FROM RESEARCH TO INDUSTRY

cea tech

Online Auto-Tuning for Performance and Energy through Micro-Architecture Dependent Code Generation

Fernando A. Endo
Laboratory: DACLE/LIALP
Doctoral school: MSTII

Thesis defense
18 September 2015

Jury:

M. Florent de Dinechin (Rapporteur)
M. Paul Kelly (Rapporteur)
M. Frédéric Pétrot (Examinateur)
M^{me} Karine Heydemann (Examinatrice)
M. Henri-Pierre Charles (Dir. de thèse)
M. Damien Couroussé (Co-encadrant)

www.cea.fr

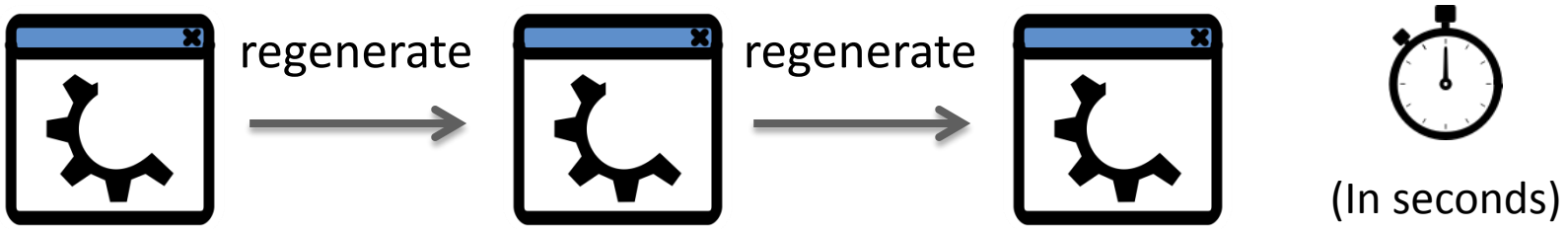
leti & list

1st part: Thesis overview

- Context
- Motivations
- Contributions

My thesis in one slide

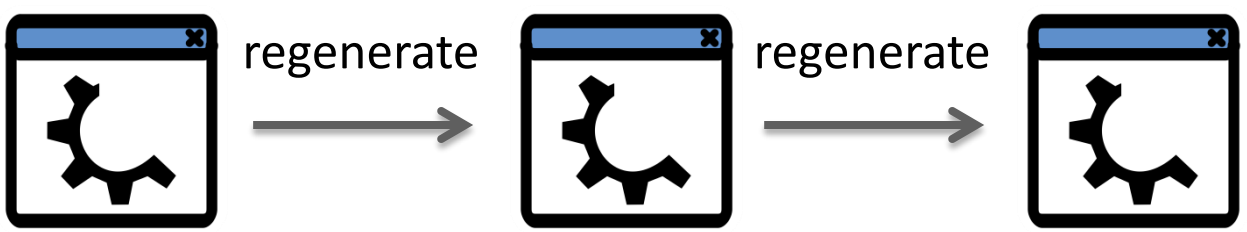
My thesis:



Performance & Energy efficiency:	+	++	+++

My thesis in one slide

My thesis:



(In seconds)

Performance & Energy efficiency: + ++ +++

Why?

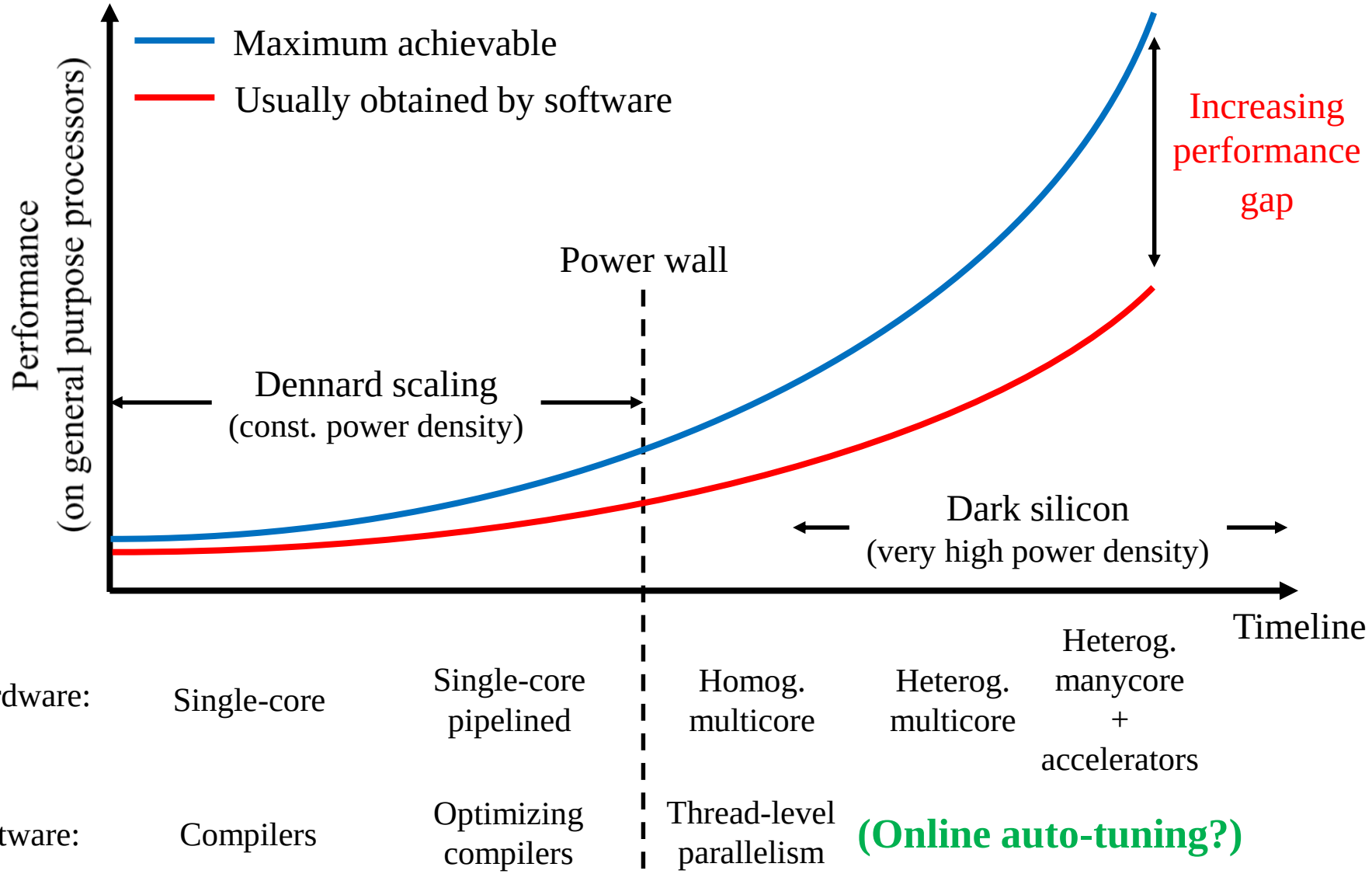
- ARM Cortex: A5/A7/A8/A9/ A12/A15/A17/ A53/A57/A72
- Scorpion
- Krait
- X-Gene
- Denver
- ThunderX
- K12
- Apple Ax: A4/A5/A6/A7/A8

Perf if Energy

Silicon tech. limitation

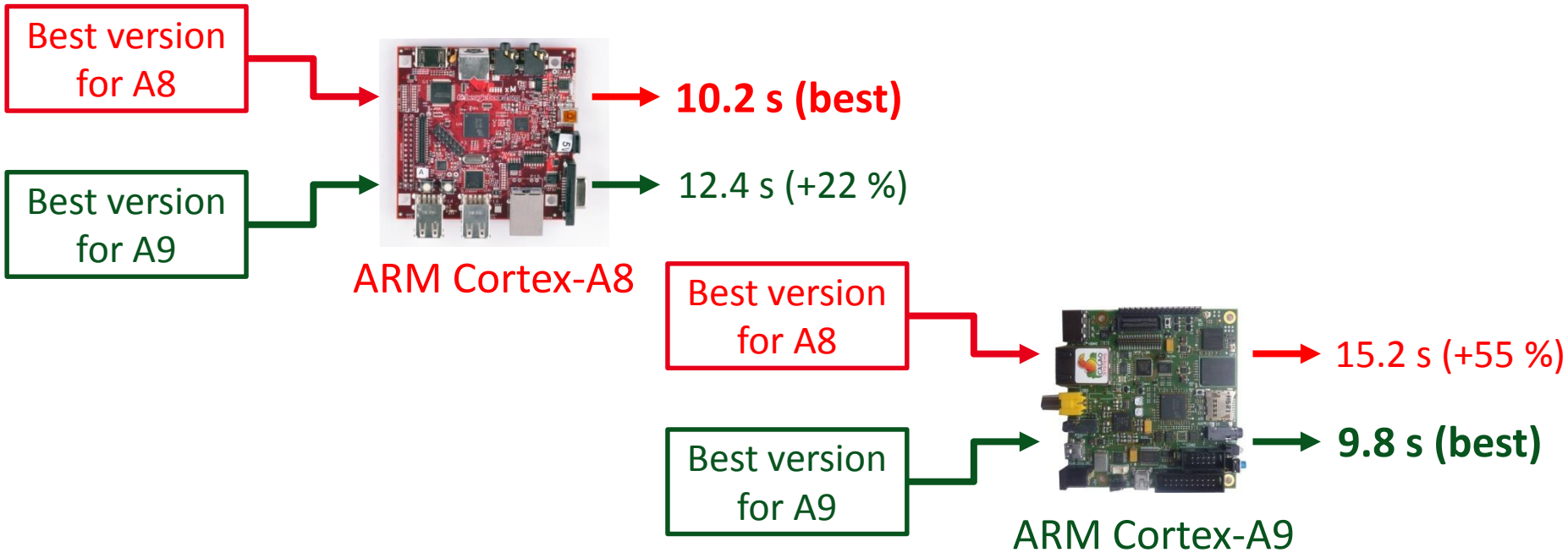
Embedded computing heterogeneity & complexity

Context: Hardware evolution



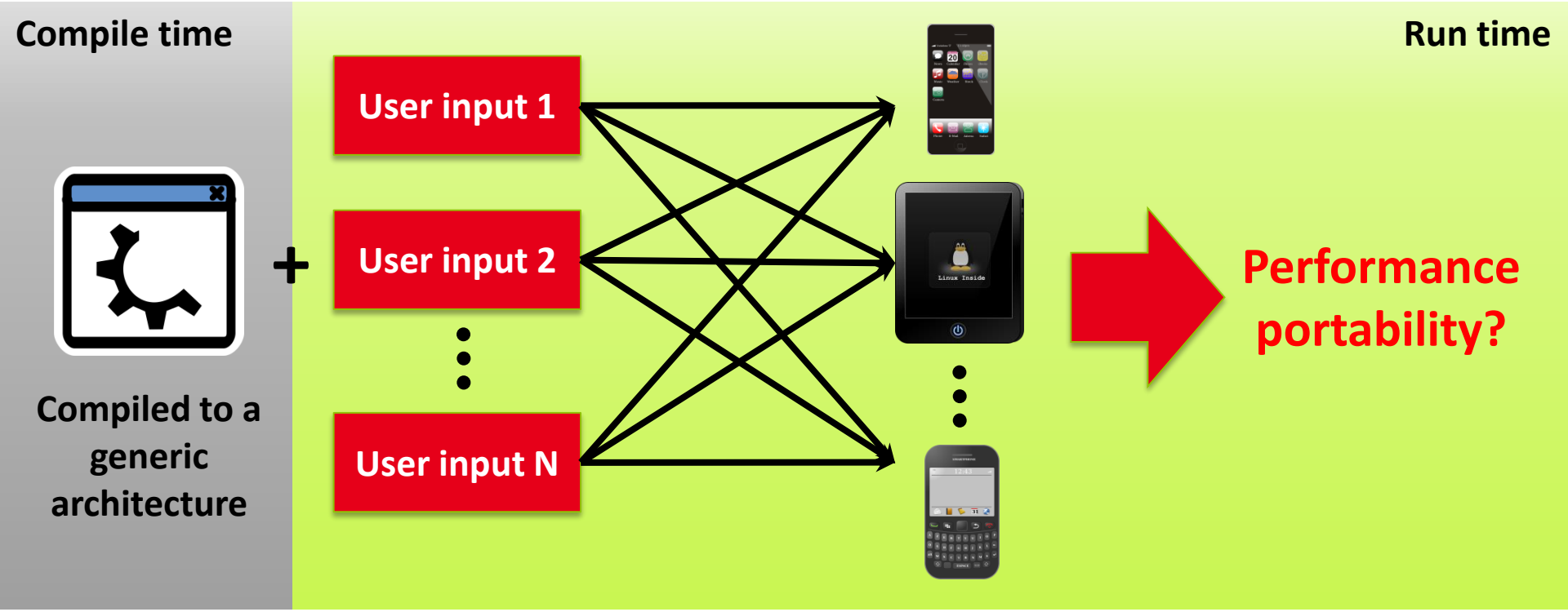
Context: Software auto-tuning

- Auto-tuning:
  - Finds the best algorithm implementations & compiler optimizations
  - Offline approaches: fixed running environment
  - **However: perf. depends on processor and input data**
    - Benchmark: Streamcluster (PARVEC)
    - **~20 hours of tuning / 1260 runs / 1 input set:**



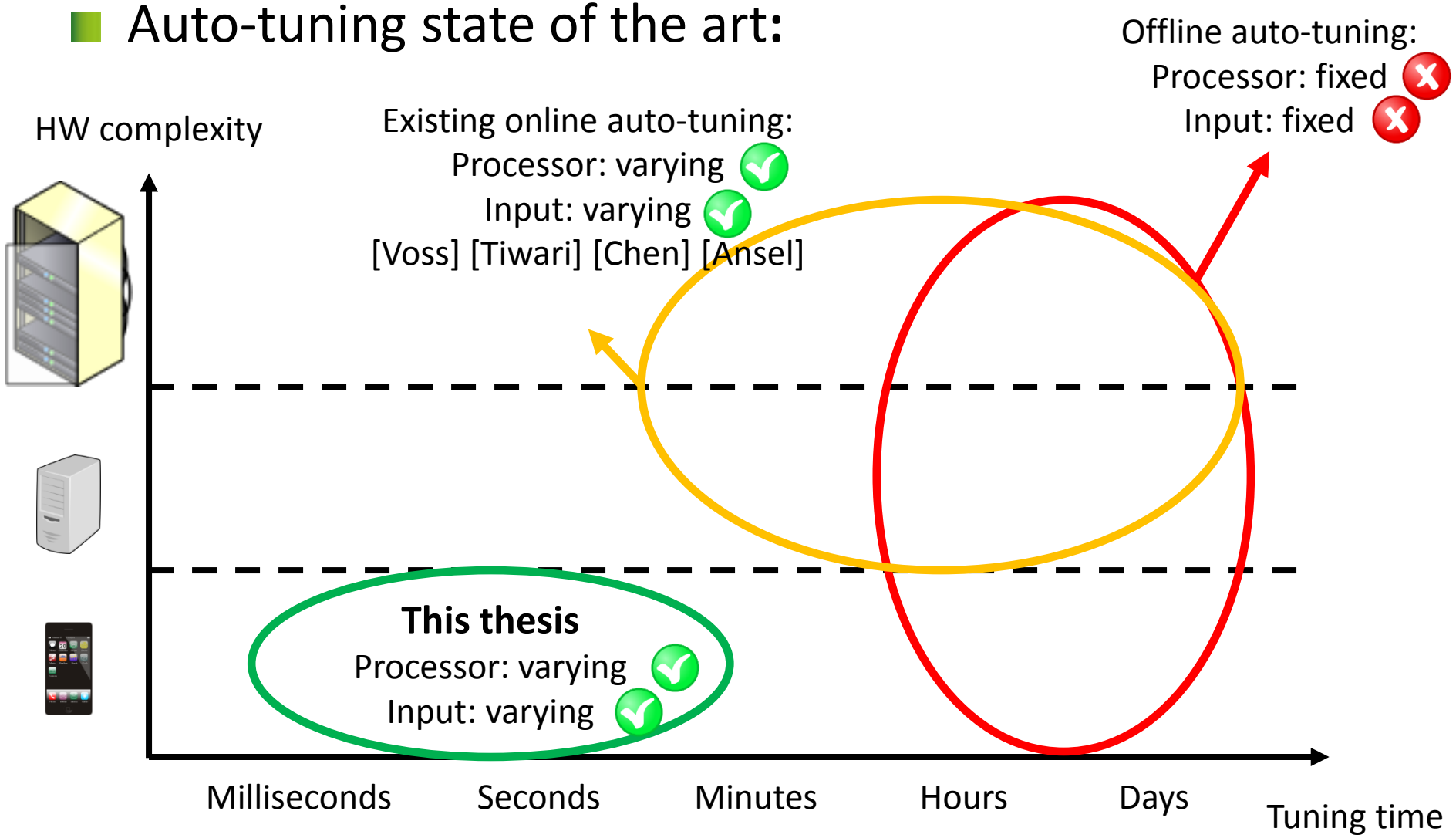
Context: Software auto-tuning

- Online auto-tuning in hand-held devices?

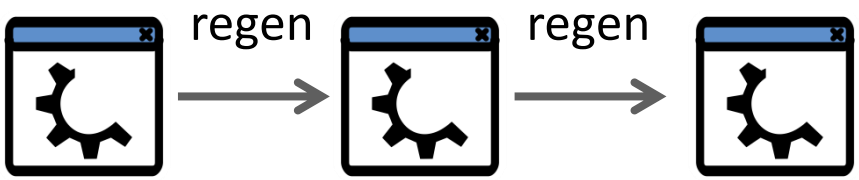


Context: Software auto-tuning

- Auto-tuning state of the art:

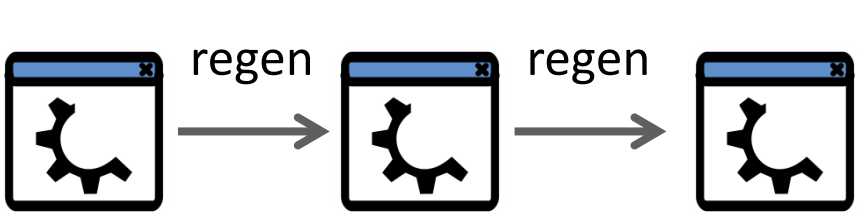


Thesis contributions

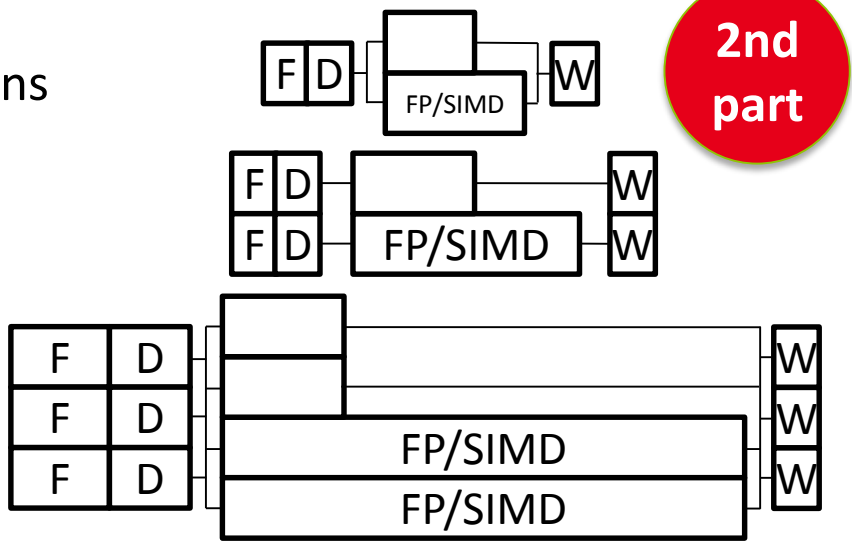


- Auto-tuning to:
  - Explore input-dep. optimizations
  - Adapt code to pipe features

Thesis contributions

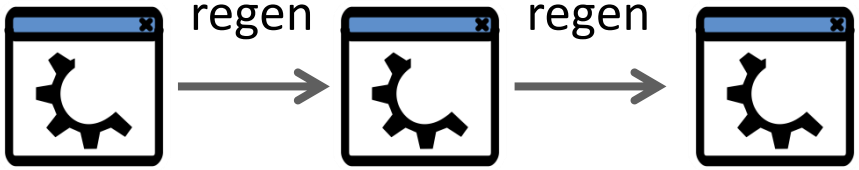


- Auto-tuning to:
  - Explore input-dep. optimizations
  - Adapt code to pipe features

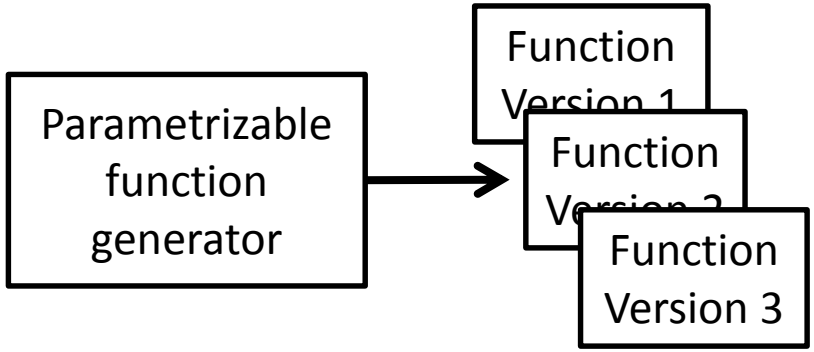


- µArch sim. for ARM:
  - Embedded core heterogeneity

Thesis contributions

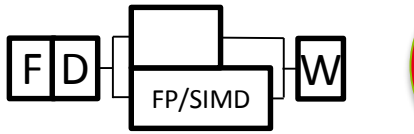


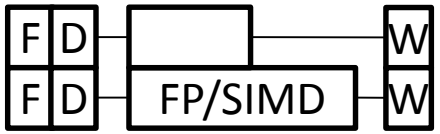
- Auto-tuning to:
  - Explore input-dep. optimizations
  - Adapt code to pipe features

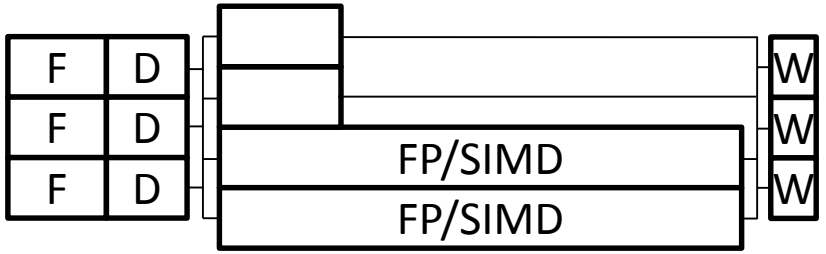


- Run-time code gen.:
  - ARM porting
  - Online auto-tun support

3rd part







2nd part

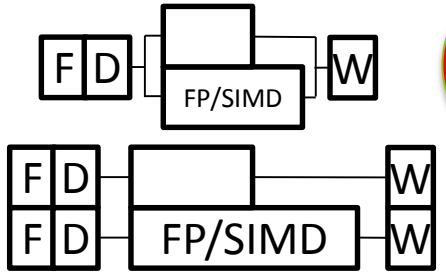
- µArch sim. for ARM:
  - Embedded core heterogeneity

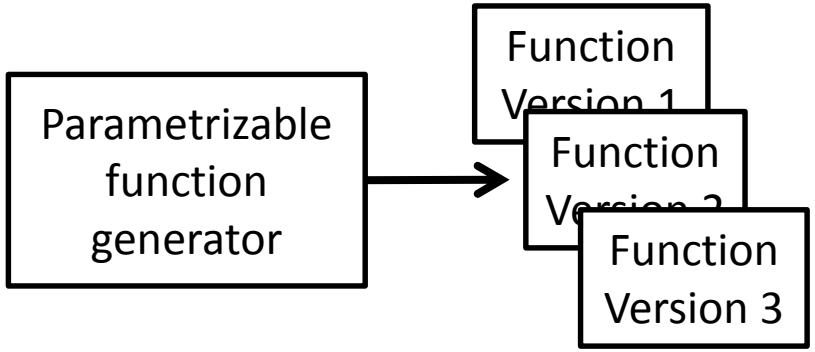
Thesis contributions

4th part

- Auto-tuning to:
  - Explore input-dep. optimizations
  - Adapt code to pipe features

2nd part





- µArch sim. for ARM:
  - Embedded core heterogeneity

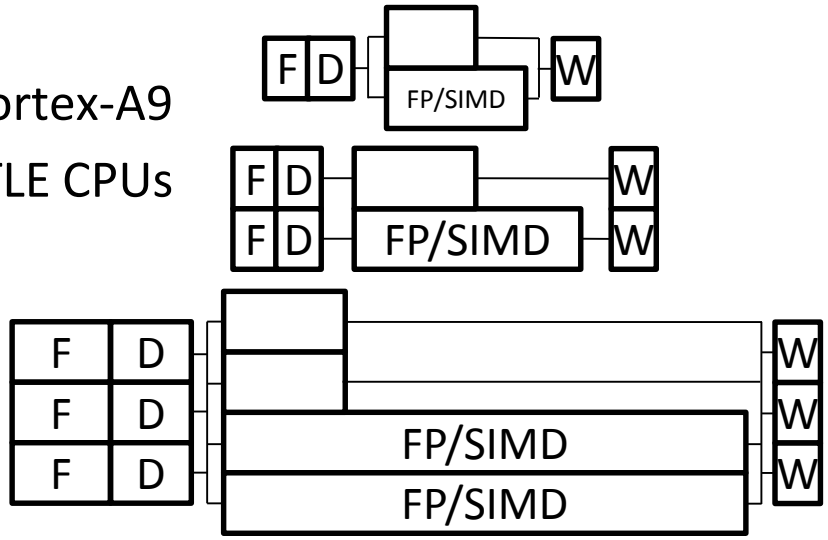
- Run-time code gen.:
  - ARM porting
  - Online auto-tun support

