Adaptive Embedded Processing with RISPP

by Jörg Henkel
together with Lars Bauer, Muhammad Shafique, …

Chair for Embedded Systems (CES)

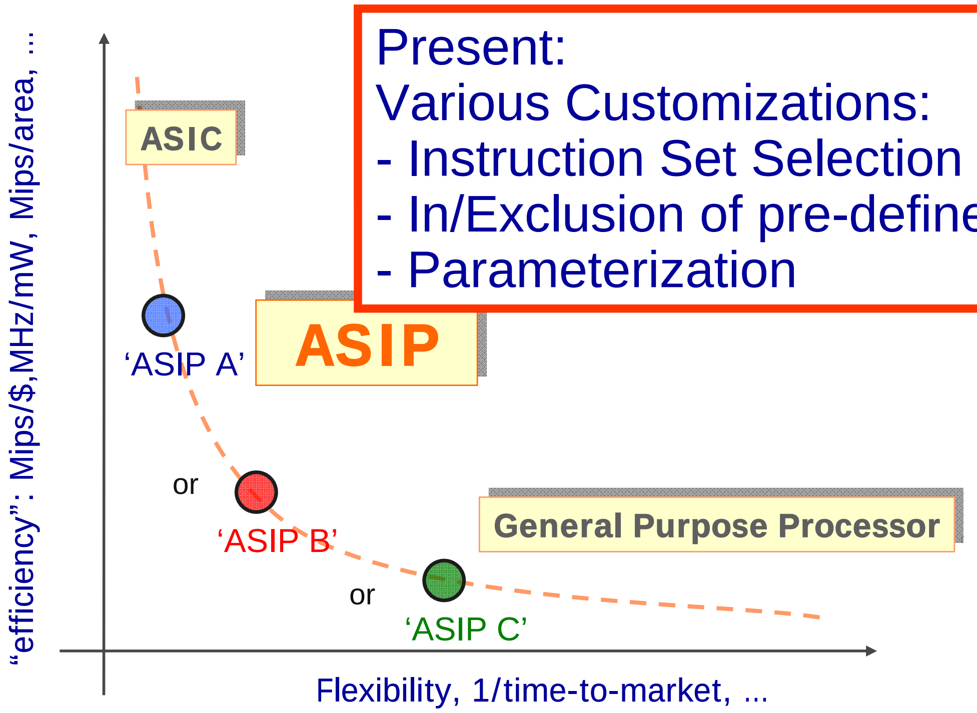
University of Karlsruhe, Germany

Development of Embedded Systems

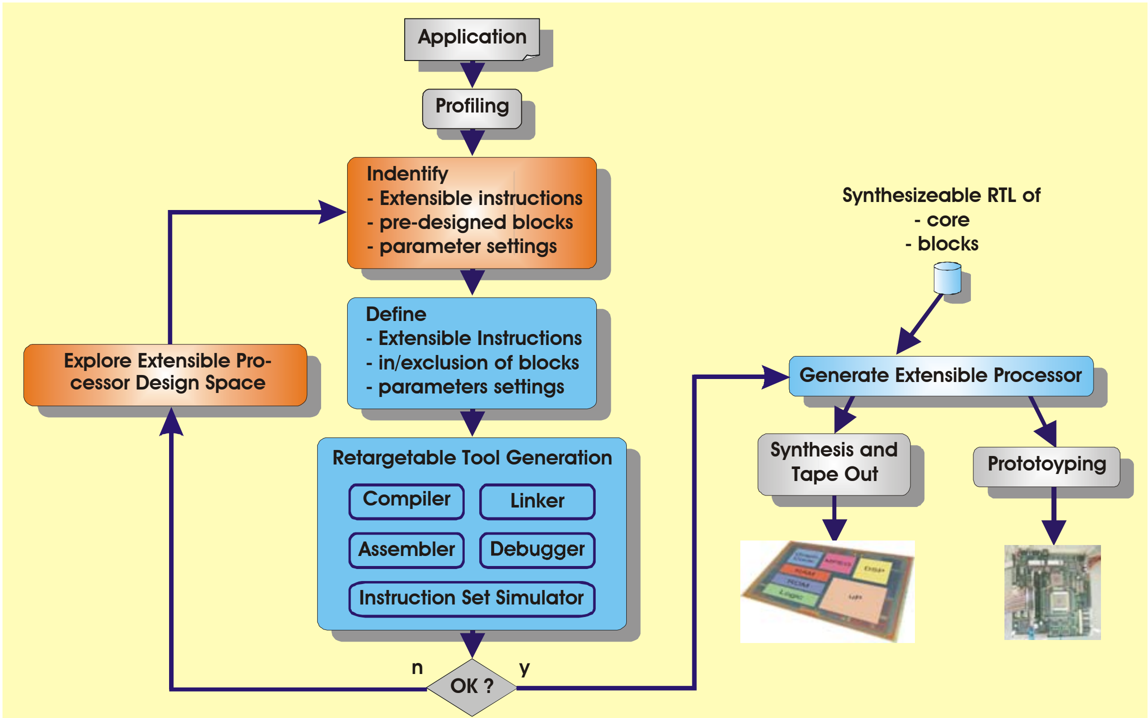
- Typical:
  - **Static analysis** of **hot spots**
  - Building **tightly optimized** system
- Nowadays:
  - Increasing complexity
  - More functionality
- Problem:
  - **Statically chosen design point** has to match all requirements
  - **Typically inefficient** for individual components (e.g. tasks or hot spots)

The place of ASIPs: from past to present

Present:
Various Customizations:
- Instruction Set Selection
- In/Exclusion of pre-defined blocks
- Parameterization

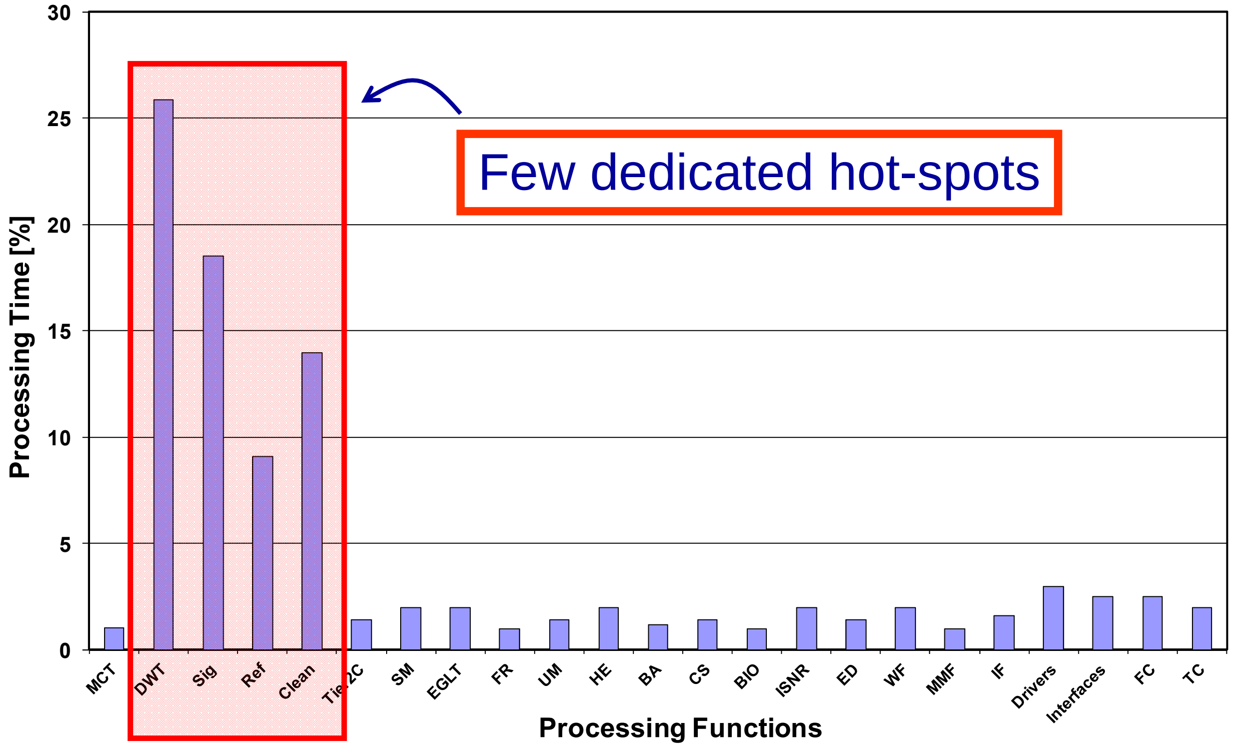

"efficiency": Mips/$,MHz/mW, Mips/area, …
ASIC
'ASIP A'
ASIP
or
'ASIP B'
General Purpose Processor
or
'ASIP C'
Flexibility, 1/time-to-market, …

State-of-the-Art ASIP Design Flow



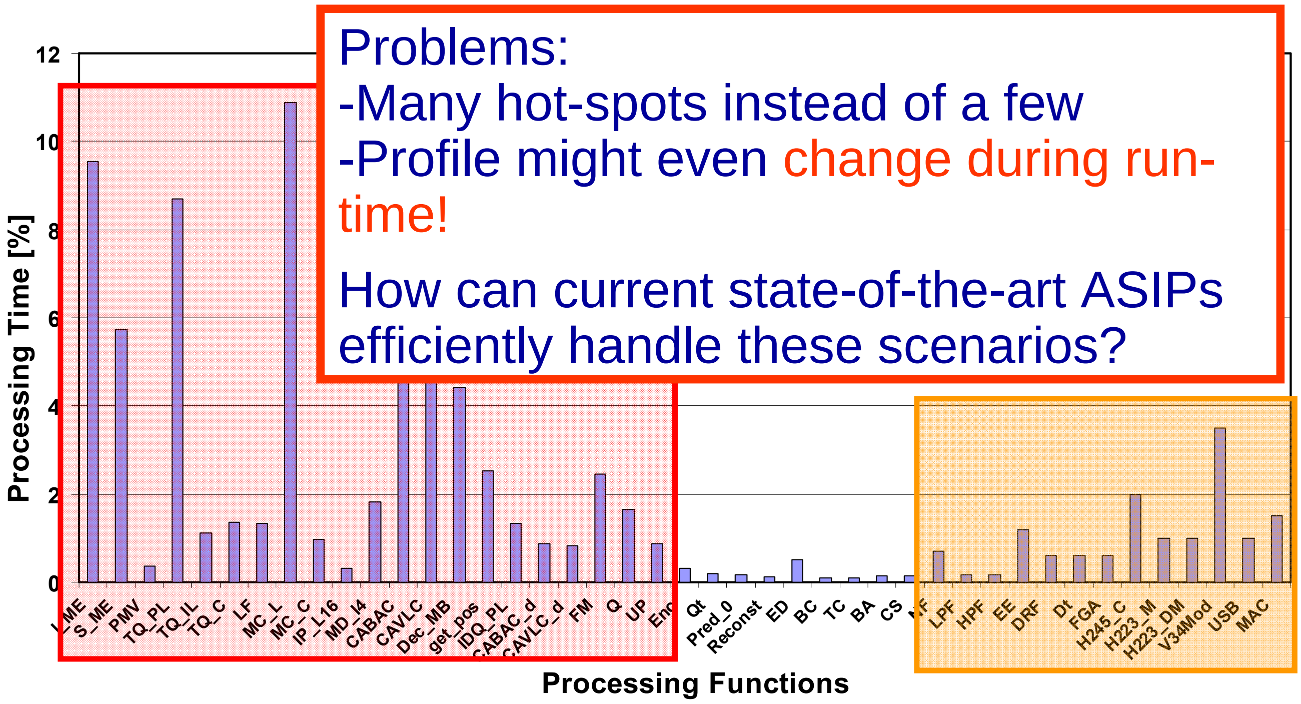
[Henkel03]

Typically …



Few dedicated hot-spots

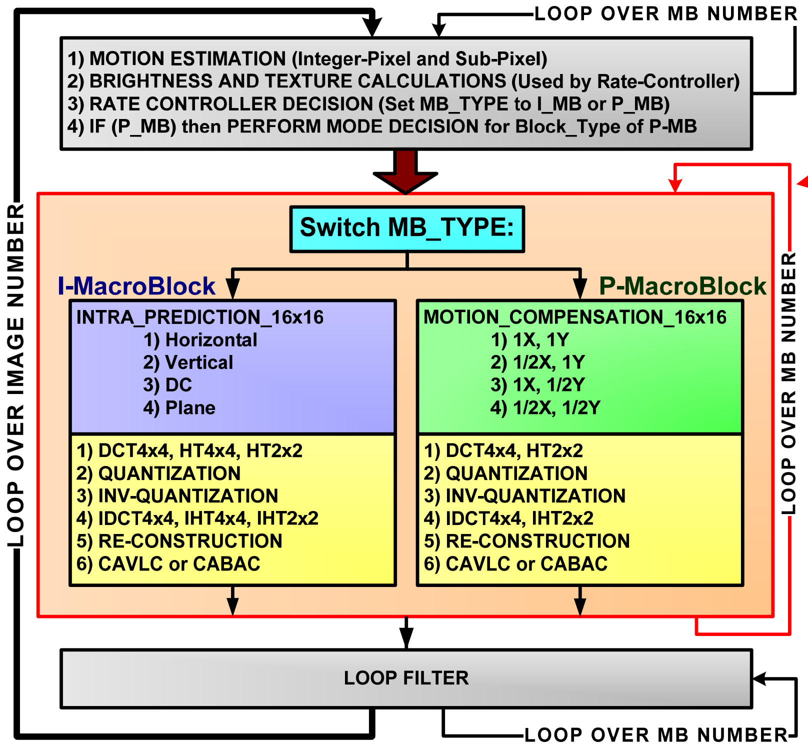
But what if …



Problems:
-Many hot-spots instead of a few
-Profile might even change during run-time!

How can current state-of-the-art ASIPs efficiently handle these scenarios?

