Mothers of Pipelines

Sava Krstić, Robert Jones, John O' Leary

Intel Corporation

PDPAR 2006

Pipeline Verification: P vs. ISA

Prove that a high-level model P of a pipelined processor is faithful to the given instruction set architecture (ISA).

- Both P and ISA are seen as transition systems; the goal is to prove that ISA in some reasonable sense simulates P.

ISA

States: $\mathcal{I} = \langle pc : \mathsf{IAddr}, rf : \mathsf{RF}, dmem : \mathsf{DMem}, imem : \mathsf{IMem} \rangle$

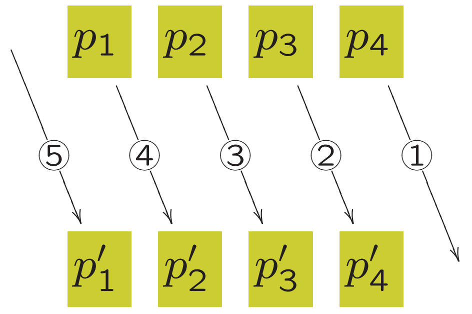
Transitions:

guard: $imem.pc = \ldots$	action
$opc1 \quad dest \quad src1 \quad src2$	$pc := pc + 4 \quad rf.dest := \mathsf{alu}\ opc1\ (rf.src1)\ (rf.src2)$
$opc2 \quad dest \quad src1 \quad imm$	$pc := pc + 4 \quad rf.dest := \mathsf{alu}\ opc2\ (rf.src1)\ imm$
$\mathsf{ld} \quad dest \quad src1 \quad offset$	$pc := pc + 4 \quad rf.dest := dmem.(rf.src1 + offset)$
$\mathsf{st} \quad src1 \quad dest \quad offset$	$pc := pc + 4 \quad dmem.(rf.dest + offset) := rf.src1$
$opc3 \quad reg \quad offset$	$pc := \begin{cases} br\_target & \text{if } br\_taken \\ pc + 4 & \text{otherwise} \end{cases}, \quad \text{where}$ $br\_target = target\ pc\ offset$ $br\_taken = taken\ opc3\ (rf.reg)$

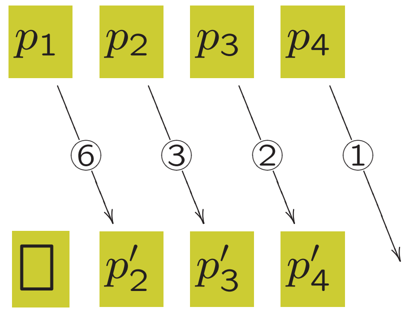
$dest, src1, src2, reg : \mathsf{Reg} \qquad imm, offset : \mathsf{Word}$
$opc1 : \{\mathsf{add}, \mathsf{sub}, \mathsf{mult}\} \qquad opc2 : \{\mathsf{addi}, \mathsf{subi}, \mathsf{multi}\} \qquad opc3 : \{\mathsf{beqz}, \mathsf{bnez}, \mathsf{j}\}$

Example: $P = DLX$

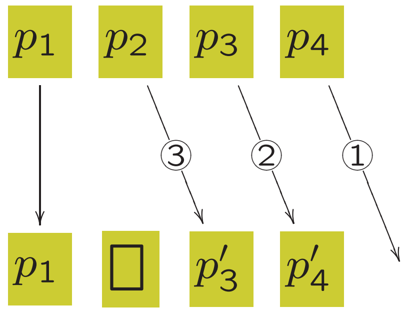
States of P: $\langle pc, rf, dmem, imem, p_1, p_2, p_3, p_4 \rangle$



[regular cycle]



[branch taken]



[stall for load]

