Cardiac conduction modeling in Hybrid Input Output Automata (HIOA)

Partha Roop, Avinash Malik, Sidharta Andalam

THE UNIVERSITY OF AUCKLAND
NEW ZEALAND
Te Whare Wānanga o Tāmaki Makaurau

BioRemulation™ Research Group
The University of Auckland

December 2016

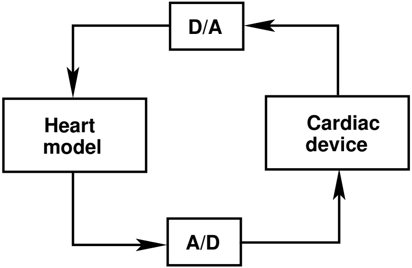
www.pretzel.ece.auckland.ac.nz/bio

Outline

Learning outcomes

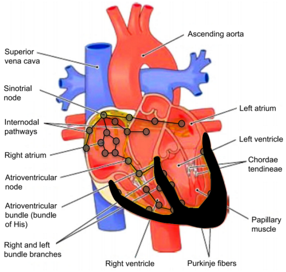
1. Understand the distinction between medical Cyber-physical Systems (CPS) and the discrete medical devices.
2. Understand the challenges in designing medical CPS.
3. Appreciate the challenges in code generation from "hybrid real-time systems".
4. Exposure to the synchronous approach and its benefits for code generation.

The challenges in modeling and closed loop validation of cardiac devices

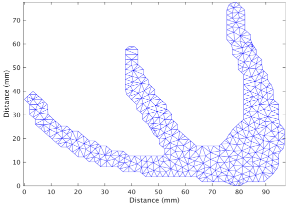


1. Need to validate the pacemaker in *closed loop* with a heart model.
2. Pacemaker is a discrete device.
3. Heart model is modal + continuous.
4. Heart model needs to execute in *real-time*, i.e., at 60-120 bpm.
5. We need to *efficiently* execute the heart model.

The heart model



(a) The heart model



(b) Triangulated ventricular myocardium

1. The nodes in Figure 1(a) indicate the fast conduction pathway.
2. The heart pulse starts from the Sinoatrial (SA) node (the so called natural pacemaker).
3. Travels first to the left and right atria.
4. Small delay at Atrioventricular (AV) node to let blood fill into ventricles.
5. Blood pumped out of ventricles in to the body.
6. The dark area indicates the ventricular myocardium wall.
7. Ventricular myocardium *approximated* using a triangular mesh (Figure 1(b)).
8. Each vertex of the triangle models a heart node.
9. Each edge models the pathway between two nodes.

Behavior of the heart node – the Action Potential (AP)

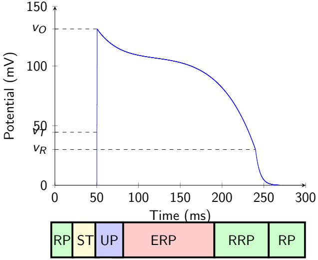
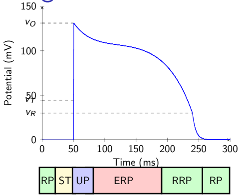


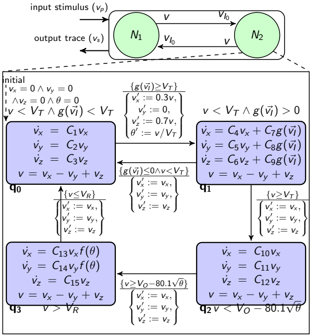
Figure: The AP of a heart node

1. Each node in the heart generates an electrical impulse.
2. This electrical impulse is called an Action Potential (AP).
3. The AP (Figure 1) has five stages.
4. Resting Period (RP): This is the steady state, when the node is awaiting activation by an external stimulus.
5. Stimulated (ST): When the external stimulus is above a threshold voltage (V_T).
6. Upstroke (UP): After continuous stimulation, the cell's voltage reaches the threshold voltage V_T, leading to a stimulus that activates neighboring nodes.
7. Effective Refractory Period (ERP): Once activated, the node cannot be activated again due to the recovery process of the ionic channels. Any new stimulus will be blocked during this refractory period.

Capturing the AP as a HIOA



(a) The AP



(b) The HIOA capturing the AP

1. Each phase of AP captured as a so called location in the HIOA.
2. Location $\mathbf{q_0}$ captures RP, $\mathbf{q_1}$ captures ST, $\mathbf{q_2}$ captures UP, and $\mathbf{q_3}$ captures ERP.
3. Continuous variables v_x, v_y, v_z evolve via Ordinary Differential Equations (ODEs) in each location, capturing the morphology of the AP.
4. Overall AP is captured using voltage variable v.
5. HIOA remains in a location until location invariant (e.g., $v < V_T \wedge g(\vec{v_l}) < V_T$) *holds*.
6. A transition is made *instantaneously* to another location upon violation of the location invariant and as long as the edge *guard* holds.
7. Actions may be performed upon taking the transition.