Example: Execution Flow of an H.264 Encoder



LOOP OVER MB NUMBER

1) MOTION ESTIMATION (Integer-Pixel and Sub-Pixel)
2) BRIGHTNESS AND TEXTURE CALCULATIONS (Used by Rate-Controller)
3) RATE CONTROLLER DECISION (Set MB_TYPE to I_MB or P_MB)
4) IF (P_MB) then PERFORM MODE DECISION for Block_Type of P-MB

Switch MB_TYPE:

I-MacroBlock

INTRA_PREDICTION_16x16
1) Horizontal
2) Vertical
3) DC
4) Plane

1) DCT4x4, HT4x4, HT2x2
2) QUANTIZATION
3) INV-QUANTIZATION
4) IDCT4x4, IHT4x4, IHT2x2
5) RE-CONSTRUCTION
6) CAVLC or CABAC

P-MacroBlock

MOTION_COMPENSATION_16x16
1) 1X, 1Y
2) 1/2X, 1Y
3) 1X, 1/2Y
4) 1/2X, 1/2Y

1) DCT4x4, HT2x2
2) QUANTIZATION
3) INV-QUANTIZATION
4) IDCT4x4, IHT2x2
5) RE-CONSTRUCTION
6) CAVLC or CABAC

LOOP OVER MB NUMBER

LOOP FILTER

LOOP OVER MB NUMBER

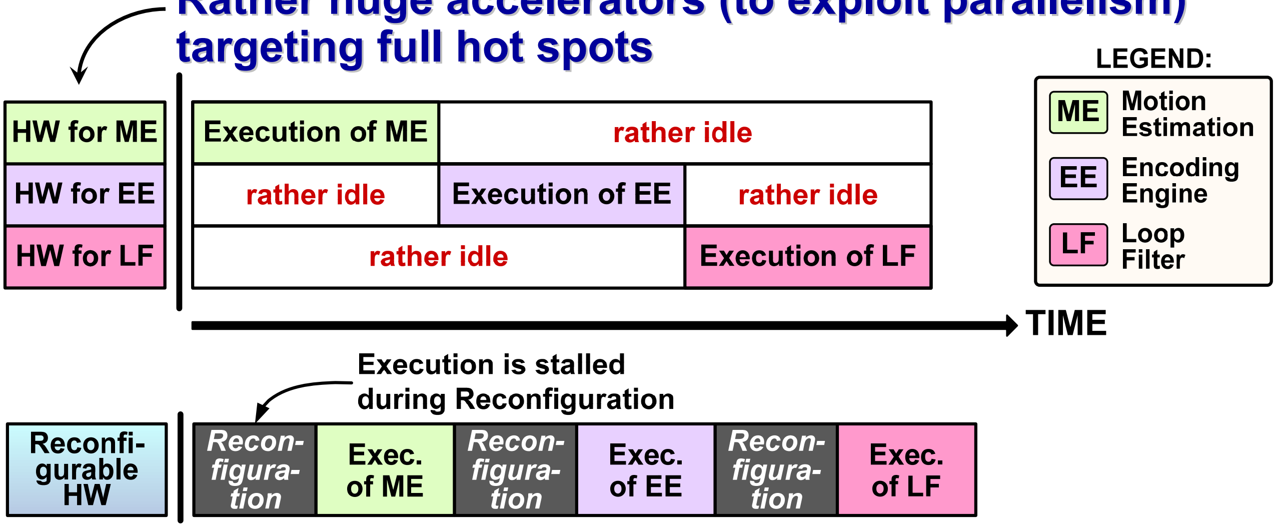
LOOP OVER IMAGE NUMBER

Main Encoding Loop:

- ❑ Iterates on Macroblocks, (i.e. 16x16 pixels)
- ❑ 2 different Macro-block-types → different computational paths
  - ❑ I-MB
  - ❑ P-MB

ASIP Efficiency problems when targeting multiple hot spots

Rather huge accelerators (to exploit parallelism) targeting full hot spots



HW for ME | Execution of ME | rather idle

HW for EE | rather idle | Execution of EE | rather idle

HW for LF | rather idle | Execution of LF

TIME

LEGEND:
ME Motion Estimation
EE Encoding Engine
LF Loop Filter

Execution is stalled during Reconfiguration

Reconfigurable HW | Reconfiguration | Exec. of ME | Reconfiguration | Exec. of EE | Reconfiguration | Exec. of LF

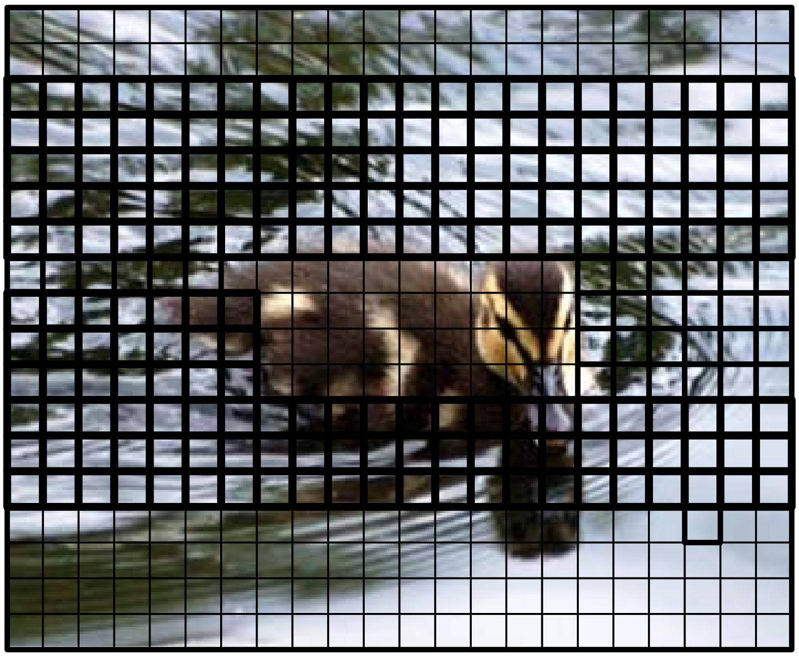
- ❑ **Reconfigurable Computing:**
  - ✚ Efficient use of hardware
  - — Potentially performance degradation due to reconfiguration time & FPGA fabric

Related Work

- **Extensible Processors:**
  - **Automatic detection & synthesis of Special Instructions**
  - **Automatic generation of Tool-Suits (simulator, compiler, …)**
  - **Early estimations to guide through the design space**
- **Reconfigurable Computing Systems:**
  - **Basic technique that enables the hardware to adapt to different requirements**
  - **Data-Flow driven systems**
  - **Control-Flow driven (typically CPU extended by a reconfigurable Co-Processor)**

Uncertainty in computation effort

- **Distribution of I- and P- Macroblocks: Prediction from previous frame or surrounding MBs?**



Macroblock (MB): 16x16 pixel Block

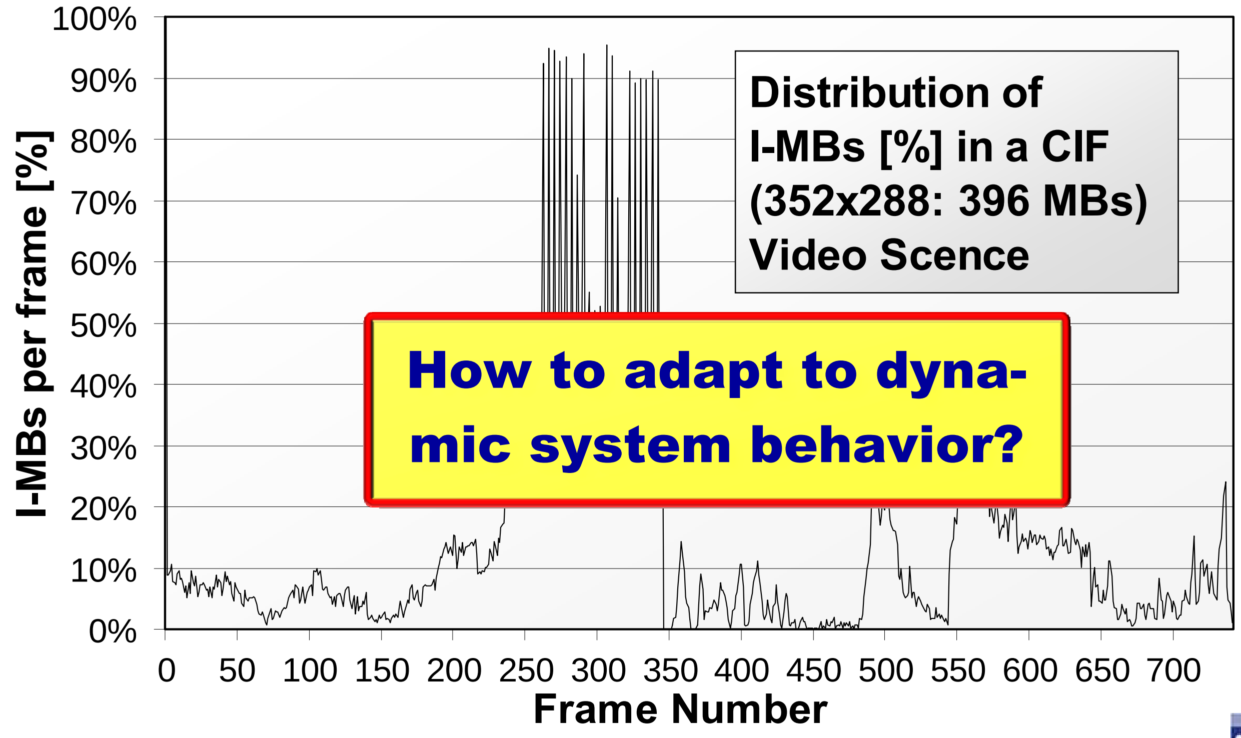
Common Intermediate Format (CIF): Image with 352x288 pixel size. 1CIF = 396 MBs

I-MB: Blocks with thick border

P-MB: Blocks with thin border

src: http://www.copyright-free-pictures.org.uk

Input-Dependent Dynamic Application Behavior



Distribution of I-MBs [%] in a CIF (352x288: 396 MBs) Video Scence

How to adapt to dynamic system behavior?

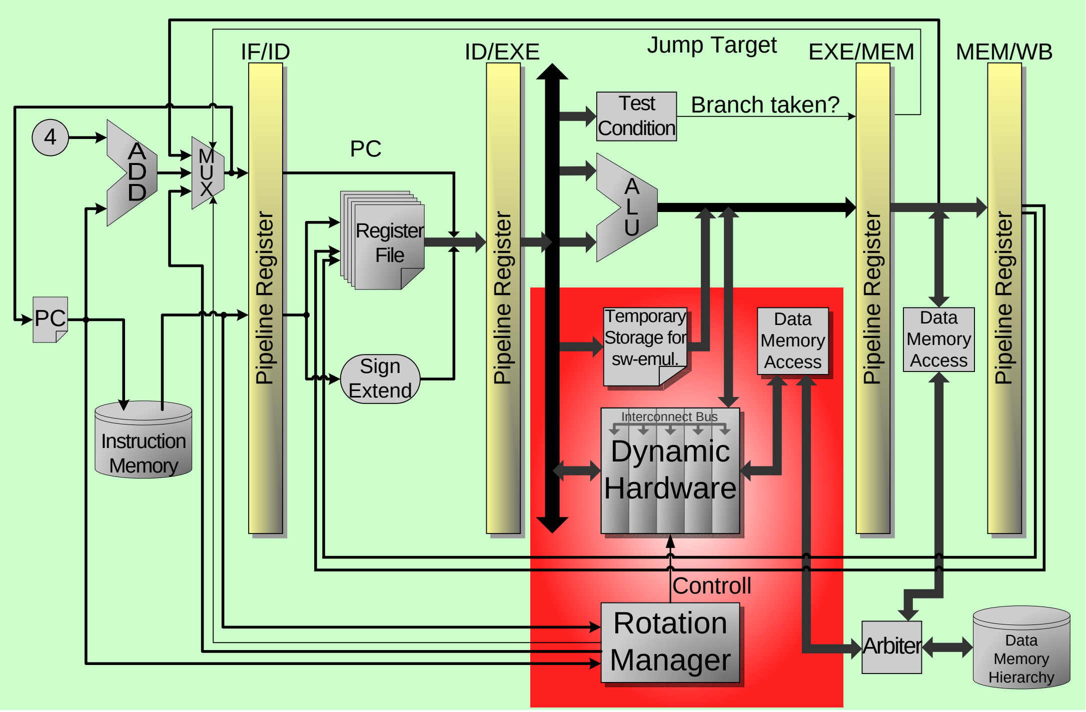
Dynamic System Behavior

- ❑ **Extensible Processor: selecting points in design space at design time**
- ❑ **Reconfigurable Computing: typically fix at compile time when and how to deploy reconfigurable hardware**

How to handle situations that are unknown at design- & compile time?

- ❑ **Depending on input data (e.g. different computational paths in a video encoder)**

Our New Concept: RISPP Basic Idea and Overview



Fundamental Processor Extension: Atom / Molecule Model

Example Special Instruction

SPECIAL IN
STR
(SIs

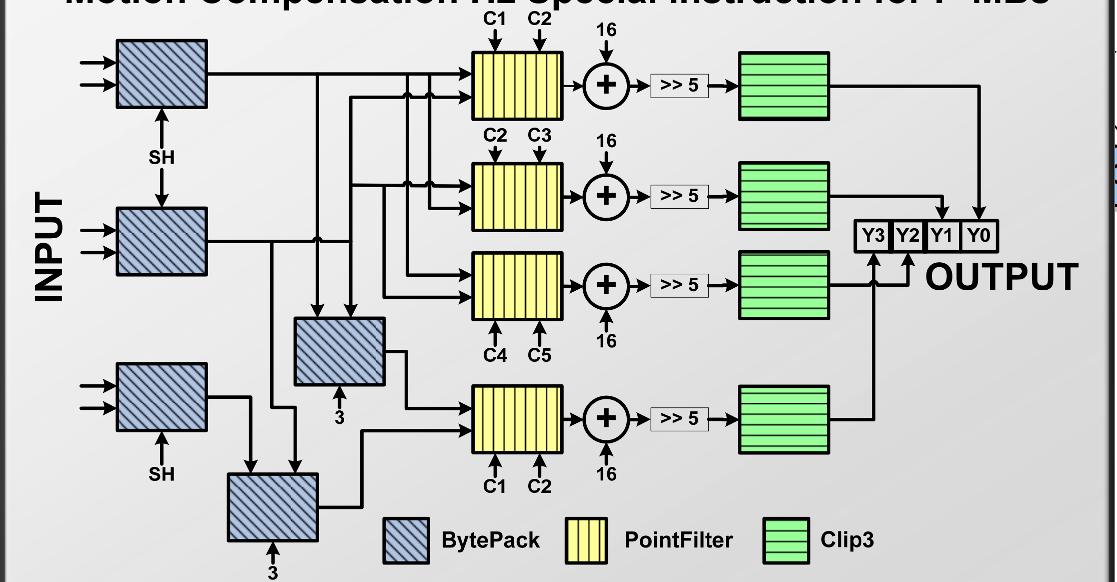
MOL
(an S
imple
by ar
Mole

ATO
(the
deno
insta
red f
Mole

- ❑ At
- ❑ M
- ❑ Sp

Csw

Motion Compensation Hz Special Instruction for P-MBs

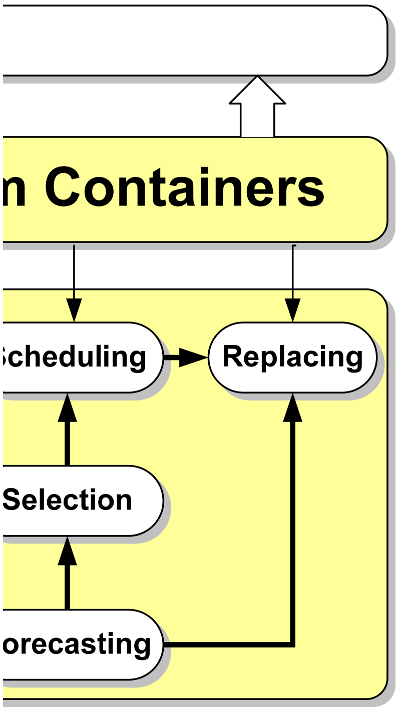


Ci = Coefficients for 6-tap Filtering
SH = Shift for Packing

Topics in RISPP

Details

- ❑ Formal Atom/Molecule Model
- ❑ Automatically Inserting Forecast Points
- ❑ Overview of the Run-Time System (State-Transition)
- ❑ Fine-tuning the Forecasts
- ❑ Selecting Molecules
- ❑ Scheduling Atom loading sequence
- ❑ Efficiency Comparison
- ❑ Hardware Prototype