① p_4 writes back to rf and retires

② p_3 does memory access

③ alu computes result/mem. address for p_2

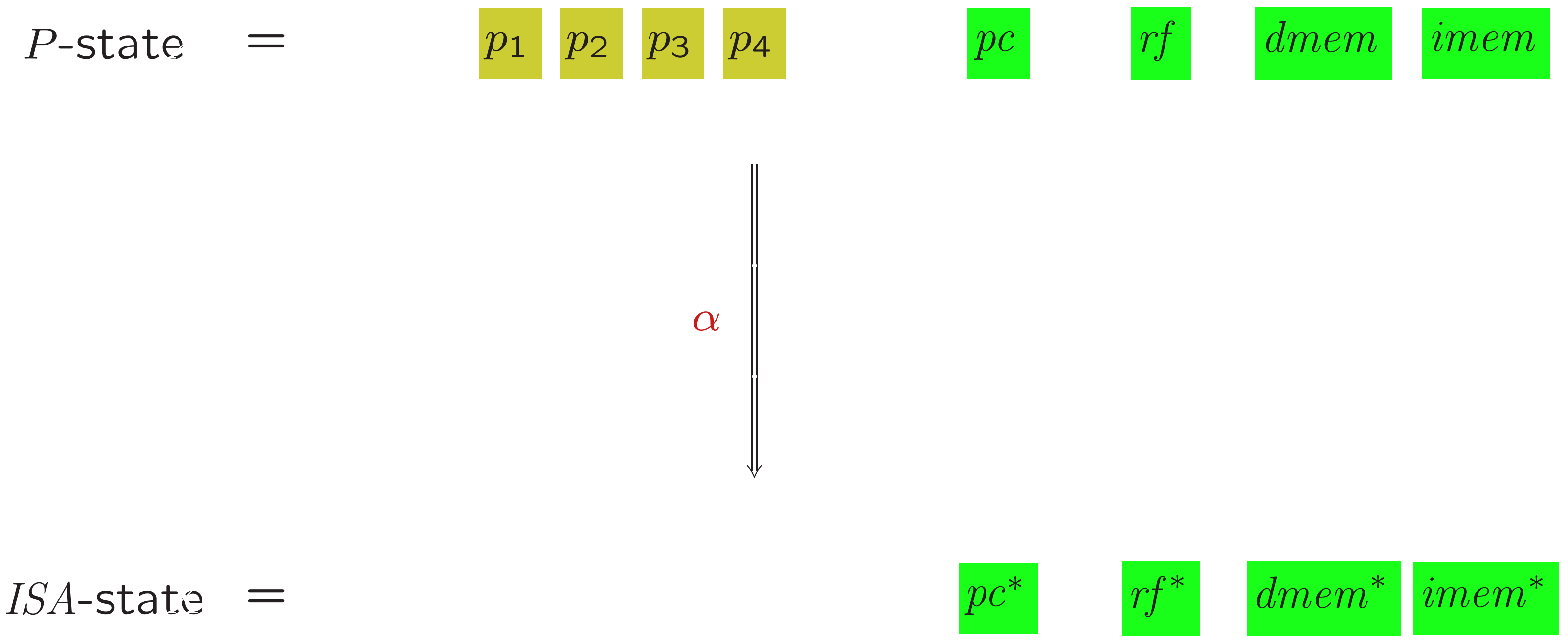
④ p_1 gets data from rf or by forwarding;
in the branch case, target/taken computed

