

## MPSoC Design Tools Research at ISS

Rainer Leupers  
Software for Systems on Silicon (SSS)  
RWTH Aachen University



Institute for Integrated Signal Processing Systems

### RWTH Aachen University background

- RWTH Aachen (established in 1870) is the leading technical university in Germany w.r.t. 3rd party funding
- 9 departments
- Approx. 30,000 students
- Engineering (electrical and mechanical) is top-ranked in virtually all recent surveys
- EE/IT department:
  - 27 professors
  - Approx. 3,000 students
- In 2007, RWTH Aachen has been officially granted „Elite University“ status in Germany’s Excellence Initiative competition

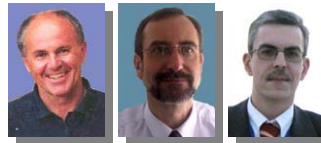
**RWTHAACHEN**  
UNIVERSITY



## ISS background

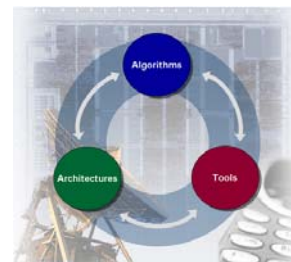
### ISS institute

- Part of RWTH Aachen EE+IT faculty
- Jointly managed by 3 professors (Meyr, Ascheid, Leupers)
- ~25 Ph.D. students, ~20 M.Sc. students + staff



### Focus on wireless communication + multimedia system design

- Interdisciplinary approach
- Basic research + tight industry cooperations
- Several EDA spin-off companies (e.g. LISATek)
- UMIC excellence cluster



© 2008 R. Leupers

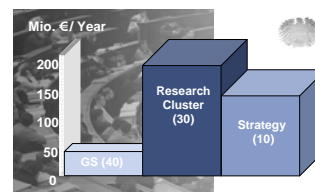
3



## The „Excellence Initiative“

### German government program

- 1,9 Bio. € total
- Funding period: 5 years
- Structure : 3 Funding lines
  - Graduate schools (1 M€ / year / topic)
  - Research cluster (6,5 M€ / year / cluster)
  - Strategy to develop top research (12.5 M€ / year / University)
- Thorough evaluation by renowned international researchers



### RWTH Aachen

- 1 Graduate school
  - Aachen Institute for Advanced Studies in Computational Engineering Sciences
- 3 Research clusters
  - Integrated Production Technology for High-Wage Countries  
Chairman: Christian Brecher (WZL)
  - Ultra high-speed Mobile Information and Communication (UMIC)**  
⇒ Chairman: Gerd Ascheid (ISS)
  - Tailor-Made Fuels from Biomass  
Chairman: Stefan Pischinger (VKA)



© 2008 R. Leupers

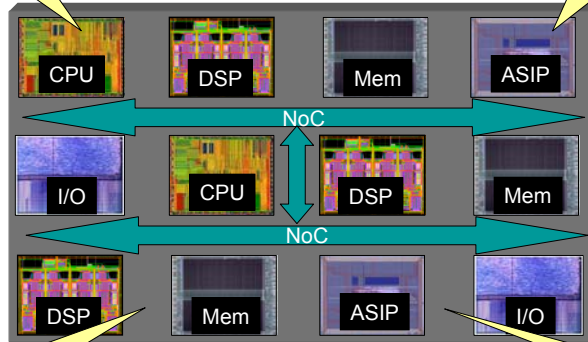
4



## MPSoC ESL tool challenges

C compilers  
• retargeting  
• code optimization

Design tools  
• profiling  
• ISA extensions  
• reconf. ASIP



Virtual platforms  
• MPSoC modeling  
• Fast hybrid simulation

MPSoC programming  
• C code partitioning  
• Task-to-PE mapping



© 2008 R. Leupers

5



## Overview

- ➔ ASIP design methodologies
- C compiler generation for ASIPs
- Automated processor customization
- Design flow for reconfigurable ASIPs
- MPSoC virtual platforms
- MPSoC programming
- Summary



© 2008 R. Leupers

6



## Customizable platforms

„As the performance of conventional microprocessors improves, they first meet and then exceed the requirements of most computing applications. Initially, performance is key. But eventually, other factors, like customization, become more important to the customer...“

[M.J. Bass, C.M. Christensen: The Future of the Microprocessor Business, IEEE Spectrum 2002]

$$\text{design budget} = (\text{semiconductor revenue}) \times (\% \text{ for R\&D})$$

growth  $\approx$  15%  $\approx$  10%

$$\# \text{ IC designs} = (\text{design budget}) / (\text{design cost per IC})$$

growth  $\approx$  15% growth  $\approx$  50-100%

[Keutzer]

- **Customizable programmable platforms**
- **application specific processors (ASIPs)**
  - **multi-processor system on chip (MPSoC)**

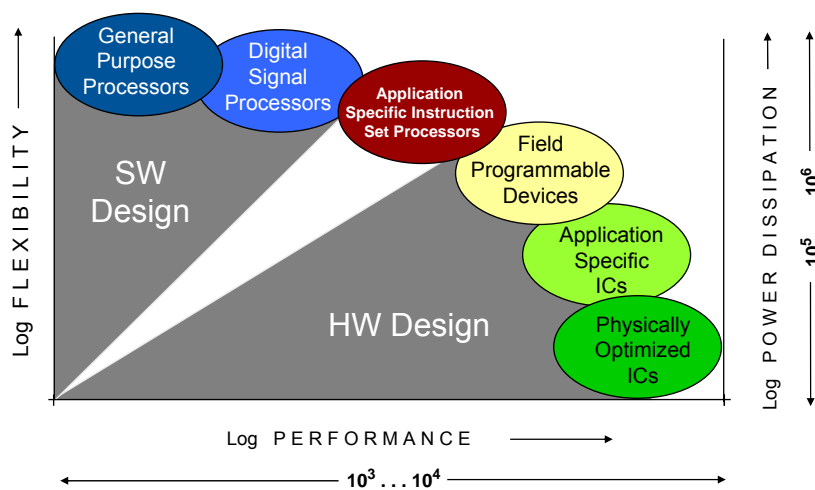


© 2008 R. Leupers

7



## Efficiency and flexibility



Source: T.Noll, RWTH Aachen



© 2008 R. Leupers

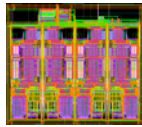
8



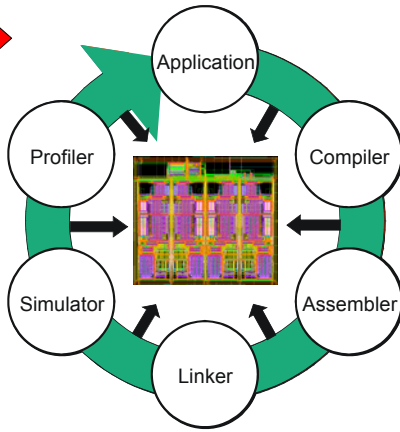
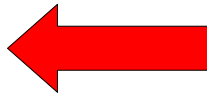
# ASIP architecture exploration



initial processor architecture



optimized processor architecture



# LISATek ASIP architecture exploration

- Unified processor model in LISA 2.0 architecture description language (ADL)
- Integrated processor development environment
- Automatic generation of:
  - SW tools
  - HW models
- Now available as CoWare Processor Designer