Fix at Design/Compile Time — Adapt at Run Time

Design Time

- ❑ Fix the available reconfigurable hardware resources
- ❑ Fix the algorithms for the run-time system

Compile Time

- ❑ Determine Special Instruction composition
- ❑ Add forecasts to the application

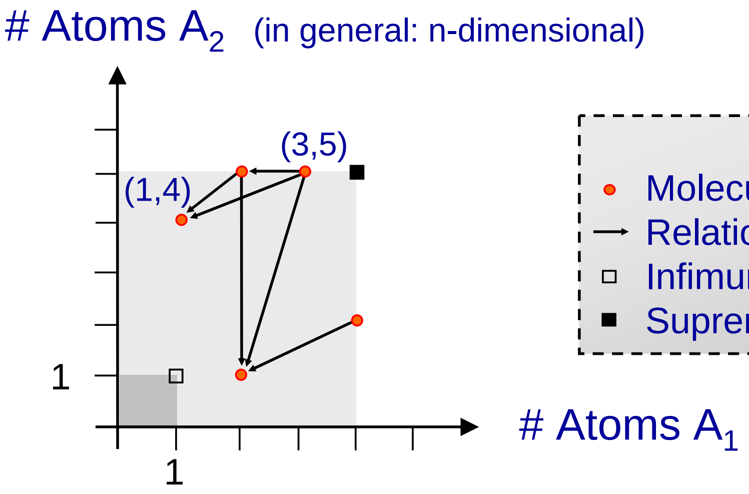
Run Time

- ❑ Adapt the forecasts
- ❑ Control SI execution
- ❑ Choose implementations (i.e. Molecules) for SIs

- ❑ Compile-time analysis: DAC`07
- ❑ Adaptation of the reconfiguration: SASO'07
- ❑ Atom/Molecule scheduling: DATE'08
- ❑ Molecule selection: DAC'08

Formal Atom/Molecule Model

Formal Atom / Molecule Model: Example

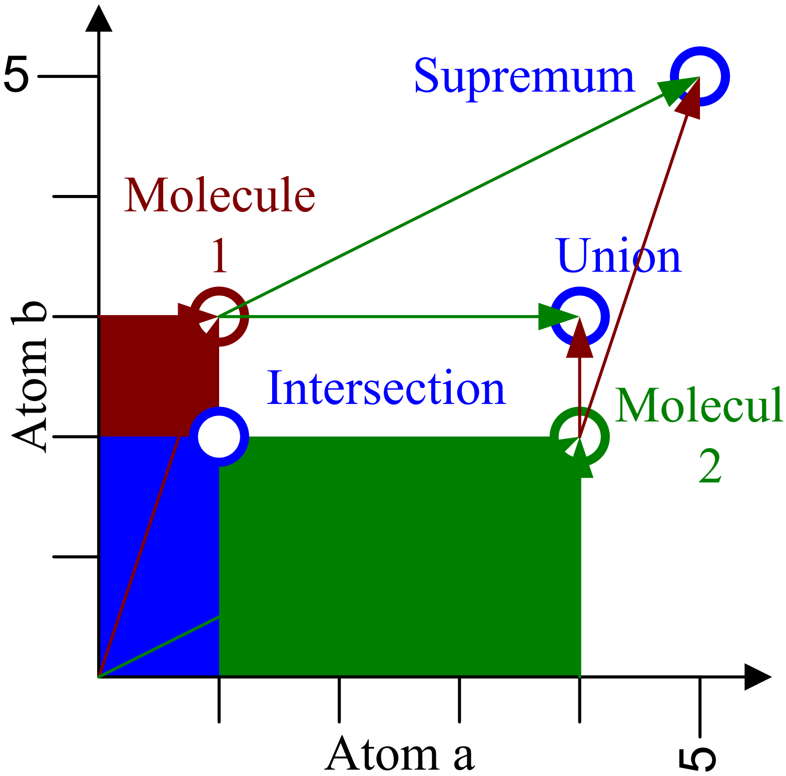


Legend:

- ● Molecule
- → Relation "is bigger or equal than"
- □ Infimum of the Molecules
- ■ Supremum of the Molecules

- **Molecule relations are e.g. needed when Molecules comprise each other**
  - **In such cases we can first configure the smallest possible Molecule with required functionality and then *upgrade* to faster implementations**

Example for base operators



Formal Atom / Molecule Model: Details

$(\mathbb{N}^n, \cup, \cap, \leq)$

- ❑ **Main data structure:**
  Set of all Molecules

$$\cup : \mathbb{N}^n \times \mathbb{N}^n \rightarrow \mathbb{N}^n$$
$$\vec{m} \cup \vec{o} := \vec{p}$$
$$p_i := \max\{m_i, o_i\}$$

- ❑ **Meta-Molecule to implement two Molecules, such that they can be executed consecutively, i.e. temporal domain (Abelian Group)**

$$\cap : \mathbb{N}^n \times \mathbb{N}^n \rightarrow \mathbb{N}^n$$
$$\vec{m} \cap \vec{o} := \vec{p}$$
$$p_i := \min\{m_i, o_i\}$$

- ❑ **Meta-Molecule for the common Atoms (indicator for *compatibility*)**

$$\vec{m} \leq \vec{o} \ \ \forall i \in [1, n] : m_i \leq o_i$$
$$\sup M := \bigcup_{\vec{m} \in M} \vec{m}$$
$$\inf M := \bigcap_{\vec{m} \in M} \vec{m}$$

- ❑ **Relation (Complete Lattice), with**
  - ❑ **Supremum: Meta-Molecule that is needed to implement all Molecules**
  - ❑ **Infimum: Meta-Molecule that is collectively needed for all Molecules**

Formal Atom / Molecule Model: Details

- **Determinant: number of Atoms needed to implement**

$$\left|\vec{m}\right| := \sum_{i\in[1,n]} m_i$$

- **Upgrading: Atoms that are additionally needed to**

$$\triangleright: \mathbb{N}^n \times \mathbb{N}^n \rightarrow \mathbb{N}^n;\ \vec{m} \triangleright \vec{o} := \vec{p},\ \ p_i := \begin{cases} o_i - m_i, & \text{if } o_i - m_i \geq 0 \\ 0 & \text{, else} \end{cases}$$

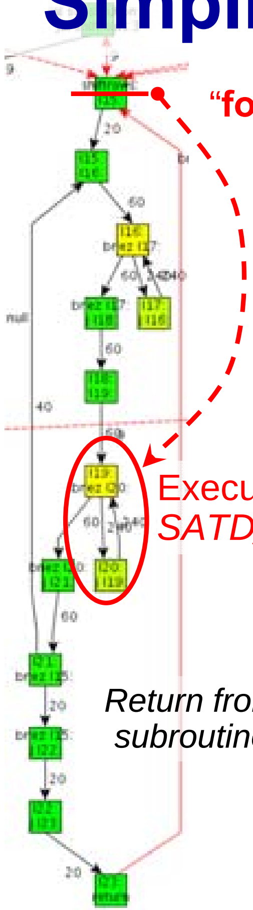
Dynamic Special Instruction (SI) Forecasting

Dynamic Special Instruction (SI) Forecasting

Why?

- Rotation (i.e. re-configuring Special Instructions) takes long time (ms)
  - => start early to prepare rotation (=> conflicts)
  - => possibility of SW execution exists if run-time system did not manage to rotate
- Many hot spots
  - Would require too many Special Instructions
  - Only a subset is available at a certain time (context-dependent)
  - => cannot determine at design time which Special Instruction to execute at which time during execution

Simplified Forecasting Example



"**forecast *SATD_4x4*, 42**"

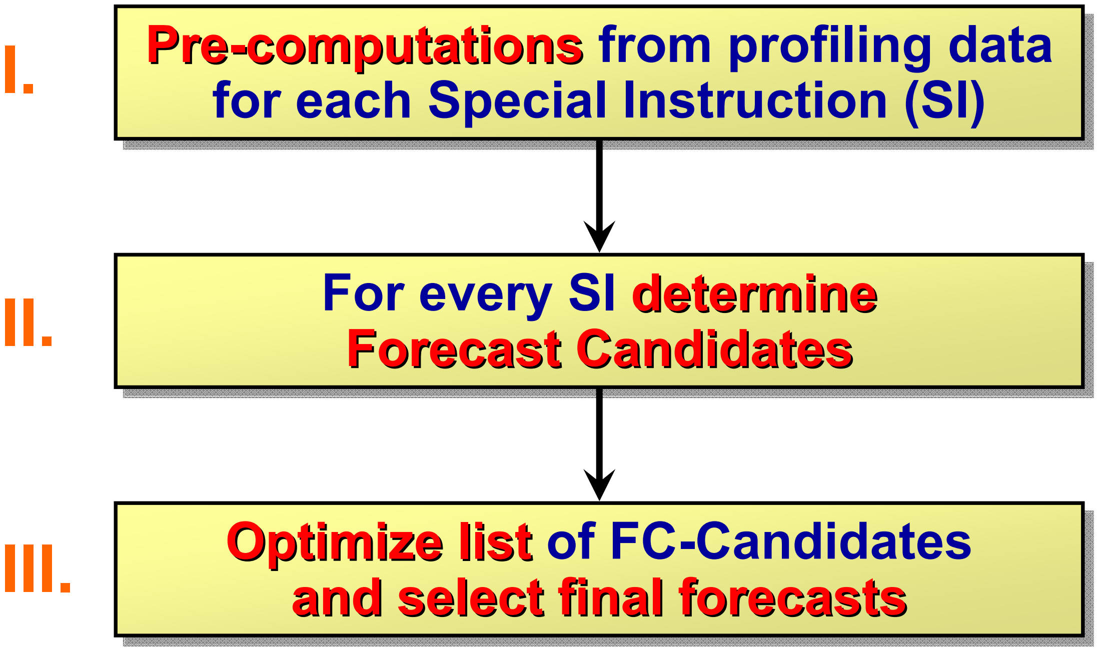
Time for Instruction set rotation

Executions of *SATD_4x4*

Return from subroutine

- **Control-flow graph**
  - **Each node is a Base-Block (BB)**
- **At compile time:**
  - **Determine points to forecast a SI**
  - **Add Forecast Instructions with forecast values (about the *SI importance*) to these points**
- **At run time:**
  - **Use the Forecasts to determine the Instruction Set rotation**
  - **Dynamically update the *importance* of the forecasted SIs**

Inserting Forecast Points (FCs): General Idea of Algorithm

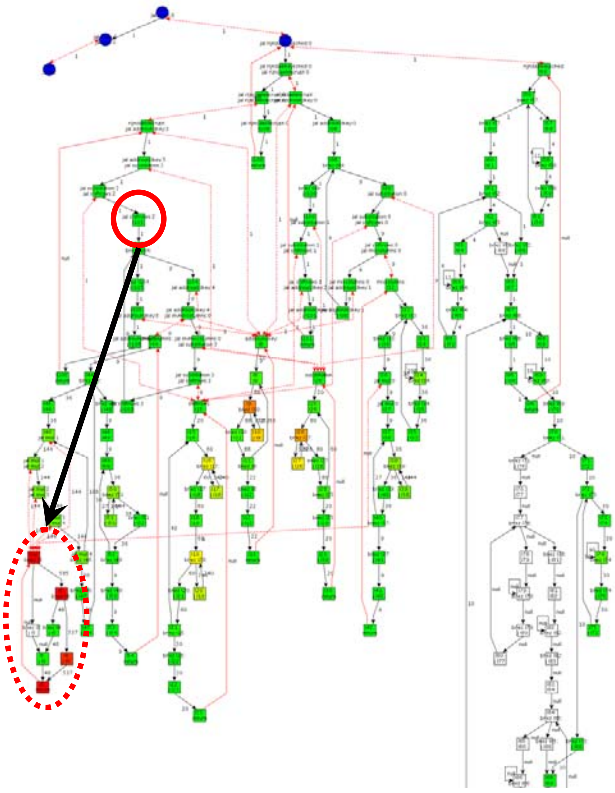


I. **Pre-computations** from profiling data for each Special Instruction (SI)

II. For every SI **determine Forecast Candidates**

III. **Optimize list** of FC-Candidates **and select final forecasts**

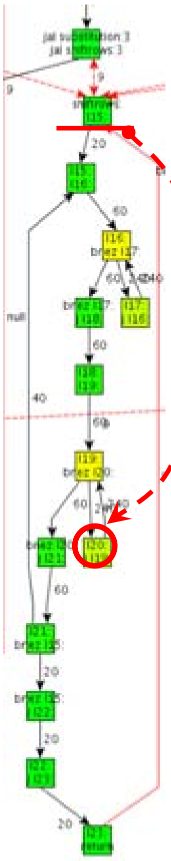
I. Pre-Computations



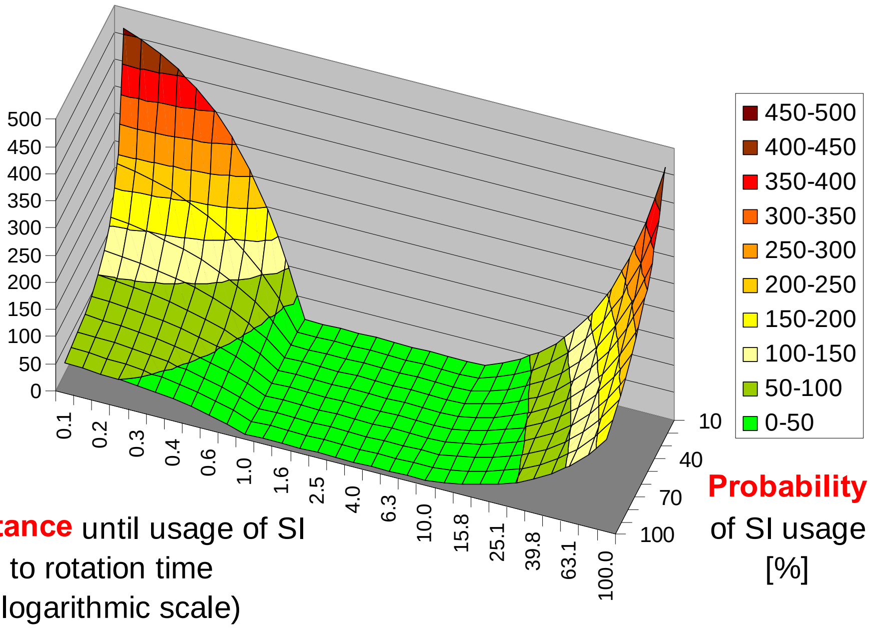
- ❑ **Pre-computations** are done on control-flow graph using profiling-information

