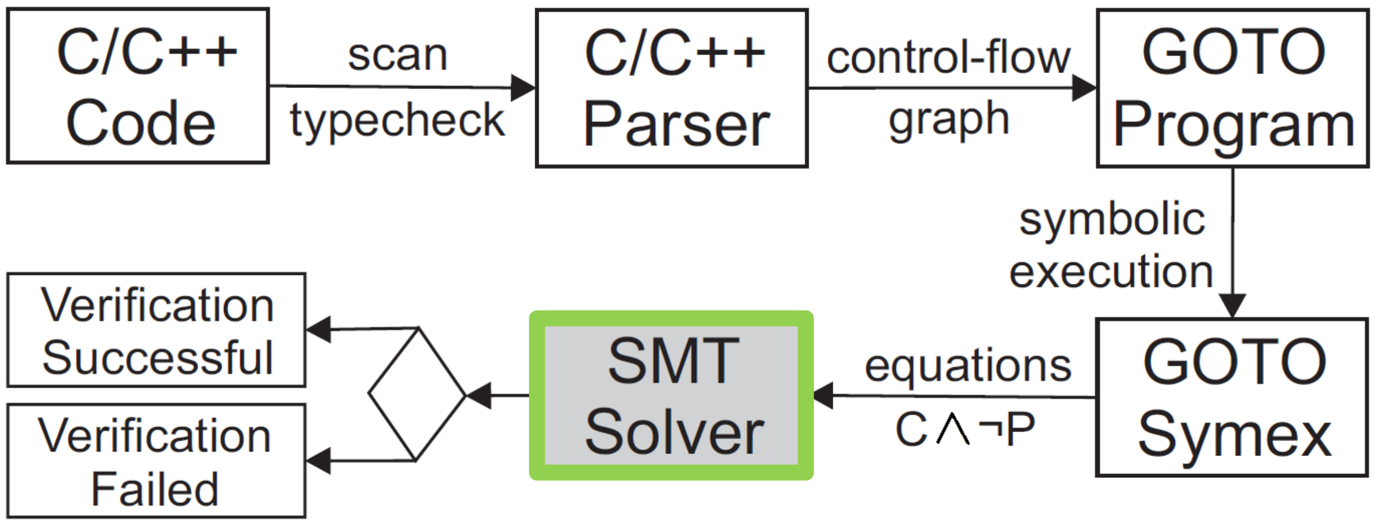

```
#include<stdio.h>
#include<assert.h>

int main(){
   int i = 0;
   for (i=0; i<50;i++){
      assert(i%8==0);
   }
}
```



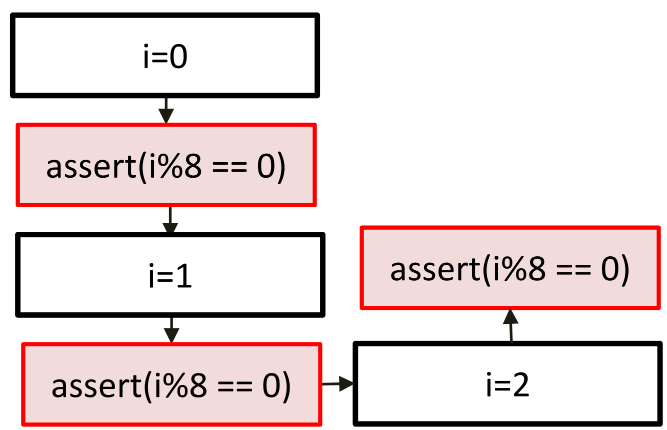
$$C := \left[\begin{array}{l} i0 := 0 \wedge \\ i1 := 1 \wedge \\ i2 := 2 \end{array}\right]$$

$$P := \left[\begin{array}{l} i0\%8 == 0 \wedge \\ i1\%8 == 0 \wedge \\ i2\%8 == 0 \end{array}\right]$$

ESBMC Architecture



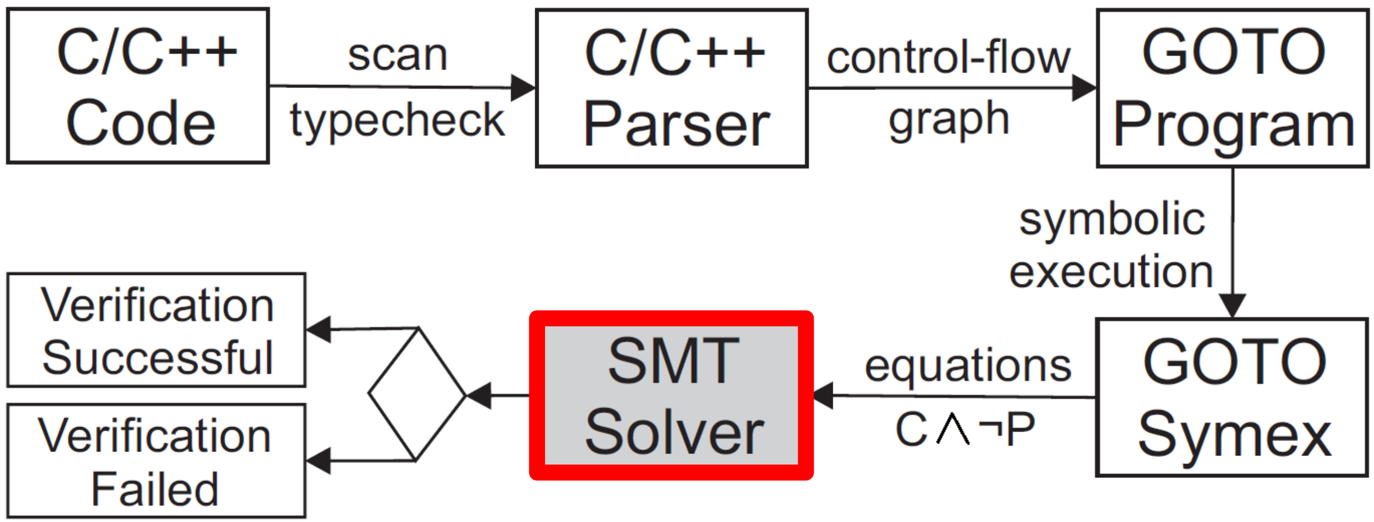

```c
#include<stdio.h>
#include<assert.h>

int main(){
   int i = 0;
   for (i=0; i<50;i++){
      assert(i%8==0);
   }
}
```



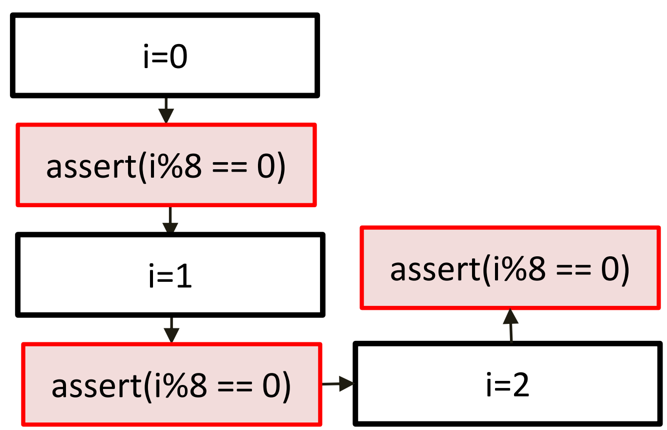
$$C := \begin{bmatrix} i0 := 0 \wedge \\ i1 := 1 \wedge \\ i2 := 2 \end{bmatrix}$$

$$P := \begin{bmatrix} i0\%8 == 0 \wedge \\ i1\%8 == 0 \wedge \\ i2\%8 == 0 \end{bmatrix}$$