⑤ new p'_1 is fetched; pc incremented

⑥ like ④, plus updating pc with computed target

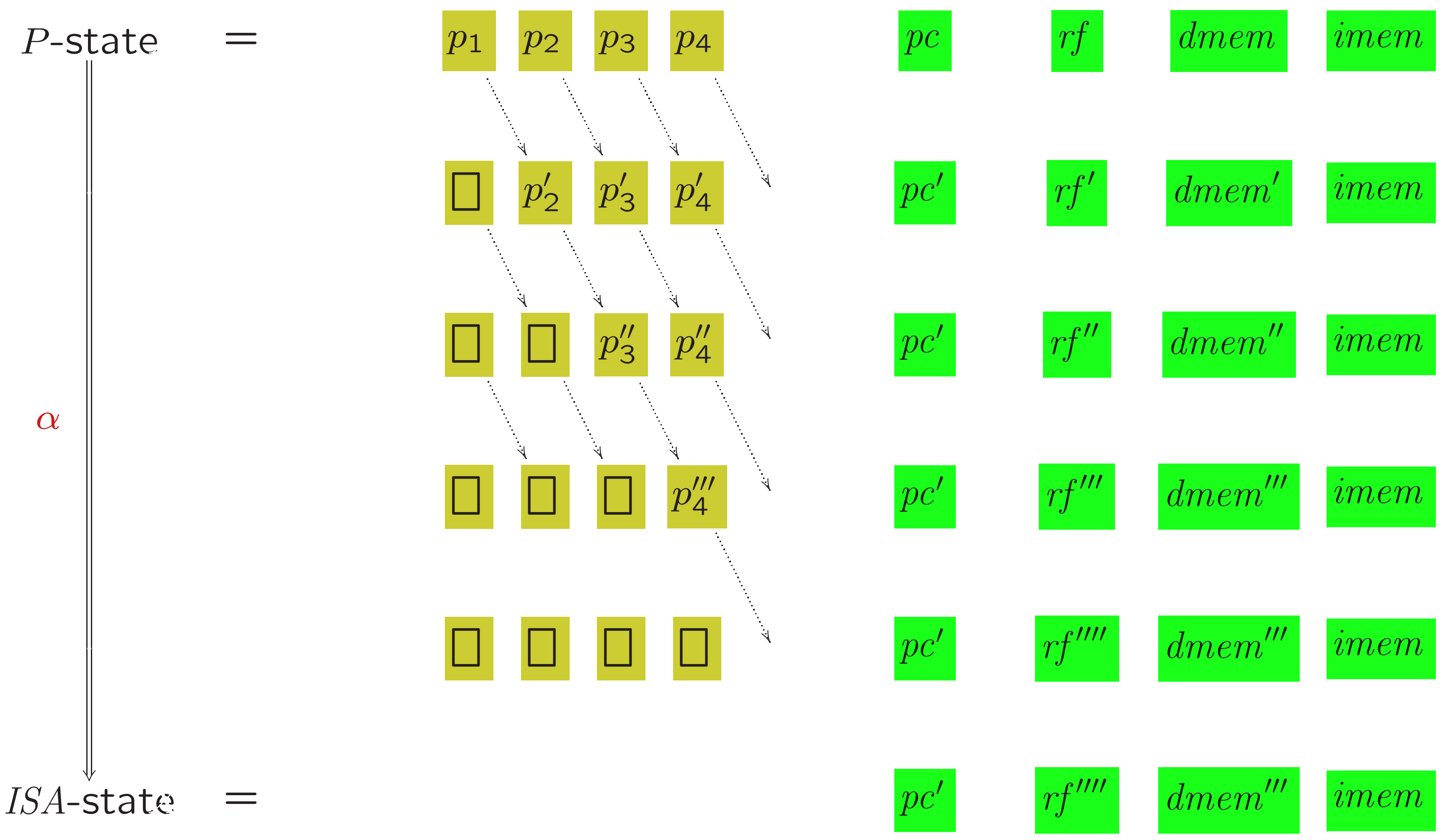
Simulating P in ISA

First need to map states of P to ISA states:



Flushing

Mapping states of P to ISA states:


P-state =
p_1 p_2 p_3 p_4 pc rf $dmem$ $imem$
□ p_2' p_3' p_4' pc' rf' $dmem'$ $imem$
□ □ p_3'' p_4'' pc' rf'' $dmem''$ $imem$
α
□ □ □ p_4''' pc' rf''' $dmem'''$ $imem$
□ □ □ □ pc' rf'''' $dmem'''$ $imem$
ISA-state = pc' rf'''' $dmem'''$ $imem$

The Burch-Dill Method for *DLX* [CAV 94]

✹ $\mathcal{D}$	*DLX* states
✹ dlx_step : $\mathcal{D} \to \mathcal{D}$	*DLX* transitions
✹ $\alpha : \mathcal{D} \to \mathcal{I}$	flushing function