3rd part

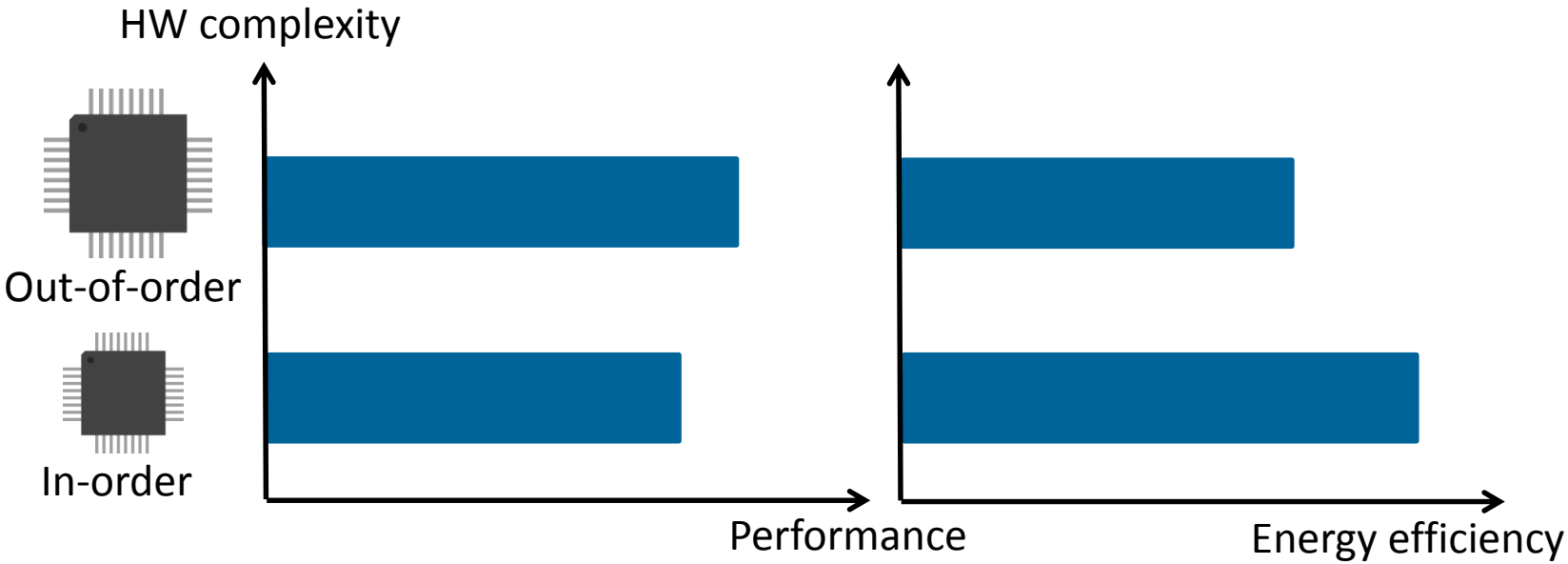
2nd Part: µArch simulator for ARM

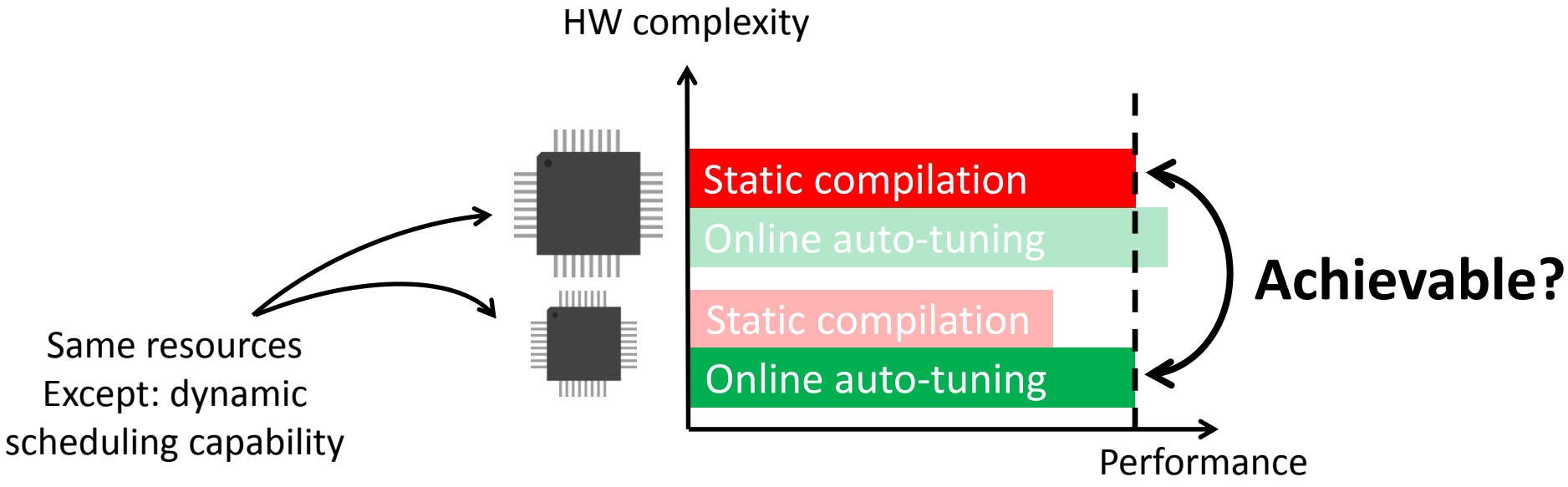
- Why simulation?
- gem5: Performance of pipe and caches
  - In-order model in gem5
- McPAT: Energy estimation
  - Better func. unit model in McPAT
- gem5 and McPAT integration
- Validations: Sim. vs real HW
  - Performance: Against Cortex-A8 & Cortex-A9
  - Area and rel. Energy: Against big.LITTLE CPUs
- Conclusions

2nd part



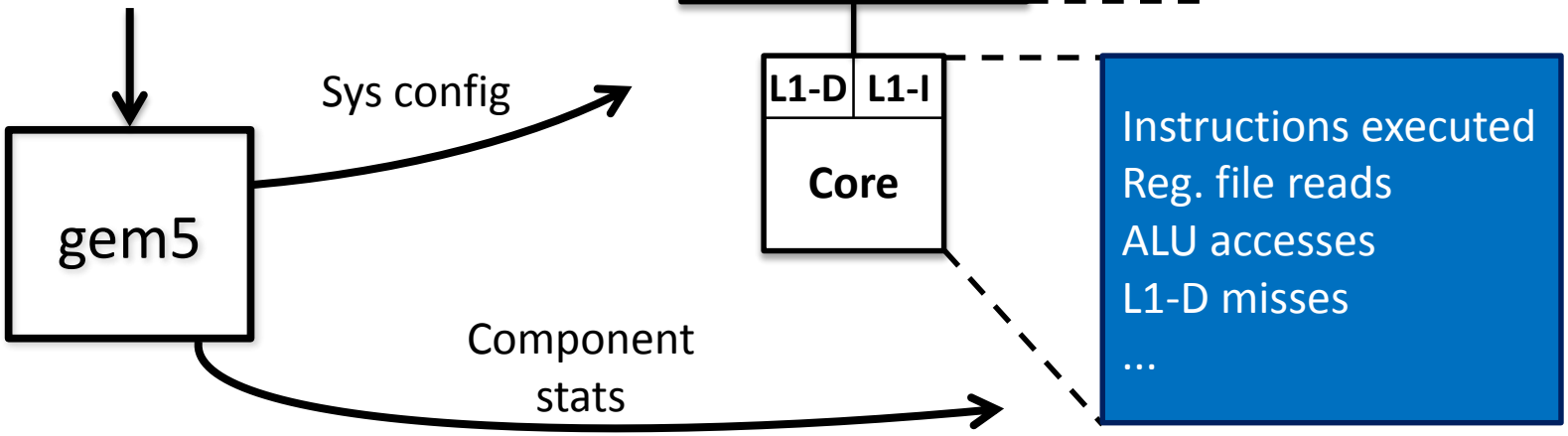
Why simulation?





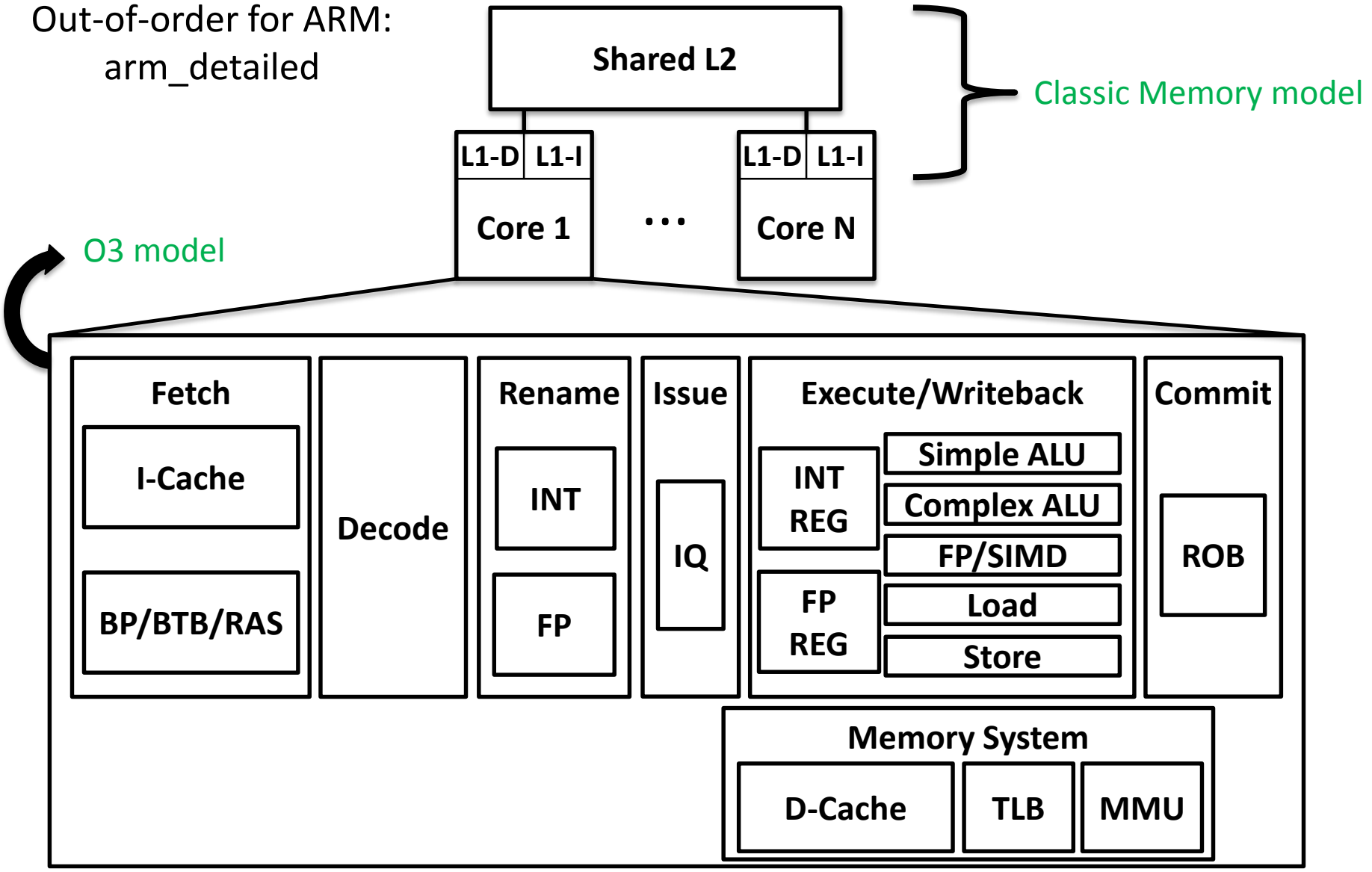
gem5 overview

- Full-system & µArch simulator: ARM, x86, Alpha
  - ARMv7-A architecture
  - Can simulate Linux boot
  - Pipeline, cache and memory behavior
  - (Internally used by ARM)



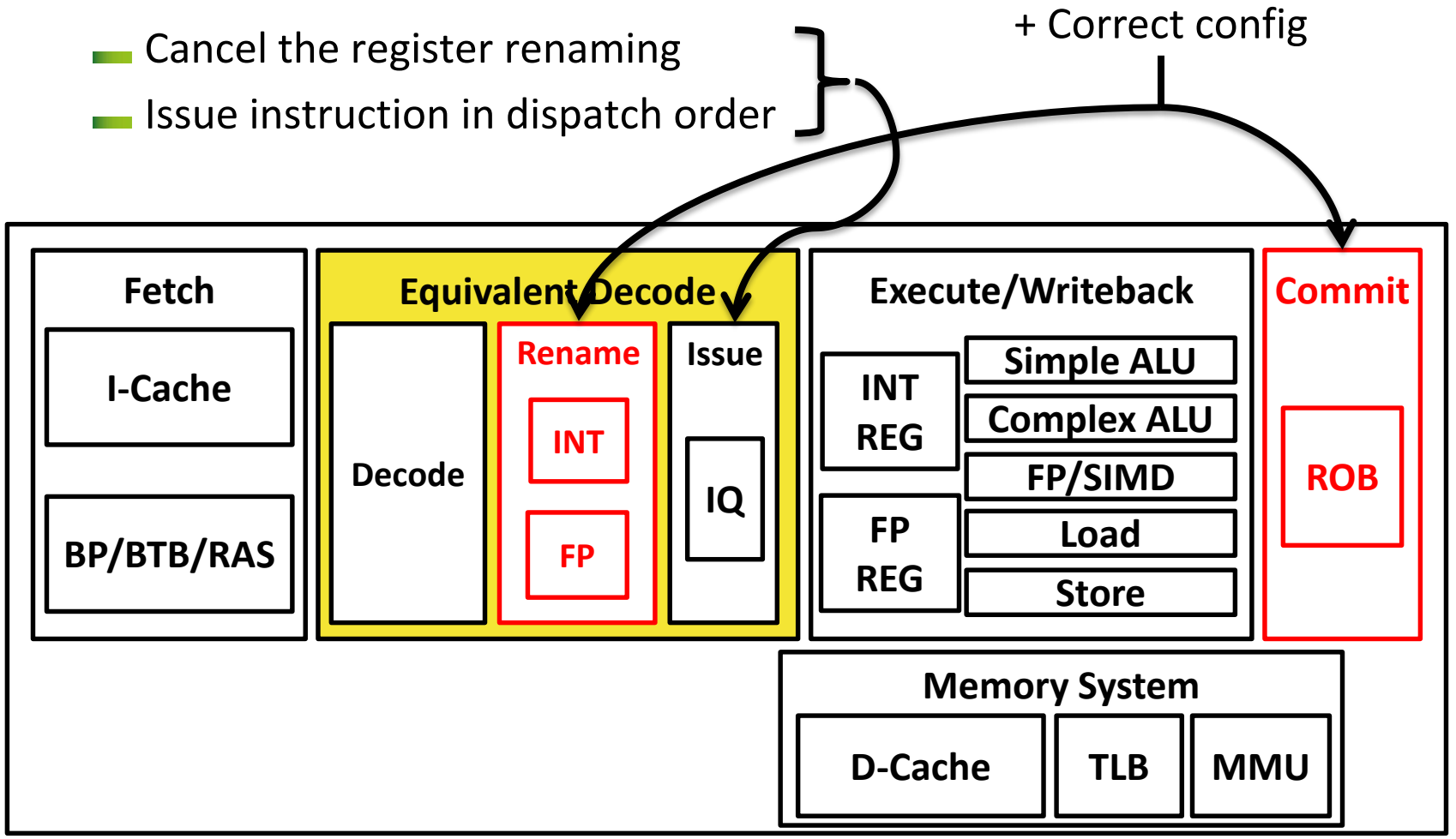
gem5 arm_detailed configuration

Out-of-order for ARM:
arm_detailed



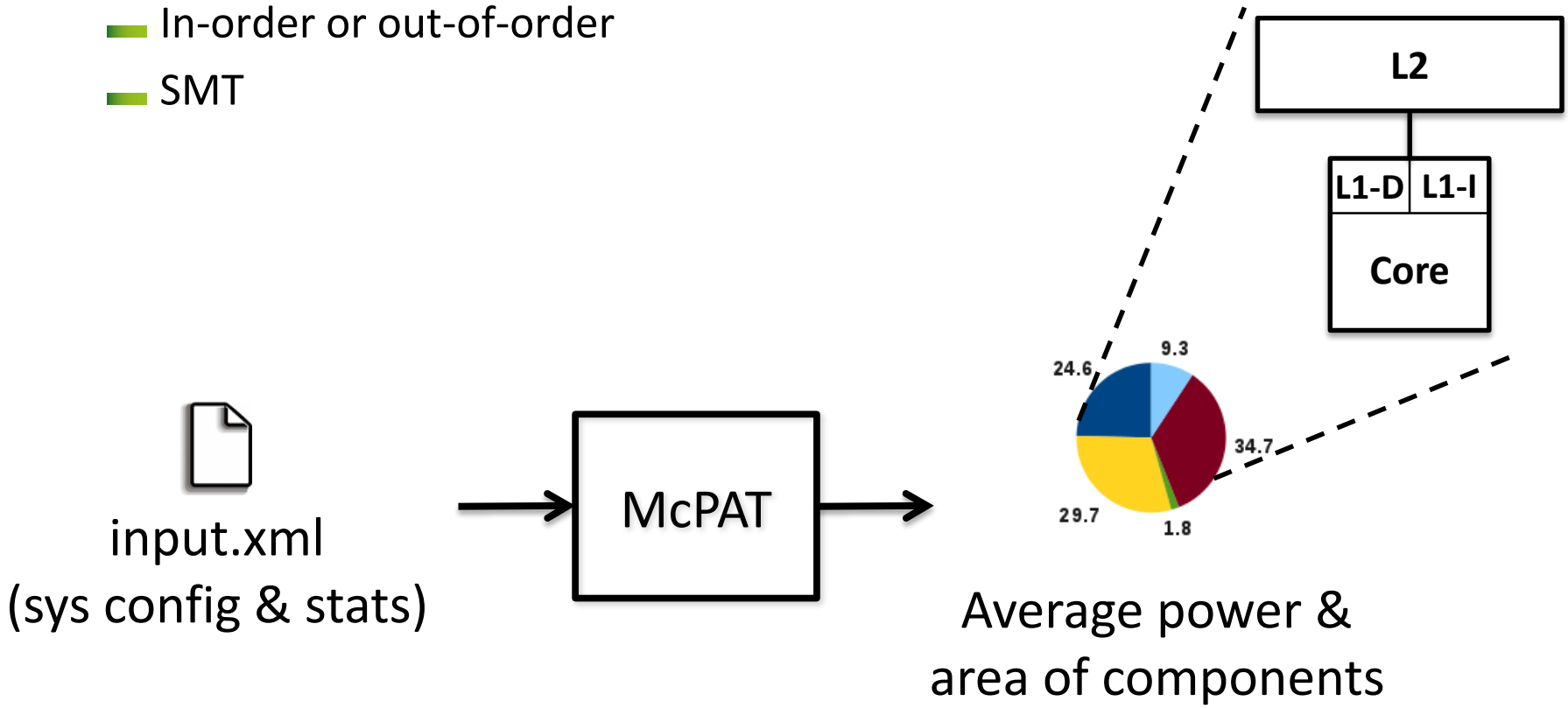
Contribution: In-order model in gem5

- Two main modifications:
  - Cancel the register renaming
  - Issue instruction in dispatch order



McPAT

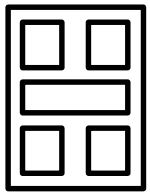
- Power and area models:
  - Multi- & manycores
  - Desktop or embedded
  - In-order or out-of-order
  - SMT

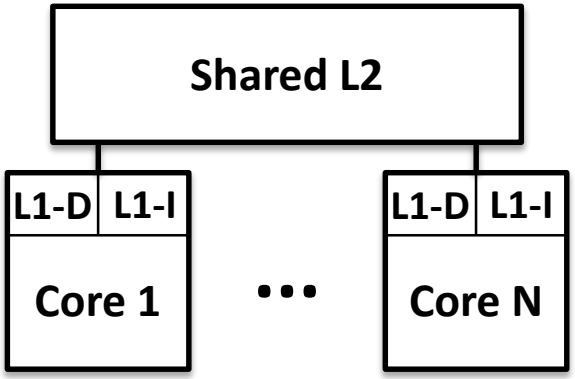


McPAT: Area & energy models

input.xml
(sys config & stats)

Component areas





Component energy cost

- F. Units: per cycle
- Other: per access

Internal chip model:

- F. Units: empirical energy & area
- Other: analytical models

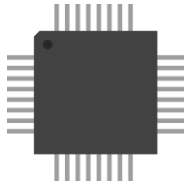
$$\text{Component avg power} = \underbrace{\text{leakage(T)}}_{\text{(Static power)}} + \underbrace{\frac{\text{energy cost} \times \text{utilizations}}{\text{runtime}}}_{\text{(Dynamic power)}}$$

Contribution: Better func. unit model in McPAT

- VFP/NEON @ 40 nm:
  - Cortex-A5: 0.15 mm^2
  - Cortex-A9: 0.98 mm^2

6.5x

Cortex-A5



Cortex-A9

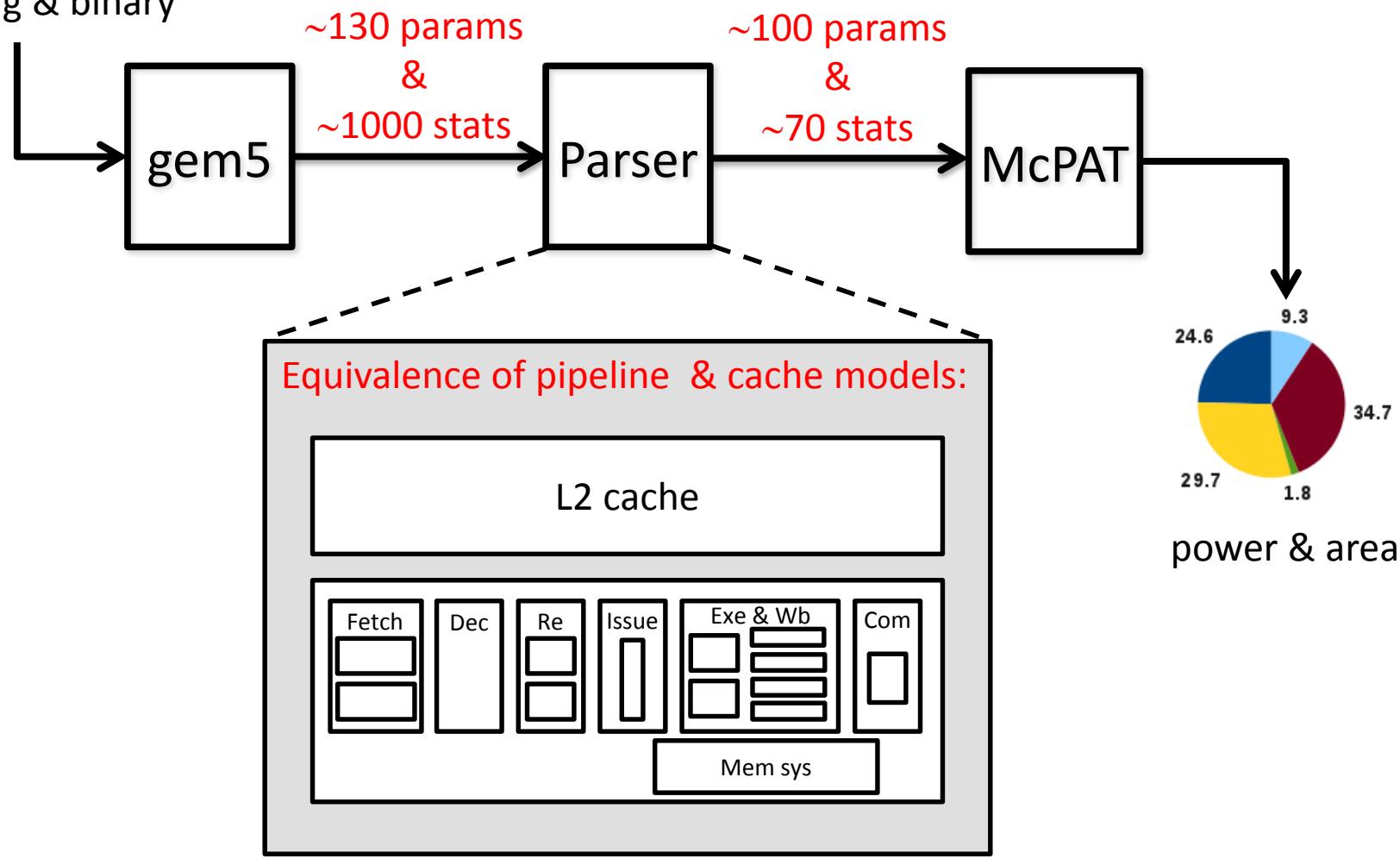
  - McPAT: always 0.97 mm^2
- Rough, but better:
  - $FU\_area \propto issue\_width^2$ [Shifer]
- Assumption:
  - $FU\_energy \propto issue\_width^2$

($issue\_width \equiv$ Number of instructions that can be issued per cycle)

($issue\_width > 1$ ⇒ Superscalar pipeline)

Contribution: gem5 and McPAT integration

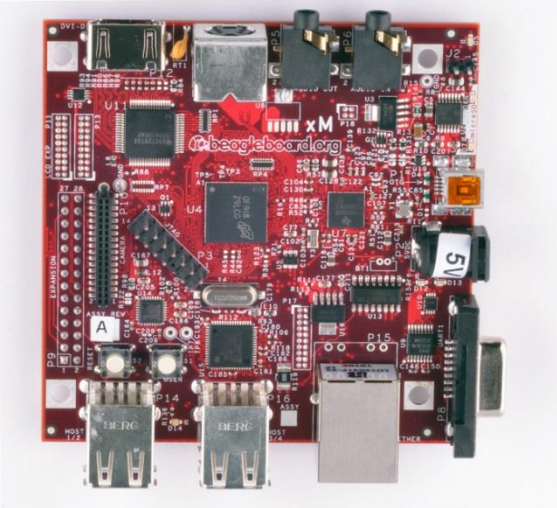
Sys config & binary



µArch simulator: Performance validation

Proposed in-order vs BeagleBoard

Original out-of-order vs Snowball



Cortex-A8

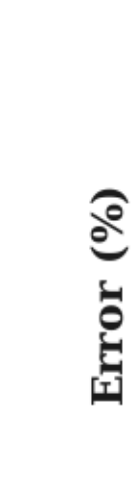
Dual Cortex-A9

(Running 10 PARSEC 3.0 benchmarks)

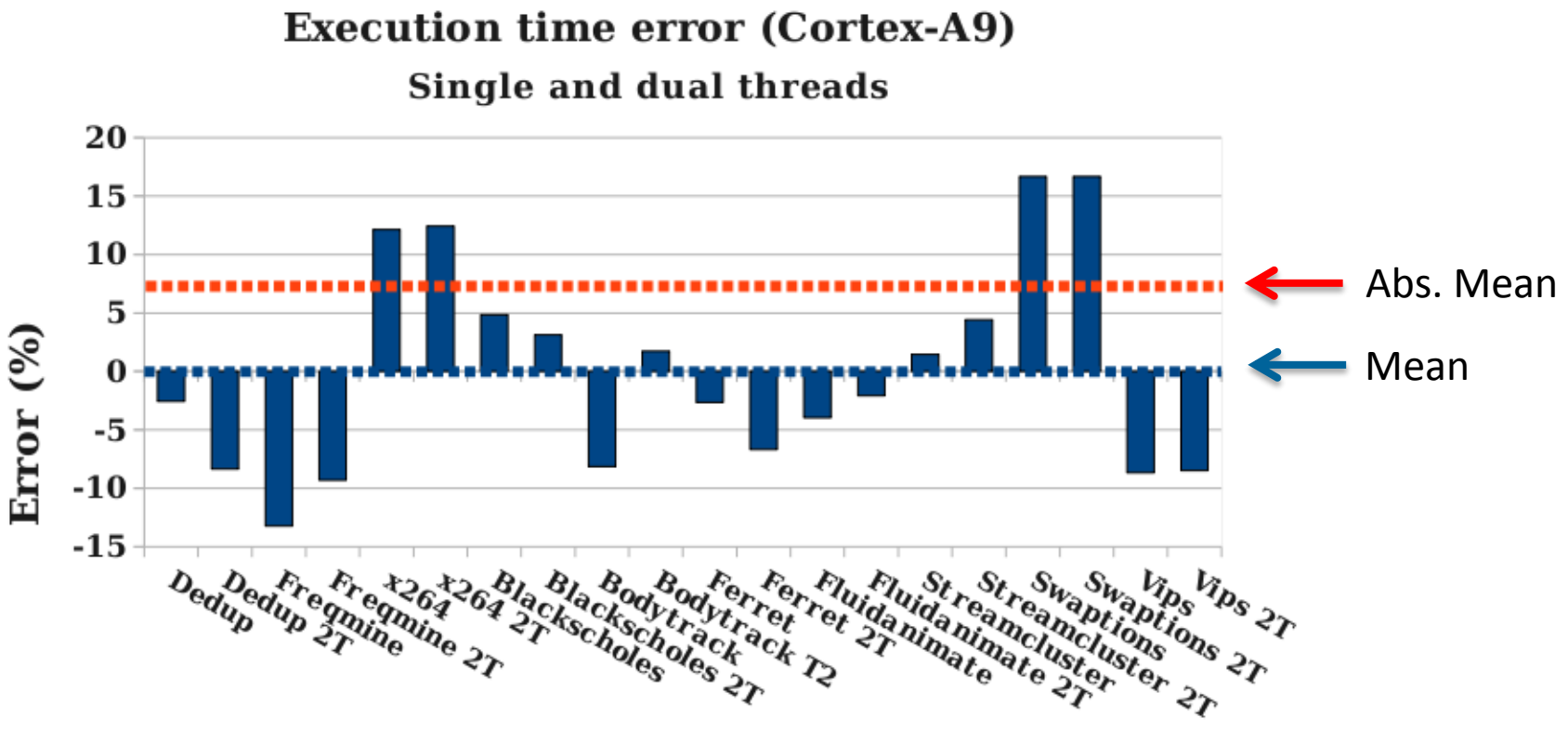
Main reason: ARM published detailed timing of EXE stage