## The LISATek™ Solution

Automated Embedded Processor Design and Software Development Tool Generation

LISATek is an automated embedded processor design and optimization environment that automates the design process from hardware design to software development. It is the creation of processor-specific software development tools. LISATek's high degree of automation enables users to design and optimize processor architectures. Moreover, it generates software development tools for processors that have not been designed using LISATek's automated hardware design capability.

LISATek automatically accelerates the design of both custom and standard processors, including the application-specific instruction set processor (ASIP) that are increasingly essential to convergent systems-on-chip (SoC) architectures. LISATek is used to develop any or wide range of processor architectures, including DSP, DSPC, MMU, VLIW and supercalar.

LISATek's generated software development environment enables the commencement of application software development prior to silicon availability, thus reducing a common bottleneck in embedded system development. The key to LISATek's automation is its Language for Instruction Set Architecture, LISA 2.0. LISA 2.0 enables the creation of a

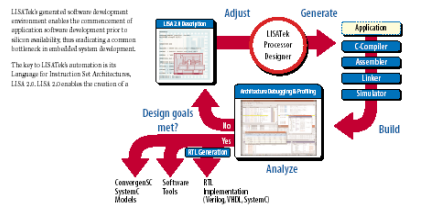
single "golden" processor model as the source for the automatic generation of the instruction set simulator (ISS), the complete suite of software development tools, and portable RTL code. The development tools, together with the customer-provided capabilities of the software simulator and debugger, enable rapid exploration of the processor architecture. LISATek enables the designer to optimize instruction set design, processor micro-architecture and memory systems, including caches.

LISATek's use of a single processor model source ensures the consistency of the ISS, software tools and RTL implementation, of simulating the hardware and debug effort necessitated by multiple independently created models with different levels of abstraction.

### HIGHLIGHTS

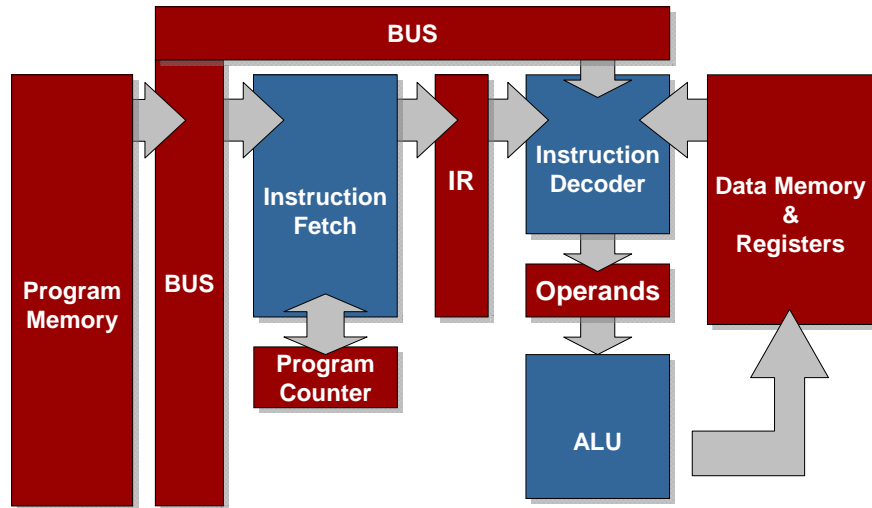
- Integrated design environment to allow processor design and software development independent, with no processor design experts required
- Single processor hardware design by users
- Hardware processor model can software tool generation—enabling processor tool development with LISA
- Access compatibility with software tools and RTL implementation
- Software development environment enables application software development prior to silicon availability

Operating at a high level of abstraction, LISATek not only eliminates the time and cost inherent in RTL-based processor design and manual tool development, but also enables processor design by non-experts.



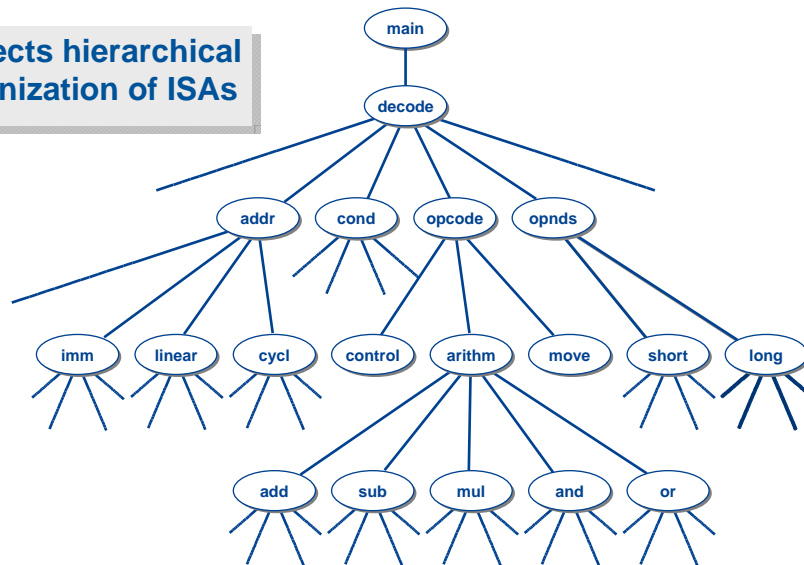


## Simplified Processor Overview



## LISA 2.0 operation hierarchy

Reflects hierarchical organization of ISAs



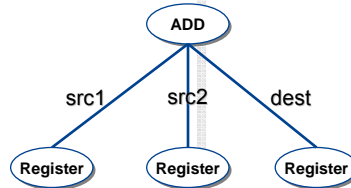
## LISA 2.0 operation example

```

OPERATION ADD
{
    DECLARE
    {
        GROUP src1, src2, dest = { Register }
    }
    CODING { 0b1011 src1 src2 dest }
    SYNTAX { "ADD" dest "," src1 "," src2 }
    BEHAVIOR { dest = src1 + src2; }
}

OPERATION Register
{
    DECLARE
    {
        LABEL index;
    }
    CODING { index }
    SYNTAX { "R" index }
    EXPRESSION { R[index] }
}
    
```