Theorem

$$\begin{tikzcd} \mathcal{D} \arrow[r, "\text{dlx\_step}"] \arrow[d, "\alpha"'] & \mathcal{D} \arrow[d, "\alpha"] \\ \mathcal{I} \arrow[r, "\text{[isa\_step]}"] & \mathcal{I} \end{tikzcd}$$

The Burch-Dill Method in General

- $\mathcal{P} = \mathcal{I} \times \text{PipeRegs}$ states of P
- $\text{p\_step} : \mathcal{P} \to \mathcal{P}$ transitions of P
- $\alpha : \mathcal{P} \to \mathcal{I}$ flushing function
- Correctness Theorem for P:

$$\begin{tikzcd}
\mathcal{P} \arrow[r, "\text{p\_step}"] \arrow[d, "\alpha"'] & \mathcal{P} \arrow[d, "\alpha"] \\
\mathcal{I} \arrow[r, "(\text{isa\_step})^*"] & \mathcal{I}
\end{tikzcd}$$

- For proof, don't need defs of alu, target, taken, and only need basic read-write array properties of rf and mem
- Correctness thm expressed in the language of SMT solvers
- . . . but it's huge for complex P. Must partition the thm.

The *MOP* Method

Put a highly-nondeterministic “machine” MOP between P and ISA and deduce the P vs. ISA correctness from P vs. MOP and MOP vs. ISA.

- MOP is derived from ISA and the features of P
- MOP is the mother of *many* pipelines:

$$\begin{tikzcd}
P_1 \arrow[rdd, dashed] & & \\
P_2 \arrow[rd, dashed] & & \\
\vdots & MOP & & ISA \arrow[ll, dashed] \\
\vdots & & \\
P_n \arrow[ruu, dashed] & &
\end{tikzcd}$$

- Out-of-order execution and non-determinism in P are OK
- Potential to systematically partition correctness theorem

MOP: Starting Observations

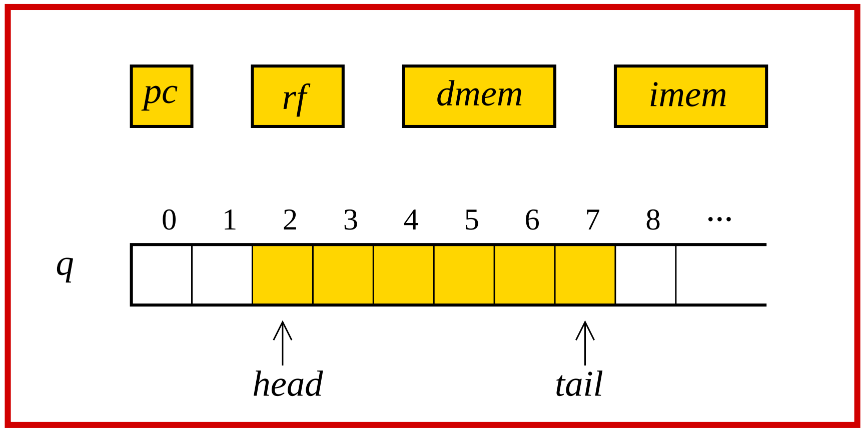
- One can formalize “instructions-in-flight”—parcels
- Parcels form a poset of finite height—progress ordering
- With every state of P we can associate a set of parcels
- At each cycle of execution of P:
  - pc, rf, mem get updated
  - some parcels get in (fetched)
  - some parcels get out (retired)
  - some parcels progress