Simulation errors of the gem5 O3 model

Real CPU is faster



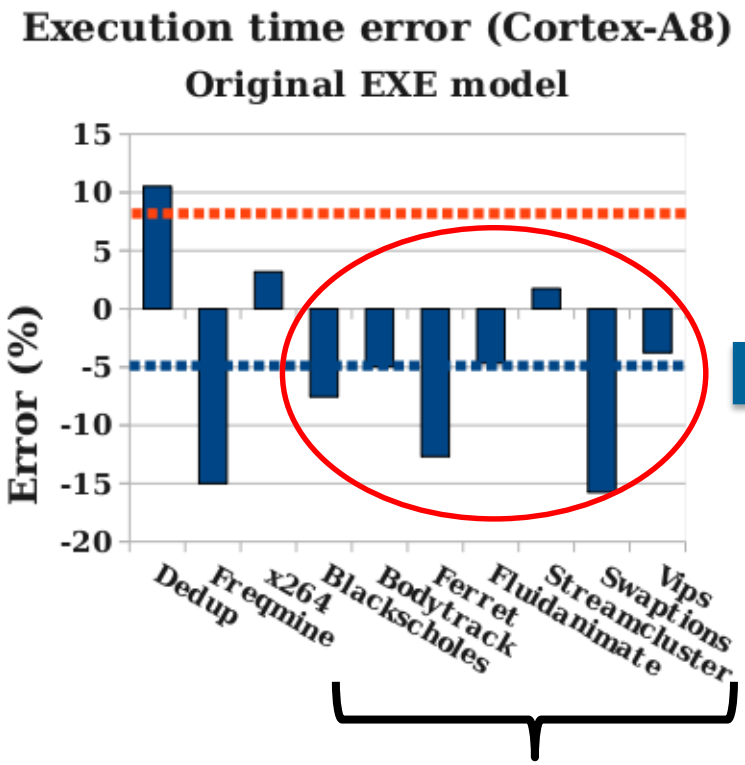
Sim. CPU is faster



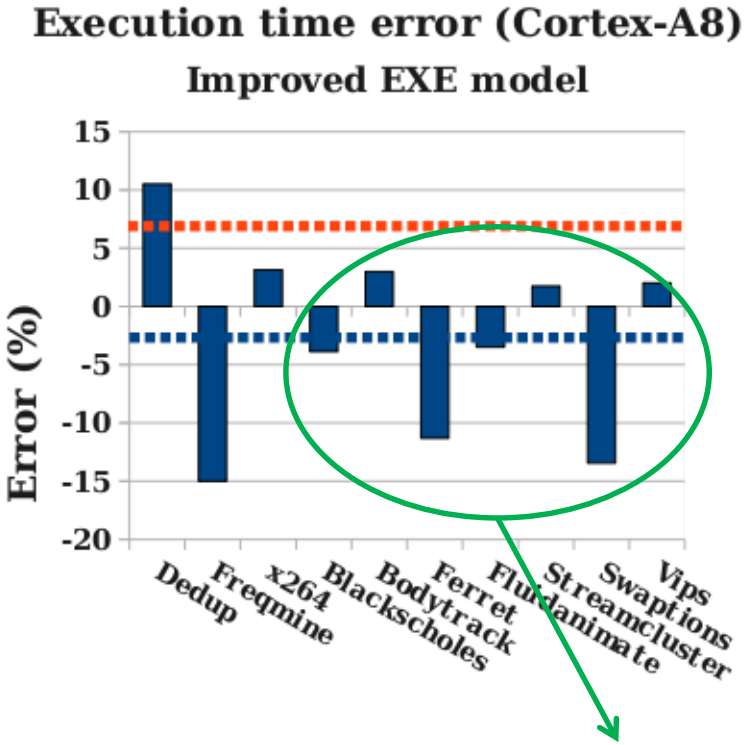
Simulation errors of the proposed in-order

Real CPU is faster

Sim. CPU is faster



FP benchs

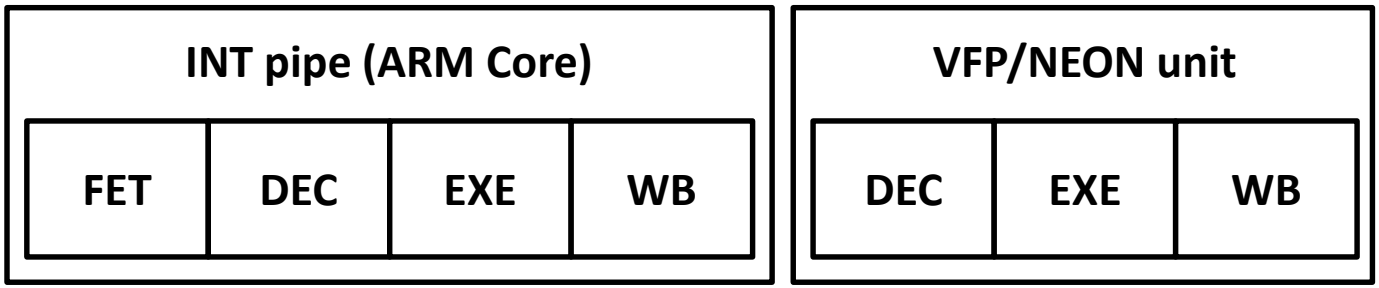


24 % of error reduction

Mean Abs. Mean

Behavior of VMOV in the Cortex-A8

- In the Cortex-A8:



INT Instructions →

FP Instructions →

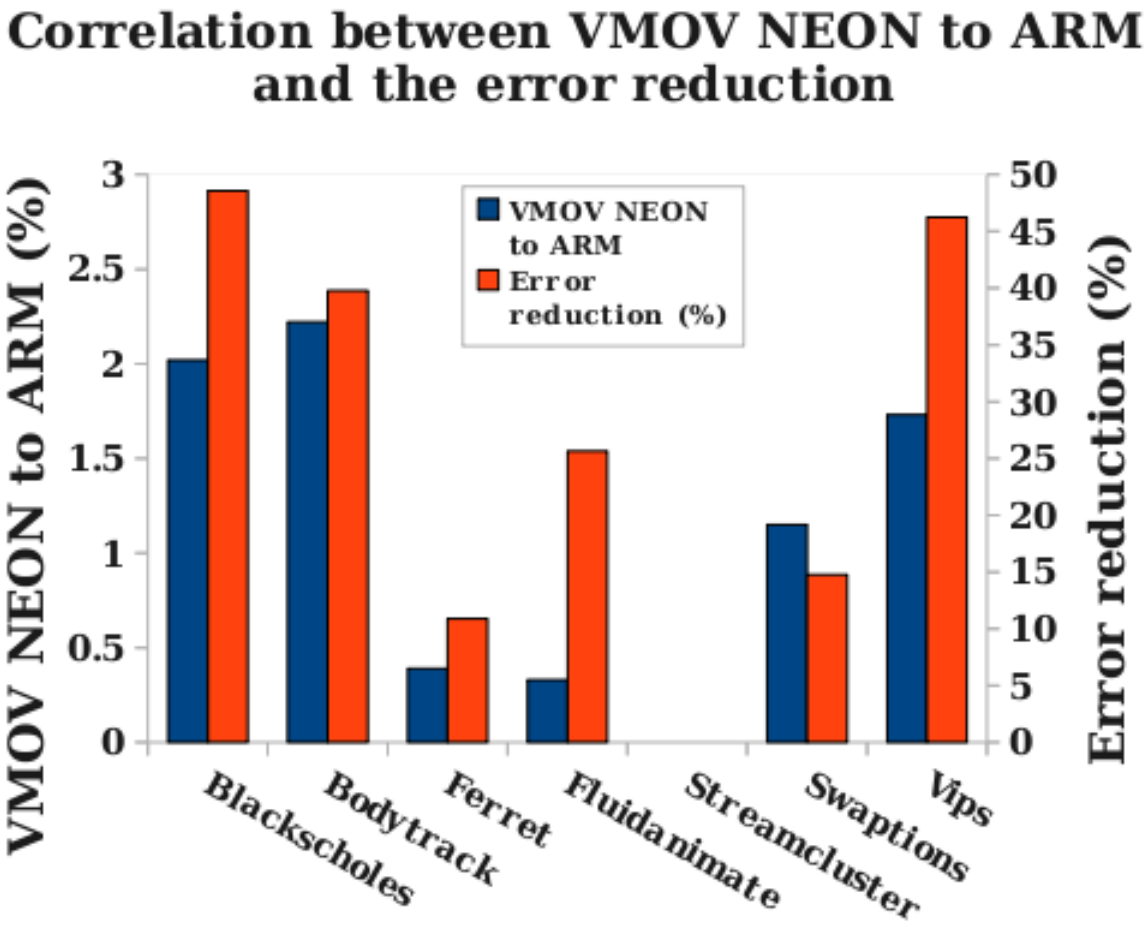
Move ARM to NEON: 4 cycles (follows the flow)

Move NEON to ARM: **> 20 cycles** (against the flow)

} Originally: same group

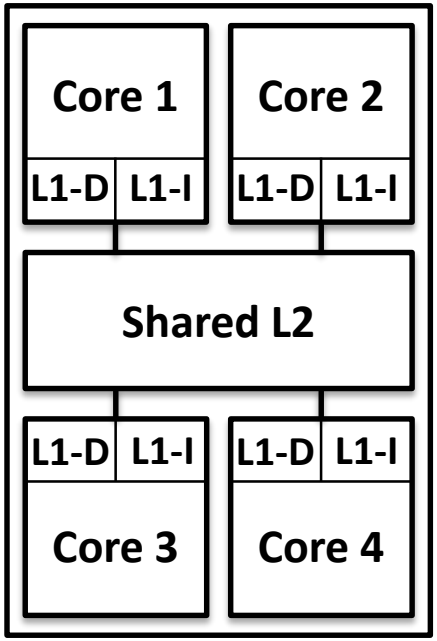
↳ **Improvement: new group (opClass)**

Error reduction thanks to a new opClass in gem5

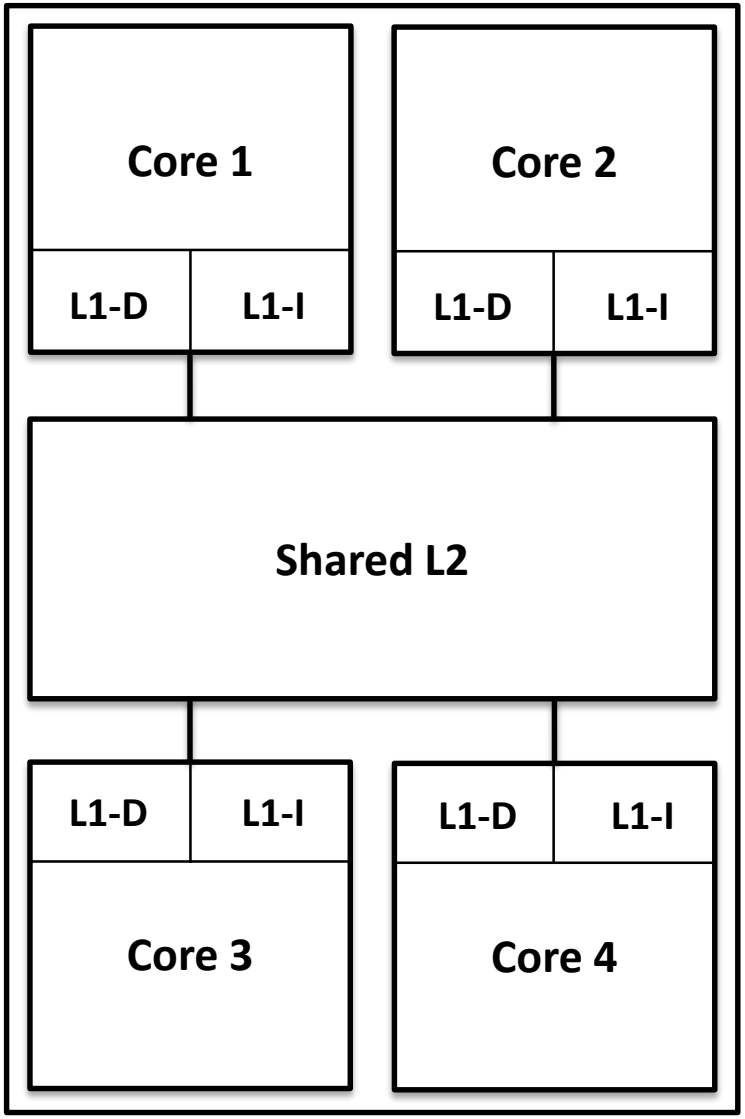


µArch simulator: Area validation

ODROID-XU3 board:
Exynos 5 octa



Cortex-A7 CPU



Cortex-A15 CPU

µArch simulator: Area validation

Cortex-A7 CPU

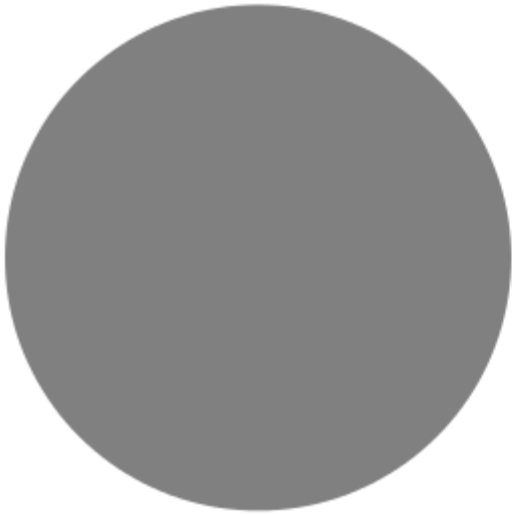
Published data

3.8 mm²
[EETimes/ISSCC]

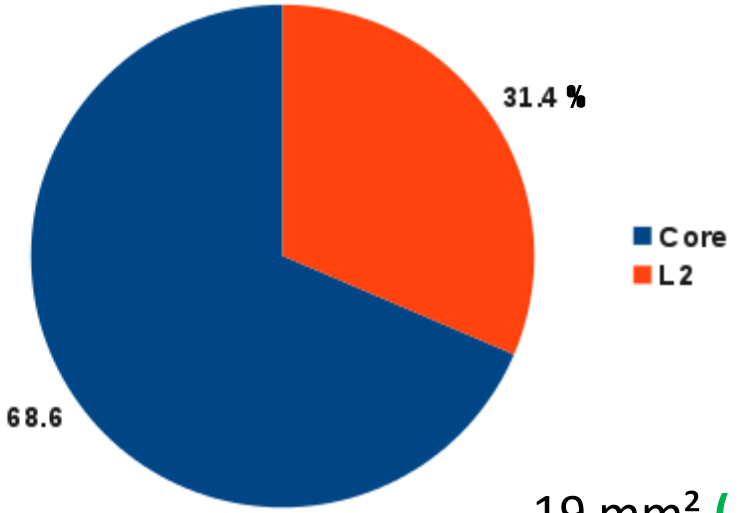
McPAT 1.0

3.3 mm² **(-13 %)**

Cortex-A15 CPU



19 mm²
[EETimes/ISSCC]



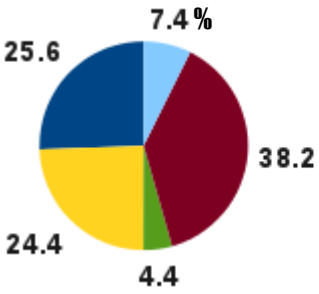
19 mm² **(-1.4 %)**

(All areas at the same scale)

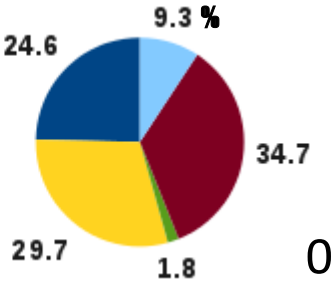
µArch simulator: Area validation

Cortex-A7

Published data



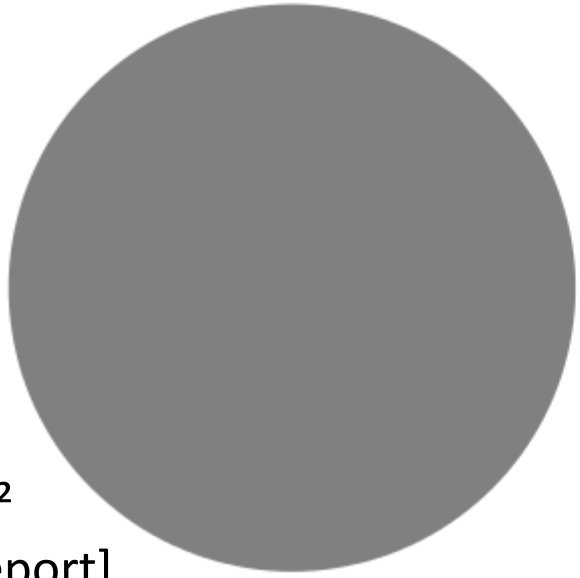
0.45 mm² [ARM]



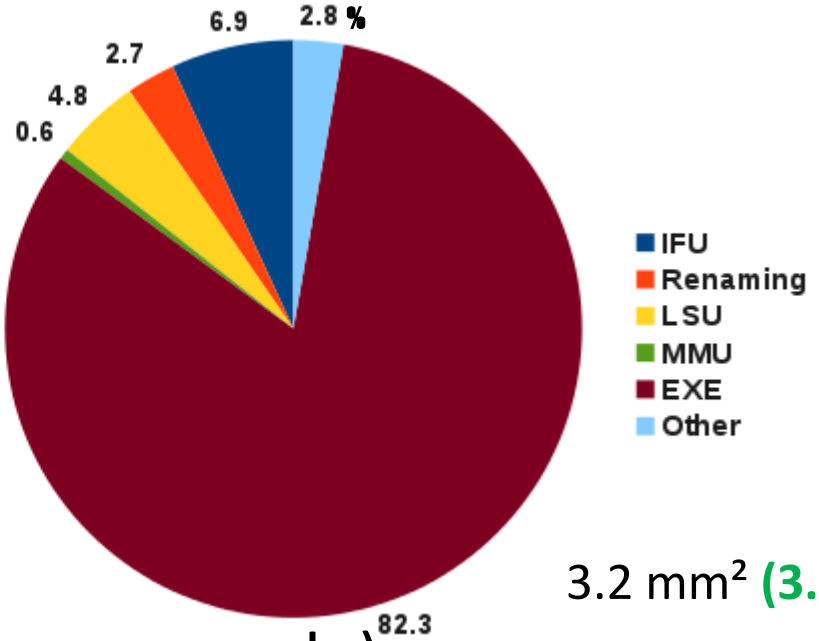
McPAT 1.0

0.45 mm² **(0 %)**

Cortex-A15



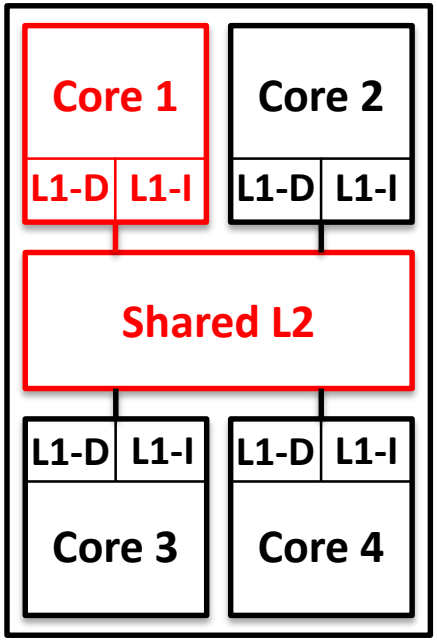
3.1 mm² [The Tech Report]



3.2 mm² **(3.6 %)**

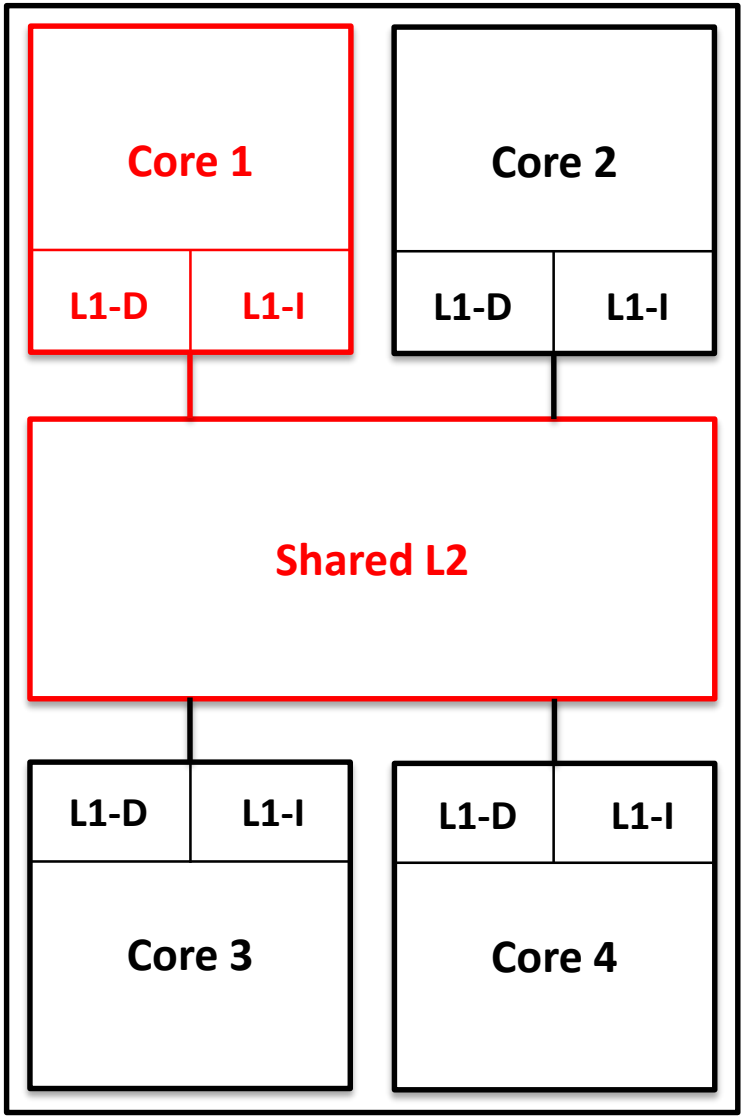
(All areas at the same scale)

µArch simulator: Rel. energy validation



Cortex-A7 CPU

Vs

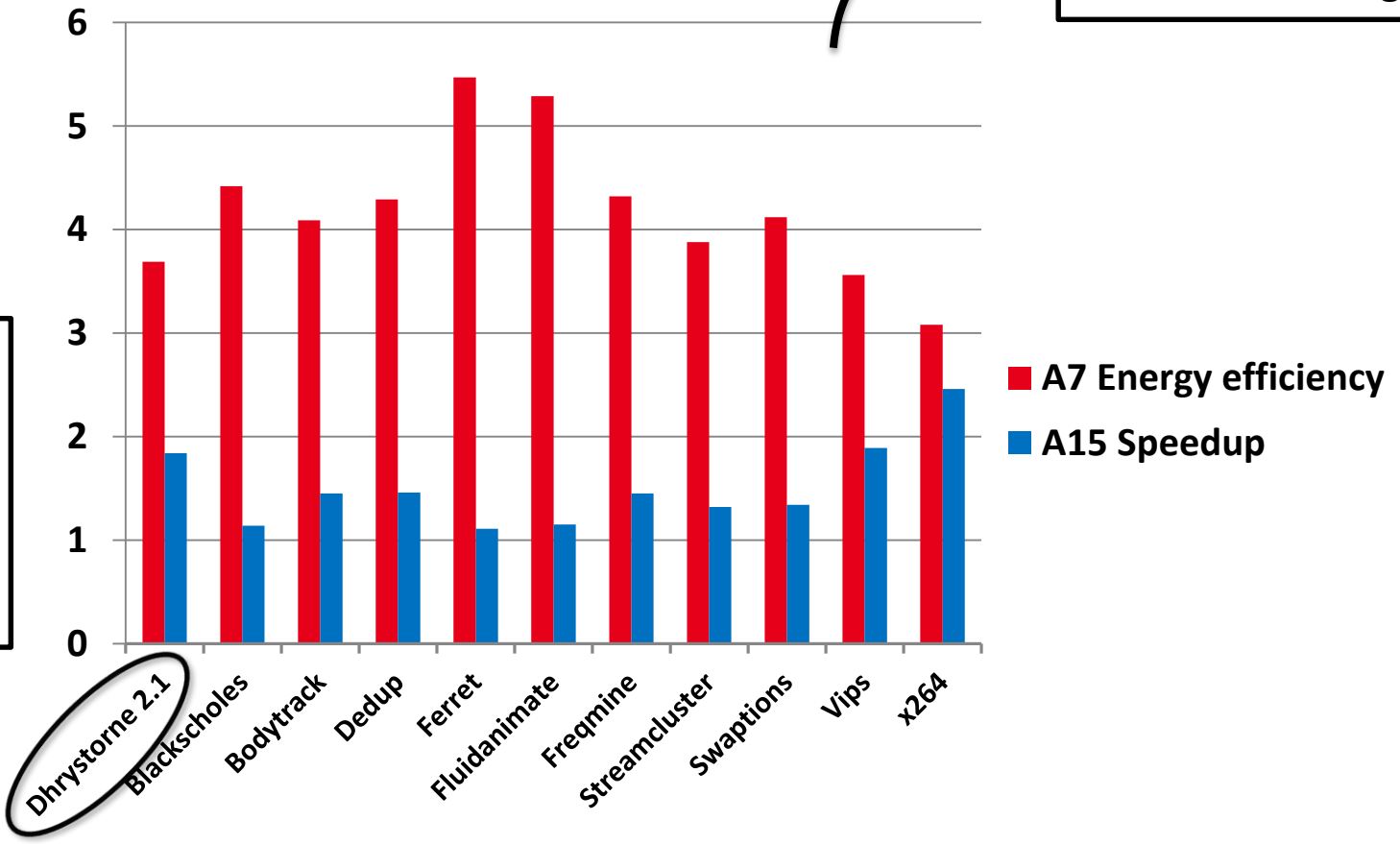


Cortex-A15 CPU

µArch simulator: Rel. energy validation

Within 6 % of ARM data [Greenhalgh]