ESBMC Architecture



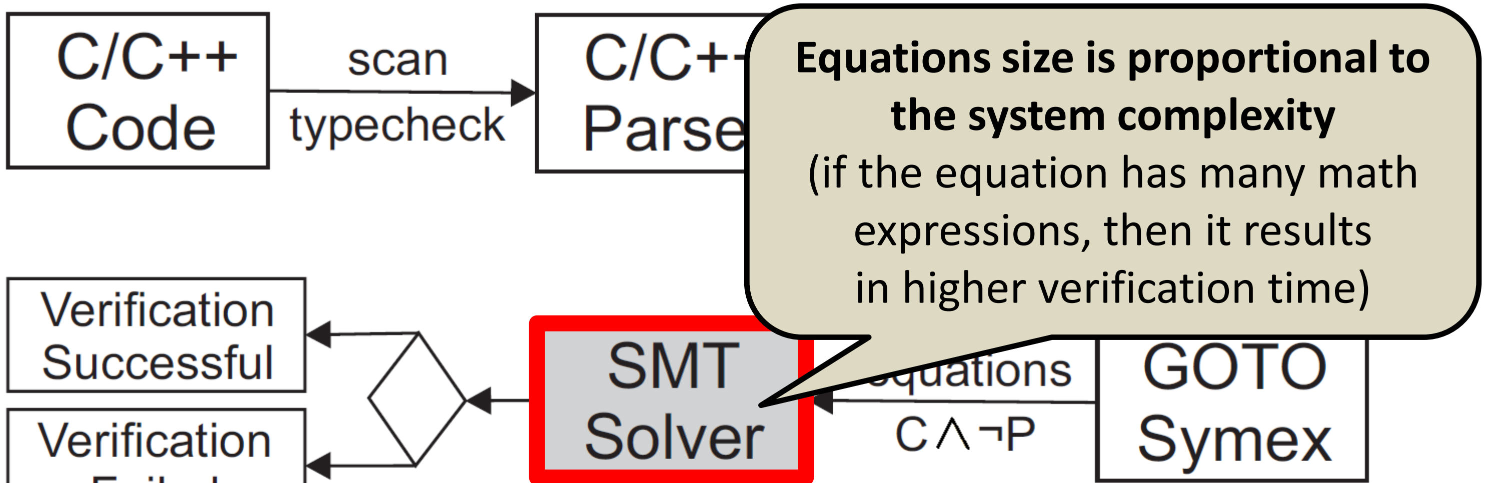

```
#include<stdio.h>
#include<assert.h>

int main(){
   int i = 0;
   for (i=0; i<50;i++){
      assert(i%8==0);
   }
}
```



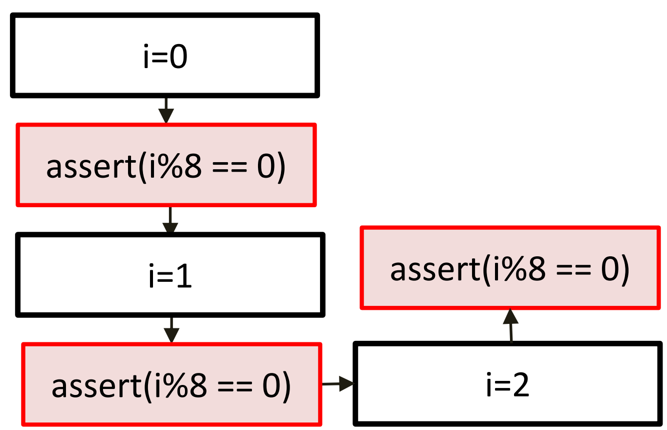
$$C := \begin{bmatrix} i0 := 0 \wedge \\ i1 := 1 \wedge \\ i2 := 2 \end{bmatrix}$$

$$P := \begin{bmatrix} i0\%8 == 0 \wedge \\ i1\%8 == 0 \wedge \\ i2\%8 == 0 \end{bmatrix}$$

ESBMC Architecture



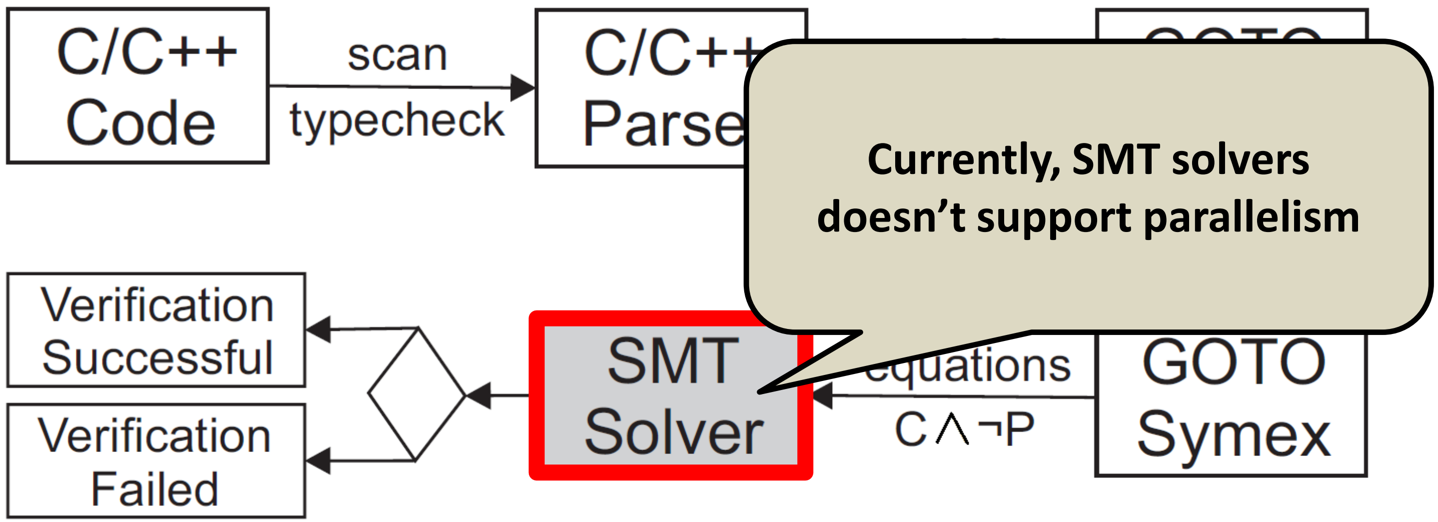

```
#include<stdio.h>
#include<assert.h>

int main(){
   int i = 0;
   for (i=0; i<50;i++){
      assert(i%8==0);
   }
}
```