← C/C++ Code



## Exploration/debugger GUI

The screenshot displays the LISA GUI interface with several panels:

- Resource Utilization:** A table showing resource usage for various components like CPU, cache, and memory.
- Code Disassembly:** A window showing assembly instructions with their addresses and disassembled forms.
- Operation List:** A list of operations with columns for Name, Cabs, Cabs/Total, Cabs/Max, and Cabs/Min.
- Control Panel:** Buttons for simulation control (Run, Stop, etc.).

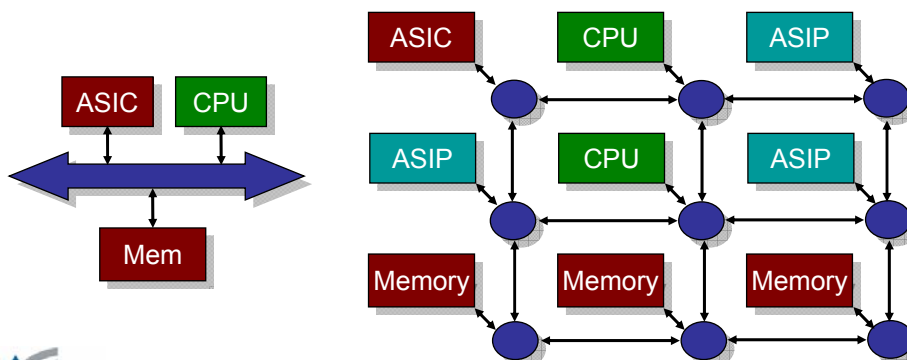
- Application simulation
- Debugging
- Profiling
- Resource utilization analysis
- Pipeline analysis
- Processor model debugging
- Memory hierarchy exploration
- Code coverage analysis
- ...

## Overview

- ASIP design methodologies
- ➔ C compiler generation for ASIPs
- Automated processor customization
- Design flow for reconfigurable ASIPs
- MPSoC virtual platforms
- MPSoC programming
- Summary

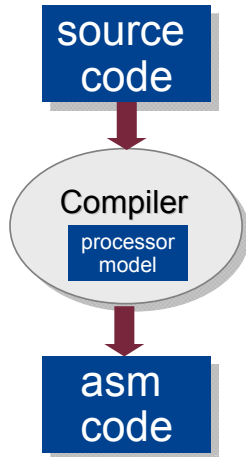
## Why care about C compilers?

- Trend towards heterogeneous multiprocessor systems-on-chip (MPSoC)
- Customized application specific instruction set processors (ASIPs) are key MPSoC components
- How to achieve efficient compiler support for ASIPs?

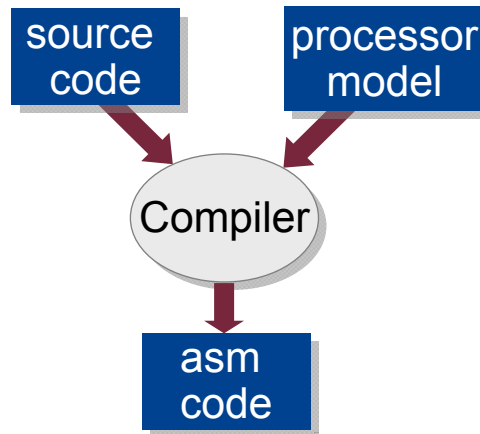


## Retargetable compilers

### Classical compiler



### Retargetable compiler

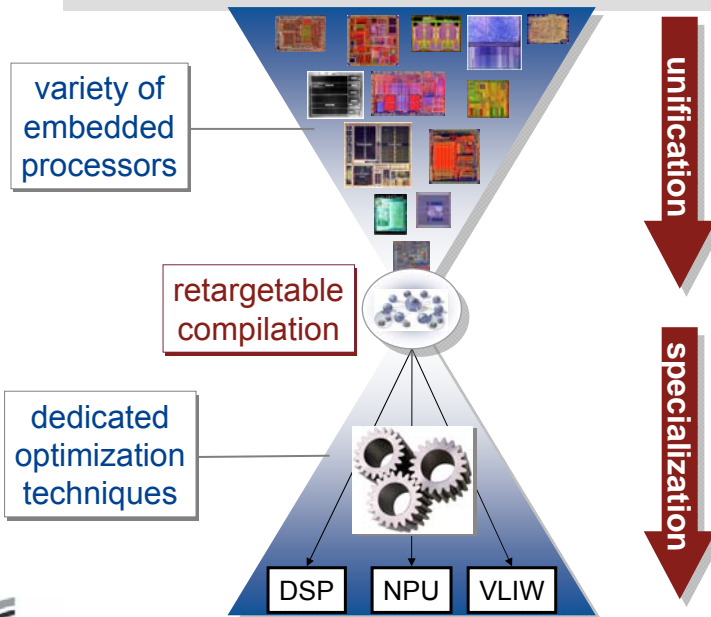


© 2008 R. Leupers

19



## Compiler flexibility/code quality trade-off



© 2008 R. Leupers

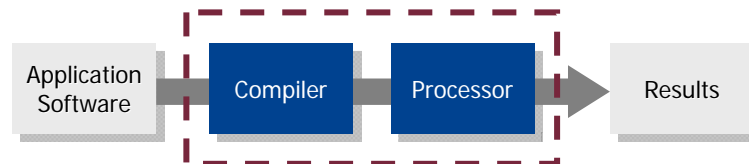
20



## C compiler in the exploration loop

### „Compiler/Architecture Co-Design“

- Efficient C-compilers **cannot** be designed for **ARBITRARY** architectures!



- Compiler and processor form a **UNIT** that needs to be optimized!
- “Compiler-friendliness“ needs to be taken into account during the architecture exploration!