The overall compilation approach

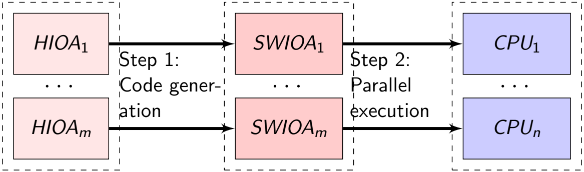


Figure: The suggested compilation approach for HIOA

Compilation approach

1. Compile each HIOA individually into a so called Synchronous Witness Input Output Automata (SWIOA).
2. Leverage synchronous model of computation to execute the individual SWIOA in parallel on multi-core systems.

A Hybrid Input Output Automata (HIOA) is
$\mathcal{H} = \langle Loc, X, V, Y, Init, f, h, Inv, E, G, R\rangle$, where:

- Loc is a finite collection of discrete locations.
- X is a finite collection of continuous state variables, with its domain represented as $\mathbf{X}$.
- V is a finite collection of input variables. We assume $V = V_D \cup V_C$, where V_D are discrete inputs and V_C are continuous inputs, with their domains $\mathbf{V}_D$, $\mathbf{V}_C$, and $\mathbf{V}$, respectively.
- Y is a finite collection of output variables. We assume that $Y = Y_D \cup Y_C$, where Y_D is a collection of discrete output variables and Y_C is a collection of continuous output variables, with their respective domains $\mathbf{Y}_D$, $\mathbf{Y}_C$, and $\mathbf{Y}$.
- $Init \subseteq \{l\} \times \mathbf{X} \times \mathbf{Y}$ such that there is exactly one $l \in Loc$, is the singleton initial state.

- $f : Loc \times \mathbf{X} \times \mathbf{V} \rightarrow \mathbb{R}^n$ is a vector field. Function $f(l, x, v)$ is globally *Lipschitz* continuous in $x \in \mathbf{X}$ and $v \in \mathbf{V}$.
- $h : Loc \times \mathbf{X} \rightarrow \mathbf{Y}$ is a vector field. Function $h(l, x)$ is globally *Lipschitz* continuous in $x \in \mathbf{X}$.
- $Inv : Loc \rightarrow 2^{\mathbf{X} \times \mathbf{V}}$ assigns to each $l \in Loc$ an invariant set.
- $E \subset Loc \times Loc$ is a collection of discrete edges.
- $G : E \rightarrow 2^{\mathbf{X} \times \mathbf{V}}$ assigns to each $e = (l, l') \in E$ a guard.
- $R : E \times \mathbf{X} \times \mathbf{V} \rightarrow 2^{\mathbf{X}}$ assigns to each $e = (l, l') \in E$, $x \in \mathbf{X}$, $v \in \mathbf{V}$ a reset relation.

Formally capturing the heart node HIOA

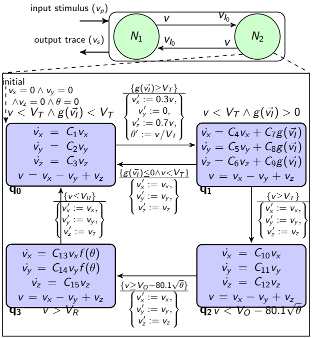


Figure: The heart node HIOA

1. $Loc = \{q_0, q_1, q_2, q_3\}$
2. $X = \{v_x, v_y, v_z, \theta\}$, with $\mathbf{X} = \mathbb{R}^4$
3. $V = V_C = \{\vec{v_l}\}$, with $\mathbf{V} = \mathbb{R}^{\|\vec{v_l}\|}$
4. $Y = Y_C = \{v\}$, with $\mathbf{Y} = \mathbb{R}$
5. $Init = \{q_0\} \times \{(0, 0, 0, 0)\} \times \{0\}$
6. An example of vector field evolving the continuous variables is given in Equation (1)
7. An example update function (h) updating the output variable v in location q_0 is given in Equation (2)
8. Location invariants and edge guards and relations are as shown in Figure 11

$$f(q_0, v_x, v_y, v_z, \theta) = \begin{bmatrix} \dot{v_x} \\ \dot{v_y} \\ \dot{v_z} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} C_1 \times v_x \\ C_2 \times v_y \\ C_3 \times v_z \\ 0 \end{bmatrix} \quad (1)$$

$$h(q_0, v_x, v_y, v_z, \theta) = v = v_x - v_y + v_z \quad (2)$$

Semantics of HIOA – Time Transition System (TTS)

The semantics of a HIOA $\mathcal{H}$ is defined by a timed transition system $TTS = \langle Q, q_0, \mathbf{V}, \mathbf{Y}, \rightarrow \rangle$.