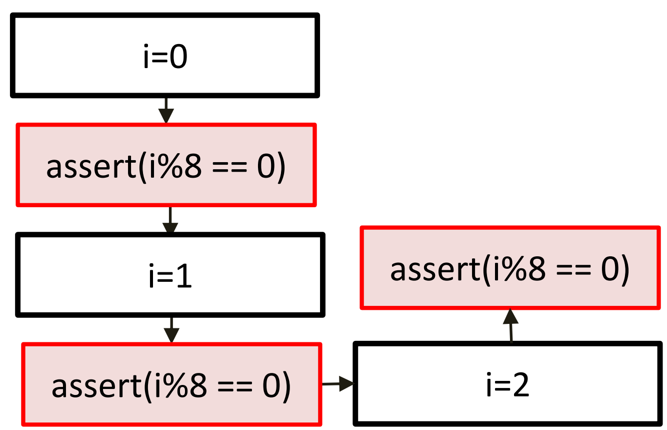
$$C := \begin{bmatrix} i0 := 0 \wedge \\ i1 := 1 \wedge \\ i2 := 2 \end{bmatrix}$$

$$P := \begin{bmatrix} i0\%8 == 0 \wedge \\ i1\%8 == 0 \wedge \\ i2\%8 == 0 \end{bmatrix}$$

ESBMC Architecture




```
#include<stdio.h>
#include<assert.h>

int main(){
   int i = 0;
   for (i=0; i<50;i++){
      assert(i%8==0);
   }
}
```



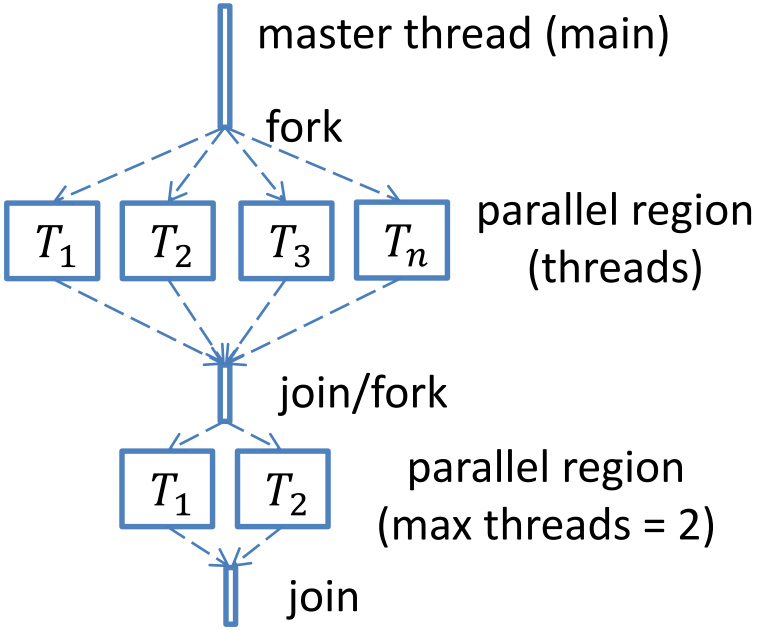
$$C := \begin{bmatrix} i0 := 0 \wedge \\ i1 := 1 \wedge \\ i2 := 2 \end{bmatrix}$$

$$P := \begin{bmatrix} i0\%8 == 0 \wedge \\ i1\%8 == 0 \wedge \\ i2\%8 == 0 \end{bmatrix}$$

Open Multi-Processing

- OpenMP (API) is a set of directives for parallel programming
  - Support for C/C++, and Fortran
  - Support for different operating systems (Windows, Linux, Mac OSX, HP-UX)
- Use the *fork-join* model
  - Threads are managed by the API
  - User customizes the execution
- Compiler directive based:

```
int k;
#pragma omp parallel for
for (k = 0; k < 10; k++)
   a[k] = 2*a[k] ;
```



ESBMC for Optimization

- The first algorithm in ANSI-C for ESBMC solves optimization problems

Initialize variables
Declare number of nodes and edges
Declare hardware cost of each node as array (h)
Declare software cost of each node as array (s)
Declare communication cost of each edge (c)
Declare the initial software cost (S_0)
Declare transposed incidence matrix graph G (E)
Define the solutions variables (x_i) as Boolean
main {
  For $TipH = 0$ to $Hmax$ do {
    Populate x_i with nondeterministic/test values
    Calculate $s(1-x) + c * |Ex|$ and store at $variable$
    Requirement insured by $ASSUME$ $(variable \leq S_0)$
    Calculate H_p cost based on value tested of x_i
    Violation check with ASSERT $(H_p > TipH)$
  }
}

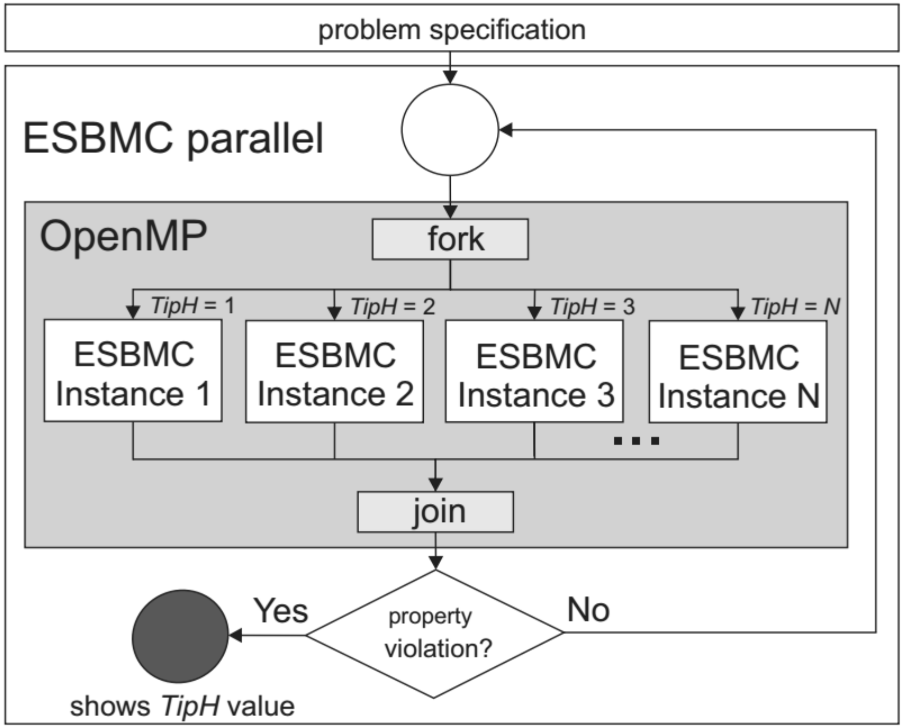
These variables are declared as matrices or vectors

This loop can generate many math equations.
It is hard for the SMT solver

We have CPU's available
Why not parallelize?