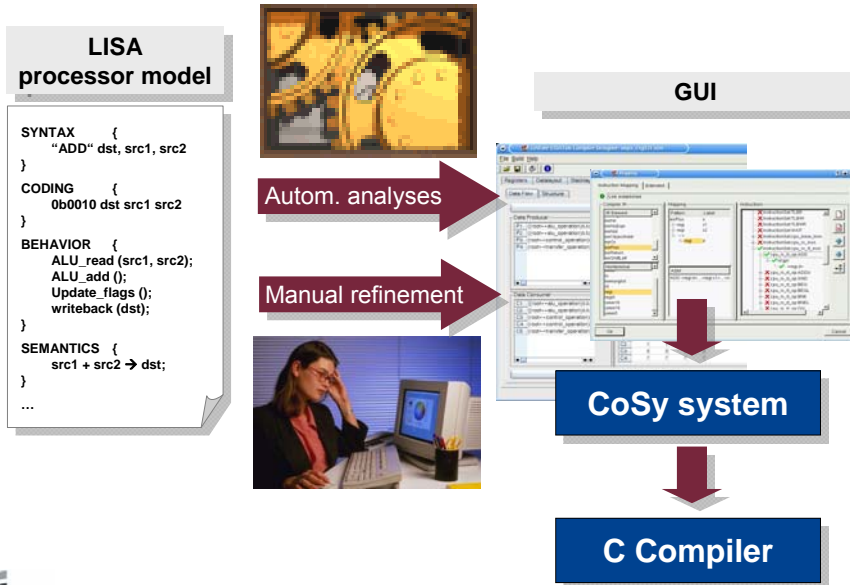
## CoSy compiler system (ACE)

- Universal retargetable C/C++ compiler
- Extensible intermediate representation (IR)
- Modular compiler organization
- Generator (BEG) for code selector, register allocator, scheduler
- Permits building working compilers quickly
- Facilities for adding target specific optimizations

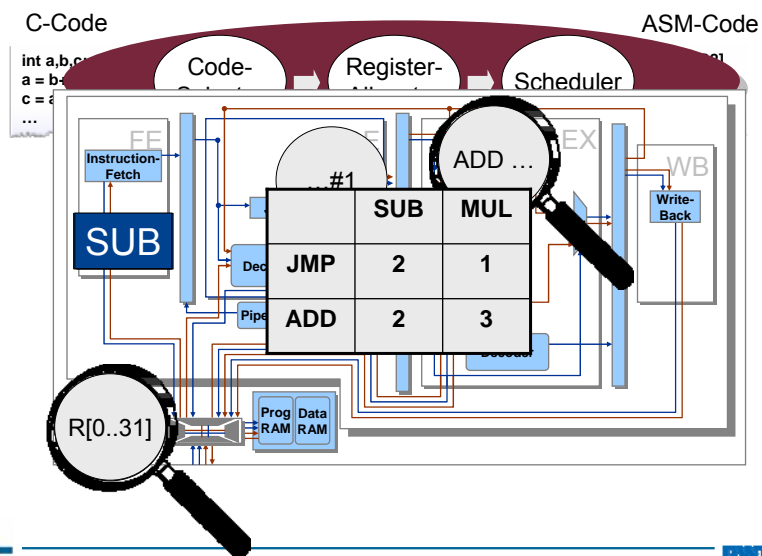


© ACE - Associated  
Compiler Experts

# LISATek C compiler generation

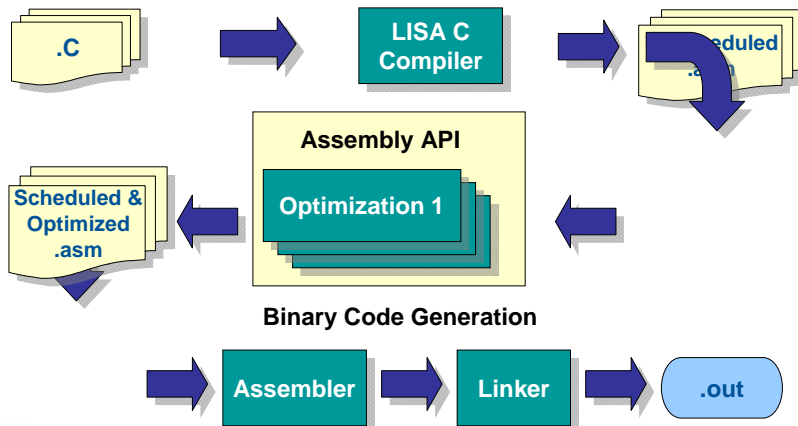


# LISATek compiler generation

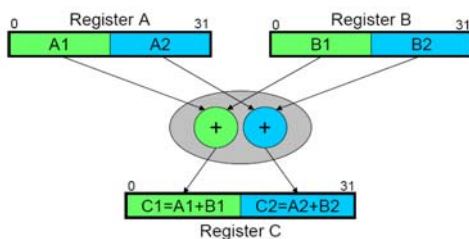


## Adding processor-specific code optimizations

- High-level (compiler IR)
  - Enabled by CoSy's engine concept
- Low-level (ASM):



## Retargetable code optimization engines



SIMD instructions

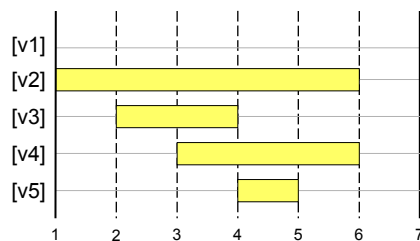
```

    if ( a > b )
    {
        // Then Block
    }
    else
    {
        //Else Block
    }
    
```

```

    CMP a,b
    JMPGT Then
    //Else Block
    JMP EndThen
    Then:
    //ThenBlock
    EndThen:
    CMPGT a,b,flag
    [ flag ] //ThenBlock
    [! flag ] //ElseBlock
    
```

conditional instructions,  
predicated execution



linear scan  
register allocation

## Overview

- ASIP design methodologies
- C compiler generation for ASIPs
- ➔ Automated processor customization
- Design flow for reconfigurable ASIPs
- MPSoC virtual platforms
- MPSoC programming
- Summary



## Generalized ASIP architecture design flow

Algorithm design  
(Matlab, SPW, ...)