Parcels

$$\mathsf{Parcel} \quad = \quad \begin{array}{l} \langle \\ \quad \mathit{instr} : \mathsf{Instr}_\perp \\ \quad \mathit{my\_pc} : \mathsf{IAddr}_\perp \\ \quad \mathit{dest}, \mathit{src1}, \mathit{src2} : \mathsf{Reg}_\perp \\ \quad \mathit{imm} : \mathsf{Word}_\perp \\ \quad \mathit{opc} : \mathsf{Opcode}_\perp \\ \quad \mathit{data1}, \mathit{data2} : \mathsf{Word}_\perp \\ \quad \mathit{res}, \mathit{mem\_addr} : \mathsf{Word}_\perp \\ \quad \mathit{tkn} : \mathsf{bool}_\perp \\ \quad \mathit{next\_pc} : \mathsf{IAddr}_\perp \\ \quad \mathit{wb} : \{\perp, \top\} \\ \quad \mathit{pc\_upd} : \{\perp, \mathbb{s}, \mathbb{m}, \top\} \\ \rangle \end{array}$$

Definition of MOP

States: $\mathcal{M} = \mathcal{I} \times \langle q : \mathbb{N} \rightarrow \text{Parcel},\ head : \mathbb{N},\ tail : \mathbb{N} \rangle$



Transitions: Atomic actions occurring in executions

Transitions

def	$i = imem.pc$	**fetch**
grd	$length = 0 \vee q.tail.pc\_upd \neq \bot$	
act	$q.(tail + 1) := empty\_parcel[instr \mapsto i, my\_pc \mapsto pc] \quad tail := tail + 1$	

def	$p = q.j$	**decode** j
grd	$head \leq j \leq tail \quad \neg(decoded\ p)$	
act	$p := decode\ p$	

- 16 more rules: **data1_rf**, **data1_forward**, **result**, **mem_addr**, **write_back** j, **load** j, **store** j, **branch_target** j, **branch_taken** j, **next_pc_branch** j, **next_pc_nonbranch** j, **pc_update**, **speculate**, **prediction_ok** j, **squash**, **retire**

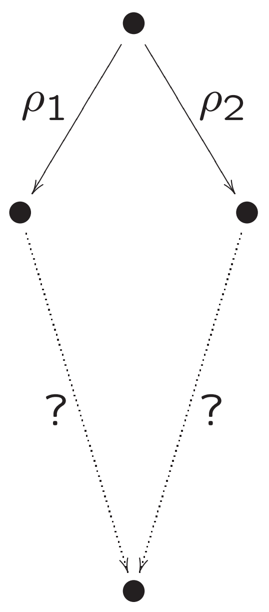
Confluence

$MOP^{\#} \quad \equiv \quad MOP$ without **fetch**

Theorem $MOP^{\#}$ is terminating.

Theorem Both MOP and $MOP^{\#}$ are locally confluent.

($\approx$ 400 little theorems)



Local Confluence: About Proof

Most cases are resolved trivially:

$$\begin{tikzcd}
& \bullet \arrow[dl, "\rho_1"'] \arrow[dr, "\rho_2"] & \\
\bullet \arrow[dr, "\rho_2"'] & & \bullet \arrow[dl, "\rho_1"] \\
& \bullet &
\end{tikzcd}
\qquad
\begin{tikzcd}
& \bullet \arrow[dl, "\rho_1"'] \arrow[dr, "\rho_2"] & \\
\bullet \arrow[rr, bend right, "\rho_2"'] & & \bullet
\end{tikzcd}$$

Some are interesting:

$$\begin{tikzcd}
& \bullet \arrow[dl, "\textbf{retire}"'] \arrow[dr, "\textbf{data\_queue}\ j"] & \\
\bullet \arrow[dr, "\textbf{data\_rf}\ j"'] & & \bullet \arrow[dl, "\textbf{retire}"] \\
& \bullet &
\end{tikzcd}$$

Flushing and Burch-Dill for MOP

- Define
  - flushed MOP state $\equiv$ its q-component is empty
- Given $m \in \mathcal{M}$, any $MOP^{\#}$ run $m \to m_1 \to m_2 \to \ldots$
  - . . . terminates . . .
  - . . . in the same final *flushed* state m'
- Define
  - $\alpha(m)$ $=$ the ISA-component of m'
- Burch-Dill **Theorem** for MOP:

$$\begin{tikzcd}
\mathcal{M} \arrow[r, "\rho"] \arrow[d, "\alpha"'] & \mathcal{M} \arrow[d, "\alpha"] \\
\mathcal{I} \arrow[r, "\text{[isa\_step]}"] & \mathcal{I}
\end{tikzcd}$$

(for any MOP rule ρ)

Simulating Microarchitectures in MOP

Theorem To verify P against ISA, it suffices to find $\beta\colon \mathcal{P} \rightarrow \mathcal{M}$ such that, for every $p \in P$, β (p_step p) is reachable from $\beta\ p$.