Average:
A15 **1.5x** faster
A7 **4x** less energy

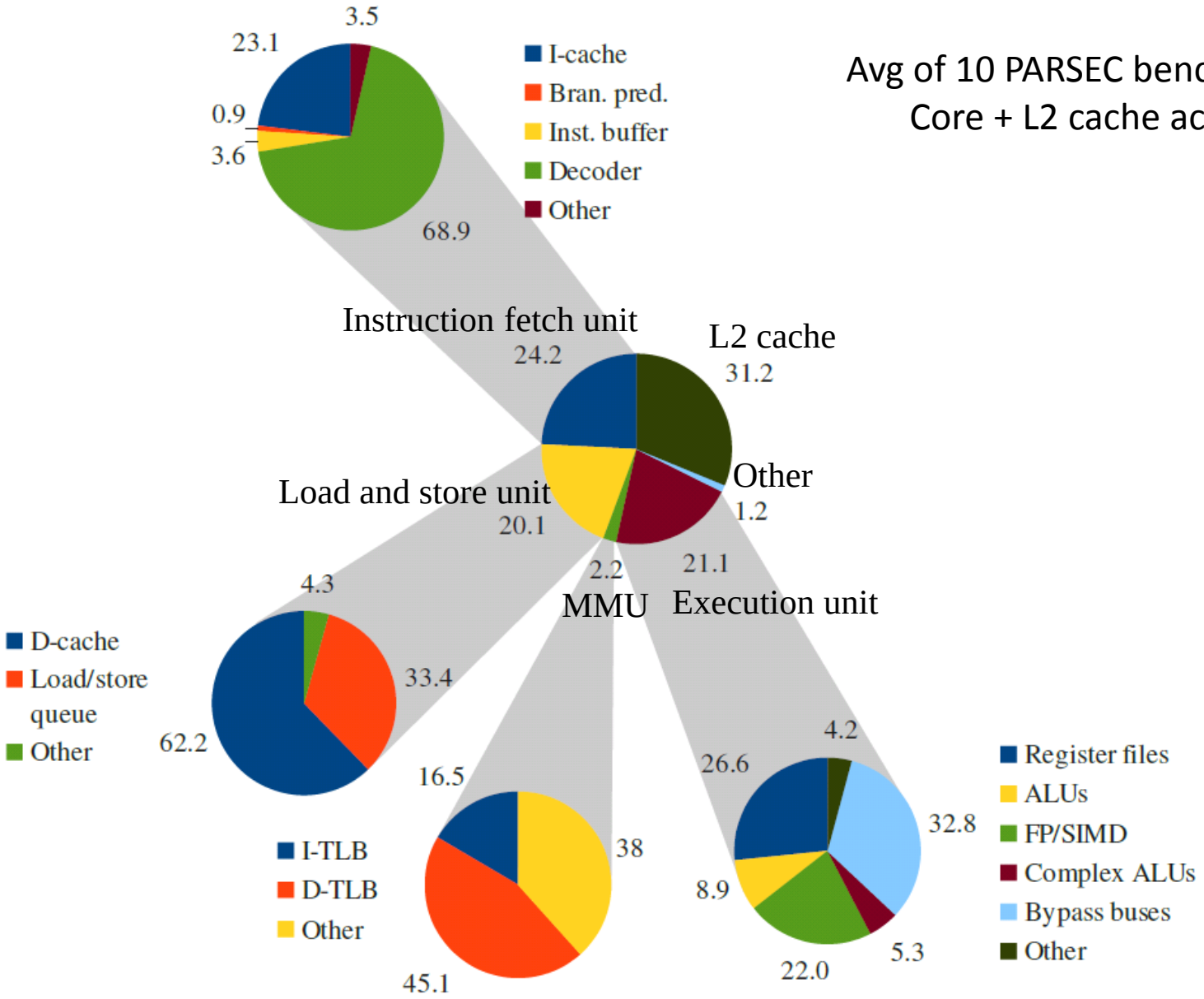


Dhrystone:
A15 is
1.8x faster
A7 uses **3.7x**
less energy

(Higher is better)

µArch simulator: Example of energy breakdown

Avg of 10 PARSEC benchmarks
Core + L2 cache active



µArch simulator: Conclusions

Timing accuracy

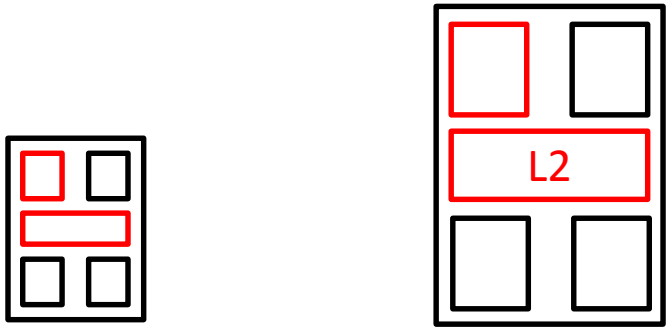
Cortex-A8 Cortex-A9

Abs. Avg: ~7 %

Previous work with gem5:
Abs. Avg. > 13 %
[Gutierrez]

State-of-the-art simulators:
Abs. Avg. > 15 %
[Desikan] [Anh]

Relative energy/perf accuracy

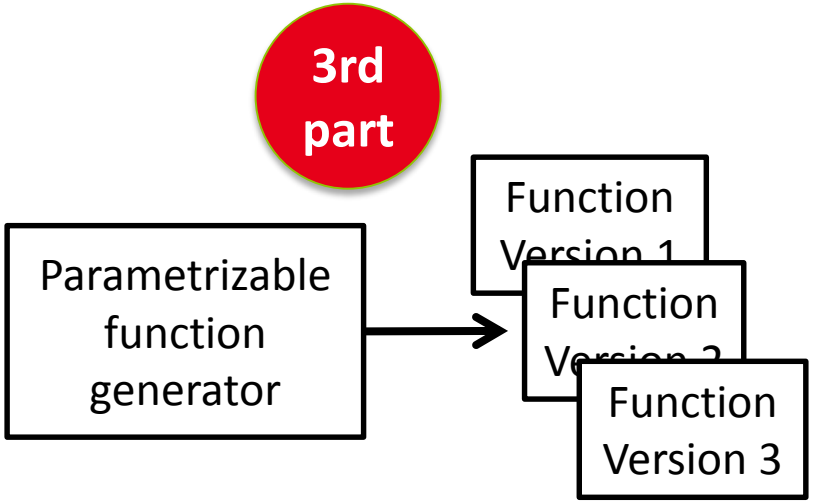


Cortex-A7 CPU Cortex-A15 CPU

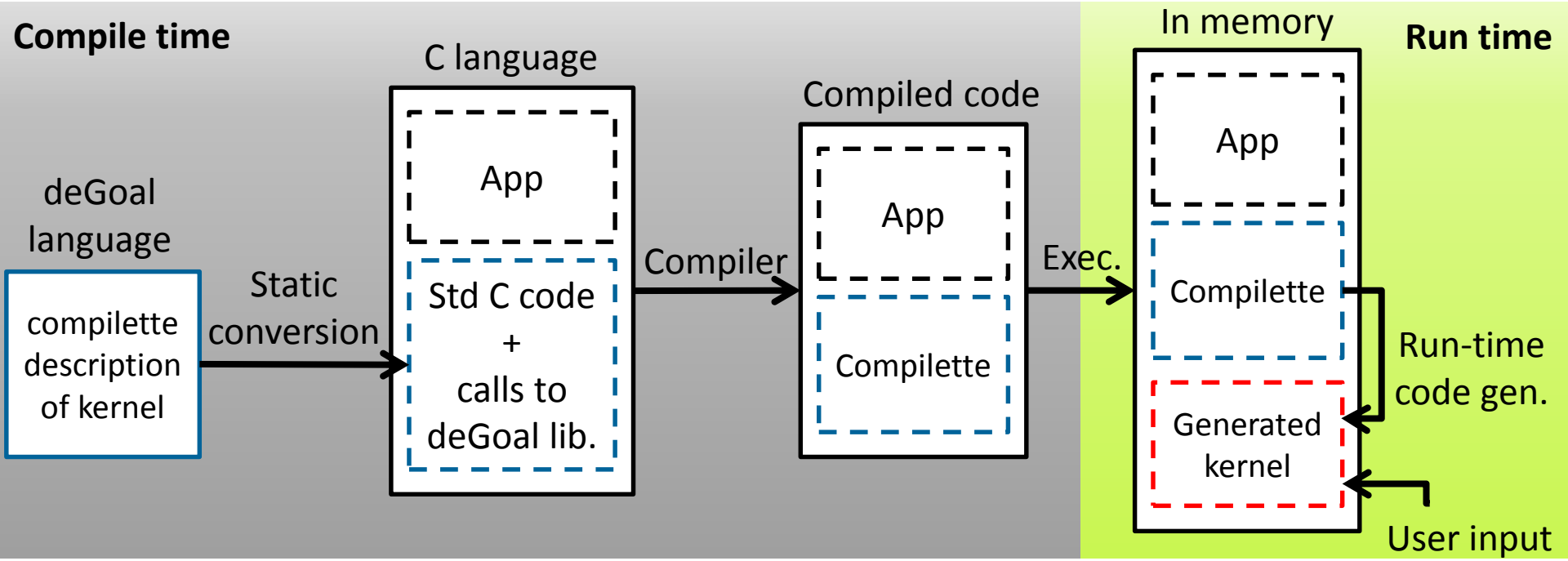
Dhrystone 2.1: < 6 %

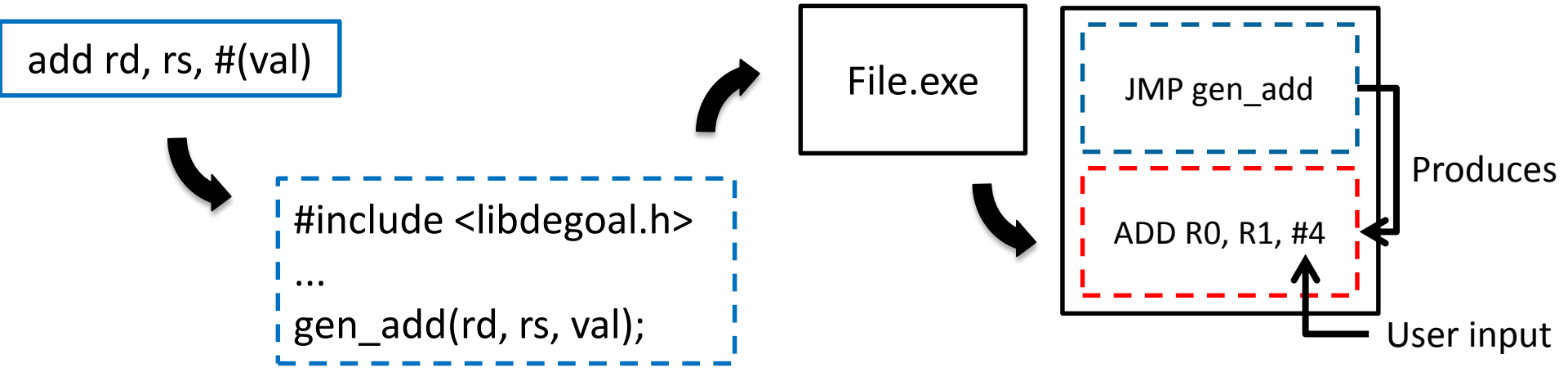
3rd part: Run-time code gen. for ARM

- Run-time code generator: deGoal
  - Overview
  - Example of deGoal code
- ARM porting and extensions
- Performance evaluation
  - 4 computing kernels, SISD & SIMD
  - gcc 4.5.2 (–O3)
- Conclusions



Run-time code generator: deGoal





Example of deGoal code

- Simple program example:

```
void gen_vector_add(void *buffer, int vec_len, int val)
{
#[
    Begin buffer Prelude vec_addr

    Type int_t int 32 #(vec_len)
    Alloc int_t v

    lw v, vec_addr
    add v, v, #(val)
    sw vec_addr, v
]#
}
```

Source to source converted to standard C code

Standard C code

Example of deGoal code

- Simple program example:

```
void gen_vector_add(void *buffer, int vec_len, int val)
{
#[
    Begin buffer Prelude vec_addr

    Type int_t int 32 #(vec_len)
    Alloc int_t v

    lw v, vec_addr
    add v, v, #(val)
    sw vec_addr, v
]#
}
```

When executed

Memory:

```
ldr r1, [r0]
add r1, #1
str r1, [r0]
add r0, #4
ldr r2, [r0]
add r2, #1
str r2, [r0]
add r0, #4
```

Example of deGoal code

- Simple program example:

```
void gen_vector_add(void *buffer, int vec_len, int val)
{
#[
   Begin buffer Prelude vec_addr

   Type int_t int 32 #(vec_len)
   Alloc int_t v

   lw v, vec_addr
   add v, v, #(val)
   sw vec_addr, v
]#
}
```

Interface: pointer to code buffer and I/O registers

Type definitions and variable allocations

Instructions

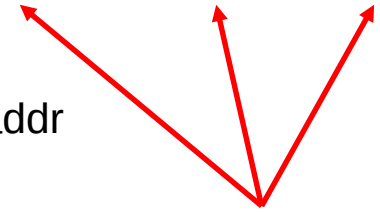
Example of deGoal code

- Simple program example:

```
void gen_vector_add(void *buffer, int vec_len, int val)
{
#[
    Begin buffer Prelude vec_addr

    Type int_t int 32 #(vec_len)
    Alloc int_t v

    lw v, vec_addr
    add v, v, #(val)
    sw vec_addr, v
]#
}
```



Determined by the application
and fixed in the final machine code

Example of deGoal code

- Simple program example:

```
void gen_vector_add(void *buffer, int vec_len, int val)
{
#[
    Begin buffer Prelude vec_addr

    Type int_t int 32 #(vec_len)
    Alloc int_t v

    lw v, vec_addr
    add v, v, #(val)
    sw vec_addr, v
]#
}
```

Example of deGoal code

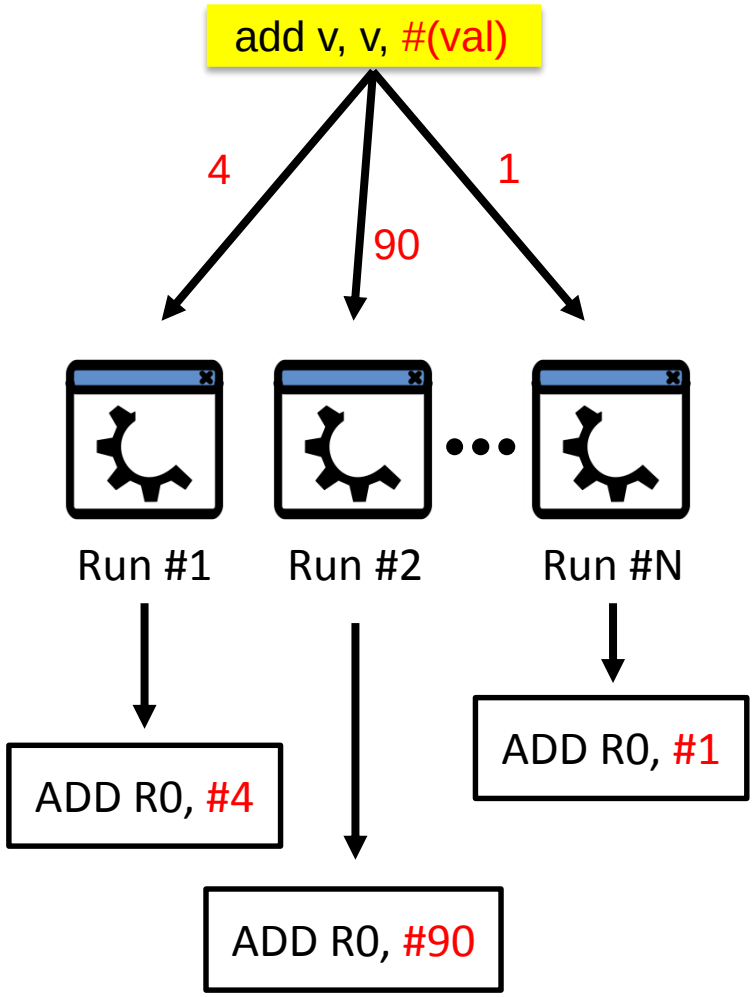
- Simple program example:

```
void gen_vector_add(void *buffer, int vec_len, int val)
{
#[
    Begin buffer Prelude vec_addr

    Type int_t int 32 #(vec_len)
    Alloc int_t v

    lw v, vec_addr
    add v, v, #(val)
    sw vec_addr, v
]#
}
```

Inline run-time constants



Example of deGoal code

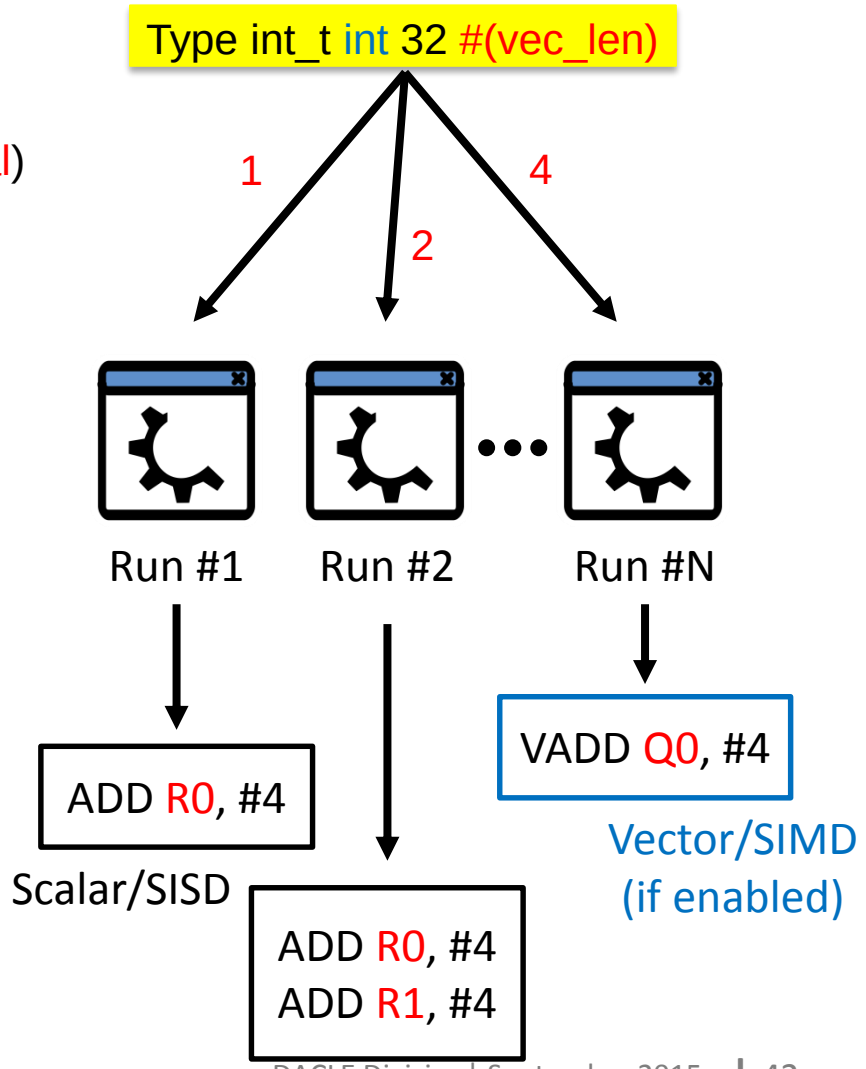
- Simple program example:

```
void gen_vector_add(void *buffer, int vec_len, int val)
{
#[
    Begin buffer Prelude vec_addr

    Type int_t int 32 #(vec_len)
    Alloc int_t v

    lw v, vec_addr
    add v, v, #(val)
    sw vec_addr, v
]#
}
```

Inline run-time constants



Example of deGoal code

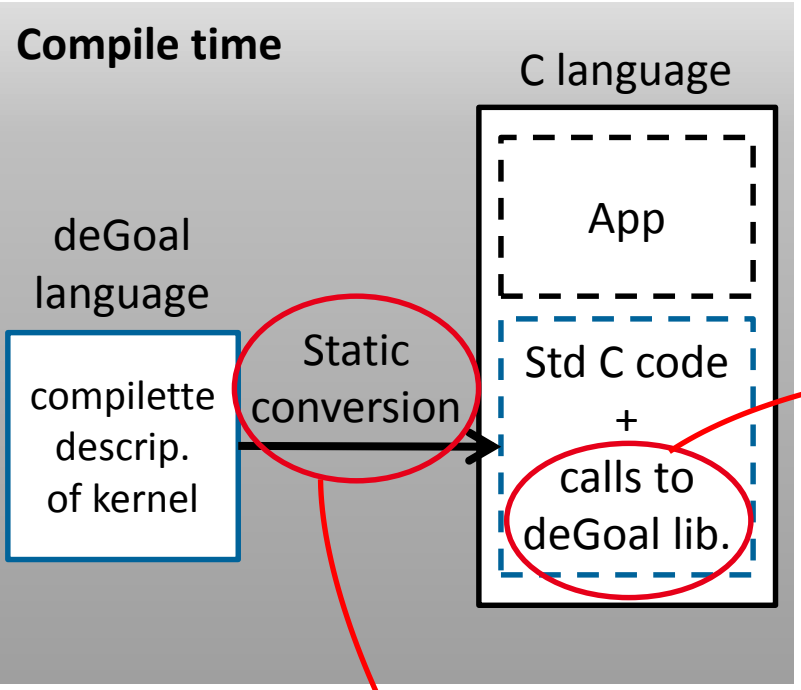
- Simple program example:

```
void gen_vector_add(void *buffer, int vec_len, int val)
{
#[
   Begin buffer Prelude vec_addr

   Type int_t int 32 1
   Alloc int_t r
]#
   int i;
   for (i = 0; i < vec_len; ++i) {
#[
      lw r, vec_addr
      add r, r, #(val)
      sw vec_addr, r
      add vec_addr, #(sizeof(int))
]#
   }
}
```

Loop unrolling:
"Copy-paste" a block of instructions

Contribution: ARM porting and extensions



degoaltoc:
Architecture-independent

~5000 lines of code

Highly tuned C library:
architecture-dependent

- ARM T32
- ARM A32
- K1
- MIPS
- MSP430
- PTX (nVIDIA)
- STxP70 (STHORM)

For ARM:
~15000 lines of code

Porting:

- **Basic ISA (integer)**
- **FP support (VFP extension)**
- **SIMD support (NEON extension)**

Extensions:

- **Configurable instruction scheduler**
- **Support for online auto-tuning**

deGoal for ARM: Performance evaluation

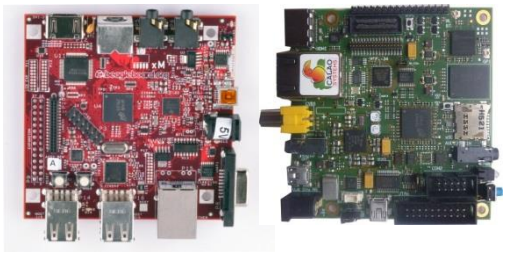
Setup

4 computing kernels:

- Linear transformation
- Interpolation
- Convolution
- Euclidean distance

(PARSEC & PARVEC suites)

Cortex-A8 Cortex-A9



Comparisons

Scalar (SISD) reference

Vectorized (SIMD) reference

Vs

- deGoal static: as ref. (no program specialization)
- deGoal dyn.: input opt. (w/ program specialization)

Scalar (SISD) reference

Vs

- deGoal static: as ref. & vectorization enabled

deGoal for ARM: Performance evaluation

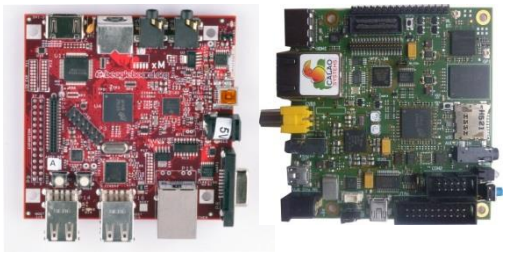
Setup

4 computing kernels:

- Linear transformation
- Interpolation
- Convolution
- Euclidean distance

(PARSEC & PARVEC suites)

Cortex-A8 Cortex-A9



Comparisons

"Raw" performance of ARM porting

Scalar (SISD) reference

Vectorized (SIMD) reference

Vs

- deGoal static: as ref. (no program specialization)
- deGoal dyn.: input opt. (w/ program specialization)

Scalar (SISD) reference

Vs

- deGoal static: as ref. & vectorization enabled

deGoal for ARM: Performance evaluation

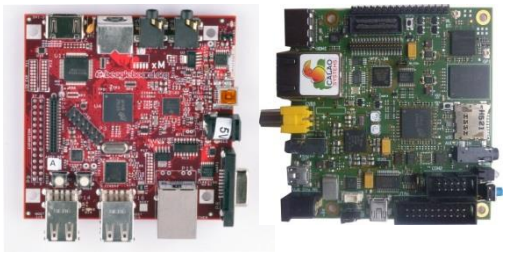
Setup

4 computing kernels:

- Linear transformation
- Interpolation
- Convolution
- Euclidean distance

(PARSEC & PARVEC suites)

Cortex-A8 Cortex-A9



Comparisons

"Raw" performance of ARM porting

Scalar (SISD) reference

Vectorized (SIMD) reference

Vs

- deGoal static: as ref. (no program specialization)
- deGoal dyn.: input opt. (w/ program specialization)

deGoal normal usage

Scalar (SISD) reference

Vs

- deGoal static: as ref. & vectorization enabled

deGoal for ARM: Performance evaluation

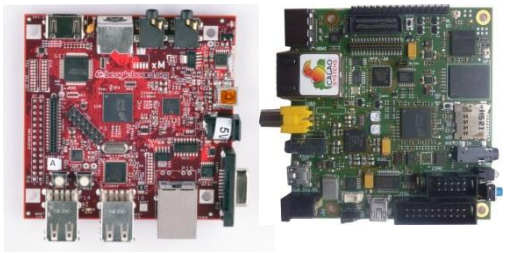
Setup

4 computing kernels:

- Linear transformation
- Interpolation
- Convolution
- Euclidean distance

(PARSEC & PARVEC suites)