- Q is of the form (l, x) where l is a location, $x \in \mathbf{X}$, and $v \in \mathbf{V}$ such that (x, v) satisfies $Inv(l)$. Q is called the state-space of $\mathcal{H}$.
- $q_0 \in Q$ of the form (l, x) and $y \in \mathbf{Y}$ such that $(l, x, y) \in Init$ is the initial state.
- $\rightarrow$ is the set of transitions consisting of either:
  1. Discrete transitions (instantaneous): For each edge $e = (l, l') \in E$, $x \in \mathbf{X}$, $v \in \mathbf{V}$, $y \in \mathbf{Y}$ we have $(l, x) \xrightarrow[x' \in R(e,x,v)]{(x,v) \in G(e)} (l', x')$ if $(l, x) \in Q$, $(l', x') \in Q$.
  2. Continuous transition (delay): For each non-negative real δ, we have $(l, x) \xrightarrow{\delta} (l, x')$ if $(l, x) \in Q$, $(l, x') \in Q$, and there is a differentiable function $F(t) = \int_0^\delta f(l, x, v)\ \forall t \in [0, \delta]$ and $\forall v \in \mathbf{V}', \mathbf{V}' \subseteq \mathbf{V}$, called the witness function, such that the following conditions hold:
     - $F(0) = x$ and $F(\delta) = x'$,
     - for all $\epsilon \in (0, \delta), (v, F(\epsilon)) \in Inv(l)$.
     - for all $\epsilon \in [0, \delta]$, $y \in \mathbf{Y}$ satisfies $h(l, F(\epsilon))$

Generating Synchronous Witness Input Output Automata (SWIOA) from HIOA

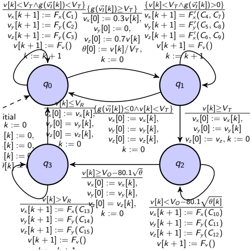


Figure: SWIOA of a heart node.

Discretized witness functions

$$\begin{bmatrix} F_x(C) \\ F_y(C) \\ F_z(C) \\ F_v() \end{bmatrix} = \begin{bmatrix} v_x[k] + \delta \times C \times v_x[k] \\ v_y[k] + \delta \times C \times v_y[k] \\ v_z[k] + \delta \times C \times v_z[k] \\ v_x[k] + v_y[k] + v_z[k] \end{bmatrix} \tag{3}$$

1. All ODEs are replaced by their numerical (Euler) solutions – the *witness functions*.
2. $k \in \mathbb{N}$ is called the logical tick – a discrete instant.
3. All ODEs values are computed at every k^{th} logical tick.
4. $\delta \in \mathbb{R}$ is called the *step size* of the solver, i.e., the time between two ticks.

SWIOA continued

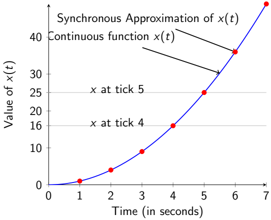


Figure: An example visualization of the synchronous approximation. $\delta = 1$ sec.

Discretized guards and actions

$$\frac{\{v[k] < V_T \wedge g(\vec{v_i}[k]) < V_T\}}{\begin{array}{c} v_x[k+1] := F_x(C_1) \\ v_y[k+1] := F_y(C_2) \\ v_z[k+1] := F_z(C_3) \\ v[k+1] := F_v() \\ k := k+1 \end{array}}$$

1. The discretized ODE solving is inspired by the synchronous model of computation (see Figure 5).
2. Every location is converted to a state in the Finite State Machine (FSM).
3. Continuous transitions, from TTS are converted to self-transitions on the SWIOA, e.g., $\mathbf{q}_0$ to $\mathbf{q}_0$ with guards and actions as shown on the left.
4. Every edge from the HIOA is present in the resultant SWIOA.
5. Guards *always* check the current tick's values.
6. Actions *always* update the next tick's values.