Proof:

$$\begin{tikzcd}
p \arrow[rrrrrrrr, "\text{p\_step}"] \arrow[d, "\beta"'] & & & & & & & & p' \arrow[d, "\beta"] & \\
m \arrow[r] \arrow[d, "\alpha"'] & \bullet \arrow[r] \arrow[dl] & \bullet \arrow[r] \arrow[d] & \bullet \arrow[r] \arrow[dl] & \bullet \arrow[r] \arrow[dll] & \bullet \arrow[r] \arrow[dr] & \bullet \arrow[r] \arrow[d] & \bullet \arrow[r] \arrow[dl] & m' \arrow[d, "\alpha"] & \longleftarrow \text{mam\_steps} \\
s \arrow[rr] & & \bullet \arrow[rrrr] & & & & \bullet \arrow[rr] & & s' & \longleftarrow \text{isa\_steps}
\end{tikzcd}$$

Partitioning the Simulation Proof

$$\begin{tikzcd}
p \arrow[rrrrrrrr, "\text{p\_step}"] \arrow[d, "\beta"'] & & & & & & & & p' \arrow[d, "\beta"] \\
m \arrow[r] & \bullet \arrow[r] & \bullet \arrow[r] & \bullet \arrow[r] & \bullet \arrow[r] & \bullet \arrow[r] & \bullet \arrow[r] & \bullet \arrow[r] & m'
\end{tikzcd}$$

- How to find a sequence of MOP transitions from m to m'?
- —Using sequence of “quasi P-states” from p to p':

$$\begin{aligned}
& \langle v_1, v_2, v_3, \ldots, v_n \rangle = p \\
\rightsquigarrow\ & \langle v_1', v_2, v_3, \ldots, v_n \rangle = p_1 \\
\rightsquigarrow\ & \langle v_1', v_2', v_3, \ldots, v_n \rangle = p_2 \\
\rightsquigarrow\ & \ldots \\
\rightsquigarrow\ & \langle v_1', v_2', v_3', \ldots, v_n' \rangle = p_{n-1} \\
\rightsquigarrow\ & \langle v_1', v_2', v_3', \ldots, v_n' \rangle = p'
\end{aligned}$$

$$v_i' = next\_v_i\ (v_1, \ldots, v_n)$$

- . . . get the corresponding MOP states $m, m_1, \ldots, m_{n-1}, m'$ and short MOP paths from m_i to m_{i+1}.

What We've Done

- Models of ISA,MOP,DLX in *reFLect*
- Local confluence proofs with CVCL
- Simulation proofs for DLX (via short paths in MOP) with CVCL

To Do

- Case study of an out-of-order processor model
- Refining the method
  - Systematic ways of defining $\beta\colon \mathcal{P} \to \mathcal{M}$ and finding short paths to connect $\beta(p)$ with $\beta(\mathsf{p\_step}\ p)$ in MOP
  - Controlling term size in subgoals (heuristics for expanding function definitions vs. treating them as uninterpreted)
- Fast and flexible SMT solvers

Selection of Related Work

- Burch & Dill [CAV 94]
- Damm & Pnueli [CHARME 97]
- Shen & Arvind [Formal Techn. for Hardware 98]
- Skakkebæk & al. [CAV 98]
- Hosabettu & al. [CAV 00]
- Lahiri & Bryant [CAV 03]
- Manolios & Srinivasan [ICCAD 05]