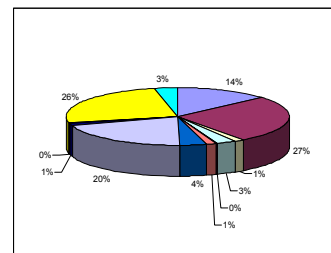
C code generation  
or implementation

initial  
architecture

architecture  
optimization

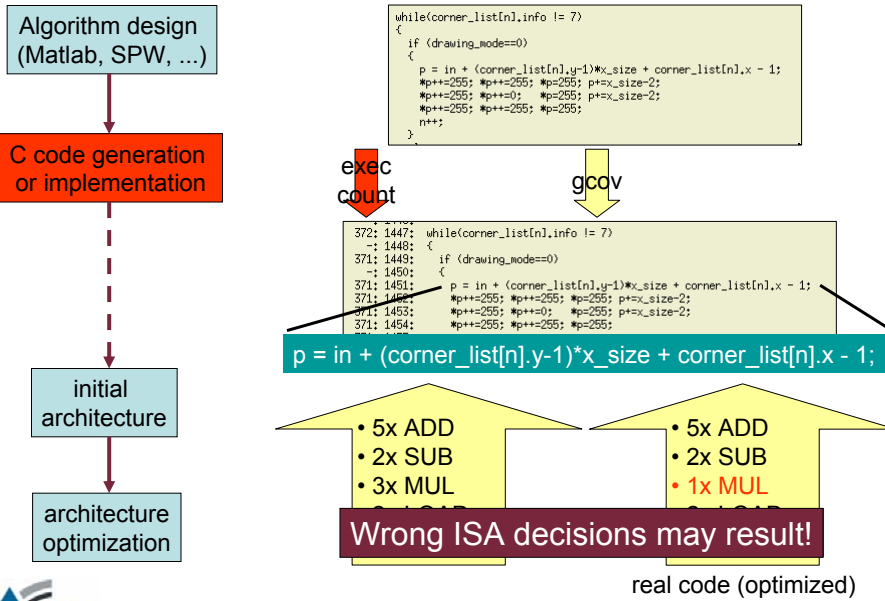


Profiling





# Do not neglect compiler optimizations

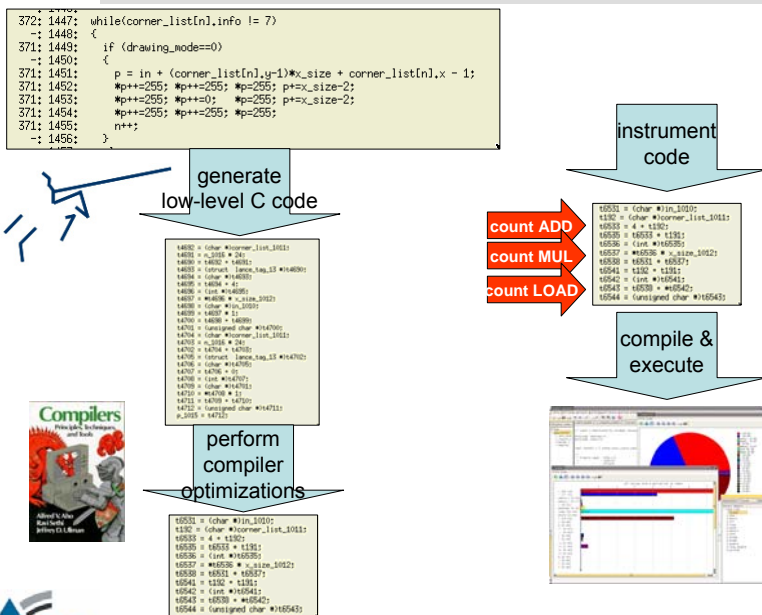


© 2008 R. Leupers

31



# μ-profiling approach



© 2008 R. Leupers

32

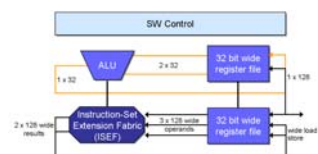
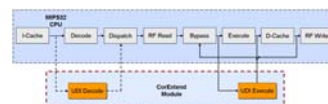
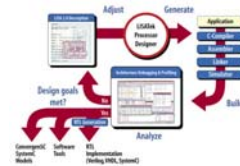


## Profiler features summary

	<i>C source level</i> (e.g. gprof)	<i>assembly level</i> (e.g. LISATek)	<i>Micro-profiler</i>
<i>primary application</i>	<b>Source code optimization</b>	<b>ISA and architecture optimization</b>	<b>ISA and architecture optimization</b>
<i>needs architectural details</i>	<b>No</b>	<b>Yes</b>	<b>No</b>
<i>speed</i>	<b>High</b>	<b>Low</b>	<b>Medium</b>
<i>Profiling granularity</i>	<b>coarse</b>	<b>fine</b>	<b>fine</b>

## ASIP design technologies revisited

- **ADL based (ASIP-from-scratch)**
  - E.g. LISATek, Target, Expression
  - Max. flexibility + efficiency, but significant design effort
- **Configurable processor cores**
  - E.g. Tensilica Xtensa, MIPS CorExtend, ARC Tangent
  - Pre-designed + pre-verified core
  - Efficiency via custom instruction set extensions (ISE)



## Processor customization tasks

### (1) Application code analysis

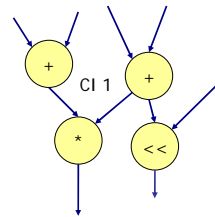
- At C source level
- Execution characteristics + hot spots

### (2) Custom instruction (CI) identification

- Optimal set of CIs to speed up hot spot under area and machine constraints
- Huge design space

### (3) CI implementation

- Interfacing with the configurable processor core
- Meet area and latency constraints



## Processor customization tasks (cont.)

### (4) SW adaptation and tools generation

- Rewrite application C code to utilize CIs
- Adapt C compiler, ISS to support CIs