- ❑ **Temporal Distance** from Base Block to SI execution
- ❑ **Probability** that the SI executions are reached
- ❑ **Number of executions** of this SI (if it is executed)

II. Forecast Decision Function (FDF)



Output: **Number of minimal SI usages** to issue a Forecast Candidate [#SI usages]



Temporal distance until usage of SI (relative to rotation time of SI with logarithmic scale) [t / T_{Rot}]

FDF-Details

$$\mathrm{FDF}(p,t) := \mathrm{offset} + \max\begin{cases} T_{Rot} - t \;/\; T_{SW} * p \\ T_{Rot} * t \;/\; p \\ 0 \end{cases}$$

$$\mathrm{offset} = \alpha * \left(E_{Rot} \big/ T_{SW} - T_{HW} \right)$$

- **Explanation and Parameter Description:**
  - **T: Time (Rot: for Rotation; SW: For SW Execution)**
  - **p: Probability**
  - **E: Energy**
  - **alpha: Parameter for Energy vs. Speedup fine-tuning**

III. Optimize list of FC Candidates

// S_1, …, S_k are the SIs of the FC Candidates in this BB

General Idea:

While the forecasted SIs in a Base Block consume *too many* area

Remove the forecast with the worst

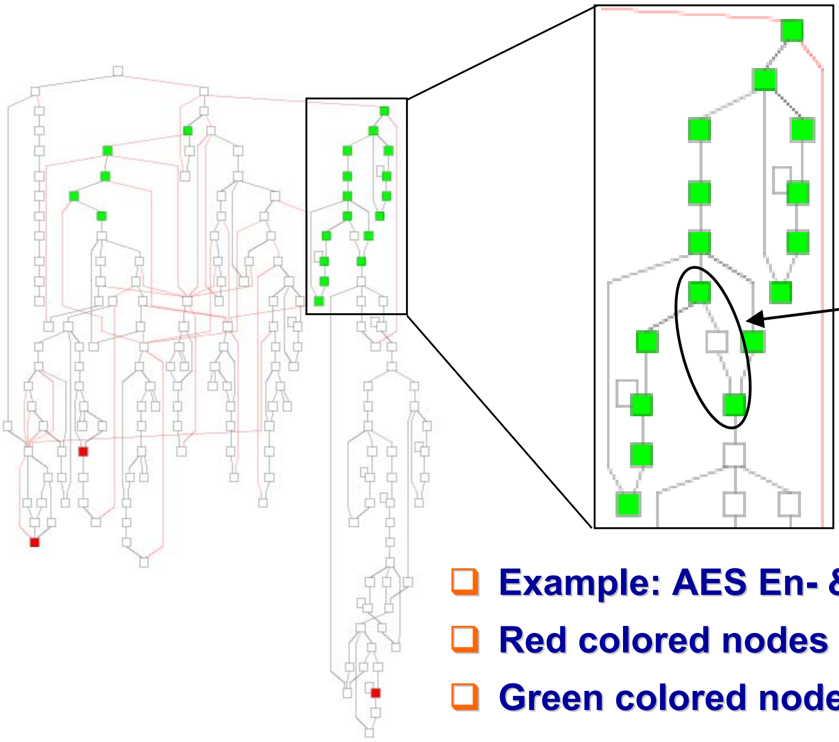
$$\frac{\textbf{Achieved Speedup}}{\textbf{Exclusively used Atoms}}$$

```
        } else {
          break;
        }
  }
```

Select final Forecasts

- **Prerequisite: SI-Termination is already added (i.e. FCs, that a SI is no longer needed)**
- **Optimization goals**
  - **As few FCs as possible, as many as needed**
  - **Choose FCs with a good trade-off between 'sufficiently early' and a 'high execution probability'**
- **For each FC-Type T start a Depth-First-Search on the transposed Base Block graph (i.e. all edges reversed)**
  - **No recursive relegation, if a node is not a FC-Candidate for the current FC-Type T**
  - **The current Node N becomes a Forecast IFF**
    - A successor S of Node N is not a FC-Candidate for Type T **AND**
    - The Path beginning with S is not soon (in terms of ms, not BBs) reaching another FC-Candidate **AND**
    - We don't have added a FC a few nodes previously

Example: Choosing FCs

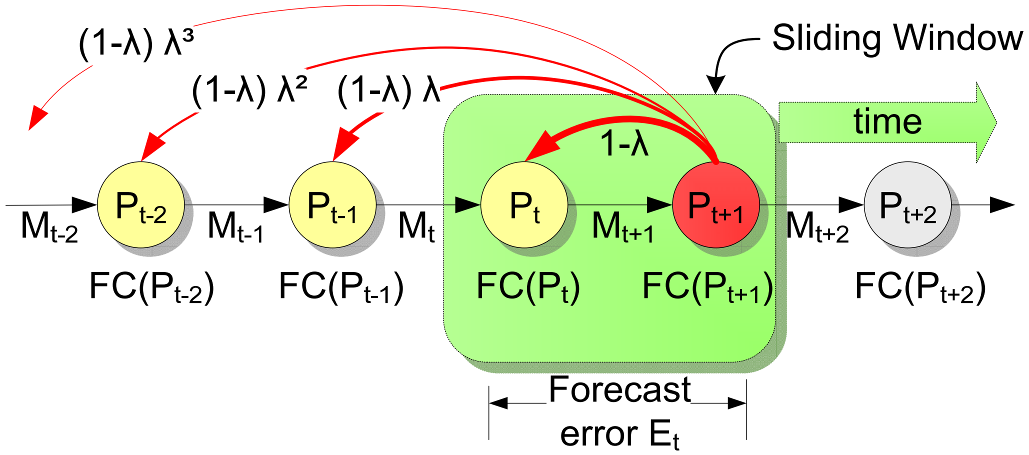


We leave the path of FC-Candidates, but immediately return to it → no extra FC needed

- Example: AES En- & Decryption
- Red colored nodes contain an SI
- Green colored nodes: FC-Candidates

Forecast fine-tuning

Adapting the Special Instruction Forecasts



- **Computing the Error:** $E_t = M_{t+1} + \gamma FC(P_{t+1}) - FC(P_t)$
- **Back-Propagating the Error:** $FC(P_t) = FC(P_t) + \alpha E_t$

Legend

P: Forecast Point
FC: Forecast Value in the Point
M: Monitored number of SI executions

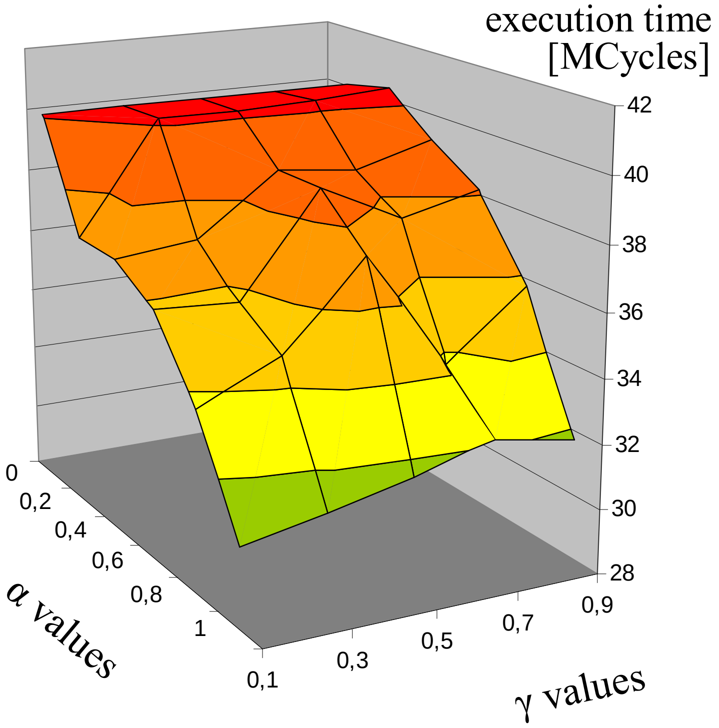
Parameter Evaluation

- **λ > 0 rapidly increases overhead**
  - **Max speedup vs. λ=0: 1.10x**
  - **λ=0 is a good performance vs. overhead compromise**

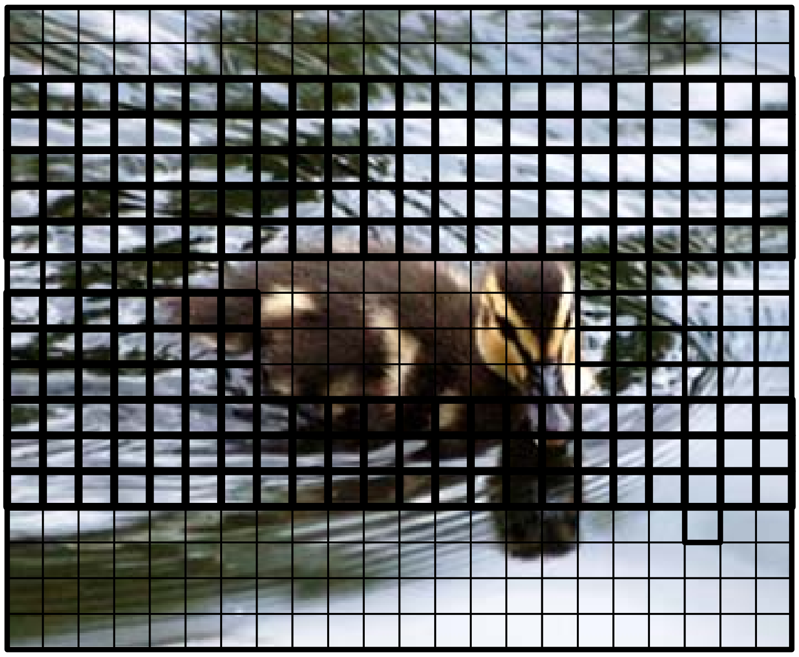
	λ=0.6	λ=1.0
Min.* speedup	1.01x	1.01x
Avg.* speedup	1.03x	1.03x
Max.* speedup	1.08x	1.10x

*comparing with λ=0

Evaluation for λ=0



Forecasting I- / P-MBs in practice



Macroblock (MB): 16x16 pixel Block

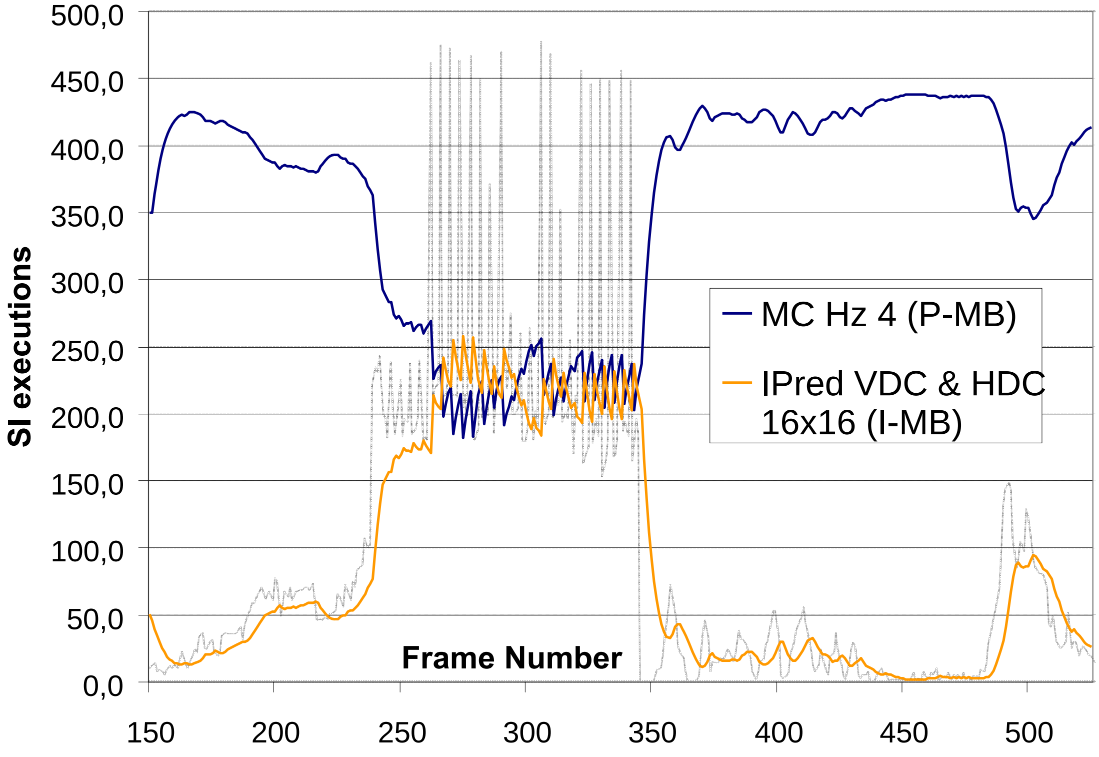
Common Intermediate Format (CIF): Image with 352x288 pixel size. 1CIF = 396 MBs