Cortex-A8 Cortex-A9



Comparisons

"Raw" performance of ARM porting

Scalar (SISD) reference

Vectorized (SIMD) reference

Vs

- deGoal static: as ref. (no program specialization)
- deGoal dyn.: input opt. (w/ program specialization)

deGoal normal usage

Scalar (SISD) reference

Vs

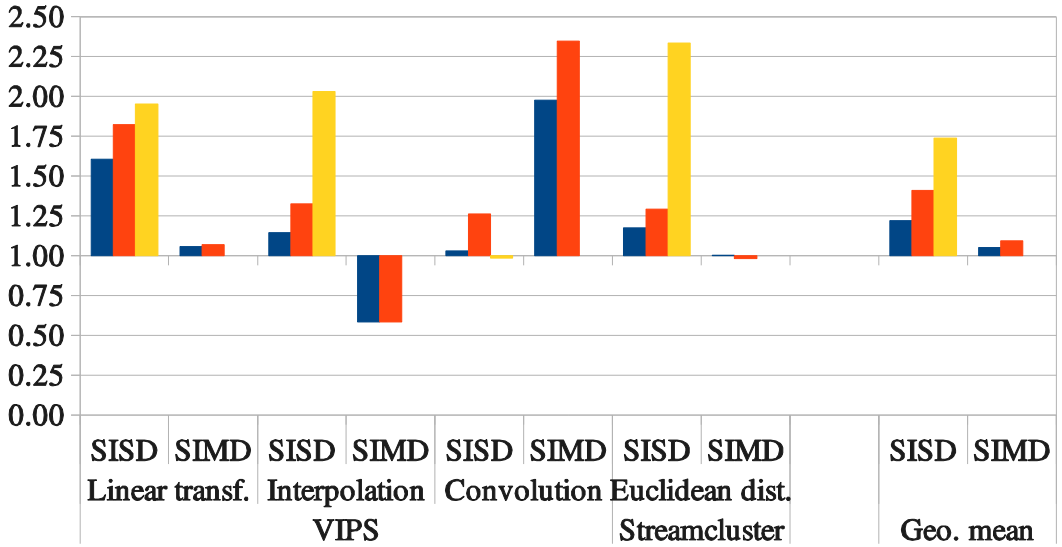
- deGoal static: as ref. & vectorization enabled

Blue version + Transparent vectorization

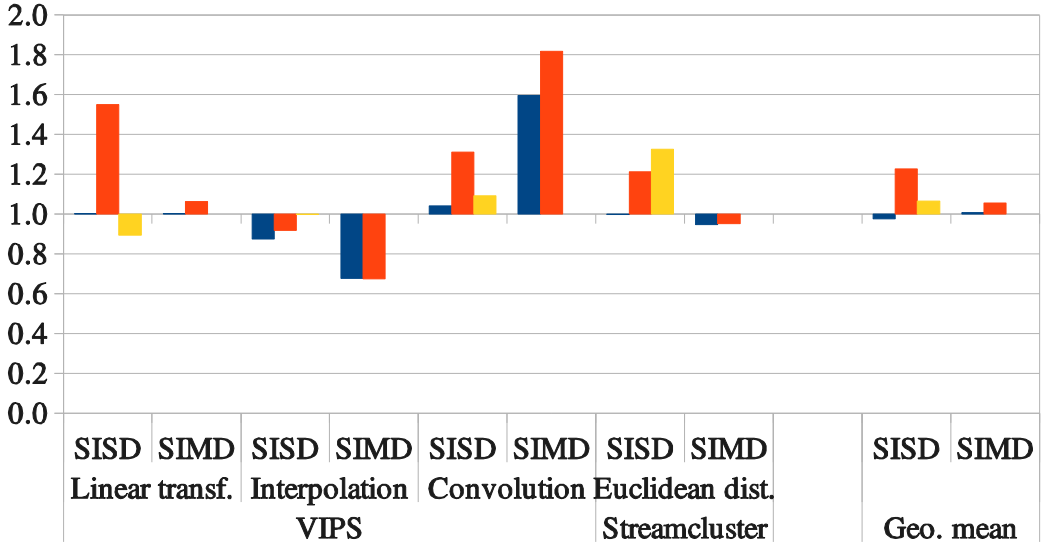
deGoal for ARM: Performance evaluation

Speedups of deGoal (higher is better) over PARSEC/PARVEC (gcc –O3)

Cortex-A8



Cortex-A9

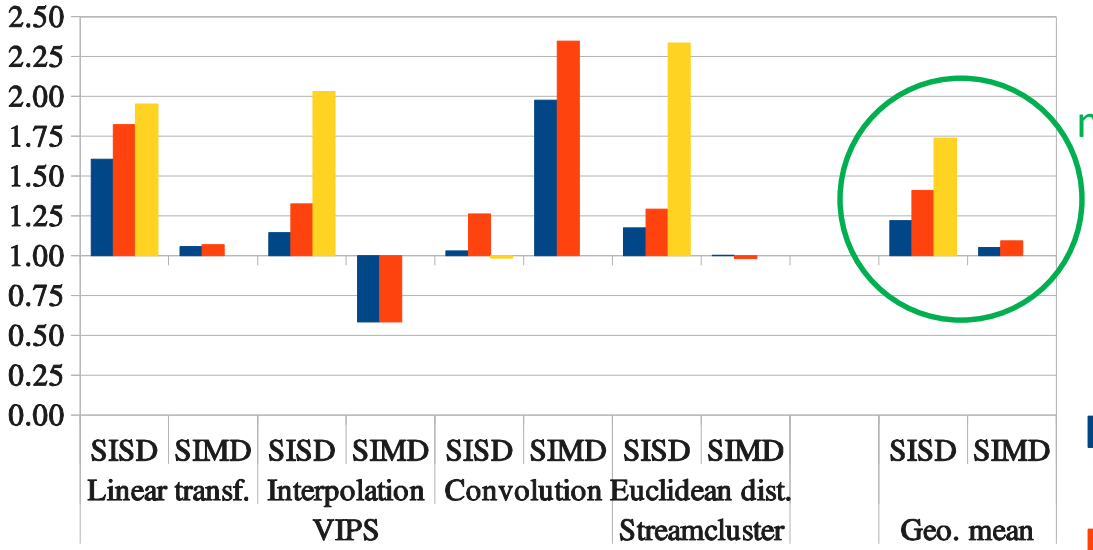


- deGoal static: as ref. (no program specialization)
- deGoal dyn.: input opt. (w/ program specialization)
- deGoal static: as ref. & vectorization enabled

deGoal for ARM: Performance evaluation

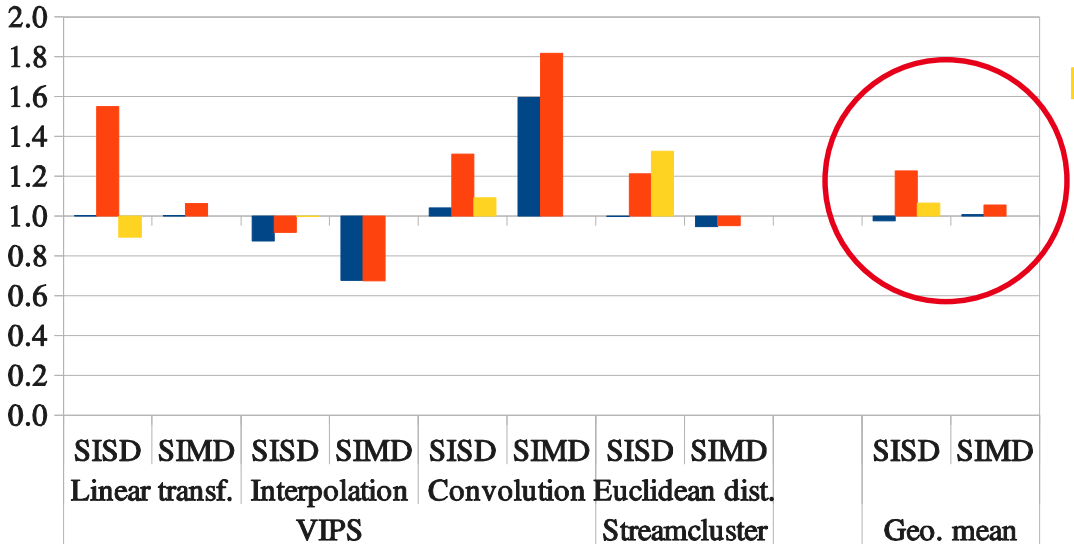
Speedups of deGoal (higher is better) over PARSEC/PARVEC (gcc –O3)

Cortex-A8



In-order pipe: benefits more from optimizations

Cortex-A9



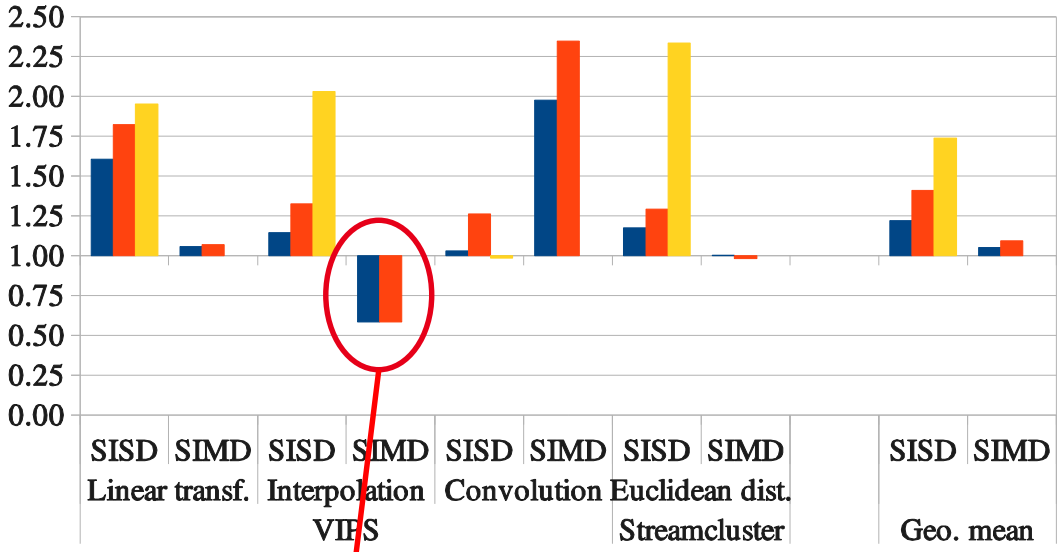
- deGoal static: as ref. (no program specialization)
- deGoal dyn.: input opt. (w/ program specialization)
- deGoal static: as ref. & vectorization enabled

Out-of-order pipe: speeds up non-optimized code

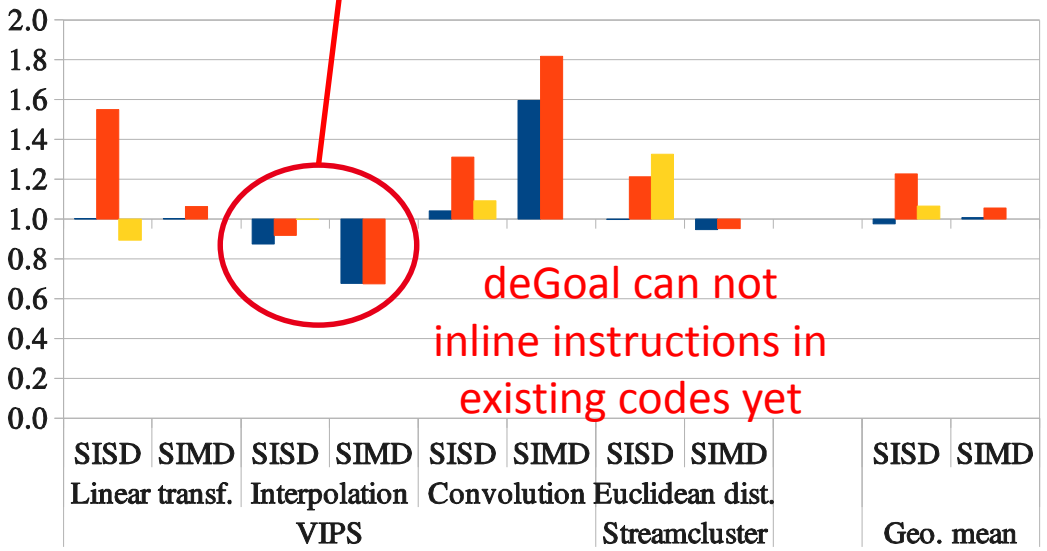
deGoal for ARM: Performance evaluation

Speedups of deGoal (higher is better) over PARSEC/PARVEC (gcc –O3)

Cortex-A8



Cortex-A9



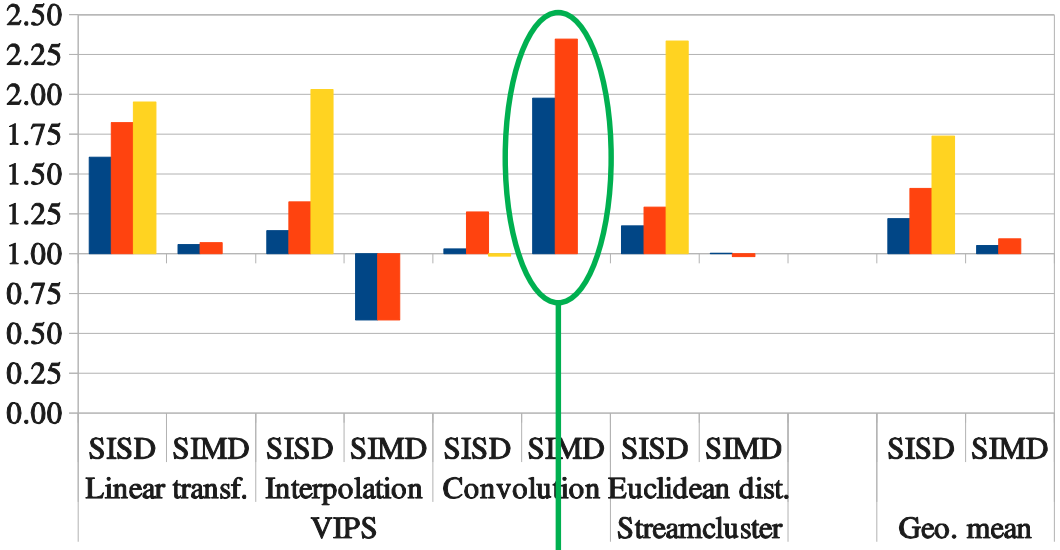
deGoal can not inline instructions in existing codes yet

- deGoal static: as ref. (no program specialization)
- deGoal dyn.: input opt. (w/ program specialization)
- deGoal static: as ref. & vectorization enabled

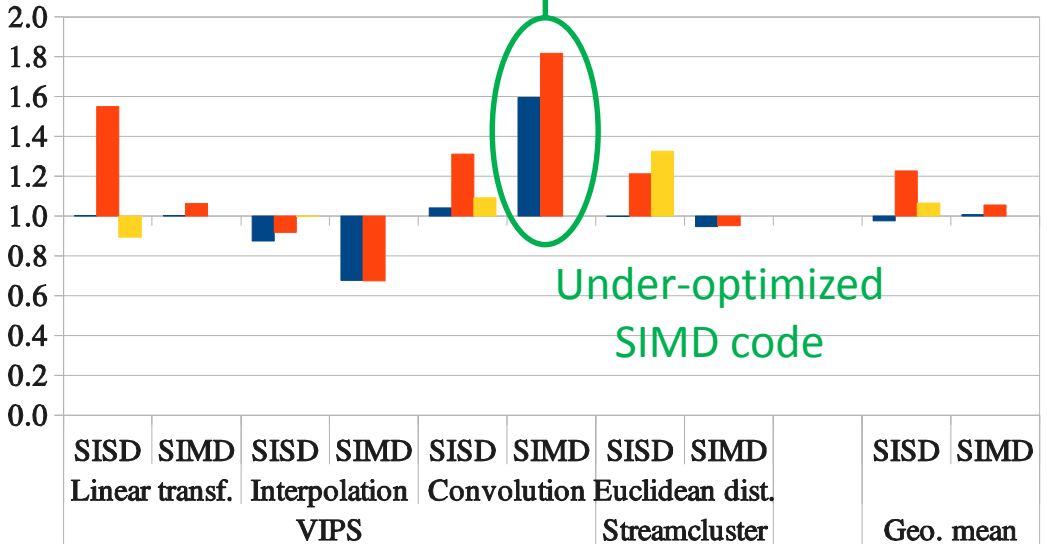
deGoal for ARM: Performance evaluation

Speedups of deGoal (higher is better) over PARSEC/PARVEC (gcc –O3)

Cortex-A8



Cortex-A9

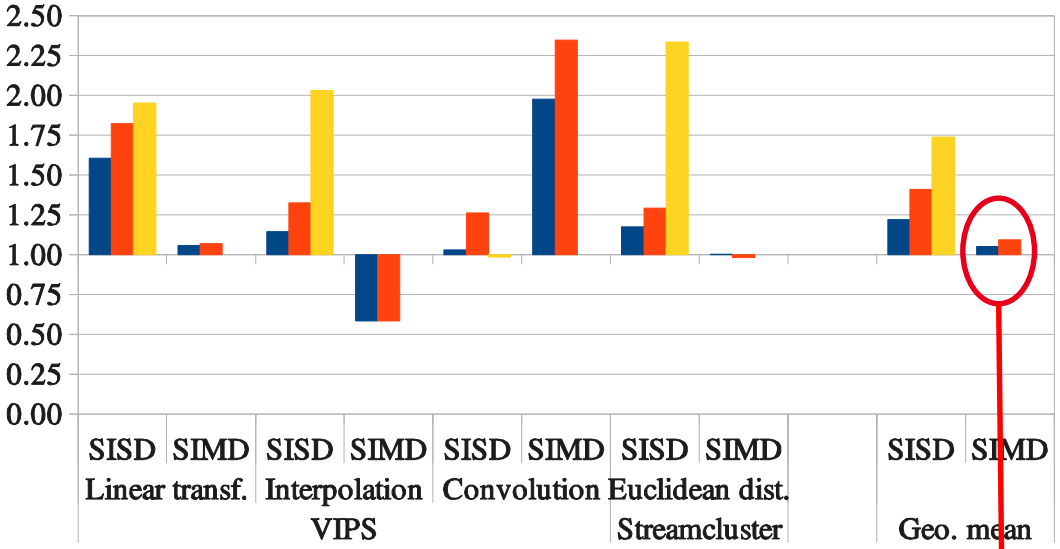


- deGoal static: as ref. (no program specialization)
- deGoal dyn.: input opt. (w/ program specialization)
- deGoal static: as ref. & vectorization enabled

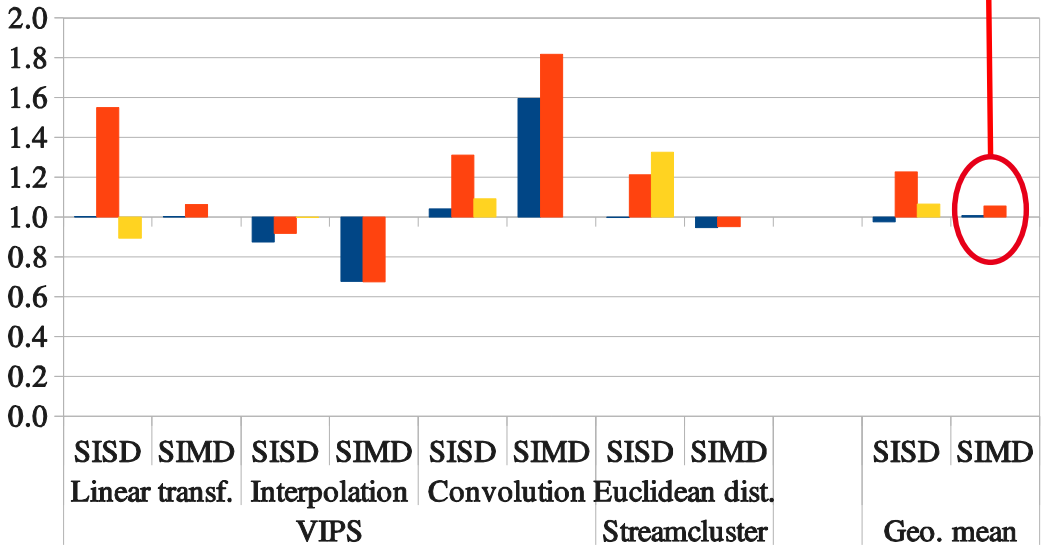
deGoal for ARM: Performance evaluation

Speedups of deGoal (higher is better) over PARSEC/PARVEC (gcc –O3)

Cortex-A8



Cortex-A9



Valid to any input set

- deGoal static: as ref. (no program specialization)
- deGoal dyn.: input opt. (w/ program specialization)
- deGoal static: as ref. & vectorization enabled

Modest SIMD speedups: Most ref. codes are already specialized (valid to limited inputs)

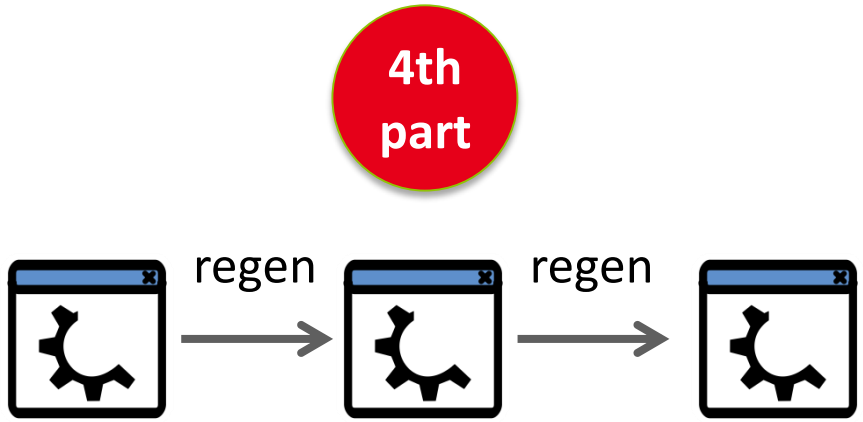
deGoal for ARM: Conclusions

- ARM porting and extensions: ~15000 lines of C code
- Same or better code quality:
  - Average speedup of **1.06** to funct. equivalent ref. codes
- Dynamic code specialization:
  - Average speedup of **1.19**
  - **Negligible dyn. overheads: < 0.02 %**

Suitable for a very fast online auto-tuner

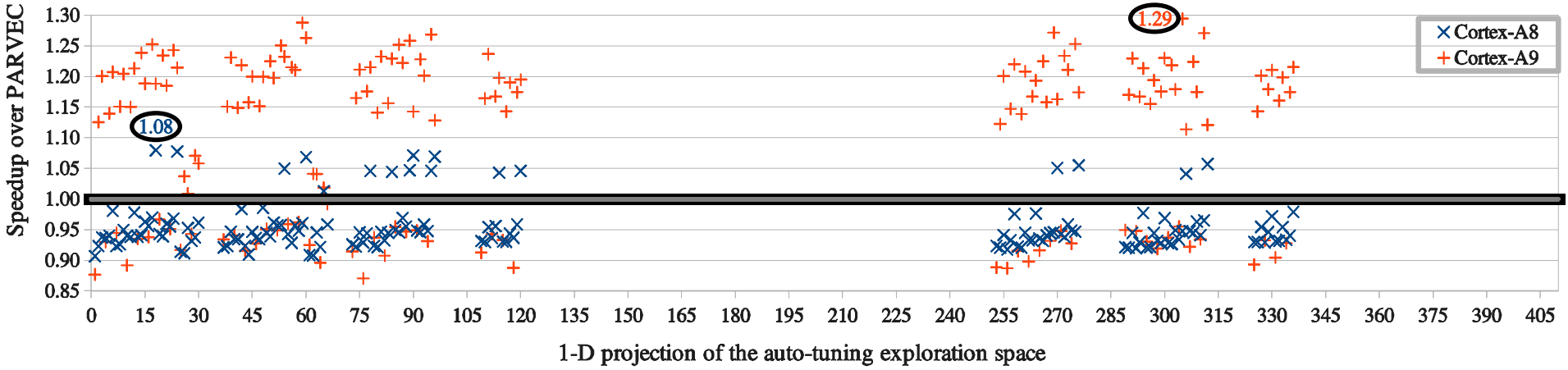
4th part: Online auto-tuning for ARM

- Why online auto-tuning?
- Online auto-tuning framework
- Example of auto-tuning code
- Results:
  - 2 HW platforms
  - 11 Simulated CPUs
  - In-order vs out-of-order
- Conclusions

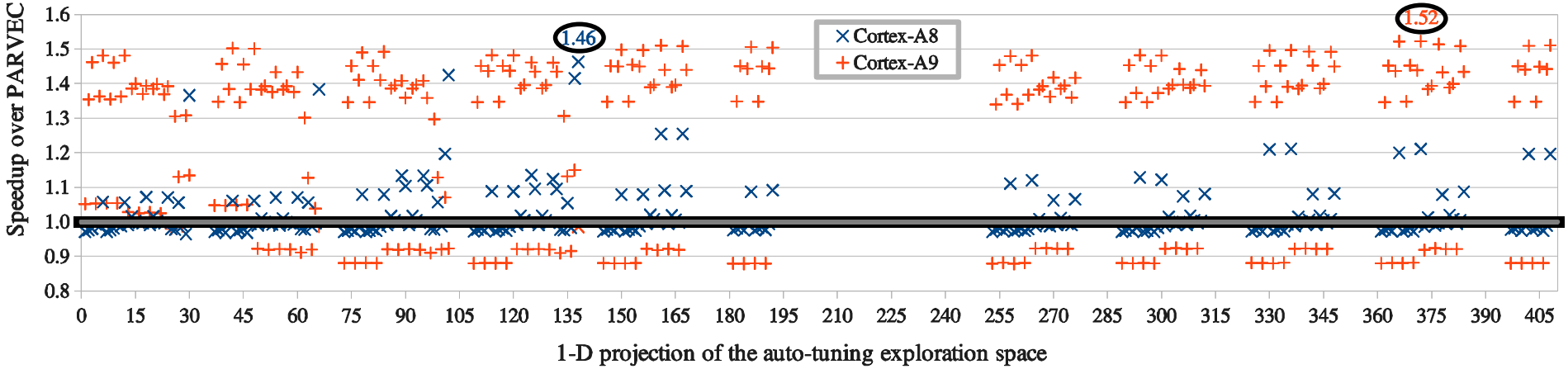


Why online auto-tuning?