### (5) HW architecture implementation

- Generate RTL HDL code for CIs
- Synthesize coprocessor for CIs

```

/*****
 * Original Code of Function F
 *****/
extern unsigned long S[4][256];
unsigned long F(unsigned long x)
{
    unsigned short a;
    unsigned short b;
    unsigned short c;
    unsigned short d;
    unsigned long y;

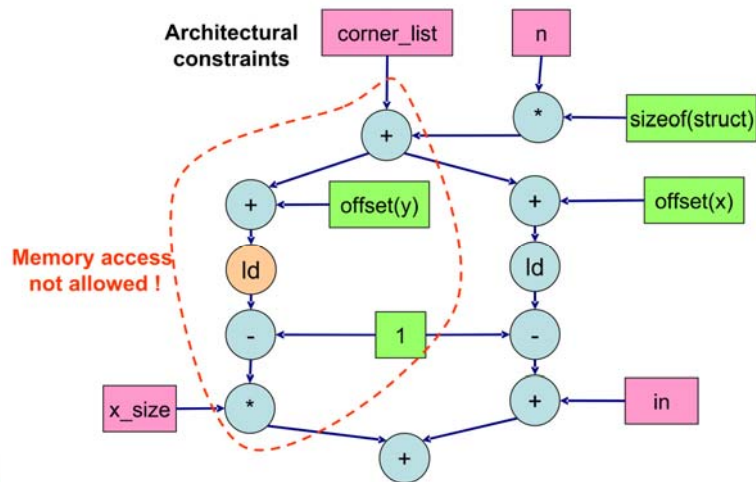
    d = x & 0x00FF;
    x >>= 8;
    c = x & 0x00FF;
    x >>= 8;
    b = x & 0x00FF;
    x >>= 8;
    a = x & 0x00FF;
    y = S[0][a] * S[1][b];
    y = y * S[2][c];
    y = y * S[3][d];

    return y;
}

/*****
 * Modified Code with CIs inserted
 *****/
extern unsigned long S[4][256];
unsigned long F(
    unsigned long x)
{
    int dummy_RS = 1, _dummy_RT = 1;
    unsigned long t58, t56, t54, t75;
    unsigned long t55, t529c1, t529c0;
    unsigned long t541c2, t553c3;
    // C Code for Block 1
    t58 = CI_1 ( x, 8);
    t56=CI_Unload_Internal (_dummy_RS, _dummy_RT, 3);
    t541c2 = t58 && 0xFF;
    t529c0 = t58 >> 8;
    t54 = CI_2 ( t529c0, t529c1);
    t75=CI_Unload_Internal (_dummy_RS, _dummy_RT, 5);
    t541c2 = t54 && 0xFF;
    t553c3 = t54 >> 8;
    t56 = CI_3 ( t553c3, t541c2);
    return t56;
}
    
```

## Custom instruction identification

- Primary goal: maximum application speedup under set of constraints



© 2008 R. Leupers



## Custom instruction identification approach

- Complex optimization problem, huge design space, back annotation required
- Optimal solution cannot be found within reasonable computation times
- Practical approach:
  - Interactive CI identification
  - Tools generate AT curve
  - For each point: use mix of ILP and heuristics to synthesize close-to-optimal CIs
  - User reviews and selects design points of interest for further fine-grained exploration
  - RTL generation and synthesis
- Short turnaround times



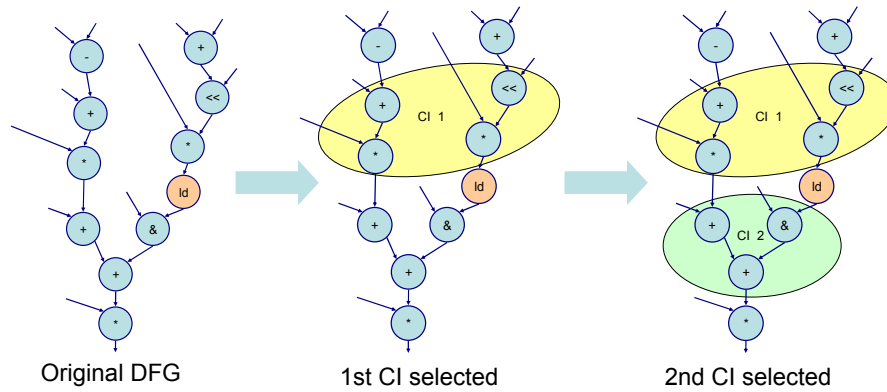
© 2008 R. Leupers

38



## Custom instruction synthesis

- Strategy: identify CIs locally optimally (ILP), but one after another (greedy)
- Library based speedup and cost estimation



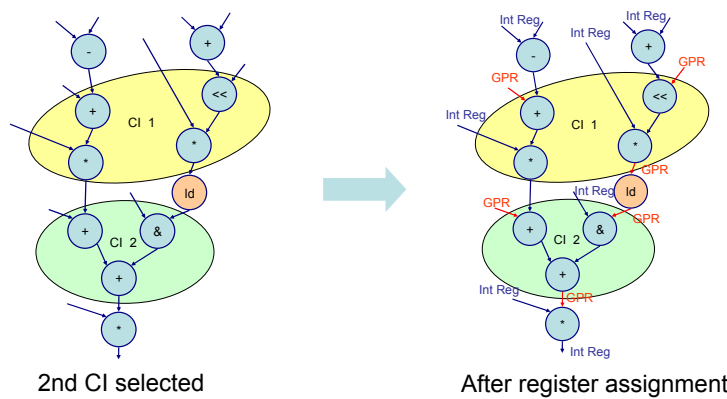
© 2008 R. Leupers

39



## Custom instruction synthesis

- Optimal (ILP) GPR and IR assignment to DFG edges for core-to-CI and CI-to-CI communication
- Heuristic sharing of IRs via lifetime analysis