□ **I-MB:** Blocks with thick border

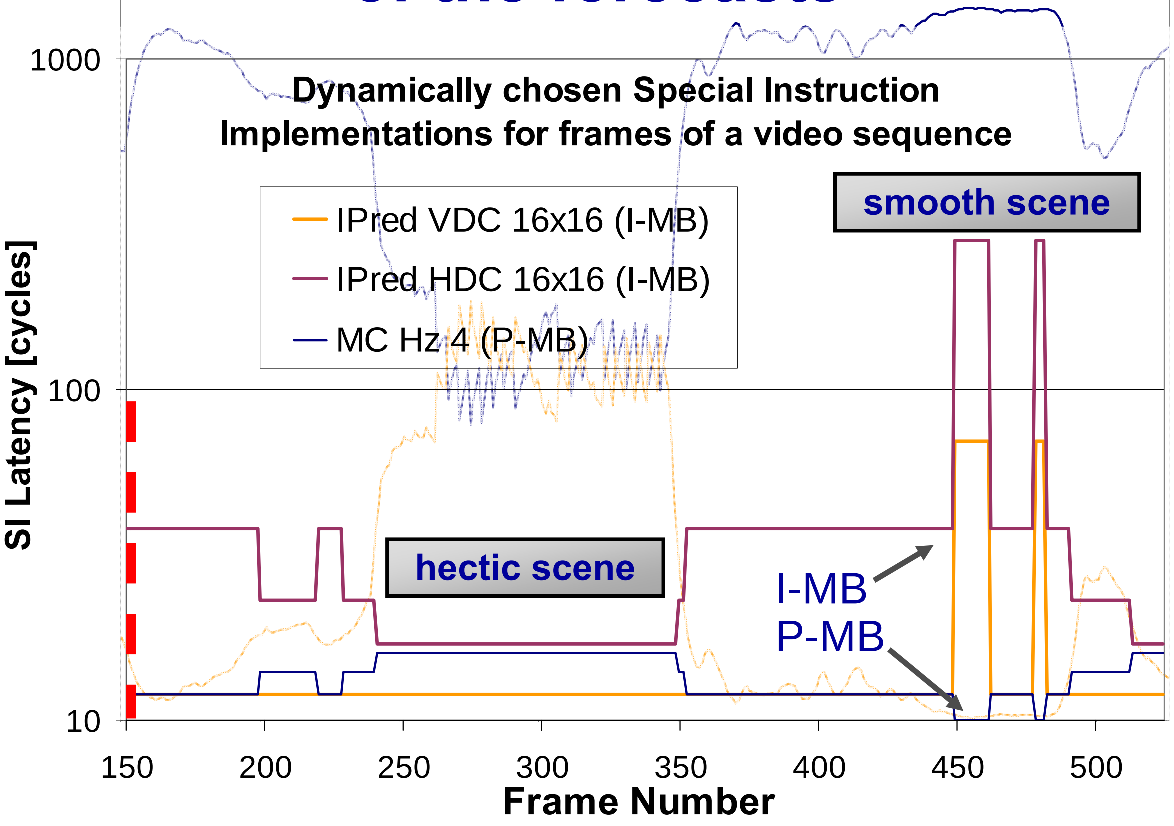
□ **P-MB:** Blocks with thin border

src: http://www.copyright-free-pictures.org.uk

Dynamically Adapting the Forecasts to the actual application requirements

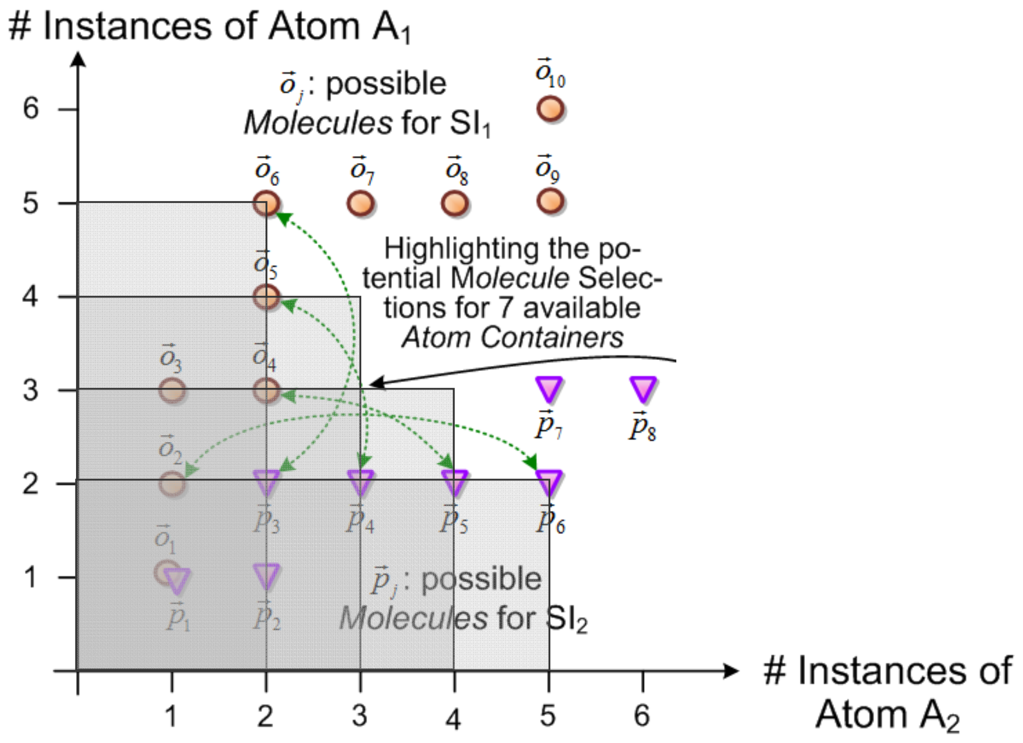


Selecting Molecules out of the forecasts



Molecule Selection

Molecule Selection



Highlighting the potential *Molecule* Selections for 7 available *Atom Containers*

Formal Selecting Molecules for SIs

- ❑ **Formalized as a Knapsack Problem**
  - → **NP-Hard**
- ❑ **Input to the selection:**

$$F = \{(M_i, f_i, t_i)\} \qquad i \text{ □ index of a certain SI}$$

- ❑ **Chose exactly one Molecule to implement a SI:**

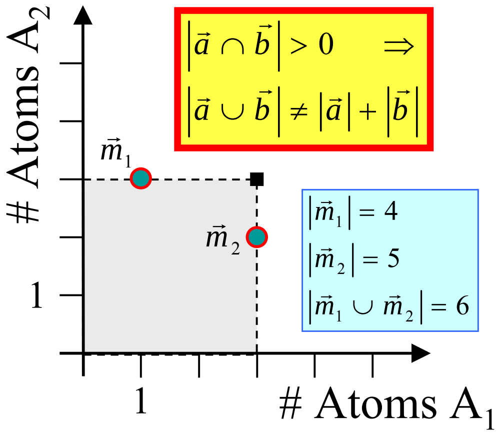
$$\forall i : |S \cap M_i| = 1$$

- ❑ **Stay within the capacity of the Knapsack:**

$$\left|\bigcup_{\vec{o} \in S} \vec{o}\right| \le N$$

Selecting Molecules for SIs (cont'd)

- Unlike traditional Knapsack: The weight of a Molecule is not constant!
  - Two Molecules might share some Atoms:
- Special Version: Set-Union Knapsack Problem

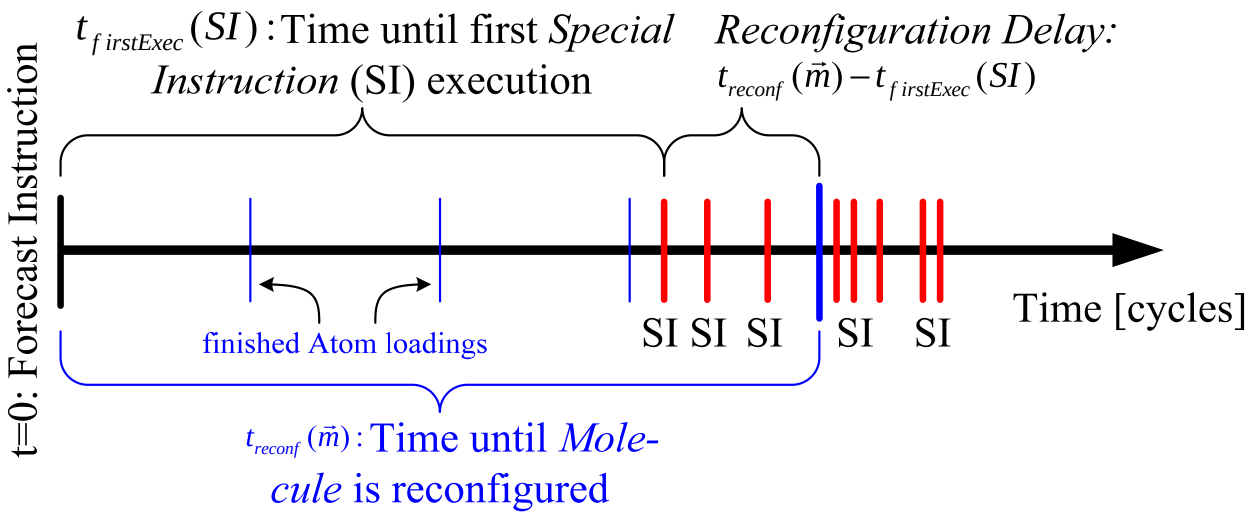


- Profit function (optimization goals):

$$\text{maximize} \sum_{\forall i:\vec{o} \in M_i \cap S} profit(\vec{o},\, f_i, t_i)$$

Relevant Selection Parameters

- Depending on the selection, the reconfiguration may finish far too late

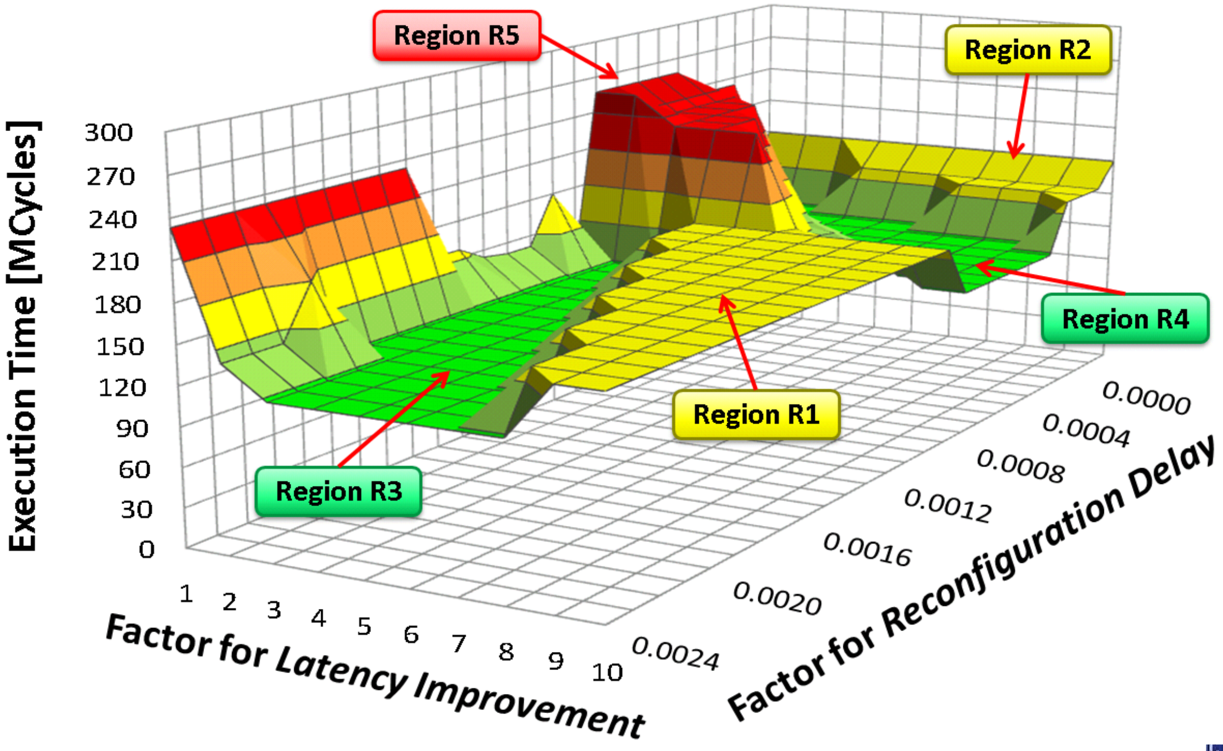

$t_{firstExec}(SI)$: Time until first *Special Instruction* (SI) execution
Reconfiguration Delay: $t_{reconf}(\vec{m}) - t_{firstExec}(SI)$
t=0: Forecast Instruction
finished Atom loadings
SI SI SI
SI
SI
Time [cycles]
$t_{reconf}(\vec{m})$: Time until *Molecule* is reconfigured

Profit function for a Molecule

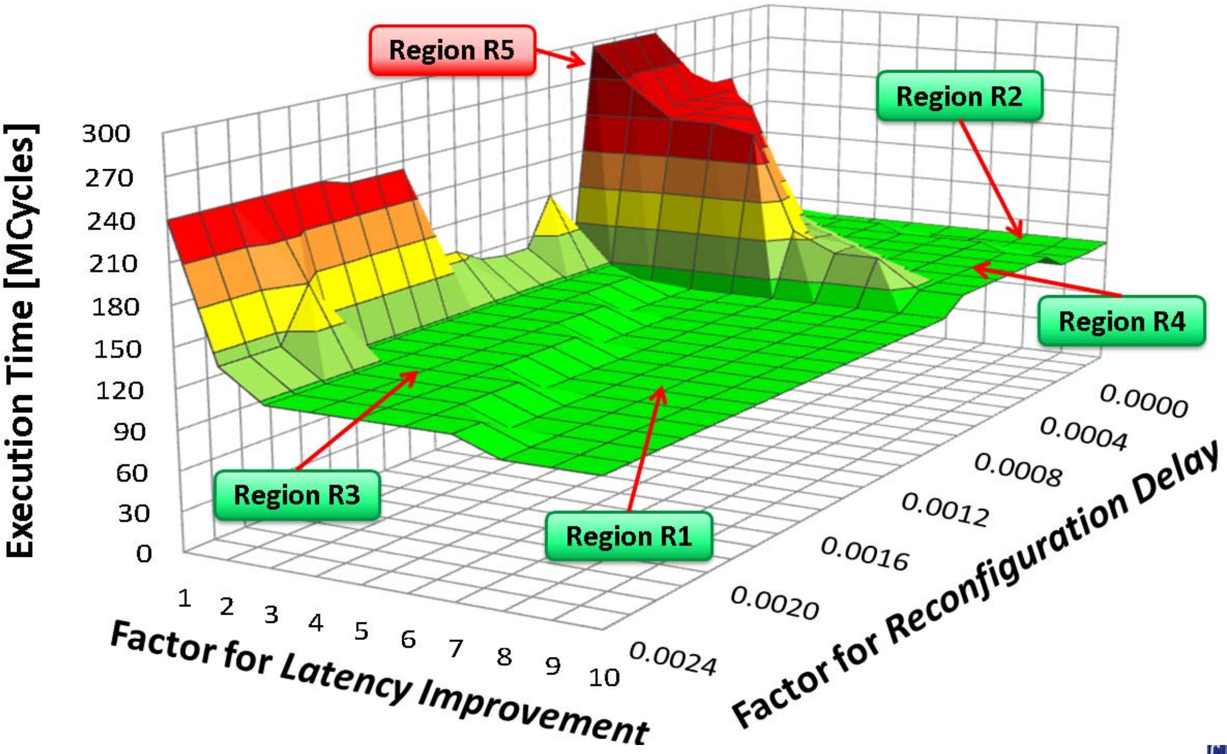
- Selection parameters L & R are used to scale the parameters
  - Latency Improvement
  - Reconfiguration Delay
- Additional parameter:
  - Expected SI execution frequency

$$profit(\vec{m}, f_i, t_i) := f_i \cdot \left(\begin{array}{l} L \cdot \left(\begin{array}{l} latency(\vec{m}_{i\_SW}) \\ -latency(\vec{m}_{ij}) \end{array} \right) - \\ R \cdot \max\left(0, \begin{array}{l} t_{reconf}(\vec{m}) \\ -t_i(SI) \end{array} \right) \end{array} \right)$$