The semantics of a HIOA $\mathcal{S}$ is a *DiscreteTimeTransitionSystem*(DTTS) $S = \langle Q, q_0, \mathbf{V}, \mathbf{Y}, \rightarrow \rangle$ where

- The state-space is Q, where any state is of the form (l, x, i, k) where l is a location, i is the initial valuation of the variables when execution begins in the location and $x[k] \in \mathbf{X}$ is the valuation at the k-th instant.
- Initial state $q_0 \in Q$ is of the form $(l, x, i, 0)$ such that $(l, x, y) \in Init$, where $y \in \mathbf{Y}$.
- Transitions $(\rightarrow)$ are of two types:
  - *Inter-location transitions* that lead to mode switches: These are of the form $(l, x[k], i, k) \xrightarrow[(x[0]) \in R(e, x[k], v[k])]{(x[k], v[k]) \in G(e) \wedge \delta} (l', x[0], i', 0)$ if $(l, x[k], i, k) \in Q$, $(l', x[k+1], i', 0) \in Q$, $e = (l, l') \in E$, $x \in \mathbf{X}$, $y \in \mathbf{Y}$, $v \in \mathbf{V}$, $i' = x[0]$.

The DTTS semantics of HIOA II

- *Intra-location transitions* made during the execution in a given mode / location: These are of the form $(l, x[k], i, k) \xrightarrow[y[k+1] \in h(l, x[k])]{(v[k], x[k]) \in Inv(l) \wedge \delta} (l, x[k+1], i, k+1)$ if $(l, x[k], i, k) \in Q$, $(l, x[k+1], i, k+1) \in Q$, $Switness(l, k, \delta, i) = x[k]$ and $Switness(l, k+1, \delta, i) = x[k+1]$, $y \in \mathbf{Y}$.

Remark

- All guards read the current tick's value of the variables.
- All the actions update the next tick's value of the variables.
- Intra **and** Inter location transitions consume time – **opposed** to TTS semantics of a HIOA.

Question

- Heart model has thousands (3081 in our case) nodes, *each* represented as a HIOA.
- How to compile and *compose* this *network* of HIOAs?
- How to *efficiently* execute the resultant composition?

Question

- Heart model has thousands (3081 in our case) nodes, *each* represented as a HIOA.
- How to compile and *compose* this *network* of HIOAs?
- How to *efficiently* execute the resultant composition?
- The currently proposed idea is to *syntactically* compose the network of HIOAs into a single HIOA, then apply the compilation semantics described before.
- What is the problem with syntactic composition?

Question

- Heart model has thousands (3081 in our case) nodes, *each* represented as a HIOA.
- How to compile and *compose* this *network* of HIOAs?
- How to *efficiently* execute the resultant composition?
- The currently proposed idea is to *syntactically* compose the network of HIOAs into a single HIOA, then apply the compilation semantics described before.
- What is the problem with syntactic composition?
- State space explosion
- For the heart model we need to build $2^{3081 \times 4}$ states and edges!

Composition Semantics of HIOAs I

Two HIOAs $\mathcal{S}_1$ and $\mathcal{S}_2$ are compatible if:

$$(Loc_1 \cup X_1) \cap (Loc_2 \cup X_2 \cup V_2 \cup Y_2) = \emptyset \tag{4}$$

$$(Loc_2 \cup X_2) \cap (Loc_1 \cup X_1 \cup V_1 \cup Y_1) = \emptyset \tag{5}$$

$$Y_1 \cap Y_2 = \emptyset \tag{6}$$

$$V_1 = V_{11} \cup V_{12}, \text{with } V_{11} = V_1 \backslash Y_2 \text{ and } V_{12} = V_1 \cap Y_2 \tag{7}$$

$$V_2 = V_{21} \cup V_{22}, \text{with } V_{22} = V_2 \backslash Y_1 \text{ and } V_{21} = V_2 \cap Y_1 \tag{8}$$

Given two compatible HIOAs, $\mathcal{S}_1$ and its semantics $S_1 = \langle Q_1, q_1, \mathbf{V}_1, \mathbf{Y}_1, \rightarrow_1 \rangle$ and $\mathcal{S}_2$ and its semantics $S_2 = \langle Q_2, q_2, \mathbf{V}_2, \mathbf{Y}_2, \rightarrow_2 \rangle$
$S_1 || S_2 = S : \langle Q, q, \mathbf{V}, \mathbf{Y} \rightarrow \rangle$ where:

- The state-space is $Q \subseteq Q_1 \times Q_2$.
- $q = (q_1, q_2)$.
- Transitions $\rightarrow$ are of the following types:

- *Inter-location transitions* of the form:

`(Rule Inter-Inter)`

$(q_1, q_2) \xrightarrow[(x_1[0]) \in R_1(e_1, x_1[k_1], \{v_1[k_1] \cup v_1'[k_1]\}) \wedge (x_2[0]) \in R_2(e_2, x_2[k_2], \{v_2[k_2] \cup v_2'[k_2]\})]{(x_1[k_1], \{v_1[k_1] \cup v_1'[k_1]\}) \in G_1(e_1) \wedge (x_2[k_2], \{v_2[k_2] \cup v_2'[k_2]\}) \in G_2(e_2) \wedge \delta}$ (q_1', q_2') where $q_1 = (l_1, x_1[k_1], i_1, k_1)$, $q_2 = (l_2, x_2[k_2], i_2, k_2)$, $q_1' = (l_1', x_1[0], i_1', 0)$, $q_2' = (l_2', x_2[0], i_2', 0)$, $i_1' = x_1[0]$, $i_2' = x_2[0]$, $e_1 = (l_1, l_1') \in E_1$, $e_2 = (l_2, l_2') \in E_2$. $x_1 \in \mathbf{X}_1$, $x_2 \in \mathbf{X}_2$, $v_1 \in \mathbf{V}_{11}$, $v_2 \in \mathbf{V}_{22}$, $v_1' \in h_2(q_2^{-1}, x_2^{-1}|_{\mathbf{V}_{12}})$, $q_2^{-1} = (l_2, x_2, i_2, k_2 - 1)$, $x_2^{-1} \in \mathbf{X}_2[k_2 - 1]$, $v_2' \in h_1(q_1^{-1}, x_1^{-1}|_{\mathbf{V}_{21}})$, $q_1^{-1} = (l_1, x_1, i_1, k_1 - 1)$, $x_1^{-1} \in \mathbf{X}_1[k_1 - 1]$.

`(Rule Inter-Intra)`

$(q_1, q_2) \xrightarrow[(x_1[0]) \in R_1(e_1, x_1[k_1], \{v_1[k_1] \cup v_1'[k_1]\}) \wedge (y_2[k_2+1] \in h_2(l_2, x_2[k_2]))]{(x_1[k_1], \{v_1[k_1] \cup v_1'[k_1]\}) \in G_1(e_1) \wedge \delta} (q_1', q_2')$

where $q_1 = (l_1, x_1[k_1], i_1, k_1)$, $q_2 = (l_2, x_2[k_2], i_2, k_2)$, $q_1' = (l_1', x_1[0], i_1', 0)$, $q_2' = (l_2, x_2[k_2 + 1], i_2, k_2 + 1)$, $i_1' = x_1[0]$. $e_1 = (q_1, q_1') \in E_1$. $q_2, q_2' \in Q_2$, $i_2' = i_2$ and $x_2' = Switness_2(l_2, k_2 + 1, \delta, i_2)$, $x_1 \in \mathbf{X}_1$, $v_1 \in \mathbf{V}_{11}$,

$v_1' \in h_2(q_2^{-1}, x_2^{-1}|_{\mathbf{V}_{12}})$, $q_2^{-1} = (l_2, x_2, i_2, k_2 - 1)$, $x_2^{-1} \in \mathbf{X}_2[k_2 - 1]$, $x_2 \in \mathbf{X}_2$, $y_2 \in \mathbf{Y}_2$.

(`Rule Intra-Inter`)

$$(q_1, q_2) \xrightarrow[(x_2[0]) \in R_2(e_2, x_2[k_2], \{v_2[k_2] \cup v_2'[k_2]\}) \wedge (y_1[k_1+1] \in h_1(l_1, x_1[k_1]))]{(x_2[k_2], \{v_2[k_2] \cup v_2'[k_2]\}) \in G_2(e_2) \wedge \delta} (q_1', q_2')$$

where $q_1 = (l_1, x_1[k_1], i_1, k_1)$, $q_2 = (l_2, x_2[k_2], i_2, k_2)$, $q_1' = (l_1', x_1[k_1 + 1], i_1', k_1 + 1)$, $q_2' = (l_2, x_2[0], i_2', 0)$, $i_2' = x_2[0]$, $e_2 = (q_2, q_2') \in E_2$. $q_1, q_1' \in Q_1$, $i_1' = i_1$ and $x_1' = Switness_1(l_1, k_1 + 1, i_1, x_1)$. $x_1 \in \mathbf{X}_1$, $v_2 \in \mathbf{V}_{22}$, $v_2' \in h_1(q_1^{-1}, x_1^{-1}|_{\mathbf{V}_{21}})$, $q_1^{-1} = (l_1, x_1, i_1, k_1 - 1)$, $x_1^{-1} \in \mathbf{X}_1[k_1 - 1]$, $x_2 \in \mathbf{X}_2$, $y_1 \in \mathbf{Y}_1$.