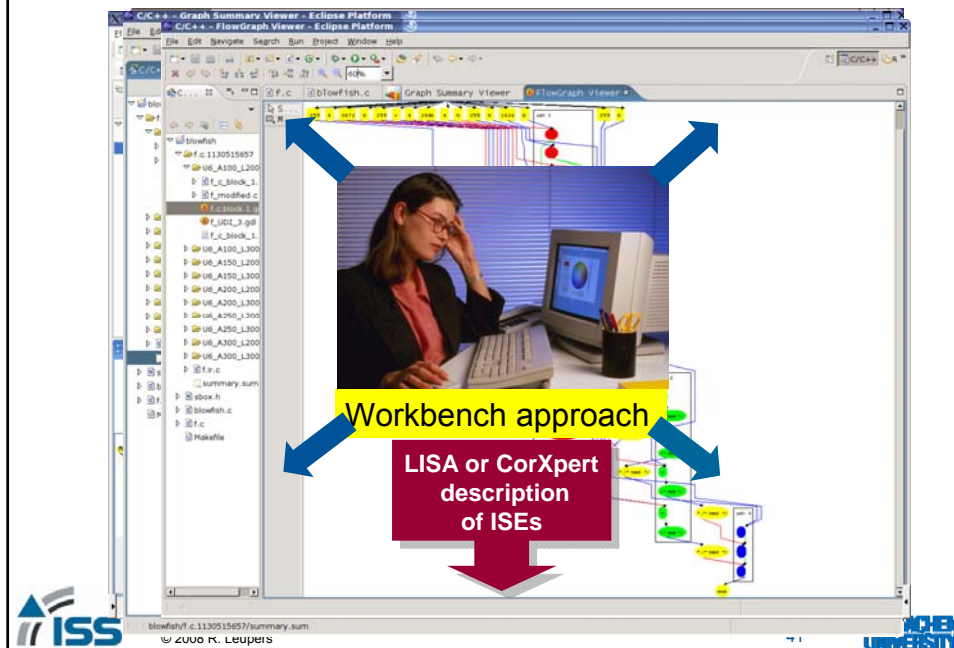


© 2008 R. Leupers

40



## ISE design tools



## Overview

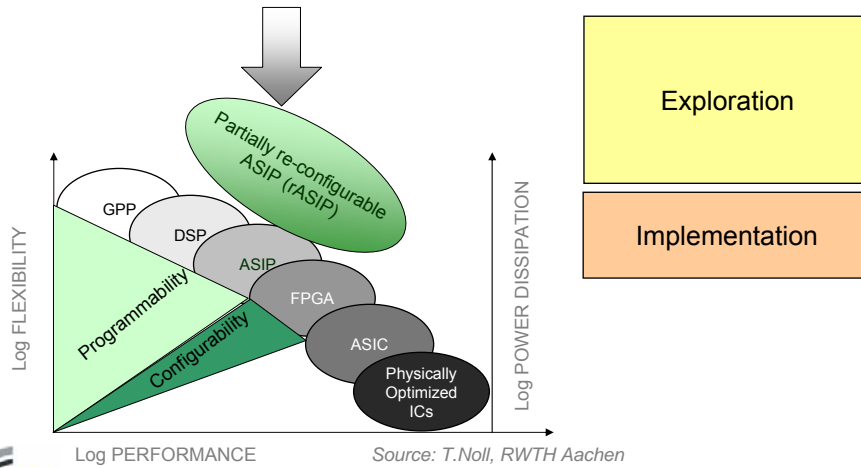
- ASIP design methodologies
- C compiler generation for ASIPs
- Automated processor customization
- ➔ Design flow for reconfigurable ASIPs
- MPSoC virtual platforms
- MPSoC programming
- Summary

## Reconfigurable ASIPs (rASIP)

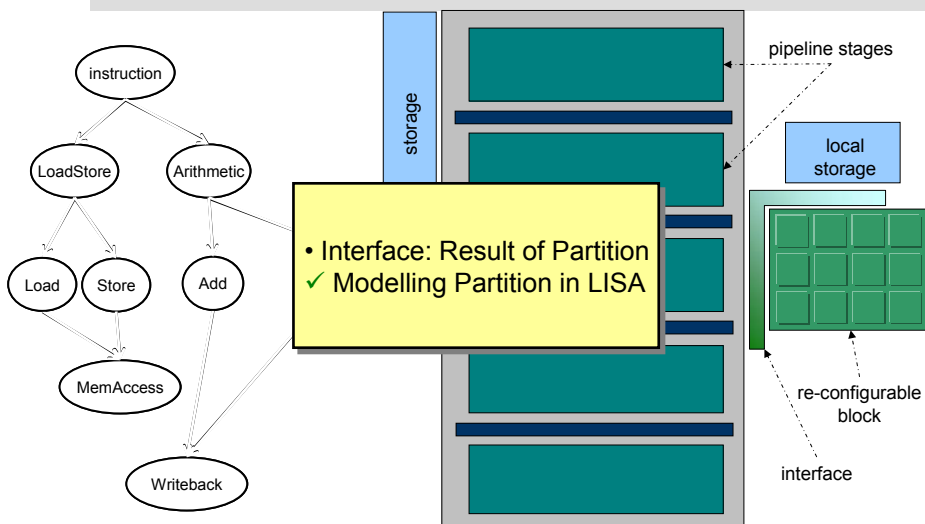
- Future trends: **Flexibility is the key**

- Increasing mask cost
- Fast-evolving applications

How to find the **right** architecture ?



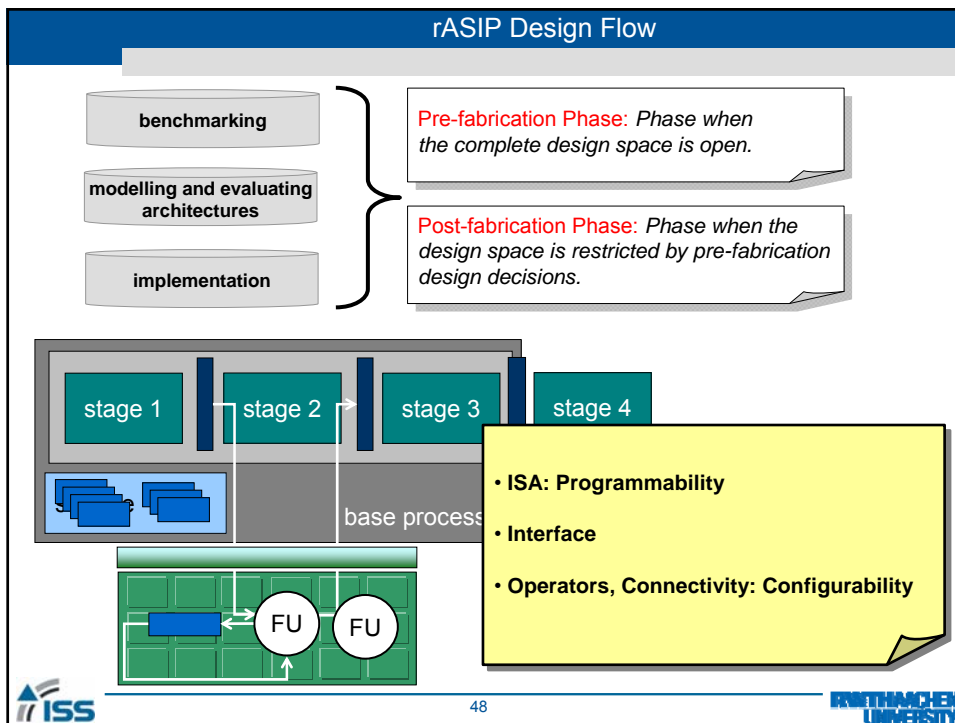
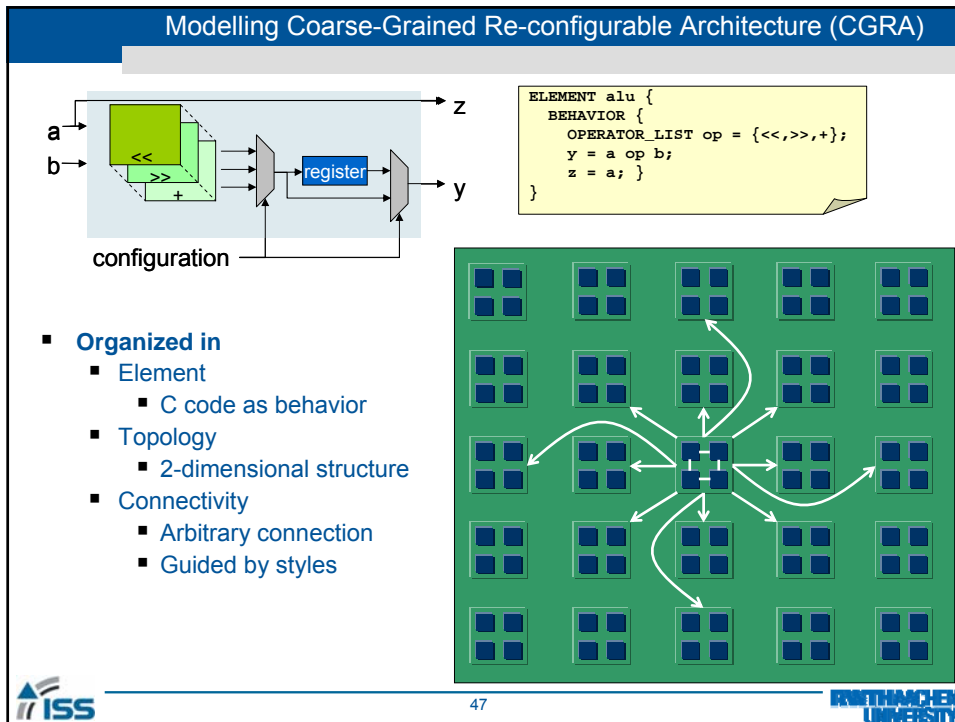
## Starting Point: Architecture Description Language (ADL) LISA