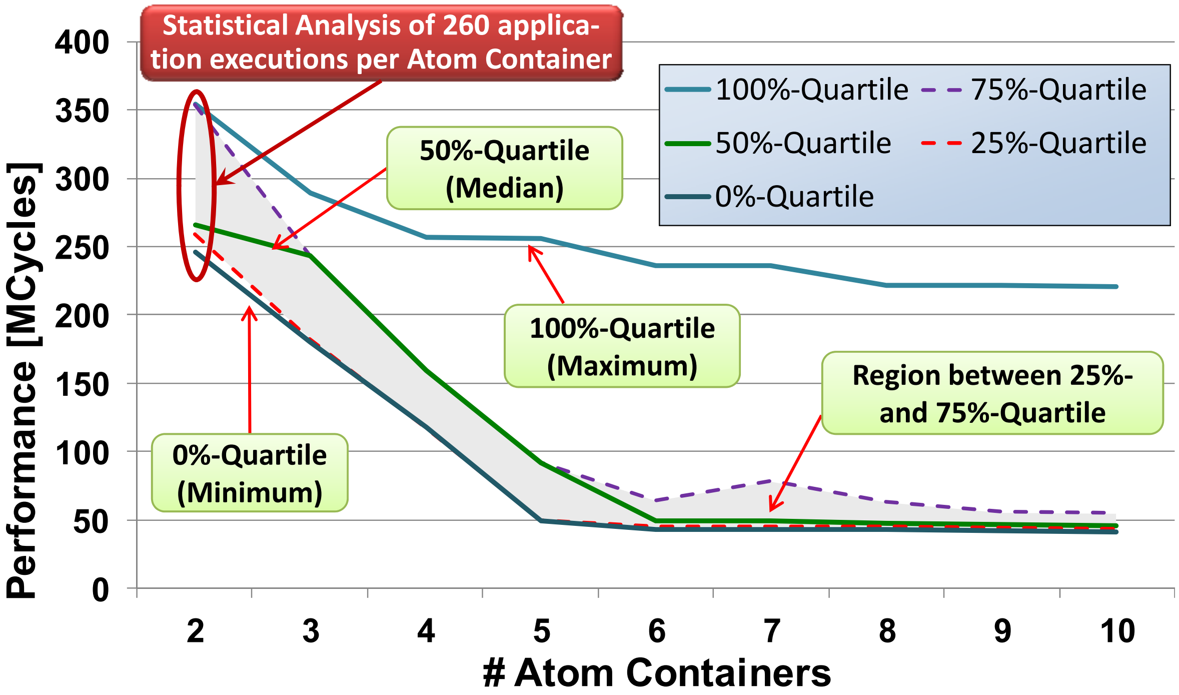
Evaluation of the profit function (greedy)



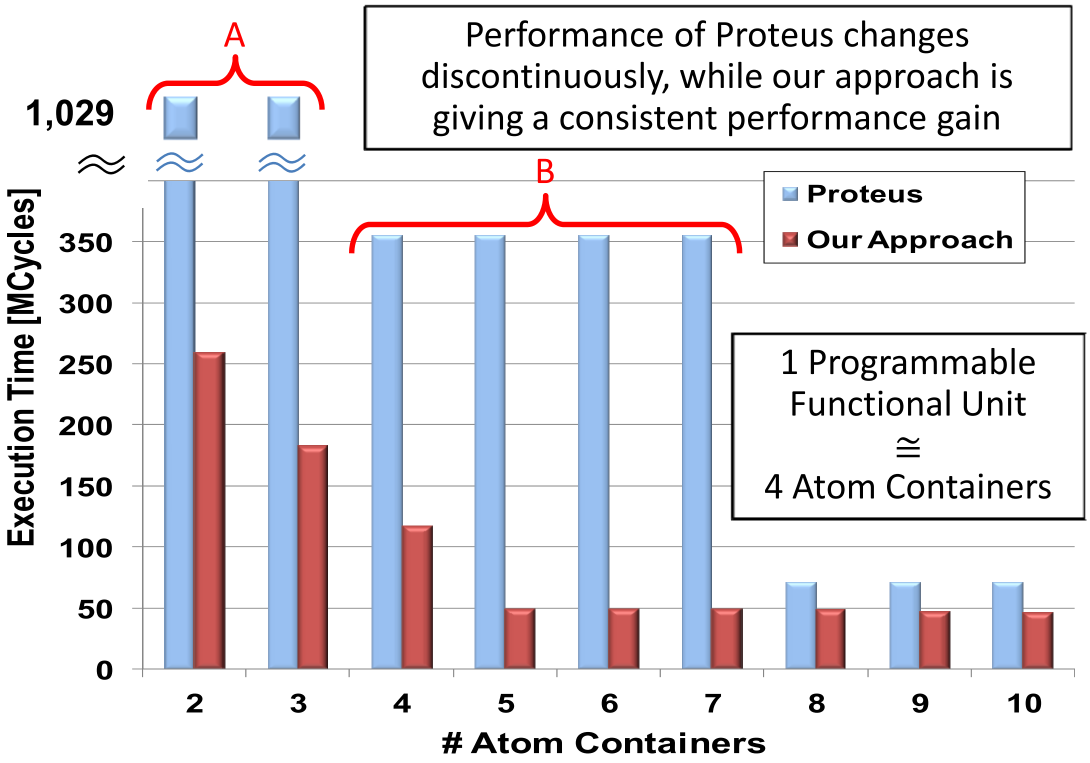
Evaluation of the profit function (optimal)



Statistical Analysis of the profit function parameters for the greedy implementation



Comparing Reconfiguration on SI-level (Proteus) with Atom-level (RISPP)



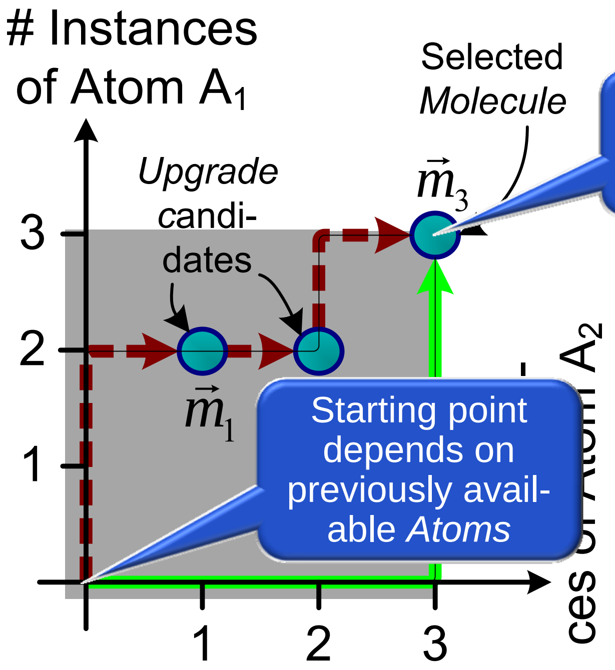
Performance of Proteus changes discontinuously, while our approach is giving a consistent performance gain

1 Programmable Functional Unit ≅ 4 Atom Containers

Atom/Molecule Scheduling

Determining Atom loading sequence

- ❑ **Problem: Reconfiguration is slow**
- ❑ **Constraint: At most one reconfiguration at a time**



Selected *Molecule* depends on changing system requirements

Starting point depends on previously avail-able *Atoms*

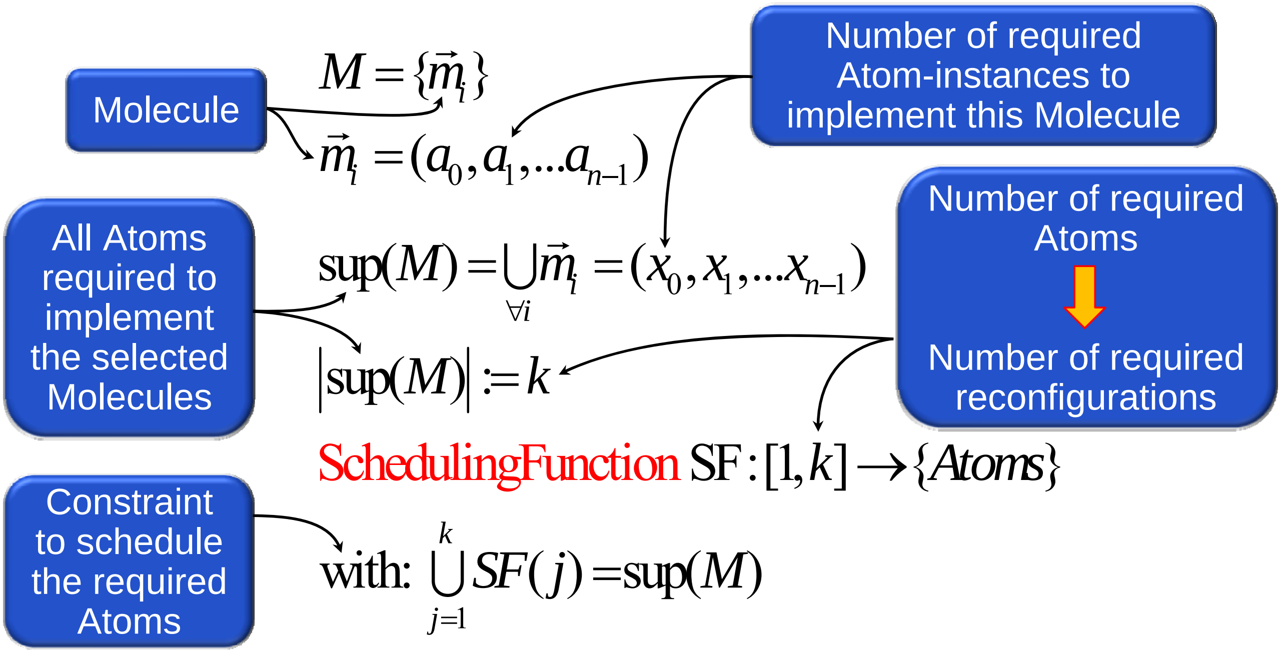
	fastest available le	
1	—	—
2	—	—
3	$\vec{m}_1$	—
4	$\vec{m}_2$	—
5	$\vec{m}_2$	$\vec{m}_2$
6	$\vec{m}_3$	$\vec{m}_3$

Overview: Implemented SIs

	Special Instruction	**# utilized *Atom*-types**	**# different possible *Molecules***
Motion Estimation (ME)	SAD	1	3
	SATD	4	20
Encoding Engine (EE)	(I)DCT	3	12
	(I)HT_2x2	1	2
	(I)HT_4x4	2	7
	MC_4	3	11
	IPred_HDC	2	4
	IPred_VDC	1	3
Filter (LF)	LF_BS4	2	5

Defining the Scheduling Problem

- Input: Set of selected Molecules *M*



Molecule

Number of required Atom-instances to implement this Molecule

All Atoms required to implement the selected Molecules

Number of required Atoms ⇩ Number of required reconfigurations

Constraint to schedule the required Atoms

$$M = \{\vec{m}_i\}$$

$$\vec{m}_i = (a_0, a_1, \ldots a_{n-1})$$

$$\sup(M) = \bigcup_{\forall i} \vec{m}_i = (x_0, x_1, \ldots x_{n-1})$$

$$|\sup(M)| := k$$

$$\text{SchedulingFunction } SF : [1,k] \rightarrow \{Atoms\}$$

$$\text{with: } \bigcup_{j=1}^{k} SF(j) = \sup(M)$$

Schedule Molecules, not Atoms

- Idea: The SI performance changes, when a new Molecule becomes available
  - To reduce the complexity and to become target oriented: determine the Molecule loading sequence

$$M = \{\vec{m}_i\}$$

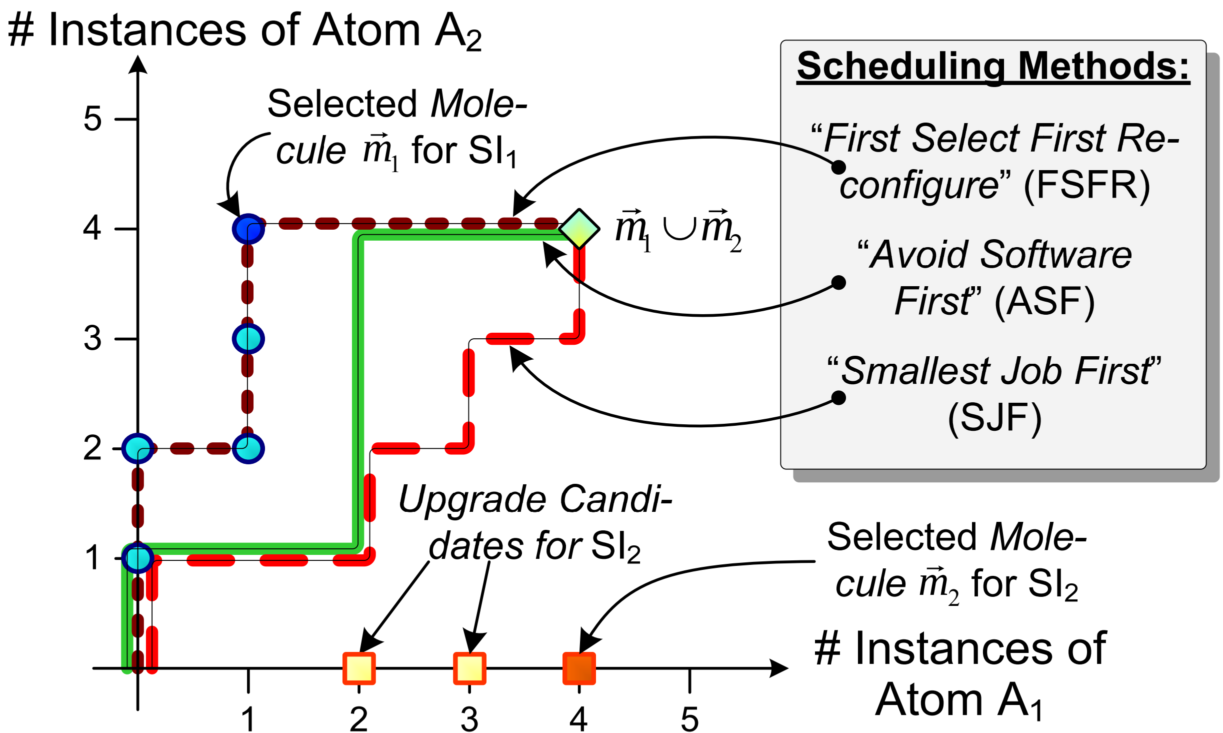
- Consider upgrade candidates (once):

$$M' = \bigcup_{\vec{m} \in M} \{\vec{o} : \vec{o} \le \vec{m} \wedge \vec{o}.getSI() = \vec{m}.getSI()\}$$

- Trim candidates (iteratively):

$$M'' = \left\{ \vec{m} \in M' : \begin{pmatrix} |\vec{a} \triangleright \vec{m}| > 0 \wedge \vec{m}.getLatency() < \\ \vec{m}.getSI().getFastestAvailable\text{-} \\ Molecule(\vec{a}).getLatency() \end{pmatrix} \right\}$$

Comparing different Scheduling Methods for 2 Selected SIs


# Instances of Atom A_2
5
4
3
2
1
Selected *Molecule* $\vec{m}_1$ for SI_1
$\vec{m}_1 \cup \vec{m}_2$
Upgrade *Candidates* for SI_2
Selected *Molecule* $\vec{m}_2$ for SI_2
# Instances of Atom A_1
1
2
3
4
5

Scheduling Methods:

"*First Select First Reconfigure*" (FSFR)

"*Avoid Software First*" (ASF)

"*Smallest Job First*" (SJF)