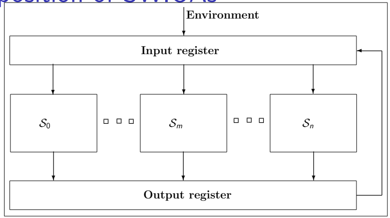
- *Intra-location transitions* of the form:

(Rule Intra-Intra)

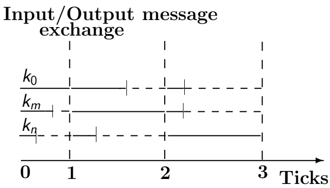
$$(q_1, q_2) \xrightarrow[(y_1[k_1+1]\in h_1(l_1, x_1[k_1]))\wedge(y_2[k_2+1]\in h_2(l_2, x_2[k_2+1]))]{(x_1[k_1], \{v_1[k_1]\cup v_1'[k_1]\})\in Inv(l_1)\wedge(x_2[k_2], \{v_2[k_2]\cup v_2'[k_2]\})\in Inv(l_2)\wedge\delta} (q_1', q_2')$$

where $q_1 = (l_1, x_1[k_1], i_1, k_1)$, $q_2 = (l_2, x_2[k_2], i_2, k_2)$, $q_1' = (l_1, x_1[k_1 + 1], i_1, k_1 + 1)$, $q_2' = (l_2, x_2[k_2 + 1], i_2, k_2 + 1)$, and $q_1, q_1' \in Q_1$. $Switness_1(l_1, k_1, \delta, i_1) = x_1[k_1]$ and $Switness_1(l_1, k_1 + 1, \delta, i_1) = x_1[k_1 + 1]$. Similarly, $q_2, q_2' \in Q_2$. $Switness_2(l_2, k_2, \delta, i_2) = x_2[k_2]$ and $Switness_2(l_2, k_2 + 1, \delta, i_2) = x_2[k_2 + 1]$ for any δ. $x_1 \in \mathbf{X}_1$, $x_2 \in \mathbf{X}_2$. $v_1' \in h_2(q_2^{-1}, x_2^{-1}|_{\mathbf{V}_{12}})$, $q_2^{-1} = (l_2, x_2, i_2, k_2 - 1)$, $x_2^{-1} \in \mathbf{X}_2[k_2 - 1]$, $v_2' \in h_1(q_1^{-1}, x_1^{-1}|_{\mathbf{V}_{21}})$, $q_1^{-1} = (l_1, x_1, i_1, k_1 - 1)$, $x_1^{-1} \in \mathbf{X}_1[k_1 - 1]$.

Composition of SWIOAs



(a) Modular execution of SWIOAs

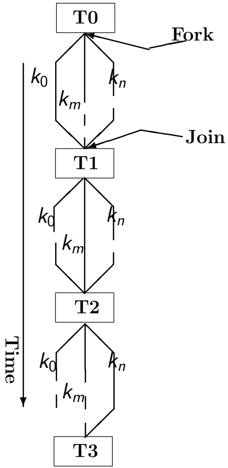


(b) SWIOA execution trace

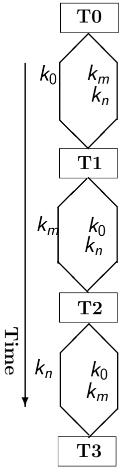
Remarks

1. Each SWIOA reads the inputs from the environment, e.g., pacemaker in case of the heart model, or other SWIOA's outputs via a shared register (or memory) called the input register.
2. Each SWIOA performs a local tick.
3. Upon completion of its local tick, each SWIOA produces outputs to a shared register (or memory).
4. Each SWIOA waits until **all** other SWIOAs have completed their individual local ticks, i.e., they *barrier synchronize*.
5. Once every SWIOA has completed a local tick, the produced outputs from the individual SWIOAs is transfered into the input register, the environment inputs are read into the input register and these steps are repeated.

Parallel execution semantics



(c) Naïve parallel implementation



(d) Ideal load balanced parallel implementation

1. Need fork join parallelism.
2. Dynamic load balancing.
   - From $T0$ to $T1$, $\mathcal{S}_m$ and $\mathcal{S}_n$ are fused together.
   - From $T1$ to $T2$, $\mathcal{S}_0$ and $\mathcal{S}_n$ are fused together.
3. Portable across backend parallel library implementations.