Multi-Core ESBMC approach

- Solution: use of OpenMP as front-end of ESBMC
- Use fork-join model provided by OpenMP
- OpenMP API creates N different instances:
  - – Instead of trying to solve the partitioning problem just once, it creates N different problems with different TipH values of hardware cost
- If a violation occurs then the optimal value was found. The threads are finished

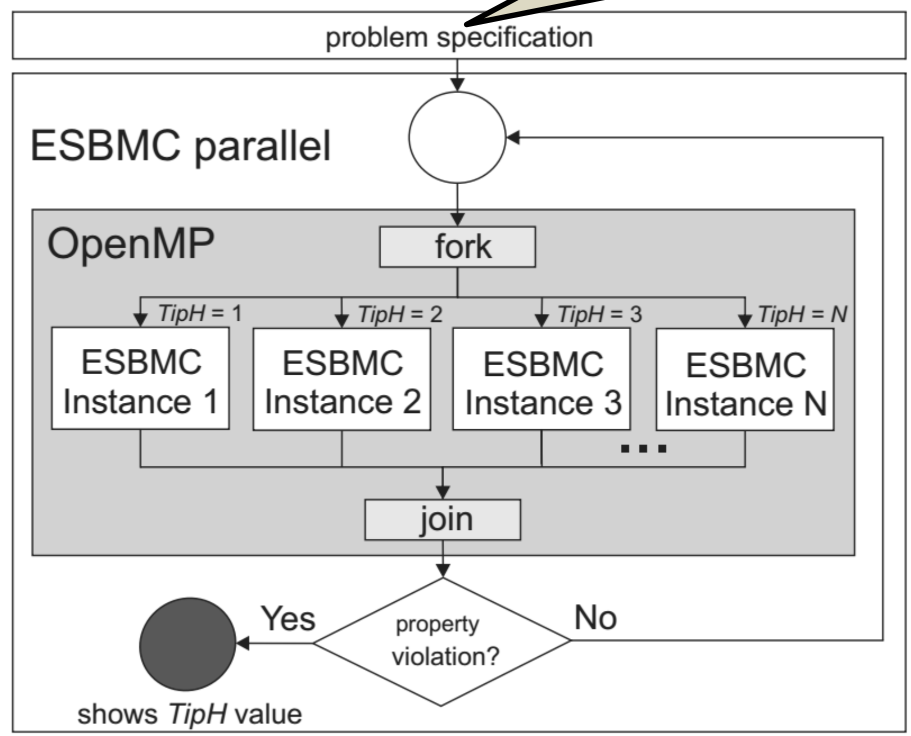


- /esbmc-parallel <$filename.c$> <$hmin\_value$> <$Hmax$>

Multi-Core ESBMC approach

- Solution: use of OpenMP as front-end of ESBMC
- Use fork-join model provided by OpenMP
- OpenMP API creates N different instances:
  - Instead of trying to solve the partitioning problem just once, it creates N different problems with different TipH values of hardware cost
- If a violation occurs then the optimal value was found. The threads are finished

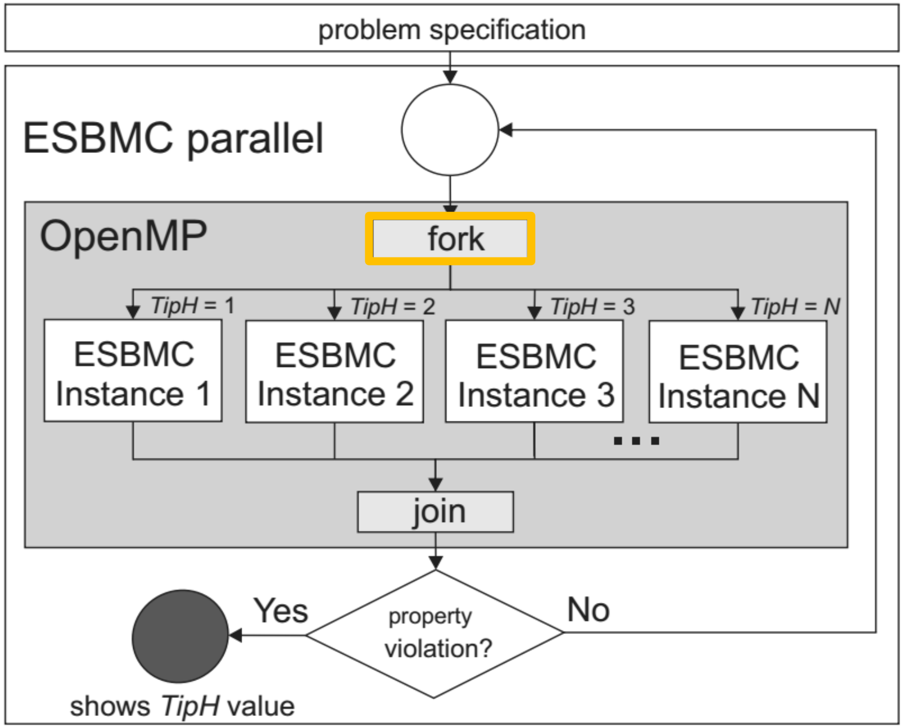
The problem is modeled to expect a TipH parameter



- /esbmc-parallel $<filename.c>$ $<hmin\_value>$ $<Hmax>$

Multi-Core ESBMC approach

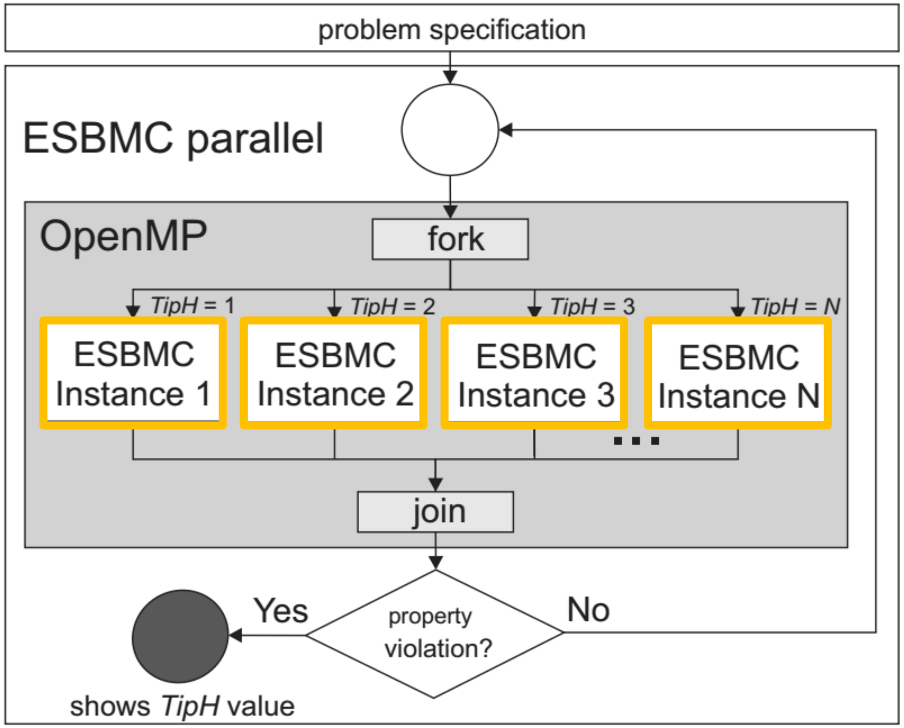
- Solution: use of OpenMP as front-end of ESBMC
- Use fork-join model provided by OpenMP
- OpenMP API creates N different instances:
  - Instead of trying to solve the partitioning problem just once, it creates N different problems with different TipH values of hardware cost
- If a violation occurs then the optimal value was found. The threads are finished