Our Scheduling Approach: Highest Efficiency First (HEF)

```
$bestBenefit \leftarrow 0$
$\forall \vec{o} \in M' \{$
    $benefit \leftarrow \vec{o}.getSI().getExpectedExecutions() \ * \ (\vec{o}.getSI().bestLatency - \vec{o}.getLatency()) / |\vec{a} \triangleright \vec{o}|$
    if $(benefit > bestBenefit) \{$
        $bestBenefit \leftarrow benefit$
        $\vec{m} \leftarrow \vec{o}$
    $\}$
$\}$
```

SI execution frequency

Additionally required Atoms

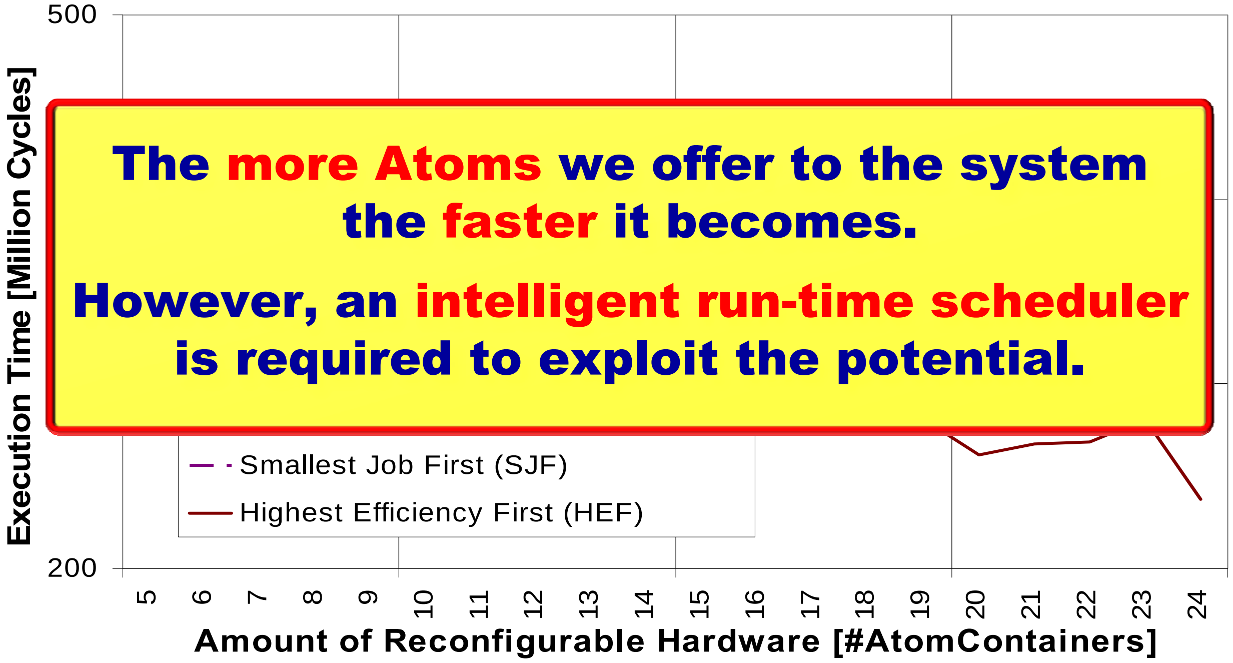
Latency improvements (per SI execution)

```
        $bestBenefit \leftarrow benefit; \ \vec{m} \leftarrow \vec{o};$
      }
  }
  // schedule the chosen Molecule
  $\forall Atoms \ a_i \in (\vec{a} \triangleright \vec{m}) \ scheduledList.push(a_i);$
  $\vec{a} \leftarrow \vec{a} \cup \vec{m};$
  $\vec{m}.getSI().bestLatency \leftarrow \vec{m}.getLatency();$
}
return $scheduledList;$
```

3. Extract the additionally needed Atoms

4. Trim the candidates

Comparing our proposed scheduling scheme



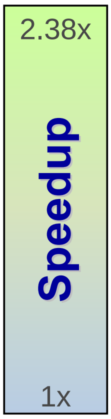
The more Atoms we offer to the system the faster it becomes.

However, an intelligent run-time scheduler is required to exploit the potential.

- Encoding 140 frames (352x288 resolution) with H.264

Speedup due to our scheduler and due to our architecture

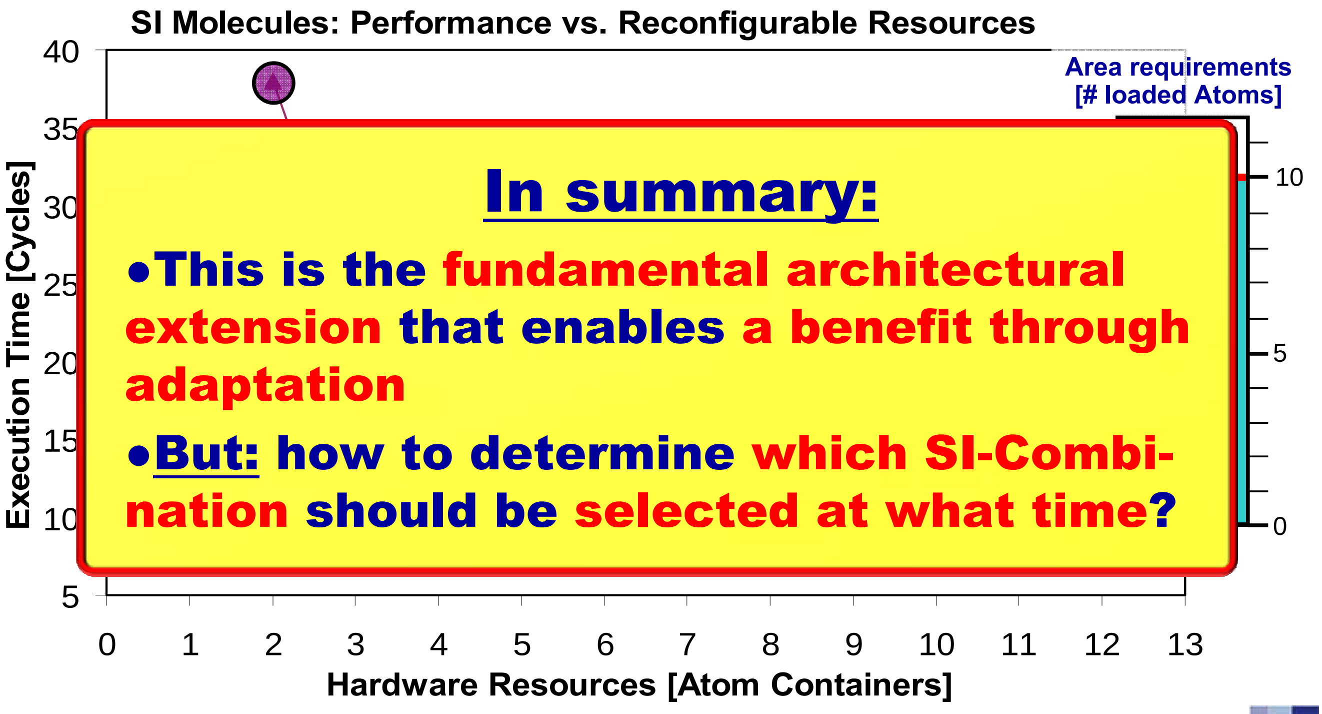
- Comparing state-of-the-art approaches (Molen) with our architecture (ASF: simple but nontrivial Scheduler)
- Evaluating, what our proposed Scheduler (HEF) achieves on top



#ACs	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
HEF vs. ASF	1.00	1.04	1.04	1.06	1.05	1.08	1.06	1.06	1.13	1.18	1.21	1.26	1.36	1.48	1.45	1.52	1.51	1.39	1.26	1.52

Efficiency / Utilization

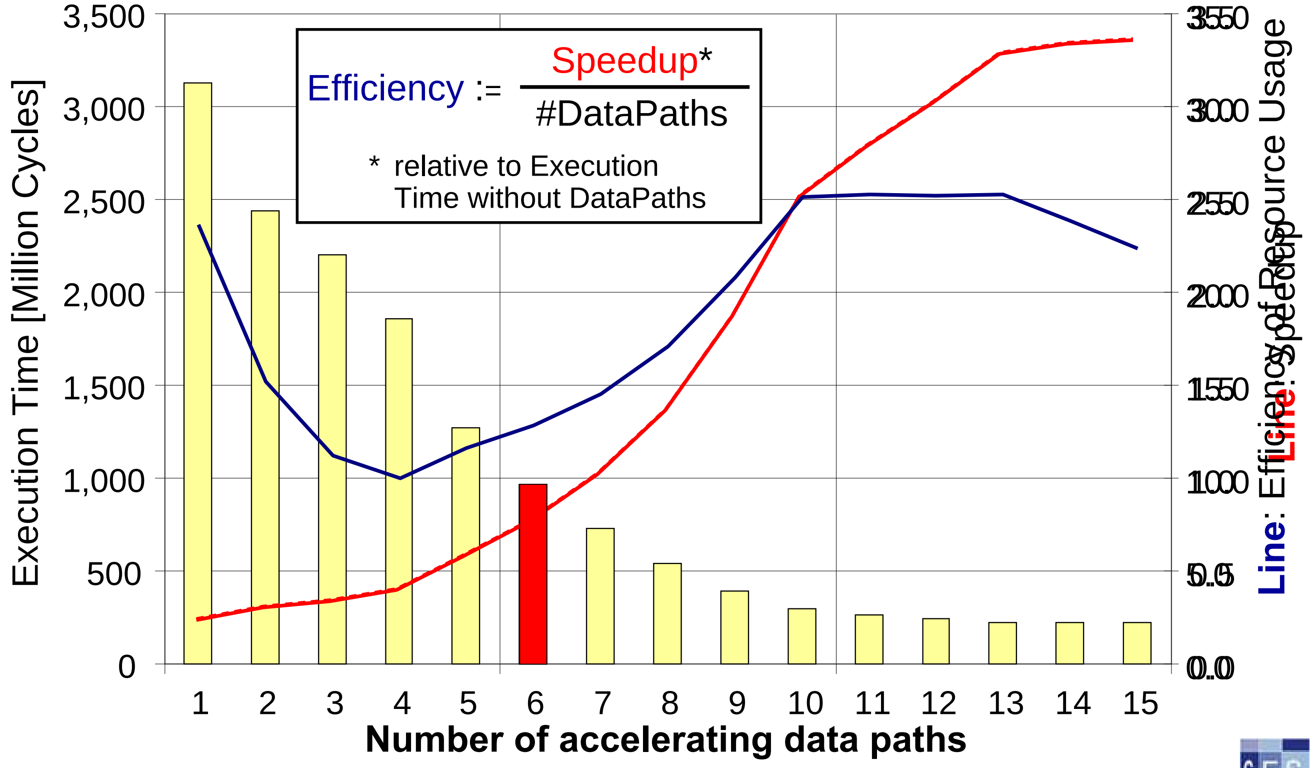
Adaptivity Through Dynamic Performance vs. Area Trade-off



In summary:

- **This is the fundamental architectural extension that enables a benefit through adaptation**
- **But: how to determine which SI-Combination should be selected at what time?**

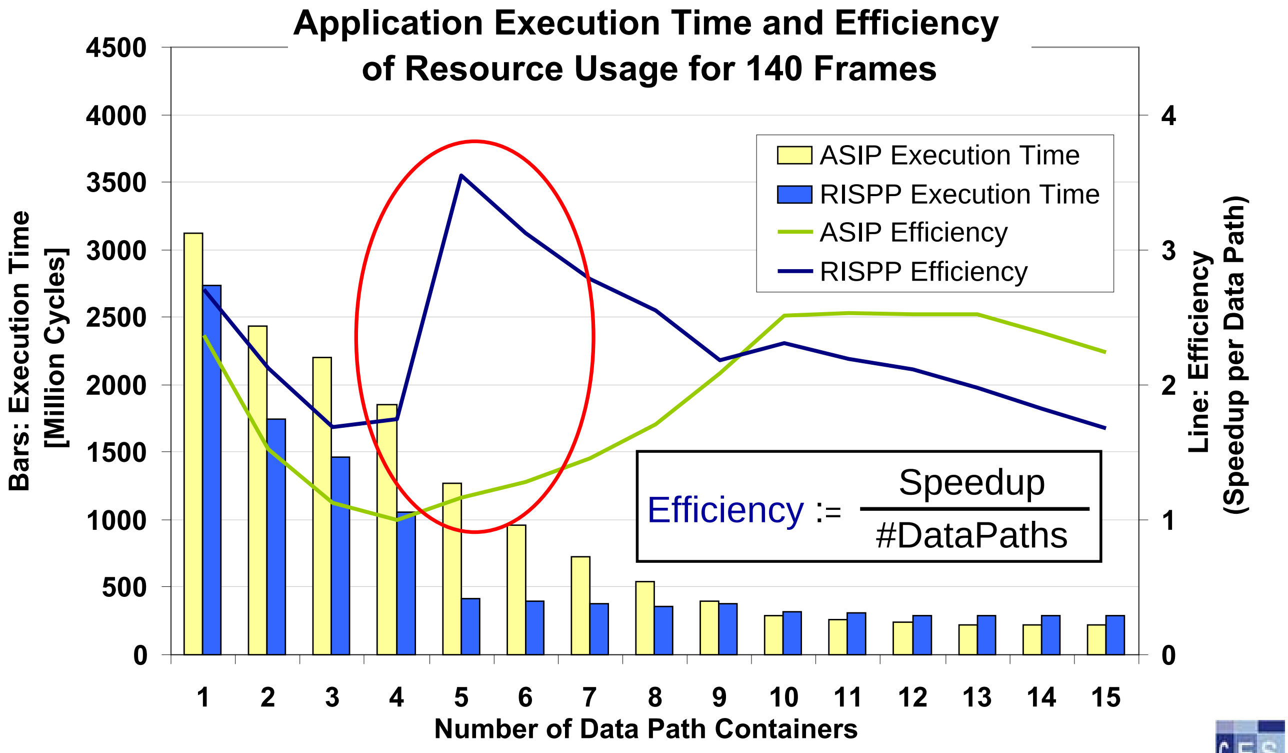
Analyzing ASIPs for varying amount of accelerating data paths



$$\text{Efficiency} := \frac{\text{Speedup*}}{\text{\#DataPaths}}$$

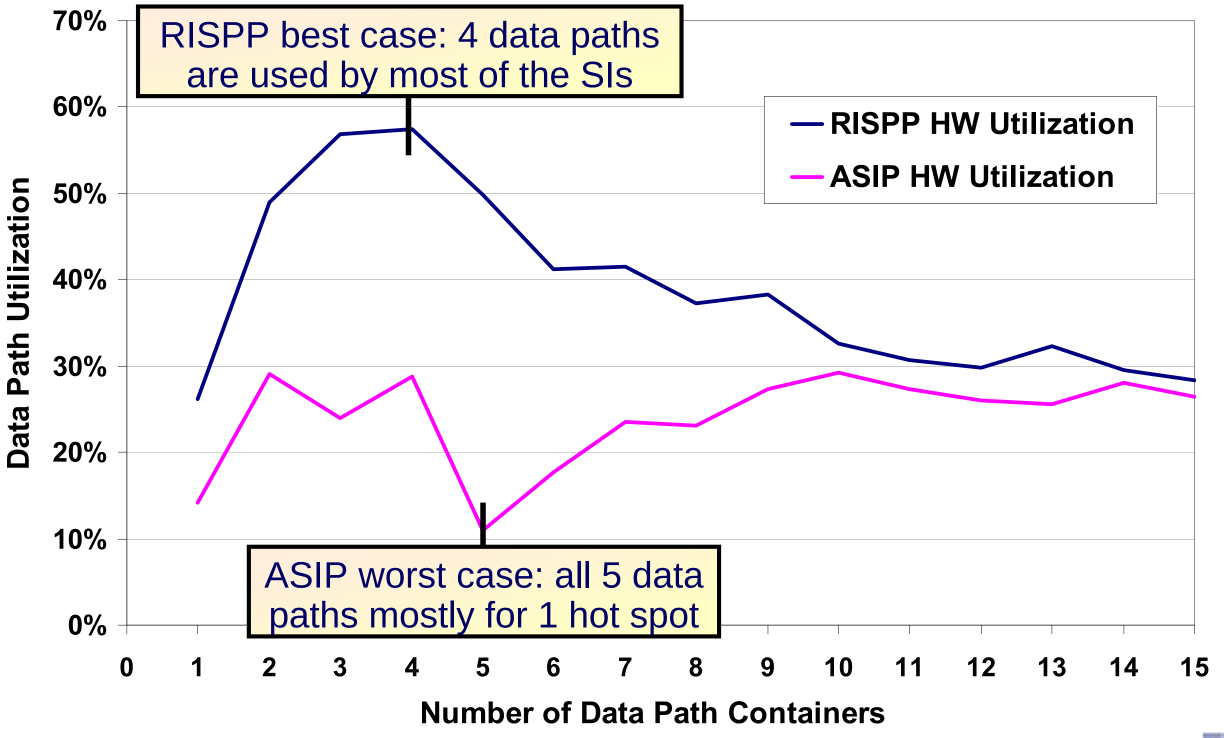
* relative to Execution Time without DataPaths