Dynamic load balancing using work stealing – Cilk and OpenMP parallelization models

```
cilk_for (int i=0; i<8; ++i)
  do_work();
```

(e) The `cilk_for` example construct

```
#pragma omp parallel for
for (int i=0; i<8; ++i)
  do_work();
```

(f) The OpenMP `for` example construct

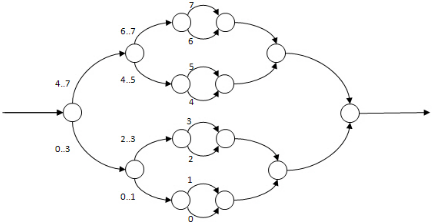


Figure: Parallelization model

Produced back end "C" code

```
// Headers including SWIOA function declaration
// and definitions

// The array of function pointers to SWIOA
int (*func[3081]) (int prev_state, int current_state);

// Store SWIOA functions into function pointer
func[0] = Sinoatrialnode;
.....

// Declare and initialize
// the arrays to store the current and previous
// states of SWIOAs
int cstate[3081]={0}, pstate[3081]={-1};

int main (void) {
 //The while loop runs forever
 while(1) {
  //Map inputs to outputs for each SWIOA
  SinoatrialnodeInput = LeftatriumOutput;
  ....
  //Do computation
  for (int i = 0; i < 3081; ++i){
   int rstate = (*func[i]) (cstate[i], pstate[i]);
   pstate[i] = cstate[i];
   cstate[i] = rstate;
  }
 }
}
```

Figure: Generated back-end "C" code snippet for the heart model.

Experimental setup

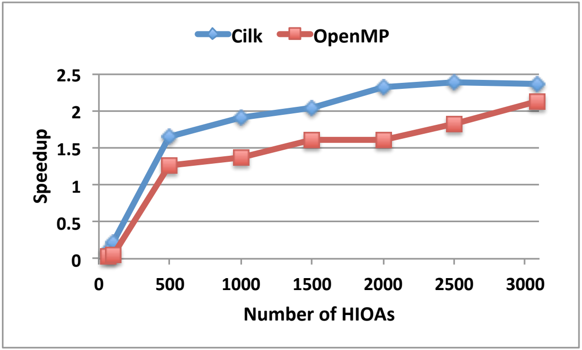
Table: Benchmark descriptions

Benchmarks	Domain	Description
Thermostat (TS)	Physics [3]	Heats a room to keep it warm.
Heart model (HM)	Biology [1]	Captures the electrical behaviour of a cardiac cell.
Water Heating system (WH)	Physics [4]	Models the heating of water
Train Gate control(TG)	Industrial automation [2]	Models the behaviour of a gate at a rail road crossing.

Experimental platform

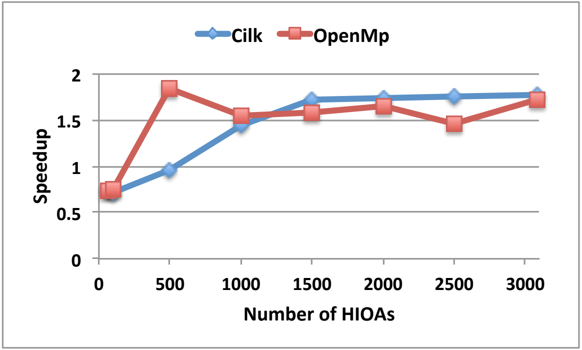
- OSX version 10.9.2
- Clang/LLVM version 4.2.1 with Cilk and OpenMP add ons.