- Best auto-tuned version is input- & µArchi-dependent



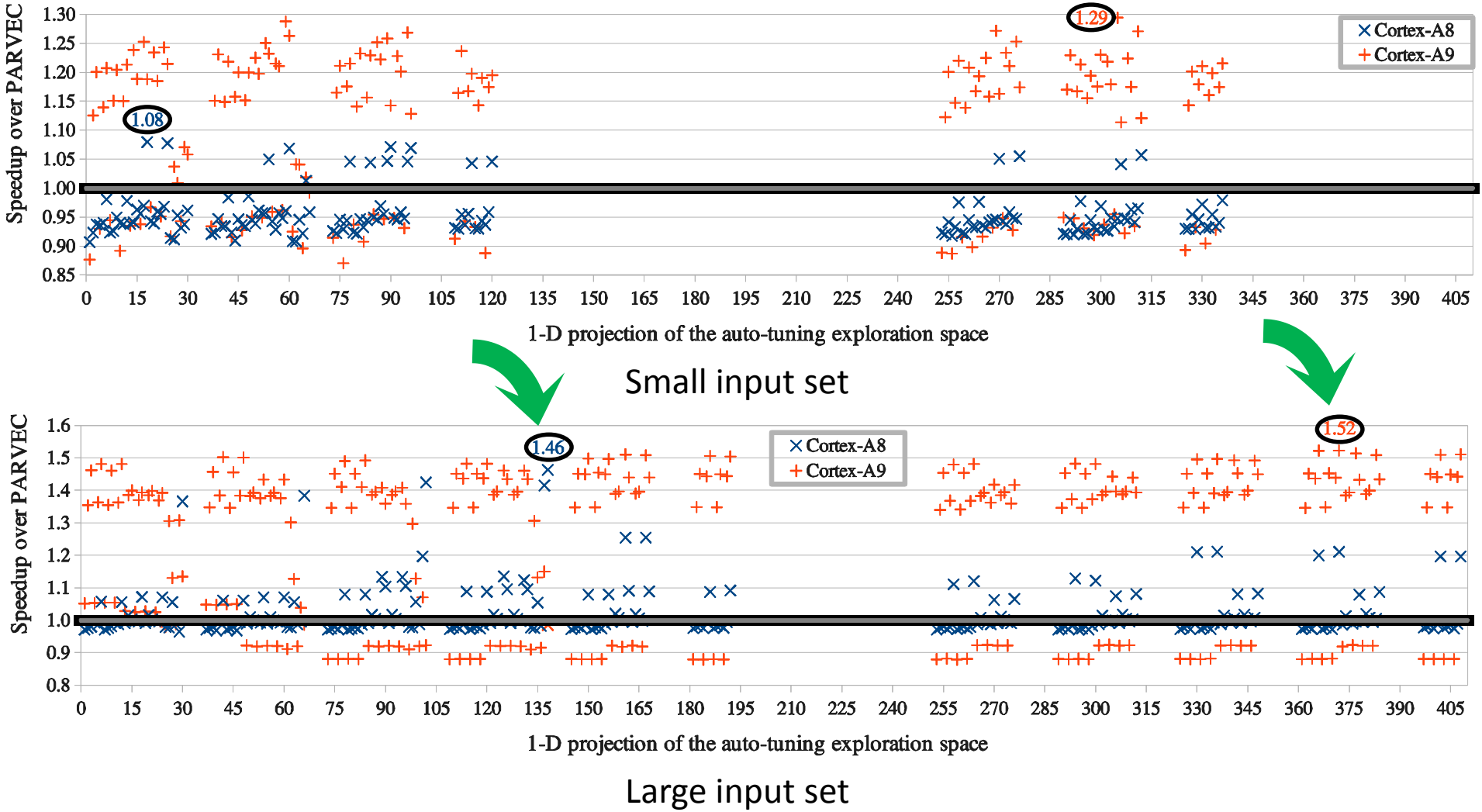
Small input set



Large input set

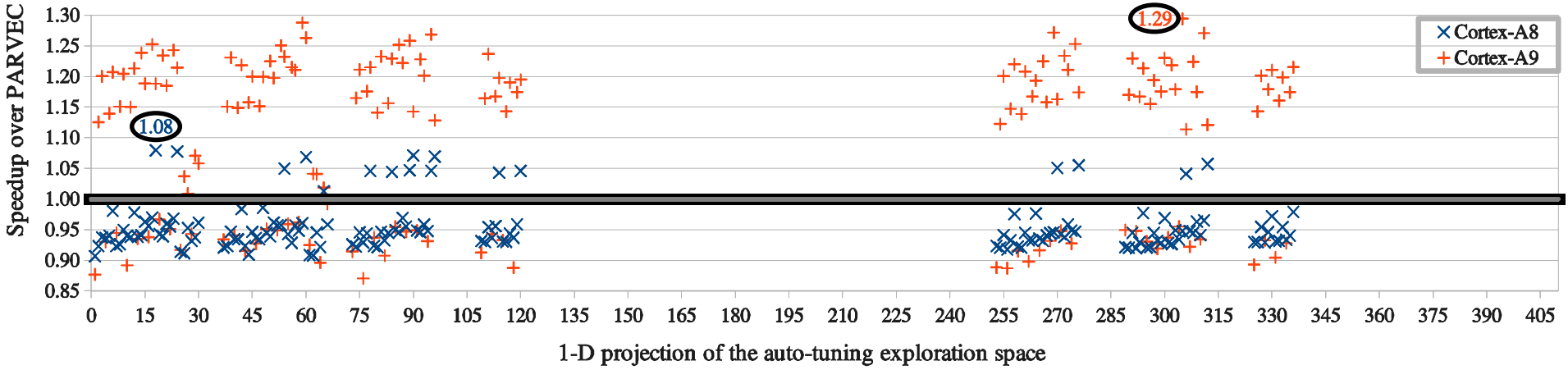
Why online auto-tuning?

- Best auto-tuned version is input- & µArchi-dependent

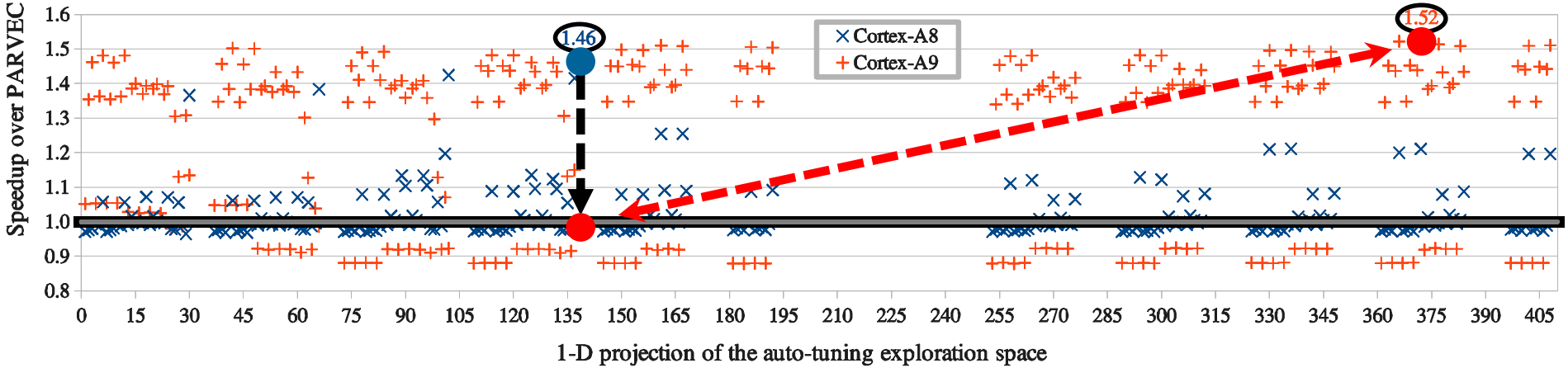


Why online auto-tuning?

- Best auto-tuned version is input- & μArchi-dependent



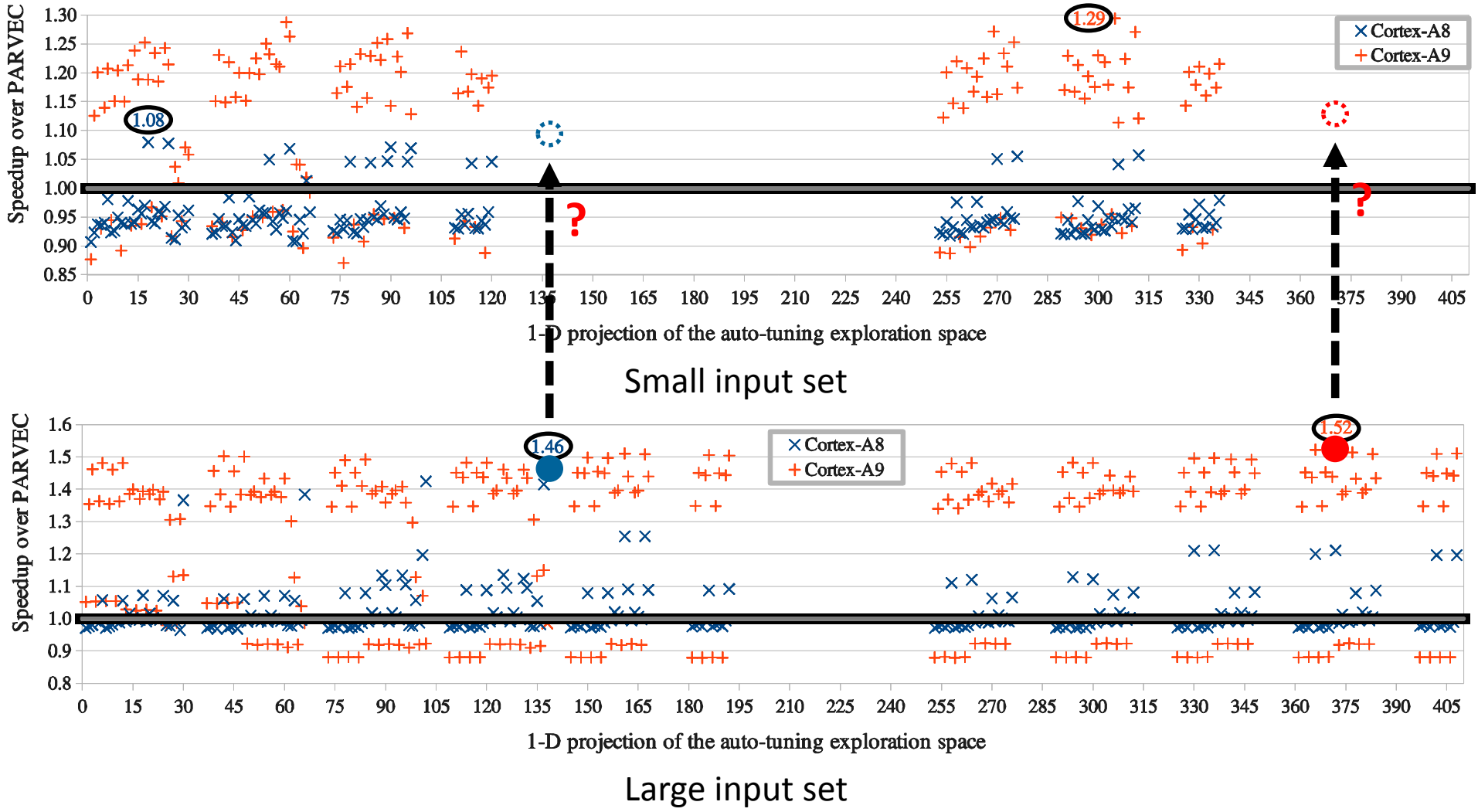
Small input set



Large input set

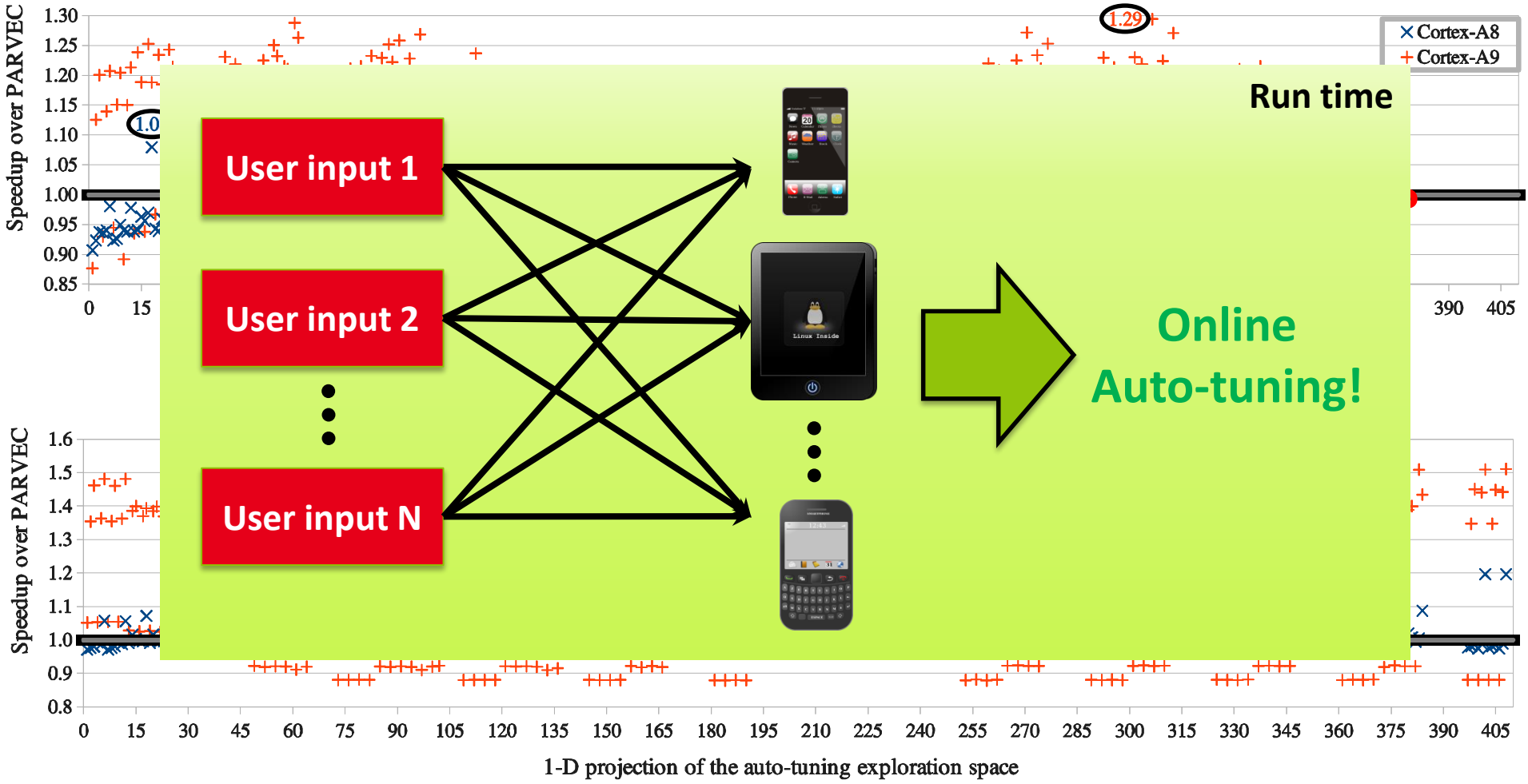
Why online auto-tuning?

- Best auto-tuned version is input- & µArchi-dependent



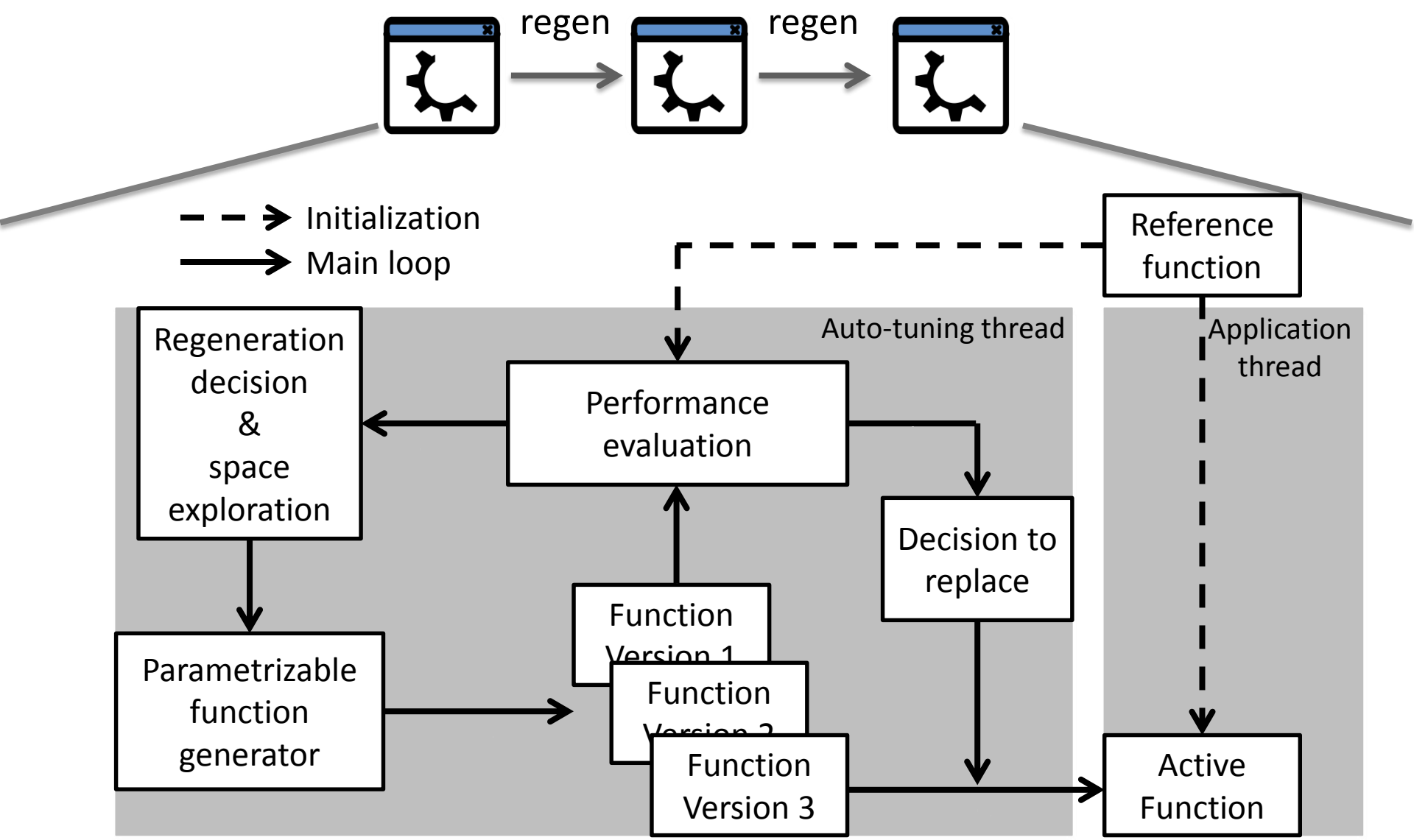
Why online auto-tuning?

- Best auto-tuned version is input- & µArchi-dependent

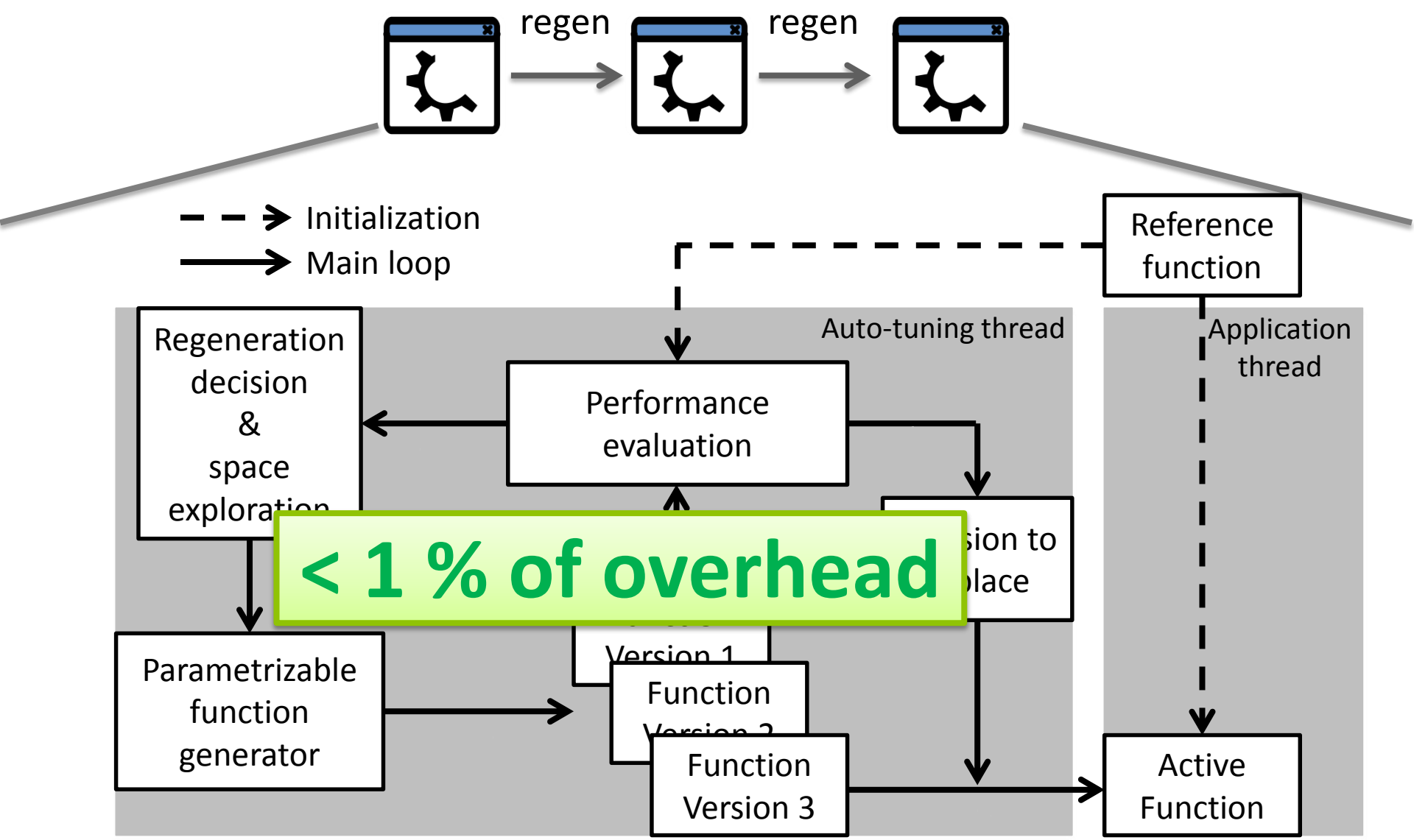


Large input set

Online auto-tuning architecture



Online auto-tuning architecture



Example of auto-tuning code

- For comparison: Reference SISD code
  - PARSEC 3.0 [Bienia]

```
float dist(Point p1, Point p2, int dim)
{
    int i;
    float result = 0.0;

    for (i = 0; i < dim; i++)
        result += (p1.coord[i] - p2.coord[i])*(p1.coord[i] - p2.coord[i]);

    return(result);
}
```

(Euclidean distance kernel in Streamcluster)

Example of auto-tuning code

- For comparison: Reference SIMD code
  - Hand vectorized in PARVEC [Cebrian]
  - Compiler intrinsics

```
float dist(Point p1, Point p2, int dim)
{
    float ret;
    int i;
    _MM_TYPE result, _aux, _diff, _coord1, _coord2;

    result = _MM_SETZERO();
    for (i=0;i<dim;i=i+SIMD_WIDTH) {
        _coord1 = _MM_LOADU(&(p1.coord[i]));
        _coord2 = _MM_LOADU(&(p2.coord[i]));
        _diff = _MM_SUB(_coord1, _coord2);
        _aux = _MM_MUL(_diff,_diff);
        result = _MM_ADD(result, _aux);
    }
    ret = (_MM_CVT_F(_MM_FULL_HADD(result, result)));
    return ret;
}
```

Example of auto-tuning code

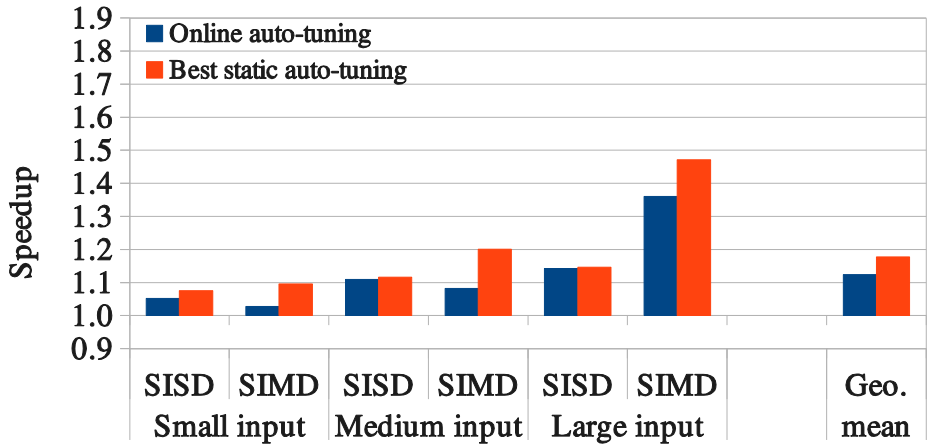
```
1  dist_gen(int dim, int vectLen, int hotUF, int coldUF,
       int pldStride)
2  {
3      numIter = function(dim, vectLen, hotUF, coldUF);
4      (...)
5  #[  loop #(numIter)                                          ]#
6      for (j = 0; j < coldUF; ++j) {
7        for (i = 0; i < hotUF; ++i) {
8  #[      lw Vc1[#(i)], coord1                                 ]#
9  #[      lw Vc2[#(i)], coord2                                 ]#
10         if (pldStride != 0) {
11 #[        pld coord1, #((vectLen-1)*4 + pldStride)           ]#
12 #[        pld coord2, #((vectLen-1)*4 + pldStride)           ]#
13         }
14 #[      sub Vc1[#(i)], Vc1[#(i)], Vc2[#(i)]                  ]#
15 #[      mac Vresult, Vc1[#(i)], Vc1[#(i)]                    ]#
16
17 #[      add coord1, coord1, #(vectLen*4)                     ]#
18 #[      add coord2, coord2, #(vectLen*4)                     ]#
19       }
20     }
21 #[  loopend                                                  ]#
22     (...)
23 #[  add result, Vresult                                      ]#
24     (...)
25 }
```

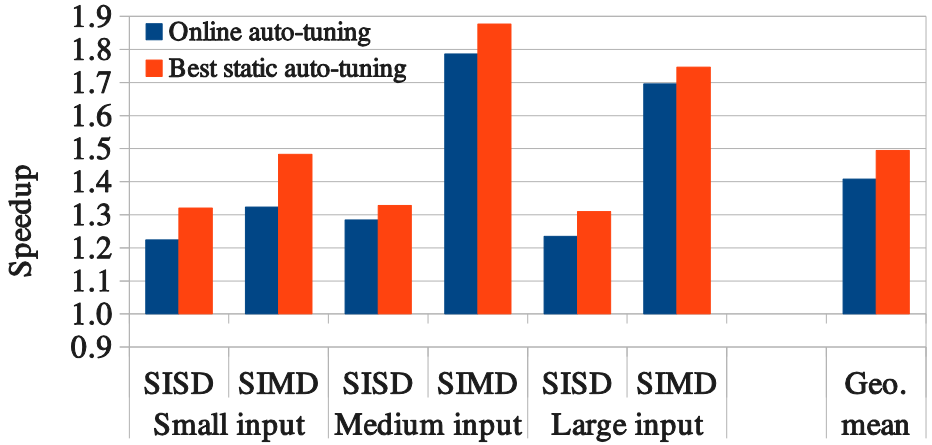
Auto-tuned params

Dynamically generated instructions

Online auto-tuning: HW results

- Highly CPU-bound benchmark (Streamcluster)