Results: execution time and efficiency of resource usage for ASIP and RISPP



$$\text{Efficiency} := \frac{\text{Speedup}}{\text{\#DataPaths}}$$

Data path utilization during hot spot execution



RISPP best case: 4 data paths are used by most of the SIs

ASIP worst case: all 5 data paths mostly for 1 hot spot

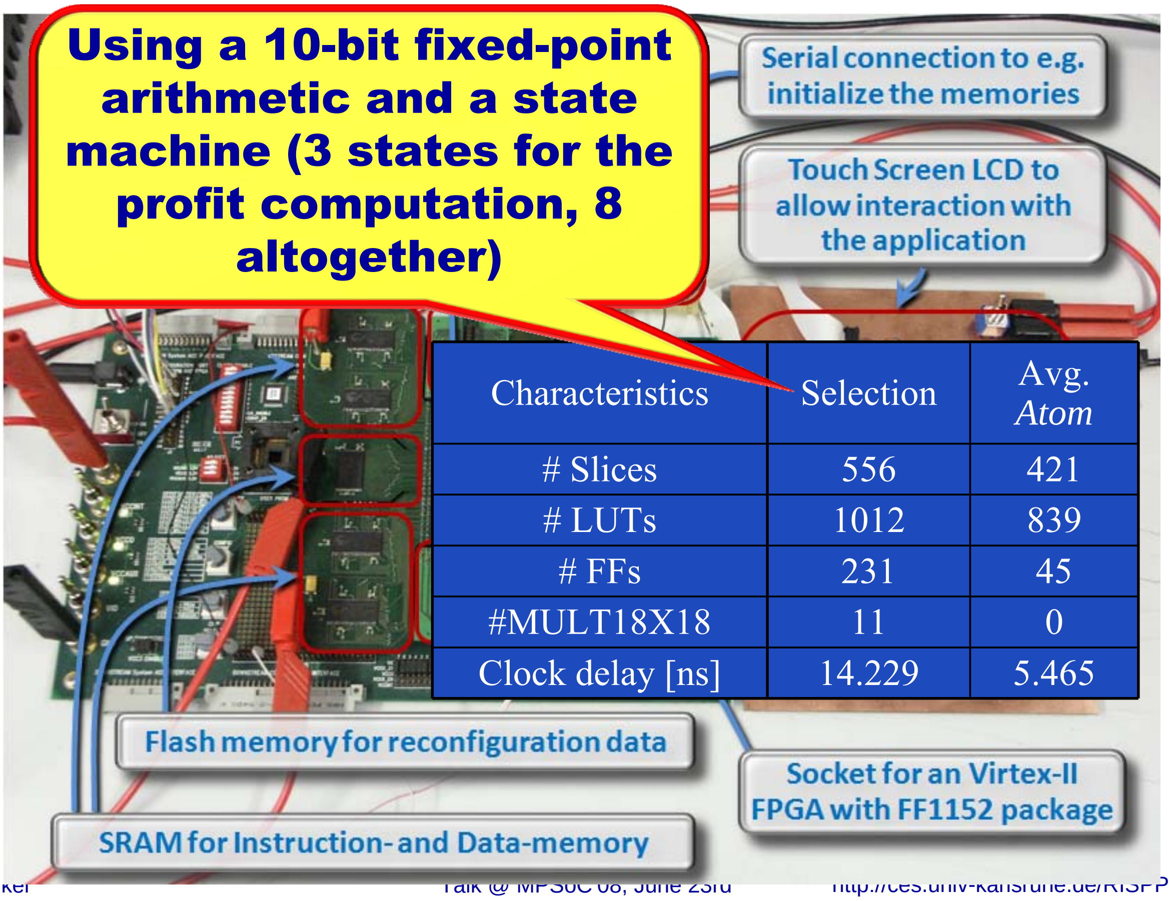
Summary of comparison of ASIP and RISPP

	ASIP			RISPP		
	Min	Avg	Max	Min	Avg	Max
Execution Time [MCycles]	220.6	999.6	3126	288.3	715.1	2734
Speedup vs. GPP	2.4	16.8	33.6	2.7	17.6	25.7
Efficiency	1.0	1.9	2.5	1.7	2.3	3.6

- ❑ **Measurements for different numbers of data paths**
  - ❑ **Avg. & max. consider many data paths/containers**
- ❑ **Only for huge number of data paths ASIP is better**

Hardware Prototype

Hardware Prototyping Board



Using a 10-bit fixed-point arithmetic and a state machine (3 states for the profit computation, 8 altogether)

Serial connection to e.g. initialize the memories

Touch Screen LCD to allow interaction with the application

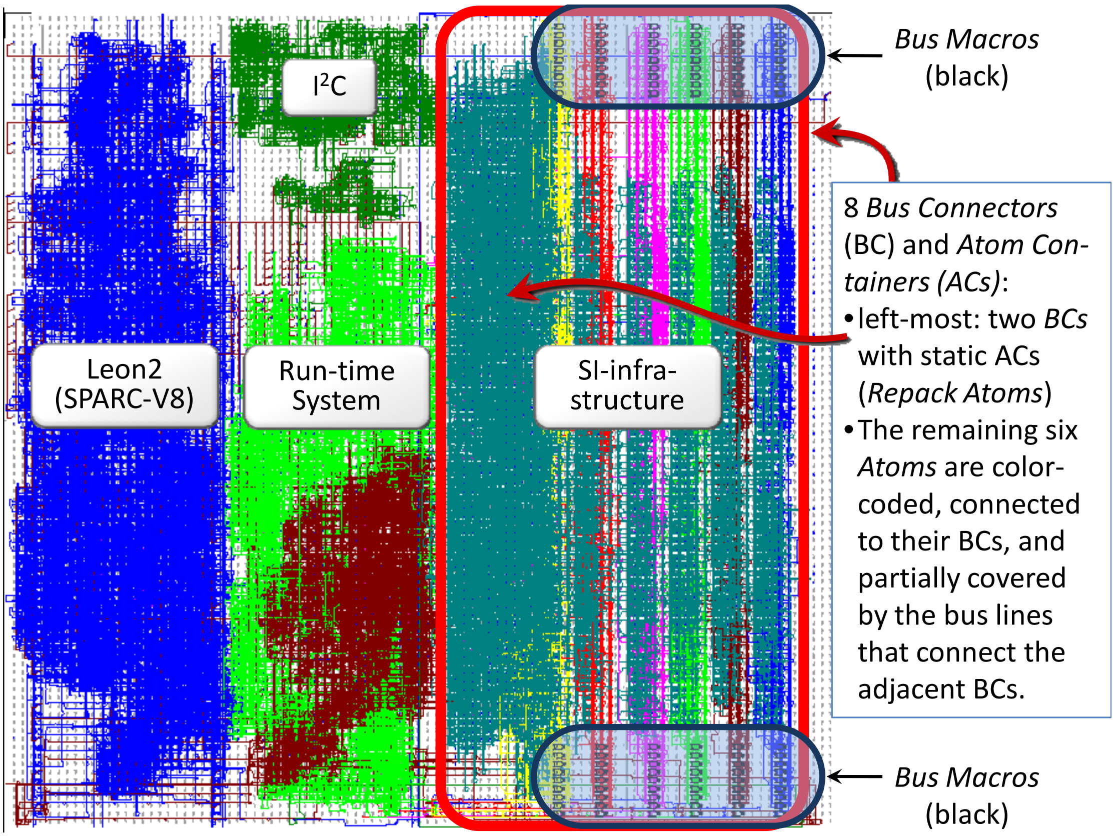
Characteristics	Selection	Avg. *Atom*
# Slices	556	421
# LUTs	1012	839
# FFs	231	45
#MULT18X18	11	0
Clock delay [ns]	14.229	5.465

Flash memory for reconfiguration data

Socket for an Virtex-II FPGA with FF1152 package

SRAM for Instruction- and Data-memory

Latest Version of Hardware Prototype



Bus Macros (black)

8 *Bus Connectors* (BC) and *Atom Containers (ACs)*:

- left-most: two *BCs* with static ACs (*Repack Atoms*)
- The remaining six *Atoms* are color-coded, connected to their BCs, and partially covered by the bus lines that connect the adjacent BCs.

Bus Macros (black)

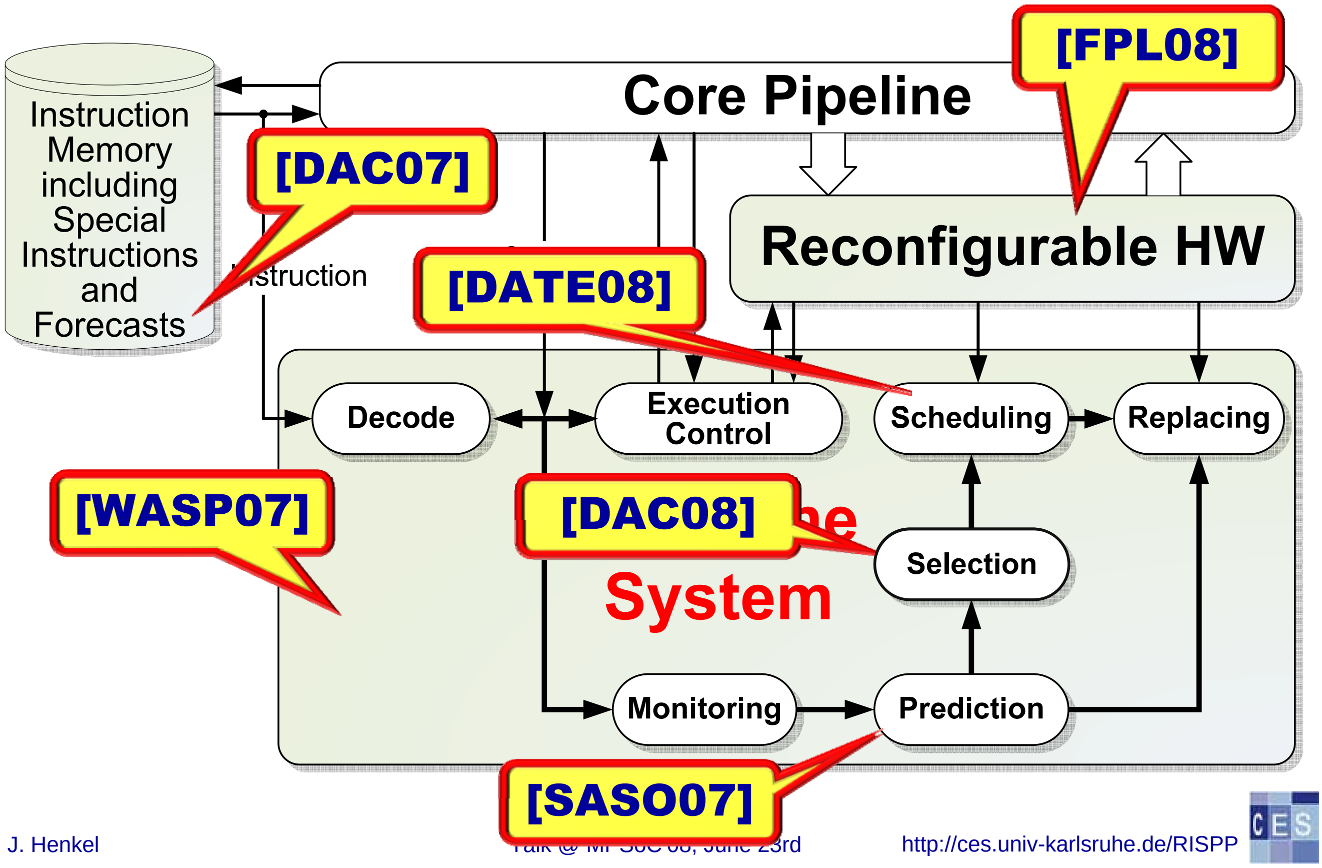
RISPP vs. ASIP

- At a fixed area ASIPs typically cannot implement all necessary special instructions in hardware. Alternative of software execution => may result in slow processing
- RISPP uses the a rotational concept instead to reconfigure the hardware for fore-coming Special Instructions
  - The rotational delay is reduced using dynamic forecasting techniques
- The rotational delay may be amortized => improved performance while still keeping the cost of hardware below ASIP level

RISPP vs. ASIP (cont'd)

- ❑ The atom/molecule model reduces the possibilities for rotation with a more fine-grained and reusable vision of Special Instructions
- ❑ Typical ASIP is very specialized => does not systematically re-use hardware components
- ❑ RISPP requires lesser hardware (in time-multiplex) than ASIPs due to Atom/Molecule model and rotational concepts
- ❑ However: fabric to execute on (e.g. FPGA) is less efficient (performance/power)

Conclusion



Instruction Memory including Special Instructions and Forecasts

Core Pipeline

[FPL08]

Reconfigurable HW

[DAC07]

struction

[DATE08]

Decode

Execution Control

Scheduling

Replacing

[WASP07]

[DAC08]

ne System

Selection

Monitoring

Prediction

[SASO07]

Conclusion

- ❑ **Current Project Status:**
  - ❑ **Almost all major parts implemented in HW**
  - ❑ **Running FPGA demonstrator**
  - ❑ **In addition a complete simulation environment**