- /esbmc-parallel <$filename.c$> <$hmin\_value$> <$Hmax$>

Multi-Core ESBMC approach

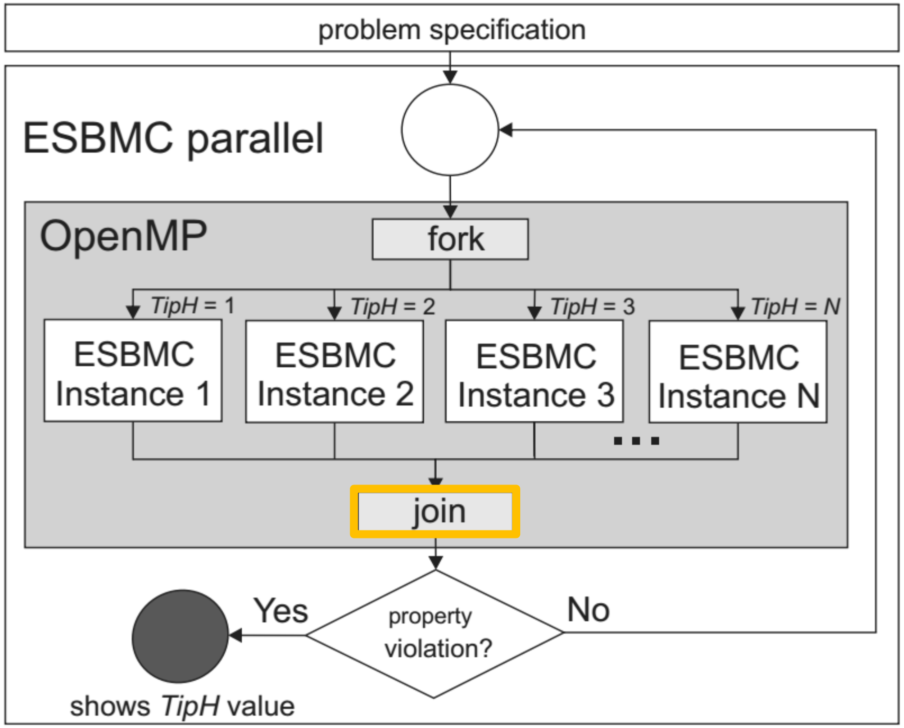
- Solution: use of OpenMP as front-end of ESBMC
- Use fork-join model provided by OpenMP
- OpenMP API creates N different instances:
  - Instead of trying to solve the partitioning problem just once, it creates N different problems with different TipH values of hardware cost
- If a violation occurs then the optimal value was found. The threads are finished



- /esbmc-parallel <*filename.c*> <*hmin_value*> <*Hmax*>

Multi-Core ESBMC approach

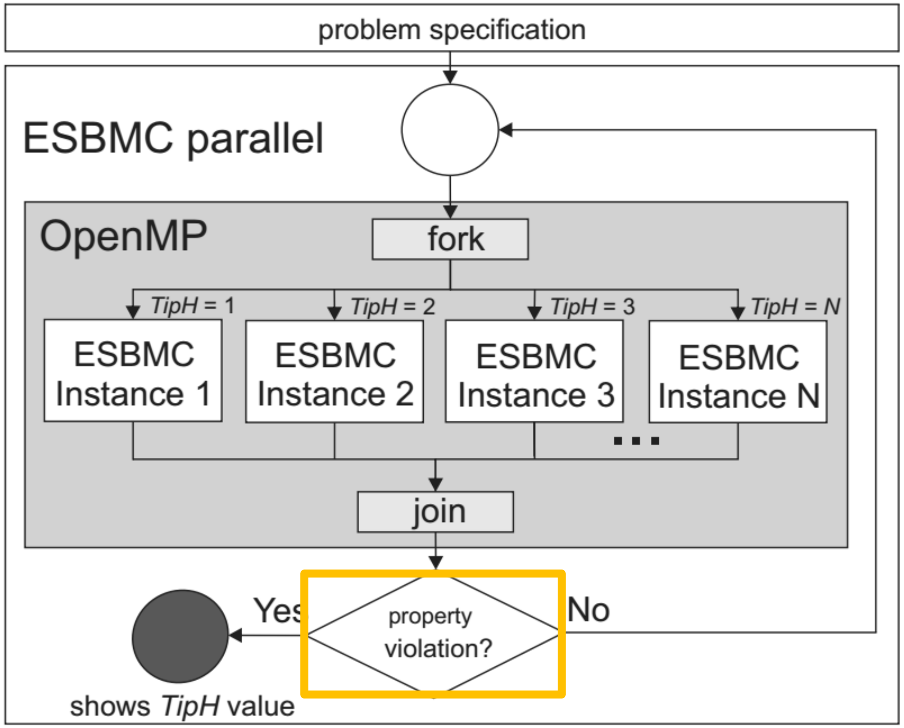
- Solution: use of OpenMP as front-end of ESBMC
- Use fork-join model provided by OpenMP
- OpenMP API creates N different instances:
  - Instead of trying to solve the partitioning problem just once, it creates N different problems with different TipH values of hardware cost
- If a violation occurs then the optimal value was found. The threads are finished



- /esbmc-parallel <$filename.c$> <$hmin\_value$> <$Hmax$>

Multi-Core ESBMC approach

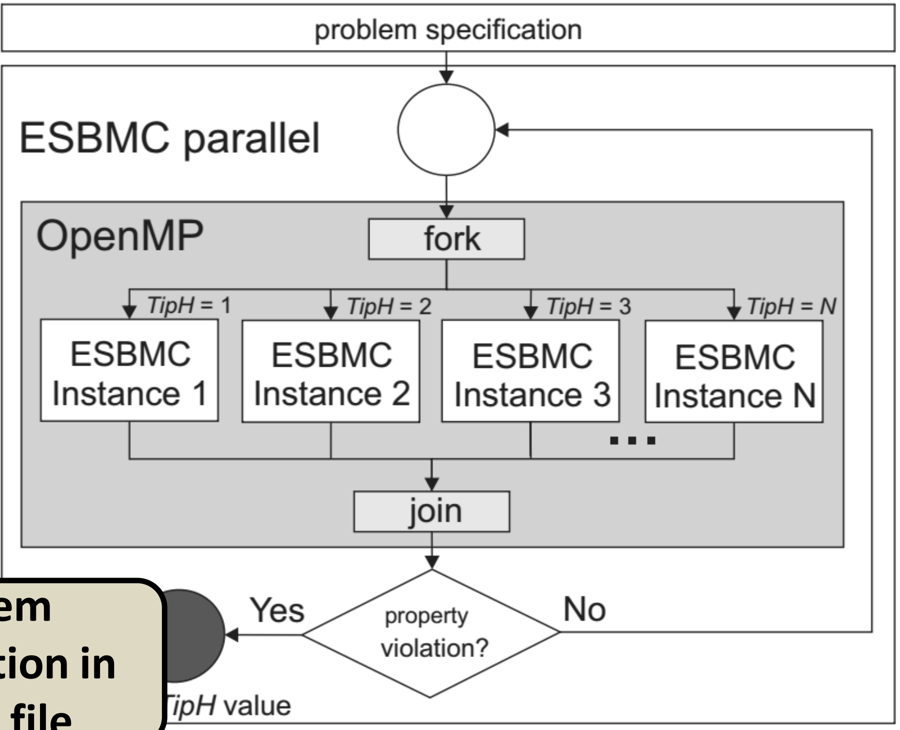
- Solution: use of OpenMP as front-end of ESBMC
- Use fork-join model provided by OpenMP
- OpenMP API creates N different instances:
  - – Instead of trying to solve the partitioning problem just once, it creates N different problems with different TipH values of hardware cost
- If a violation occurs then the optimal value was found. The threads are finished