Cortex-A8

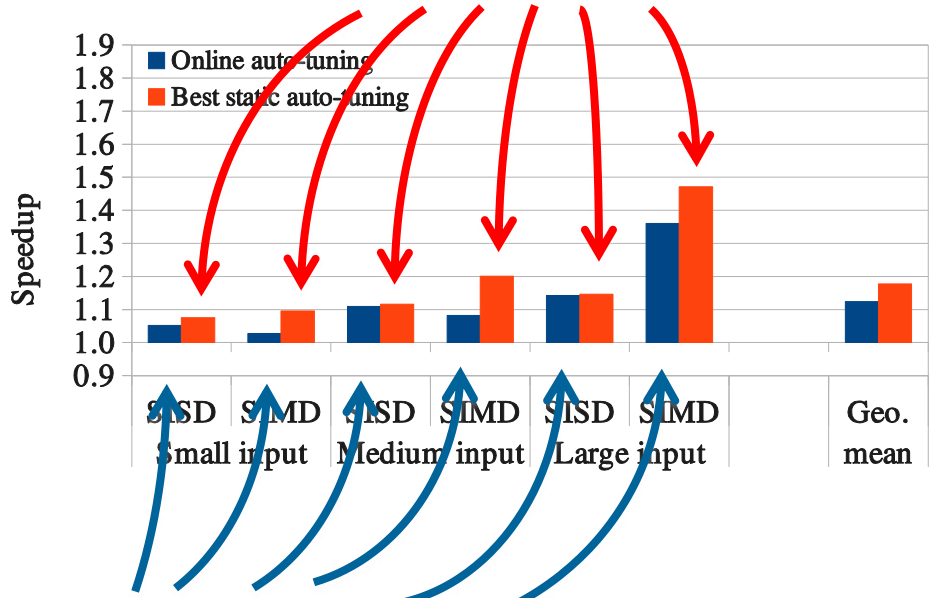
Avg speedup: 1.26
(all overheads included)

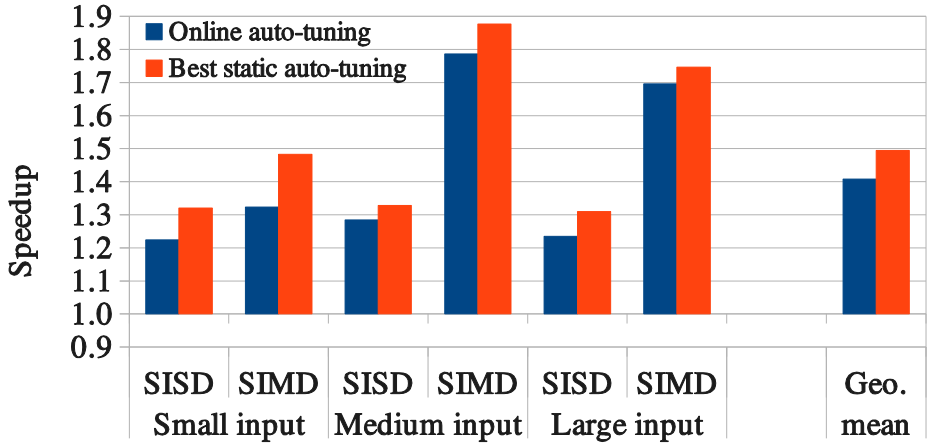
Cortex-A9

Online auto-tuning: HW results

- Highly CPU-bound benchmark (Streamcluster)

Several hours to find





Found during prog. execution (2.5 s to ~40 s) (up to 73 versions explored)

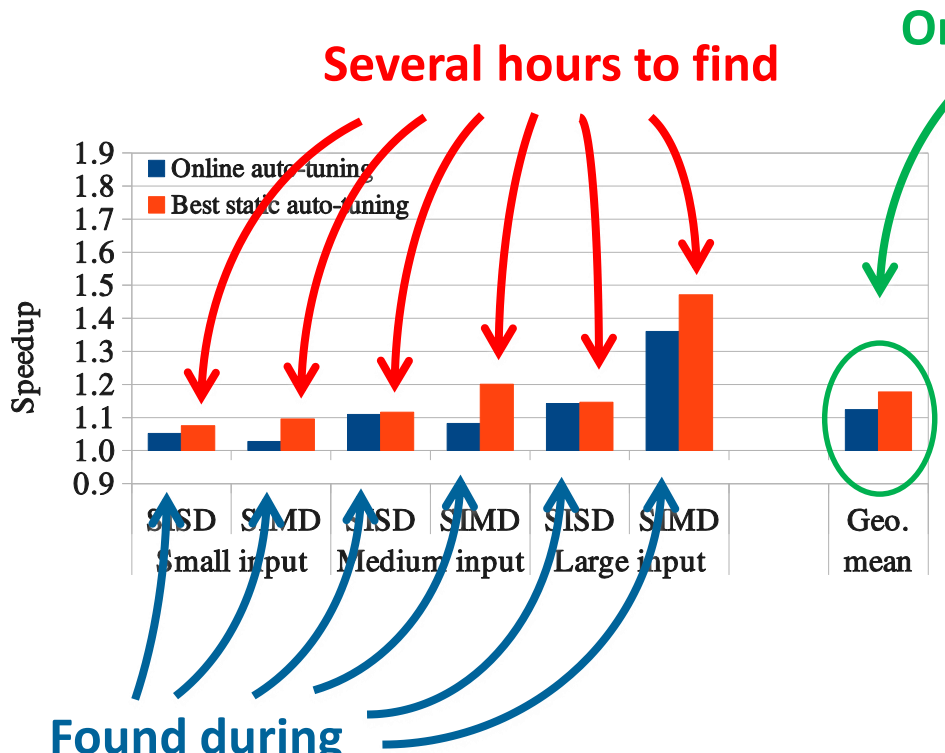
Cortex-A8

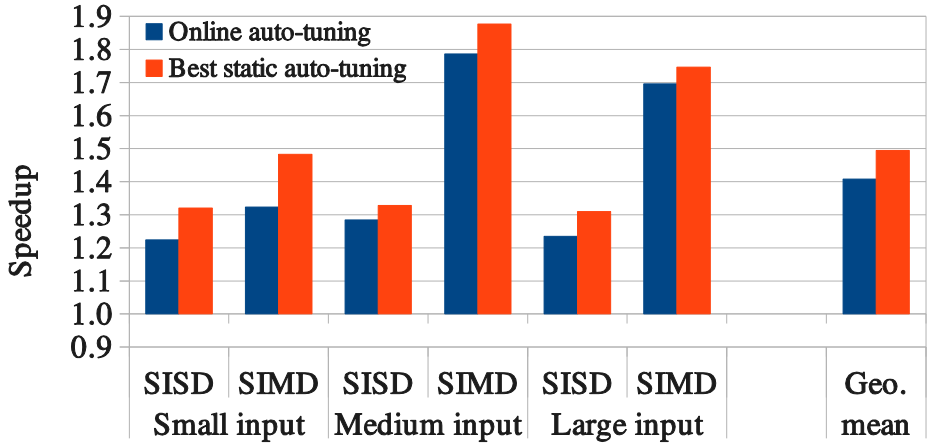
Avg speedup: 1.26
(all overheads included)

Cortex-A9

Online auto-tuning: HW results

- Highly CPU-bound benchmark (Streamcluster)





Found during prog. execution (2.5 s to ~40 s) (up to 73 versions explored)

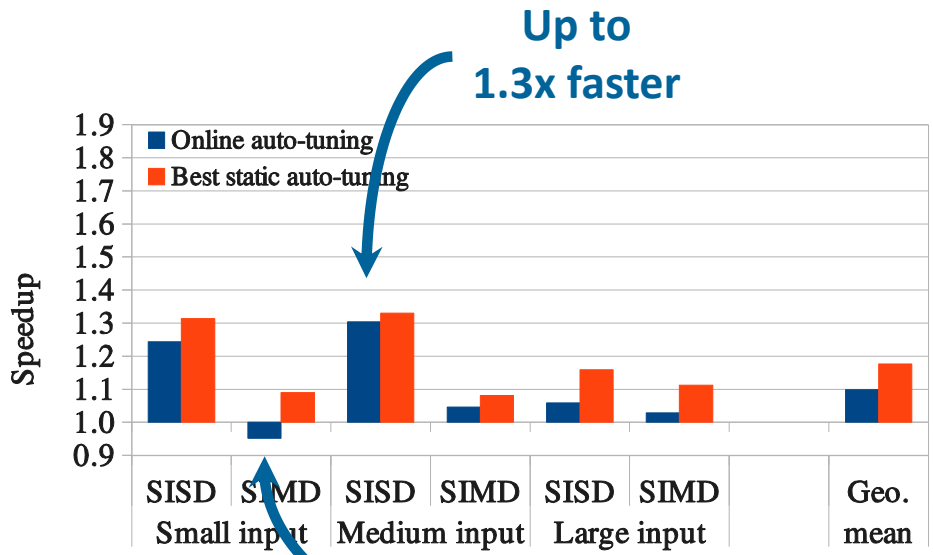
Cortex-A8

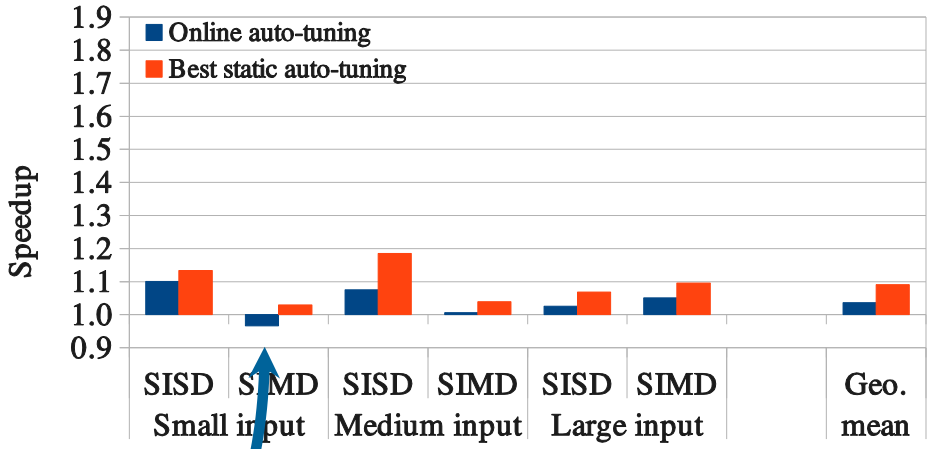
Avg speedup: 1.26
(all overheads included)

Cortex-A9

Online auto-tuning: HW results

- Highly memory-bound benchmark (VIPS im_lintra)





Bench. run-time ~500 ms,
(~30 versions explored)

Cortex-A8

Avg speedup: 1.07
(all overheads included)

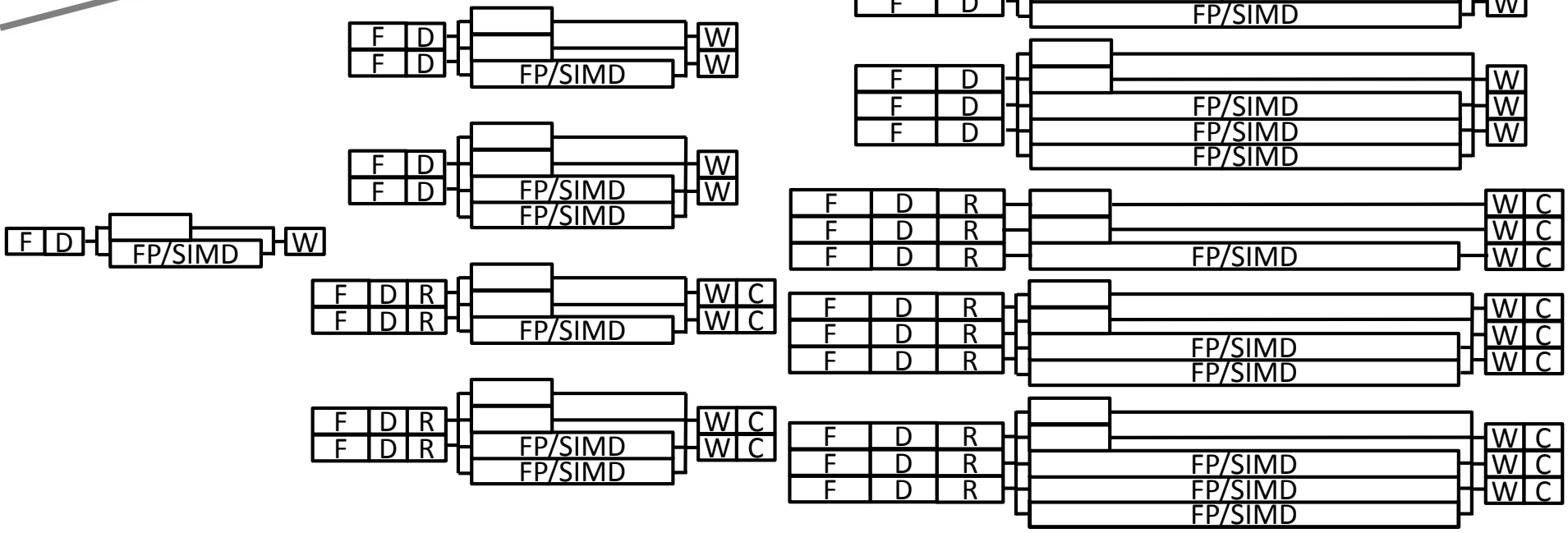
Cortex-A9

Online auto-tuning: Sim. results

Real HW:
2 cores

Cortex-A8 Cortex-A9

gem5 + McPAT:
11 CPUs



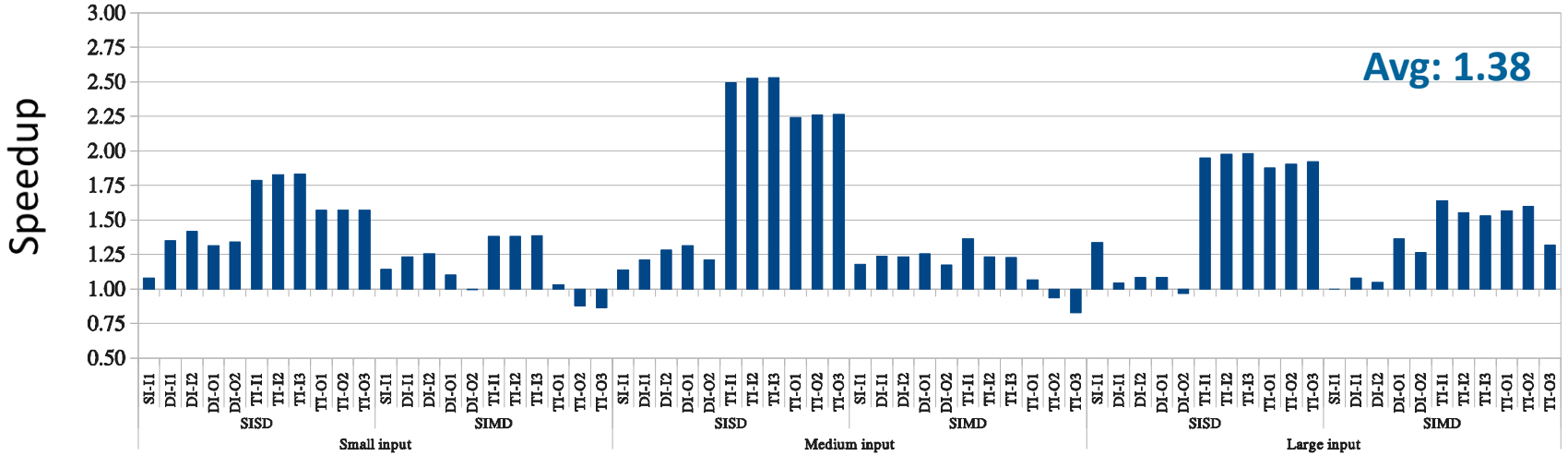
In-order

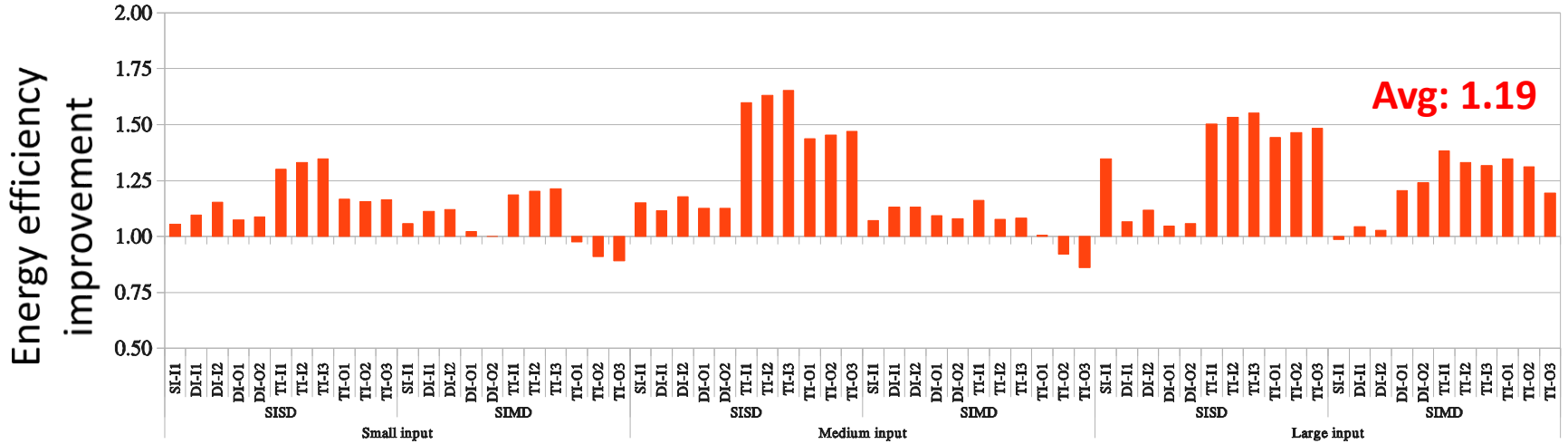
In-order & out-of-order

In-order & out-of-order

Online auto-tuning: Sim. results

- Simulation of CPU-bound bench: 66 running configs

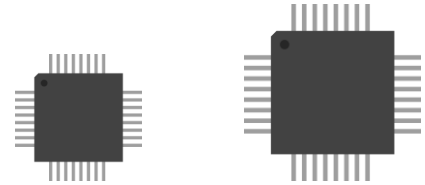


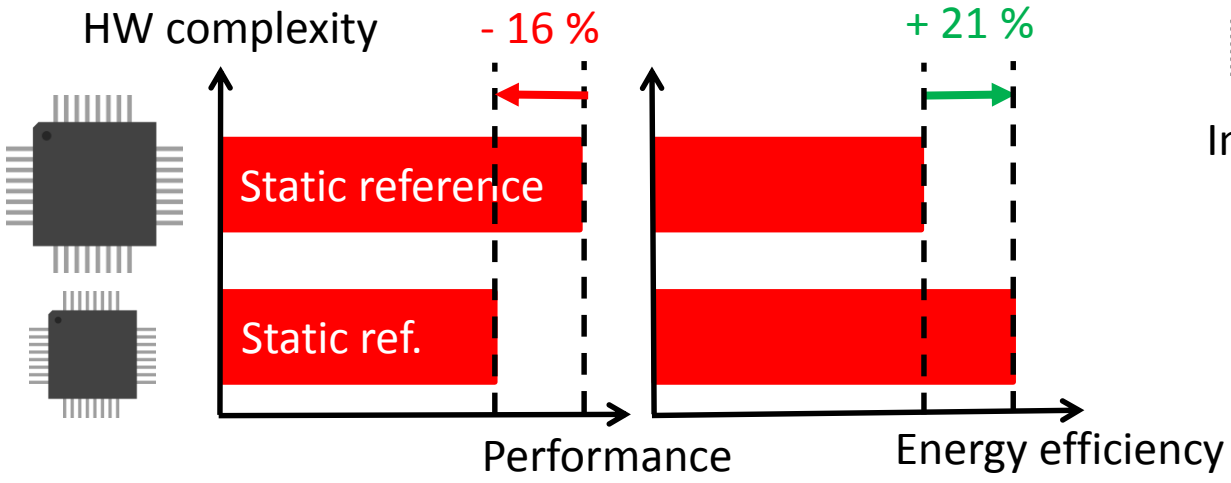


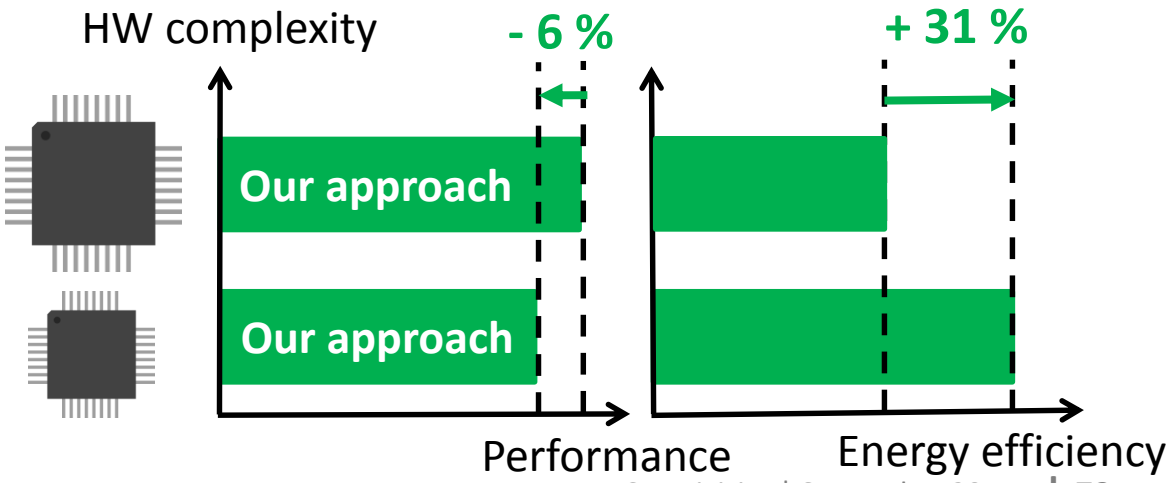
Online auto-tuning: Sim. results

- In-order vs out-of-order:
  - CPU-bound bench.

Area overhead OoO: 12 %



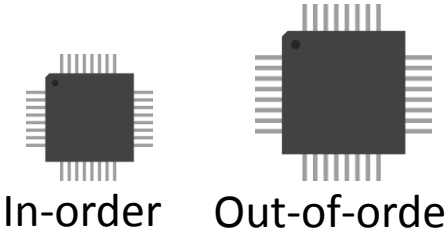


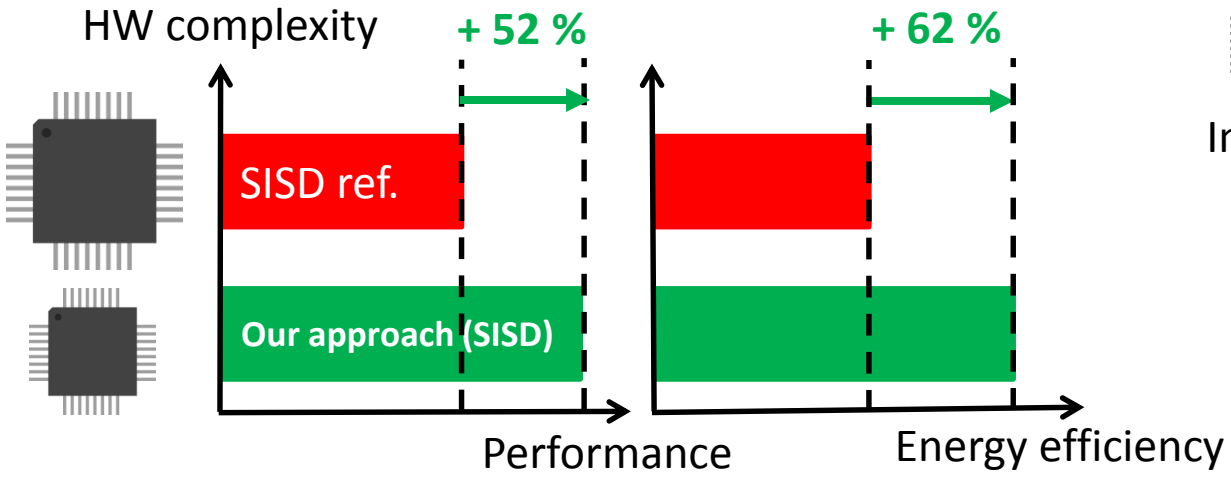


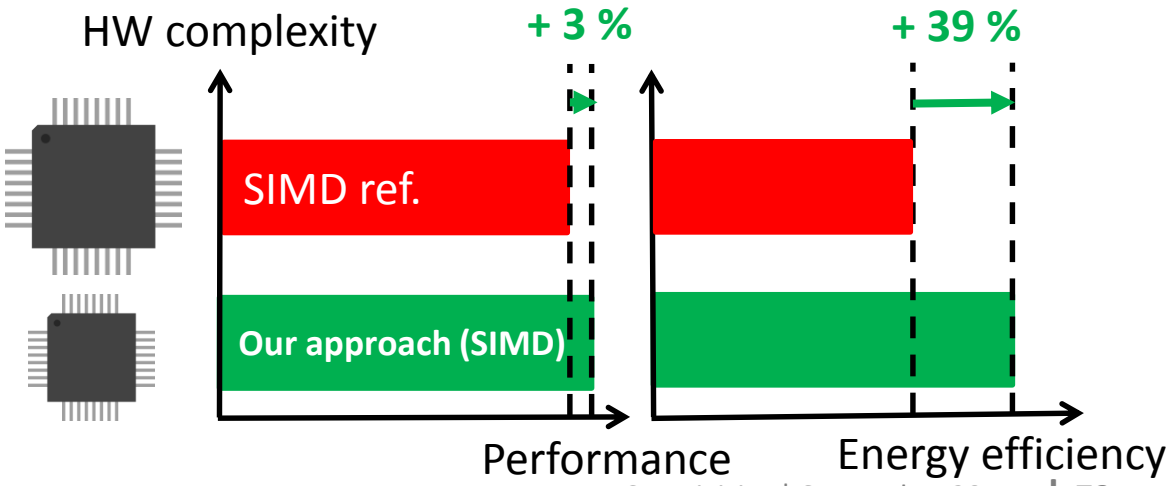
Online auto-tuning: Sim. results

- In-order vs out-of-order:
  - CPU-bound bench.