Results I



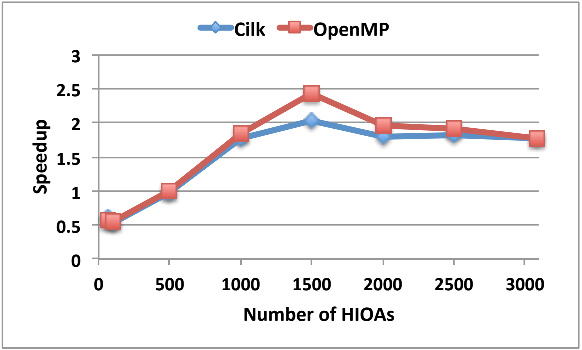
(a) Thermostat

Results II



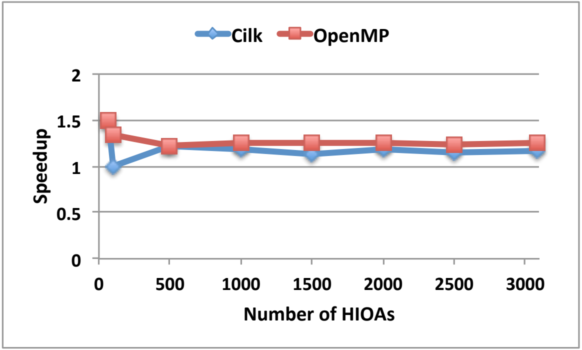
(b) Heart model

Results III



(c) Water Heating system

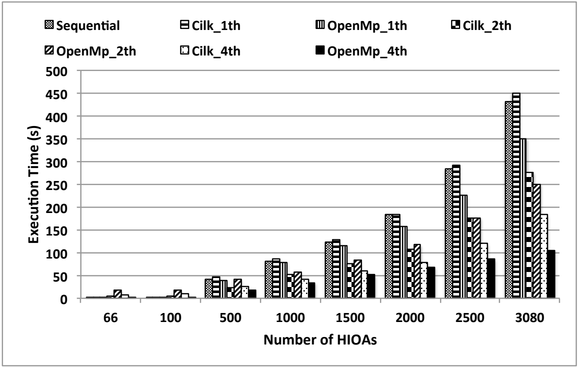
Results IV



(d) Train Gate control

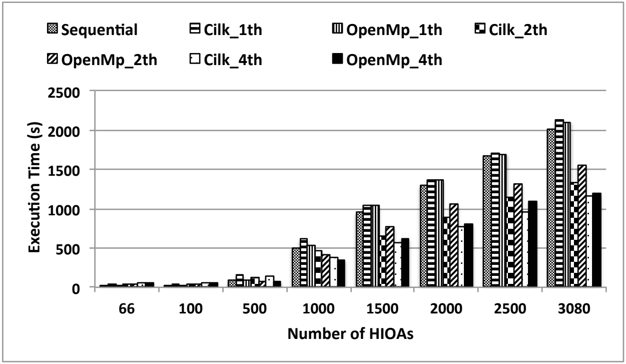
Figure: Cilk vs. OpenMP speedup normalized to sequential execution time on 4 cores

Results with varying cores/threads I



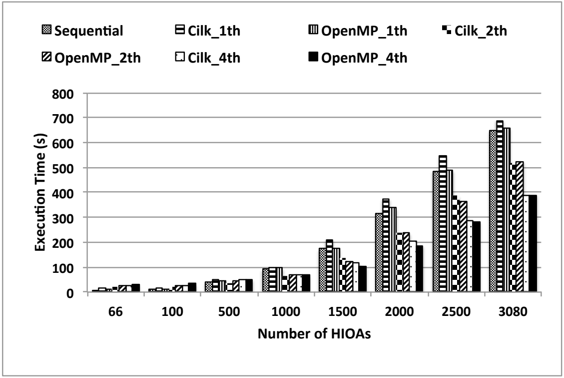
(a) Thermostat

Results with varying cores/threads II



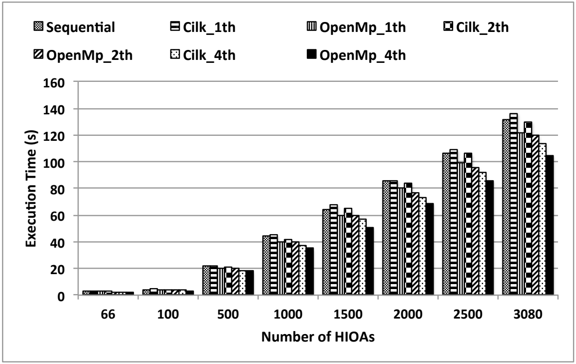
(b) Heart model

Results with varying cores/threads III



(c) Water Heating system

Results with varying cores/threads IV



(d) Train Gate control

Figure: Cilk vs. OpenMP speedup with varying number of threads/cores

Conclusions

1. Presented a modular compilation framework for large system designed in Hybrid Input Output Automata (HIOA).
2. Presented a parallel execution framework for HIOA – the very first to our knowledge to present a parallel execution framework.
3. The compilation approach (and associated semantics) are inspired by the synchronous model of computation.
4. We see speedup from $1.4\times$ to $2.5\times$ over sequential execution.

- Nathan Allen et al. “Modular code generation for emulating the electrical conduction system of the human heart”. In: *2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016, Dresden, Germany, March 14-18, 2016*. 2016, pp. 648–653.
- Costello Brennon and Elliott Joshua. *Hybrid Systems*. Tufts University, Course EE194, Lecture. http://www.eecs.tufts.edu/~khan/Courses/Spring2013/EE194/Lecs/ Hybrid_ Systems_Presentation_Elliott_Costello.pdf. 2013.
- Hespanha Joao Pedro. *How to describe a hybrid system? Formal models for hybrid system*. University of California at Santa Barbara, Course ECE229, Lecture 2. http://www.ece.ucsb.edu/~hespanha/ece229/Lectures/ Lecture2.pdf. 2005.
- J-F Raskin. “Handbook of Networked and Embedded Control Systems”. In: Springer, 2005. Chap. An introduction to hybrid automata, pp. 491–517.