- /esbmc-parallel <$filename.c$> <$hmin\_value$> <$Hmax$>

Multi-Core ESBMC approach

- Solution: use of OpenMP as front-end of ESBMC
- Use fork-join model provided by OpenMP
- OpenMP API creates N different instances:
  - Instead of trying to solve the partitioning problem just once, it creates N different problems with different TipH values of hardware cost
- If a violation occurs then the optimal value was found. T threads are finished

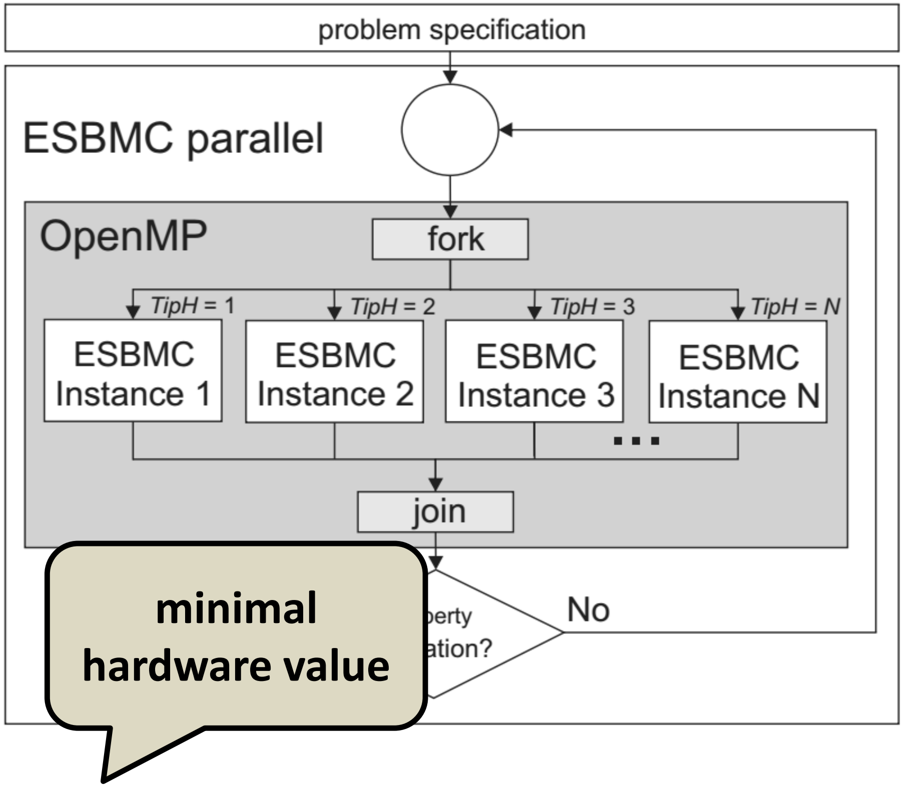


problem specification in ANSI-C file

- /esbmc-parallel <*filename.c*> <*hmin_value*> <*Hmax*>

Multi-Core ESBMC approach

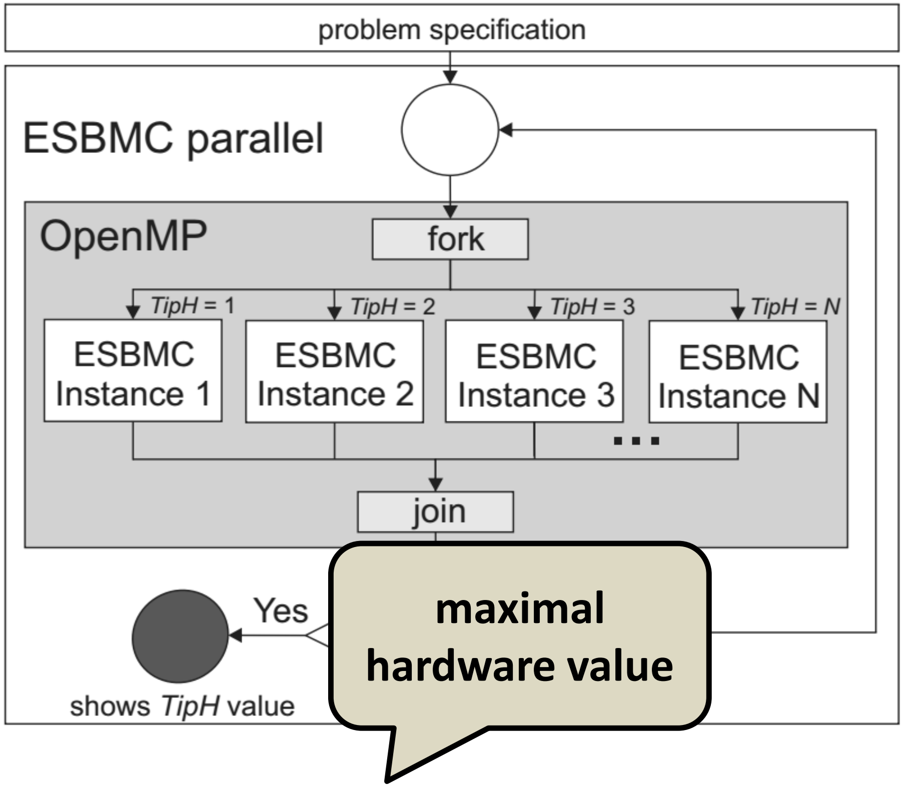
- Solution: use of OpenMP as front-end of ESBMC
- Use fork-join model provided by OpenMP
- OpenMP API creates N different instances:
  - – Instead of trying to solve the partitioning problem just once, it creates N different problems with different TipH values of hardware cost
- If a violation occurs then the optimal value was found. The threads are finished



- /esbmc-parallel $<filename.c>$ $<hmin\_value>$ $<Hmax>$

Multi-Core ESBMC approach

- Solution: use of OpenMP as front-end of ESBMC
- Use fork-join model provided by OpenMP
- OpenMP API creates N different instances:
  - – Instead of trying to solve the partitioning problem just once, it creates N different problems with different TipH values of hardware cost
- If a violation occurs then the optimal value was found. The threads are finished



- /esbmc-parallel <$filename.c$> <$hmin\_value$> <$Hmax$>

Experimental Evaluation

- Set up
  - Desktop with 64-bit Ubuntu 14.04 LTS, 15GB of RAM and i7 Intel (8-cores) processor with 3.40 GHz of clock
  - ESBMC v1.24
  - SMT solver: Boolector v. 2.0.1
  - MathWorks MATLAB R2013a (GA and ILP)
  - Time out (TO) = 7200 sec
  - Memory out (MO) = 15GB
- Use 7 benchmarks (with different number of nodes)
- Compare with ESBMC, ESBMC Multi-Core, ILP, and GA
- Each time is the average of three measured times
  - (92% of statistical confidence)