Area overhead OoO: 12 %





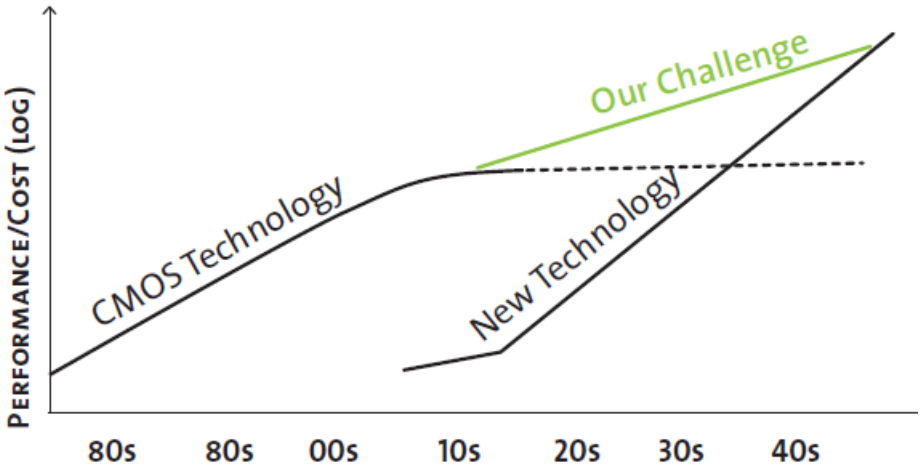


Online auto-tuning: Conclusions

- First online auto-tuner in short-running apps
- HW results:
  - Average speedup of **1.16**
  - Streamcluster:
    - Run-time code spec. alone: **1.10**
    - Run-time code spec. & auto-tun: **1.26**
  - Average overhead: **0.9 %**
- Sim. results: Online auto-tuning “replaces” HW OoO
  - Usually better performance in in-order
  - Energy reduced by **33 %**
  - (Simulation time: 3 to 15 h)

Thesis prospects: µArch simulation

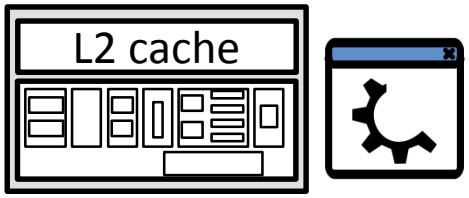
Scaling without Technology Help



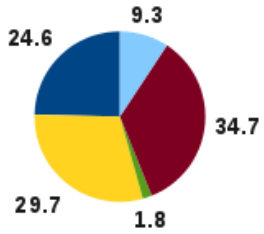
[Hill & Kozyrakis '12]

Today:
- CPU+GPGPU
- Heterogeneous multicores
- Fixed + reconfig. logic
- Approximate computing

Future?
- Special funct. units
- Accelerators in the pipe
- Near-threshold cores
- EOLE architecture [Perais]

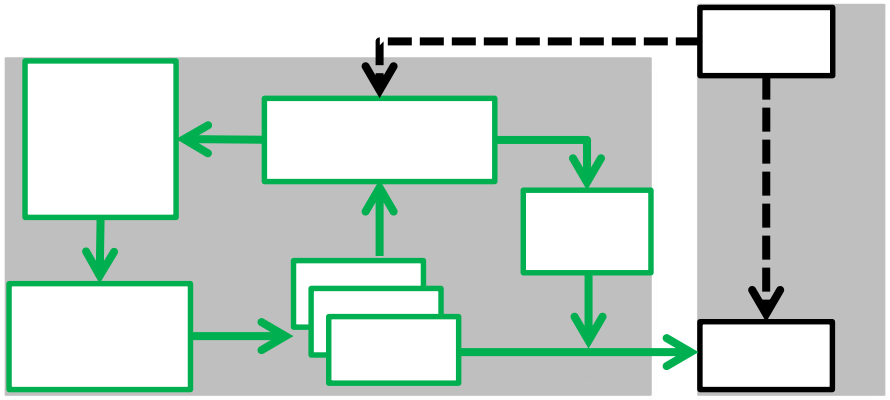


Sys config & binary

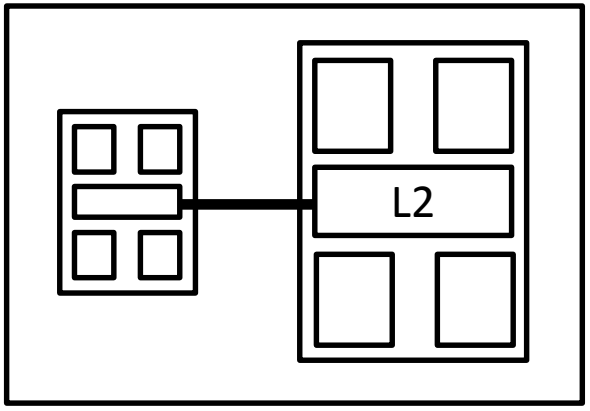


power & area

Thesis prospects: Online auto-tuning



< 1 % of overhead per core



Scalability to heterogeneous multi/manycores

Download Now

Install-time auto-tuning
(in ahead-of-time compilers)

```
1  dist_gen(int dim, int vectLen, int hotUF, int coldUF,
       int pldStride)
2  {
3      numIter = function(dim, vectLen, hotUF, coldUF);
4      (...)
5  #[  loop #(numIter)                                    ]#
6      for (j = 0; j < coldUF; ++j) {
7        for (i = 0; i < hotUF; ++i) {
8  #[      lw Vc1[#(j)], coord1                          ]#
9  #[      lw Vc2[#(i)], coord2                          ]#
10       if (pldStride != 0) {
11 #[       pld coord1, #((vectLen-1)*4 + pldStride)     ]#
12 #[       pld coord2, #((vectLen-1)*4 + pldStride)     ]#
13       }
14 #[      sub Vc1[#(i)], Vc1[#(i)], Vc2[#(i)]           ]#
15 #[      mac Vresult, Vc1[#(i)], Vc1[#(i)]             ]#
16
17 #[      add coord1, coord1, #(vectLen*4)              ]#
18 #[      add coord2, coord2, #(vectLen*4)              ]#
19       }
20     }
21 #[  loopend                                           ]#
22     (...)
23 #[  add result, Vresult                               ]#
24     (...)
25 }
```

C-like auto-tuning language

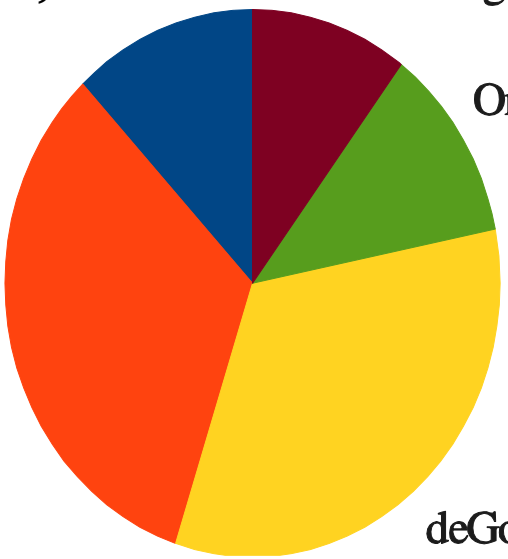
deGoal compilettes
+
Auto-tun lib calls

Thesis results

35-page internal report

170-page dissertation

Submitted: ACM TACO



SAMOS XIV & RAPIDO '15 (HiPEAC)

CC 2014 & DCE '15 (HiPEAC)

13 external requests (Microsoft, LIRMM, USA universities) & 2 internal requests (shared GIT created)

Post-doc with A. Seznec, a famous pipeline researcher

References (order of appearance)

- Voss, M. J. & Eigenmann, R. ADAPT: Automated De-Coupled Adaptive Program Transformation. *Proceedings of the Proceedings of the 2000 International Conference on Parallel Processing*, IEEE Computer Society, 2000, 163-
- Tiwari, A. & Hollingsworth, J. K. Online Adaptive Code Generation and Tuning. Proceedings of the 2011 *IEEE International Parallel & Distributed Processing Symposium*, IEEE Computer Society, 2011, 879-892
- Chen, Y.; Fang, S.; Eeckhout, L.; Temam, O. & Wu, C. Iterative Optimization for the Data Center. *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ACM, 2012, 49-60
- Ansel, J.; Pacula, M.; Wong, Y. L.; Chan, C.; Olszewski, M.; O'Reilly, U.-M. & Amarasinghe, S. SiblingRivalry: Online Autotuning Through Local Competitions. *Proceedings of the 2012 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, ACM, 2012, 91-100
- Shifer, E. & Weiss, S. Low-Latency Adaptive Mode Transitions and Hierarchical Power Management in Asymmetric Clustered Cores. *ACM Transactions on Architecture and Code Optimization*, ACM, 2008, 10, 10:1-10:25
- EETimes Slideshow: Samsung cagey on smartphone SoC at ISSCC http://www.eetimes.com/document.asp?doc_id=1263082&page_number=2 [Accessed: 5 November 2014]
- ARM website. Cortex-A7 Processor.

References (order of appearance)

- The Tech Report. AMD's A4-5000 'Kabini' APU reviewed. http://techreport.com/review/24856/amd-a4-5000-kabini-apu-reviewed/11 [Accessed: 5 November 2014].
- Greenhalgh, P. Big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7 ARM White paper, 2011
- Gutierrez, A.; Pusdesris, J.; Dreslinski, R. G.; Mudge, T.; Sudanthi, C.; Emmons, C. D.; Hayenga, M. & Paver, N. Sources of Error in Full-System Simulation. *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2014, 13-22
- Desikan, R.; Burger, D. & Keckler, S. W. Measuring Experimental Error in Microprocessor Simulation. *Proceedings of the 28th Annual International Symposium on Computer Architecture*, ACM, 2001, 266-277
- Ahn, J. H.; Li, S.; Seongil, O. & Jouppi, N. P. McSimA+: A Manycore Simulator with Application-level+ Simulation and Detailed Microarchitecture Modeling. *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013, 74-85
- Bienia, C. Benchmarking Modern Multiprocessors Princeton University, 2011
- Cebrian, J. M.; Jahre, M. & Natvig, L. Optimized Hardware for Suboptimal Software: The Case for SIMD-aware Benchmarks 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2014, 66-75
- Perais, A. and Seznec, A. EOLE: Paving the Way for an Effective Implementation of Value Prediction. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 481–492, June 2014.

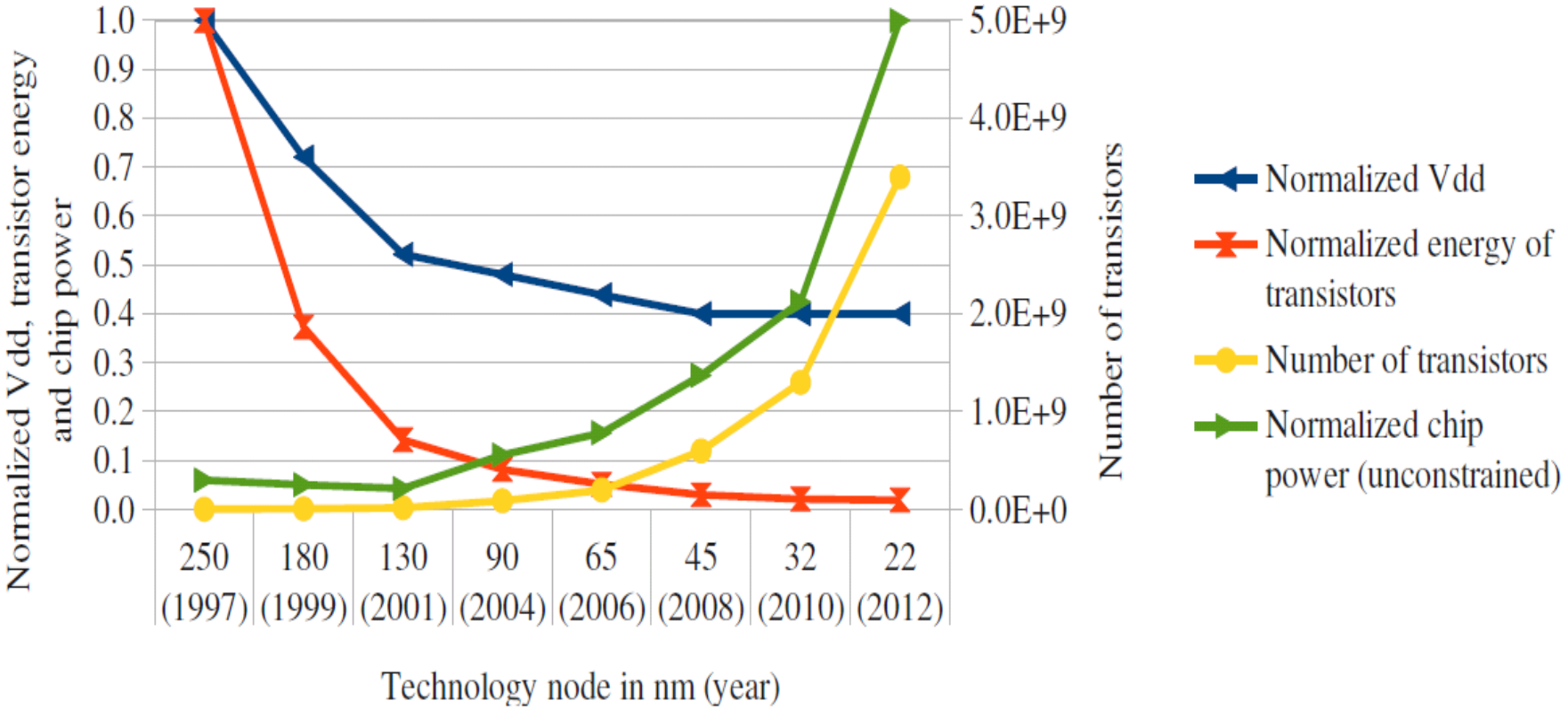
Personal bibliography

- Charles, H.-P.; Couroussé, D.; Lomüller, V.; Endo, F. A. & Gauguey, R. deGoal a Tool to Embed Dynamic Code Generators into Applications. ***Compiler Construction***, Springer Berlin Heidelberg, 2014, 8409, 107-112
- Endo, F. A.; Couroussé, D. & Charles, H.-P. Micro-architectural simulation of in-order and out-of-order ARM microprocessors with gem5. *2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (**SAMOS XIV**)*, 2014, 266-273
- Endo, F. A.; Couroussé, D. & Charles, H.-P. Micro-architectural Simulation of Embedded Core Heterogeneity with gem5 and McPAT. *Proceedings of the 2015 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools (**RAPIDO '15**)*, ACM, 2015, 7:1-7:6
- Endo, F. A.; Couroussé, D. & Charles, H.-P. Towards a dynamic code generator for run-time self-tuning kernels in embedded applications. To appear in the *4th International Workshop on Dynamic Compilation Everywhere (**DCE' 15**)*. January 2015
- Endo, F. A. Génération dynamique de code pour l'optimisation énergétique. To appear online. **PhD thesis**. September 2015.

Power is almost not scaling down anymore

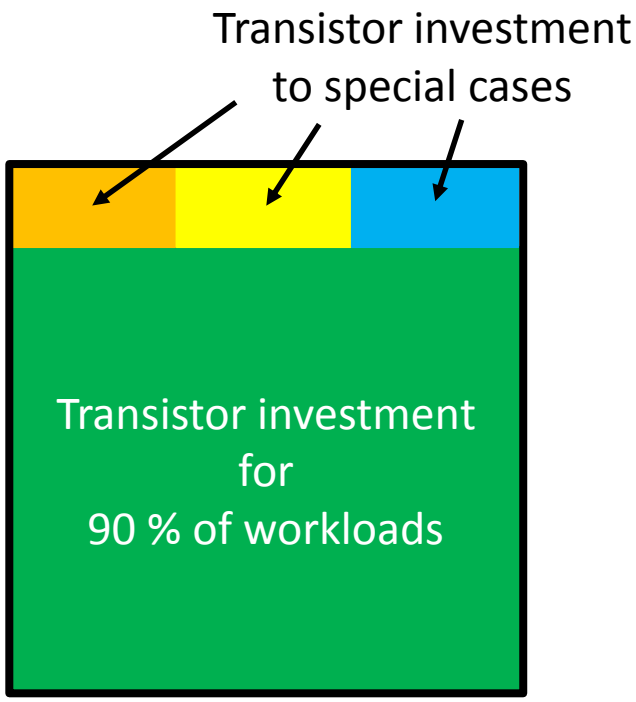
Perf if Energy

Silicon tech. limitation

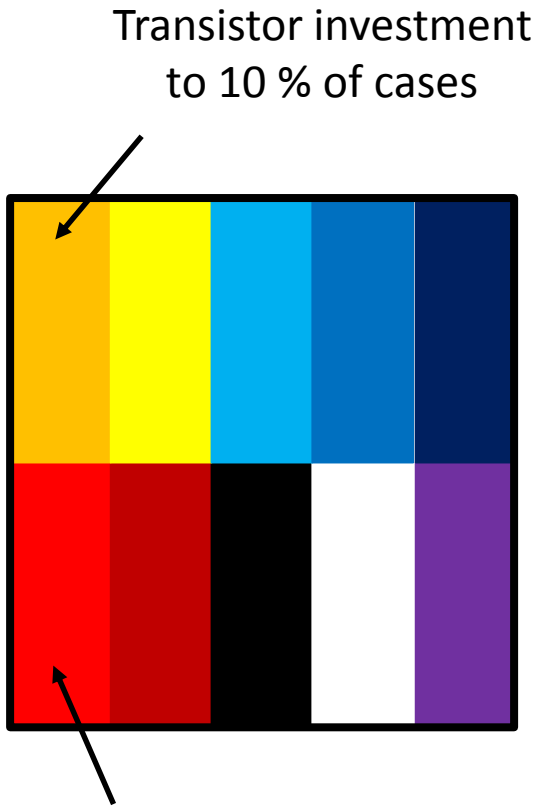


Trend in GP computing: From 90/10 to 10x10

Processors today

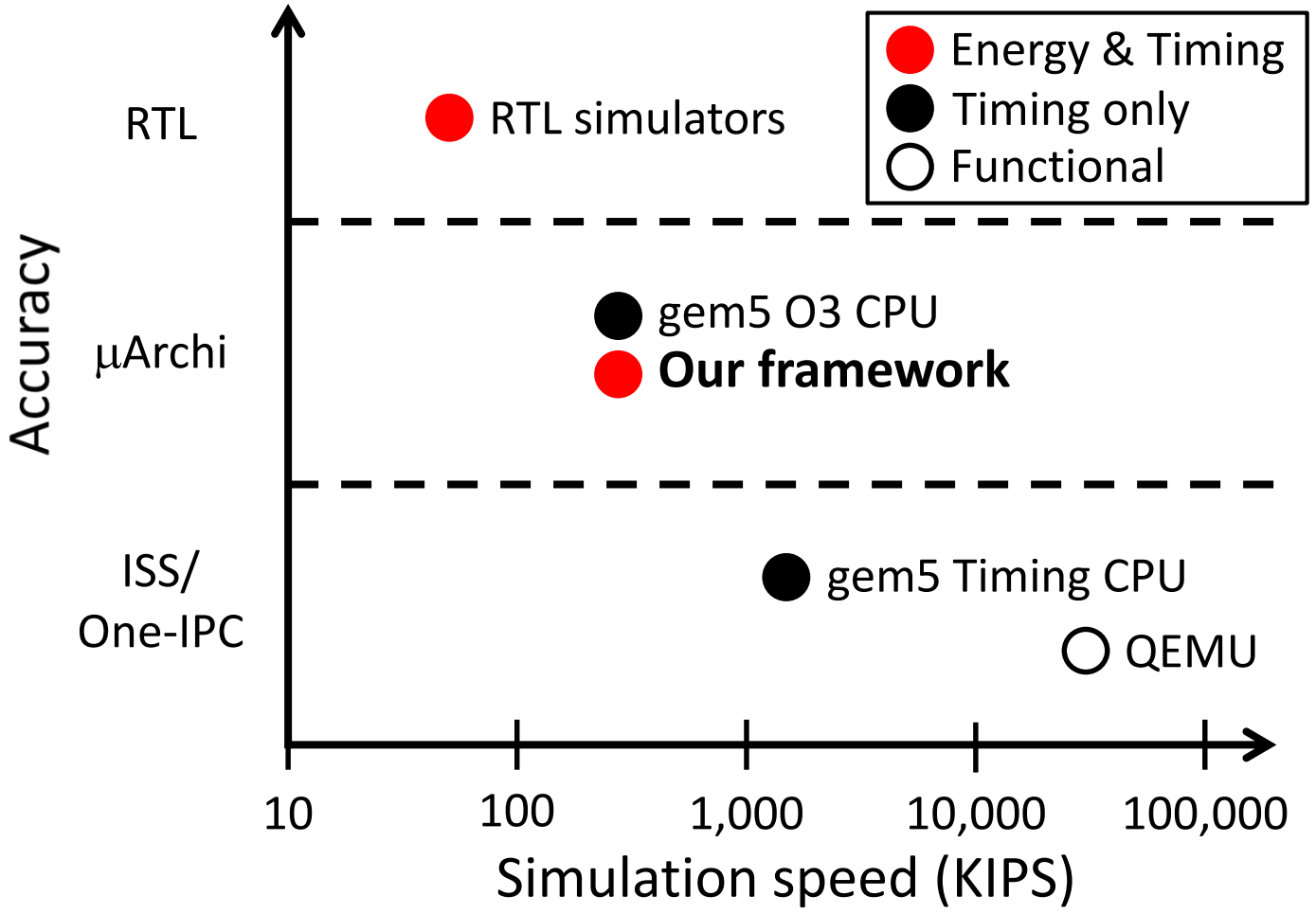


Processors tomorow



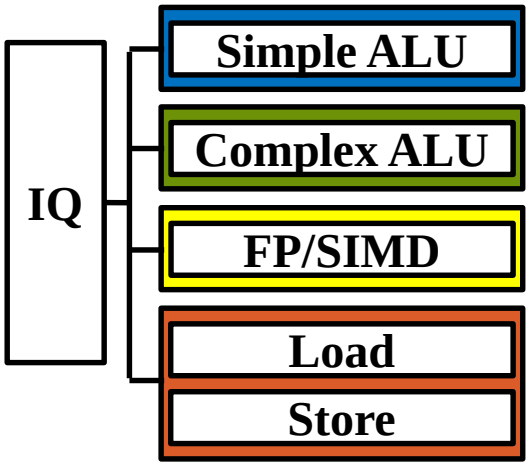
So on...

Abstraction level of simulators

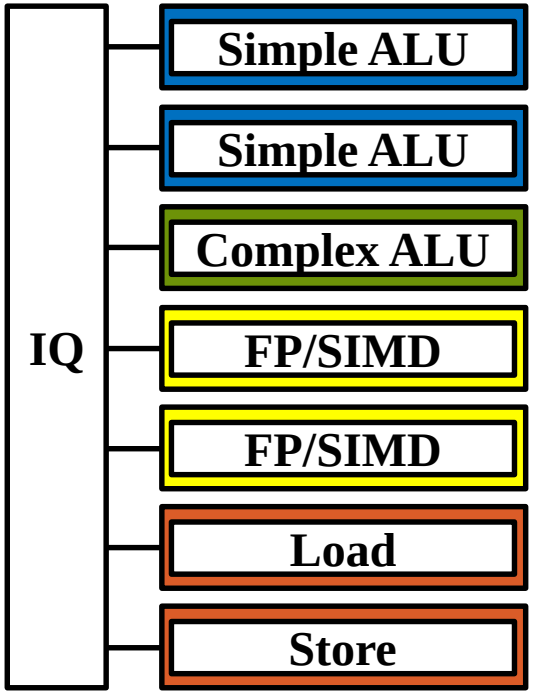


gem5 EXE stage model

Cortex-A7



Cortex-A15



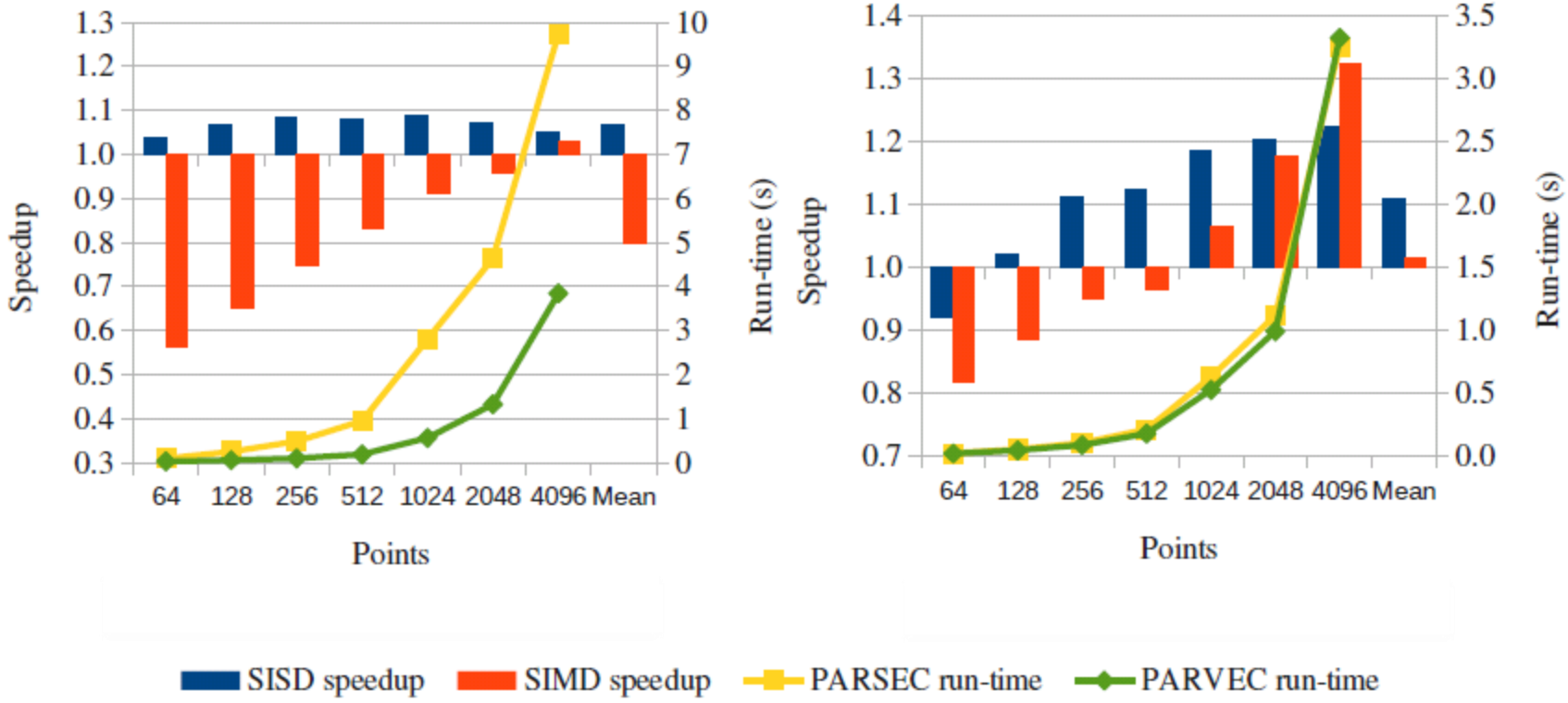
gem5 EXE stage model

Example of FU	gem5 opClass	Example of instructions
Simple ALU	IntAlu	MOV, ADD, SUB, AND, ORR
Complex ALU	IntMult	MUL, MLA
	IntDiv	UDIV, SDIV
FP/SIMD Unit[1]	SimdFloatAdd	VADD, VSUB
	SimdFloatMult	VMUL, VNMUL
	SimdFloatMultAcc	VMLA, VMLS, VNMLA, VNMLS
Load Unit	MemRead	LDR, VLDR
Store Unit	MemWrite	STR, VSTR

Stats of online auto-tuning in the A8 & A9

Bench.	Input set	Explo-rable versions	Exploration limit in one run	Run-time regeneration and space exploration						
				Kernel calls	Explored		Overhead to bench. run-time		Duration to kernel life	
					A8	A9	A8	A9	A8	A9
Stream-cluster	dim=32	390	43-49	5315388	49	49	0.2 % (11 ms)	0.4 % (9.2 ms)	13 / 4.4 %	32 %
	dim=64	510	55-61		58	61	0.2 % (17 ms)	0.3 % (15 ms)	6.3 / 2.7 %	22 %
	dim=128	630	67-73		67	73	0.2 % (30 ms)	0.2 % (26 ms)	5.6 / 1.8 %	15 %
VIPS	Small	858	106-112	1200	44	28	4.2 % (26 ms)	2.5 % (12 ms)	100 %	100 %
	Medium	330	39-45	2336	40	42	0.9 % (14 ms)	1.0 % (14 ms)	18 %	66 %
	Large	596	73-79	5500	75	71	0.3 % (71 ms)	0.8 % (78 ms)	28 %	86 %

Online auto-tuning: Very small workload



Cortex-A8

Cortex-A9