Experimental Evaluation

- Set up
  - Desktop with 64-bit Ubuntu 14.04 LTS, 15GB of RAM and i7 Intel (8-cores) processor with 3.40 GHz of clock
  - ESBMC v1.24
  - SMT solver: Boolector v. 2.0.1
  - MathWorks MATLAB R2013a (GA and ILP)
  - Time out (TO) = 7200 sec
  - Memory out (MO) = 15GB

- Use 7 benchmarks (with different number of nodes)
- Compare with ESBMC, ESBMC Multi-Core, ILP, and GA
- Each time is the average of three measured times
  - (92% of statistical confidence)

Experimental Evaluation

- Set up
  - Desktop with 64-bit Ubuntu 14.04 LTS, 15GB of RAM and i7 Intel (8-cores) processor with 3.40 GHz of clock
  - ESBMC v1.24
  - SMT solver: Boolector v. 2.0.1
  - MathWorks MATLAB R2013a (GA and ILP)
  - Time out (TO) = 7200 sec
  - Memory out (MO) = 15GB

- Use 7 benchmarks (with different number of nodes)
- Compare with ESBMC, ESBMC Multi-Core, ILP, and GA
- Each time is the average of three measured times
  - (92% of statistical confidence)

Results

		CRC32	Patricia	Dijkstra	Clustering	RC6	Fuzzy	Mars
	Nodes	25	21	26	150	329	261	417
	Edges	32	48	69	331	448	422	600
	S_0	20	10	20	50	600	4578	300
Exact Solution	Hp	15	47	31	241	692	13820	876
	Sp	19	4	19	46	533	4231	297
ILP	Time(s)	2	1	2	649	1806	TO	5429
	Hp	15	47	31	241	692	-	876
GA	Time(s)	7	7	9	340	2050	1372	5000
	Error	13%	0,0%	29,0%	2%	-7%	-38%	-28%
ESBMC	Time(s)	30	314	325	MO	MO	MO	MO
	Hp	15	47	31	-	-	-	-
Multi-core ESBMC	Time(s)	2	6	7	1609	TO	TO	TO
	Hp	15	47	31	241	-	-	-
ESBMC Relative Speedup		14	54	47		-	-	-

Results

Bechmark

		CRC32	Patricia	Dijkstra	Clustering	RC6	Fuzzy	Mars
	Nodes	25	21	26	150	329	261	417
	Edges	32	48	69	331	448	422	600
	S_0	20	10	20	50	600	4578	300
Exact Solution	Hp	15	47	31	241	692	13820	876
	Sp	19	4	19	46	533	4231	297
ILP	Time(s)	2	1	2	649	1806	TO	5429
	Hp	15	47	31	241	692	-	876
GA	Time(s)	7	7	9	340	2050	1372	5000
	Error	13%	0,0%	29,0%	2%	-7%	-38%	-28%
ESBMC	Time(s)	30	314	325	MO	MO	MO	MO
	Hp	15	47	31	-	-	-	-
Multi-core ESBMC	Time(s)	2	6	7	1609	TO	TO	TO
	Hp	15	47	31	241	-	-	-
ESBMC Relative Speedup		14	54	47		-	-	-

Results

Initial Software Cost

		CRC32	Patricia	Dijkstra	Clustering	RC6	Fuzzy	Mars
	Nodes	25	21	26	150	329	261	417
	Edges	32	48	69	331	448	422	600
	S_0	20	10	20	50	600	4578	300
Exact Solution	Hp	15	47	31	241	692	13820	876
	Sp	19	4	19	46	533	4231	297
ILP	Time(s)	2	1	2	649	1806	TO	5429
	Hp	15	47	31	241	692	-	876
GA	Time(s)	7	7	9	340	2050	1372	5000
	Error	13%	0,0%	29,0%	2%	-7%	-38%	-28%
ESBMC	Time(s)	30	314	325	MO	MO	MO	MO
	Hp	15	47	31	-	-	-	-
Multi-core ESBMC	Time(s)	2	6	7	1609	TO	TO	TO
	Hp	15	47	31	241	-	-	-
ESBMC Relative Speedup		14	54	47		-	-	-

Results

Hardware Partitioned Cost (solution)

		CRC32	Patricia	Dijkstra	Clustering	RC6	Fuzzy	Mars
	des	25	21	26	150	329	261	417
	Edges	32	48	69	331	448	422	600
	S_0	20	10	20	50	600	4578	300
Exact Solution	Hp	15	47	31	241	692	13820	876
	Sp	19	4	19	46	533	4231	297
ILP	Time(s)	2	1	2	649	1806	TO	5429
	Hp	15	47	31	241	692	-	876
GA	Time(s)	7	7	9	340	2050	1372	5000
	Error	13%	0,0%	29,0%	2%	-7%	-38%	-28%
ESBMC	Time(s)	30	314	325	MO	MO	MO	MO
	Hp	15	47	31	-	-	-	-
Multi-core ESBMC	Time(s)	2	6	7	1609	TO	TO	TO
	Hp	15	47	31	241	-	-	-
ESBMC Relative Speedup		14	54	47		-	-	-

Results

Software Partitioned Cost (solution)

		CRC32	Patricia	Dijkstra	Clustering	RC6	Fuzzy	Mars
	des	25	21	26	150	329	261	417
	ges	32	48	69	331	448	422	600
	$_0$	20	10	20	50	600	4578	300
Exact Solution	Hp	15	47	31	241	692	13820	876
	Sp	19	4	19	46	533	4231	297
ILP	Time(s)	2	1	2	649	1806	TO	5429
	Hp	15	47	31	241	692	-	876
GA	Time(s)	7	7	9	340	2050	1372	5000
	Error	13%	0,0%	29,0%	2%	-7%	-38%	-28%
ESBMC	Time(s)	30	314	325	MO	MO	MO	MO
	Hp	15	47	31	-	-	-	-
Multi-core ESBMC	Time(s)	2	6	7	1609	TO	TO	TO
	Hp	15	47	31	241	-	-	-
ESBMC Relative Speedup		14	54	47		-	-	-

Results

4 approaches

		CRC32	Patricia	Dijkstra	Clustering	RC6	Fuzzy	Mars
	Nodes	25	21	26	150	329	261	417
	Edges	32	48	69	331	448	422	600
	S_0	20	10	20	50	600	4578	300
[illegible]ct Sc[illegible]tion	Hp	15	47	31	241	692	13820	876
	Sp	19	4	19	46	533	4231	297
ILP	Time(s)	2	1	2	649	1806	TO	5429
	Hp	15	47	31	241	692	-	876
GA	Time(s)	7	7	9	340	2050	1372	5000
	Error	13%	0,0%	29,0%	2%	-7%	-38%	-28%
ESBMC	Time(s)	30	314	325	MO	MO	MO	MO
	Hp	15	47	31	-	-	-	-
Multi-core ESBMC	Time(s)	2	6	7	1609	TO	TO	TO
	Hp	15	47	31	241	-	-	-
ESBMC Relative Speedup		14	54	47		-	-	-

Results

Best Performance

		CRC32	Patricia	Dijkstra	Clustering	RC6	Fuzzy	Mars
	Nodes	25	21	26	150	329	261	417
	Edges	32	48	69	331	448	422	600
	S_0	20	10	20	50	600	4578	300
...ct So...tion	Hp	15	47	31	241	692	13820	876
	Sp	19	4	19	46	533	4231	297
ILP	Time(s)	2	1	2	649	1806	TO	5429
	Hp	15	47	31	241	692	-	876
GA	Time(s)	7	7	9	340	2050	1372	5000
	Error	13%	0,0%	29,0%	2%	-7%	-38%	-28%
ESBMC	Time(s)	30	314	325	MO	MO	MO	MO
	Hp	15	47	31	-	-	-	-
Multi-core ESBMC	Time(s)	2	6	7	1609	TO	TO	TO
	Hp	15	47	31	241	-	-	-
ESBMC Relative Speedup		14	54	47		-	-	-

Results

		CRC32	Patricia	Dijkstra	Clustering	RC6	Fuzzy	Mars
	Nodes	25	21	26	150	329	261	417
	Edges	32	48	69	331	448	422	600
	S_0	20	10	20	50	600	4578	300
Exact	Hp	15	47	31	241	692	13820	876
	Sp	19	4	19	46	533	4231	297
IL	Time(s)	2	1	2	649	1806	TO	5429
	Hp	15	47	31	241	692	-	876
GA	Time(s)	7	7	9	340	2050	1372	5000
	Error	13%	0,0%	29,0%	2%	-7%	-38%	-28%
ESBMC	Time(s)	30	314	325	MO	MO	MO	MO
	Hp	15	47	31	-	-	-	-
Multi-core ESBMC	Time(s)	2	6	7	1609	TO	TO	TO
	Hp	15	47	31	241	-	-	-
ESBMC Relative Speedup		14	54	47		-	-	-

Solved all, but with errors

Results

		CRC32	Patricia	Dijkstra	Clustering	RC6	Fuzzy	Mars
	Nodes	25	21	26	150	329	261	417
	Edges	32	48	69	331	448	422	600
	S_0	20	10	20	50	600	4578	300
Exact Solution	Hp	15	47	31	241	692	13820	876
	Sp	19	4	19	46	533	4231	297
ILP	Time(s)	2	1	2	649	1806	TO	5429
	Hp	15	47	31	241	692	-	876
	Time(s)	7	7	9	340	2050	1372	5000
	Error	13%	0,0%	29,0%	2%	-7%	-38%	-28%
ESBMC	Time(s)	30	314	325	MO	MO	MO	MO
	Hp	15	47	31	-	-	-	-
Multi-core ESBMC	Time(s)	2	6	7	1609	TO	TO	TO
	Hp	15	47	31	241	-	-	-
ESBMC Relative Speedup		14	54	47		-	-	-

Worst Performance

Results

		CRC32	Patricia	Dijkstra	Clustering	RC6	Fuzzy	Mars
	Nodes	25	21	26	150	329	261	417
	Edges	32	48	69	331	448	422	600
	S_0	20	10	20	50	600	4578	300
Exact Solution	Hp	15	47	31	241	692	13820	876
	Sp	19	4	19	46	533	4231	297
ILP	Time(s)	2	1	2	649	1806	TO	5429
	Hp	15	47	31	241	692	-	876
GA	Time(s)	7	7	9	340	2050	1372	5000
	Error	13%	0,0%	29,0%	2%	-7%	-38%	-28%
E[illegible]MC	Time(s)	30	314	325	MO	MO	MO	MO
	Hp	15	47	31	-	-	-	-
Multi-core ESBMC	Time(s)	2	6	7	1609	TO	TO	TO
	Hp	15	47	31	241	-	-	-
ESBMC Relative Speedup		14	54	47		-	-	-

Speedup over ESBMC

Conclusions

- 1st generation of co-design:
  - Above 400 nodes: none
  - Until 400 nodes: ILP
  - Until 150 nodes: ESBMC
  - GA (error issues)
- ILP e GA: easier to use but ESBMC: no cost (BSD license)
- MC-ESBMC has better performance than Sequential ESBMC (speedup from 14 until 54 and no memory out)
- 150 nodes is a realistic problem? All depends on the granularity of problem modeling

Future Work

- ESBMC: study the possibilities to decrease the time to solution (solver included)
- Use of ESBMC to more complex types of architecture, including more then one CPU (2nd generation of co-design)

Conclusions

- 1st generation of co-design:
  - Above 400 nodes: none
  - Until 400 nodes: ILP
  - Until 150 nodes: ESBMC
  - GA (error issues)
- ILP e GA: easier to use but ESBMC: no cost (BSD license)
- MC-ESBMC has better performance than Sequential ESBMC (speedup from 14 until 54 and no memory out)
- 150 nodes is a realistic problem? All depends on the granularity of problem modeling

Future Work

- ESBMC: study the possibilities to decrease the time to solution (solver included)
- Use of ESBMC to more complex types of architecture, including more then one CPU (2nd generation of co-design)

Conclusions

- 1st generation of co-design:
  - Above 400 nodes: none
  - Until 400 nodes: ILP
  - Until 150 nodes: ESBMC
  - GA (error issues)
- ILP e GA: easier to use but ESBMC: no cost (BSD license)
- MC-ESBMC has better performance than Sequential ESBMC (speedup from 14 until 54 and no memory out)
- 150 nodes is a realistic problem? All depends on the granularity of problem modeling

Future Work

- ESBMC: study the possibilities to decrease the time to solution (solver included)
- Use of ESBMC to more complex types of architecture, including more then one CPU (2nd generation of co-design)

Conclusions

- 1^{st} generation of co-design:
  - Above 400 nodes: none
  - Until 400 nodes: ILP
  - Until 150 nodes: ESBMC
  - GA (error issues)
- ILP e GA: easier to use but ESBMC: no cost (BSD license)
- MC-ESBMC has better performance than Sequential ESBMC (speedup from 14 until 54 and no memory out)
- 150 nodes is a realistic problem? All depends on the granularity of problem modeling

Future Work

- ESBMC: study the possibilities to decrease the time to solution (solver included)
- Use of ESBMC to more complex types of architecture, including more then one CPU (2^{nd} generation of co-design)

Conclusions

- 1st generation of co-design:
  - Above 400 nodes: none
  - Until 400 nodes: ILP
  - Until 150 nodes: ESBMC
  - GA (error issues)
- ILP e GA: easier to use but ESBMC: no cost (BSD license)
- MC-ESBMC has better performance than Sequential ESBMC (speedup from 14 until 54 and no memory out)
- 150 nodes is a realistic problem? All depends on the granularity of problem modeling

Future Work

- ESBMC: study the possibilities to decrease the time to solution (solver included)
- Use of ESBMC to more complex types of architecture, including more then one CPU (2nd generation of co-design)

Thank you for your attention!

Contacts:
alessandro.b.trindade@gmail.com
hussamaismail@gmail.com
lucascordeiro@ufam.edu.br