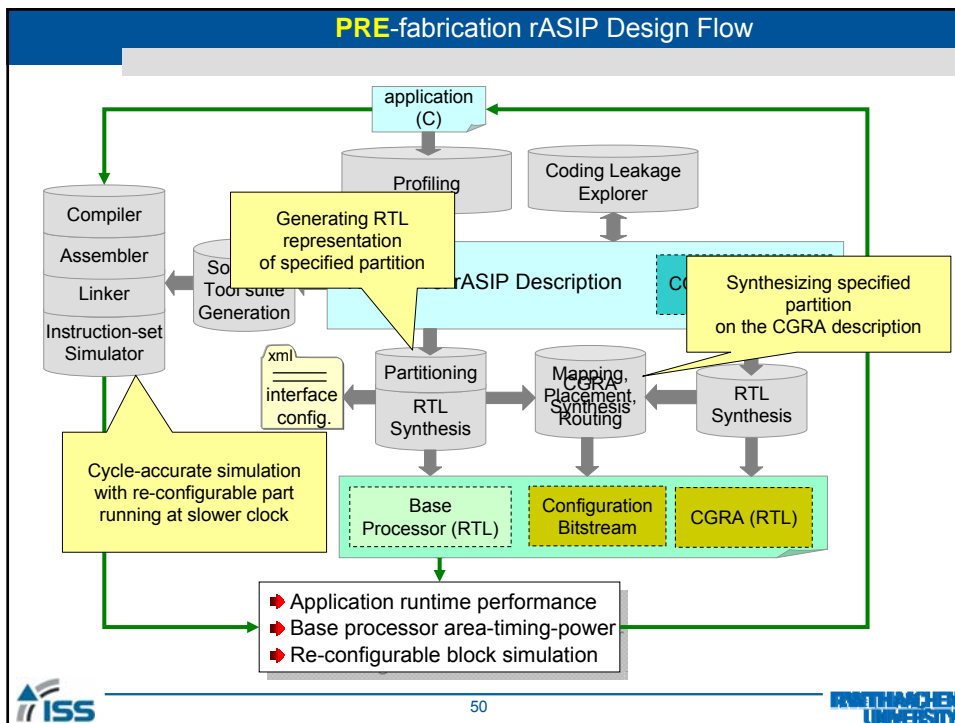
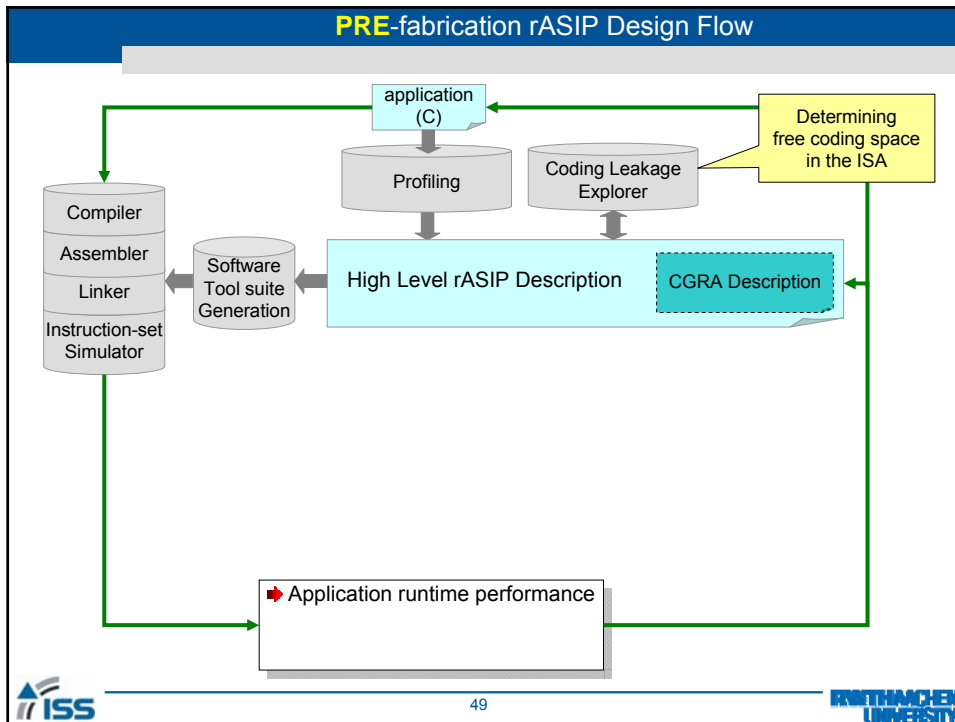


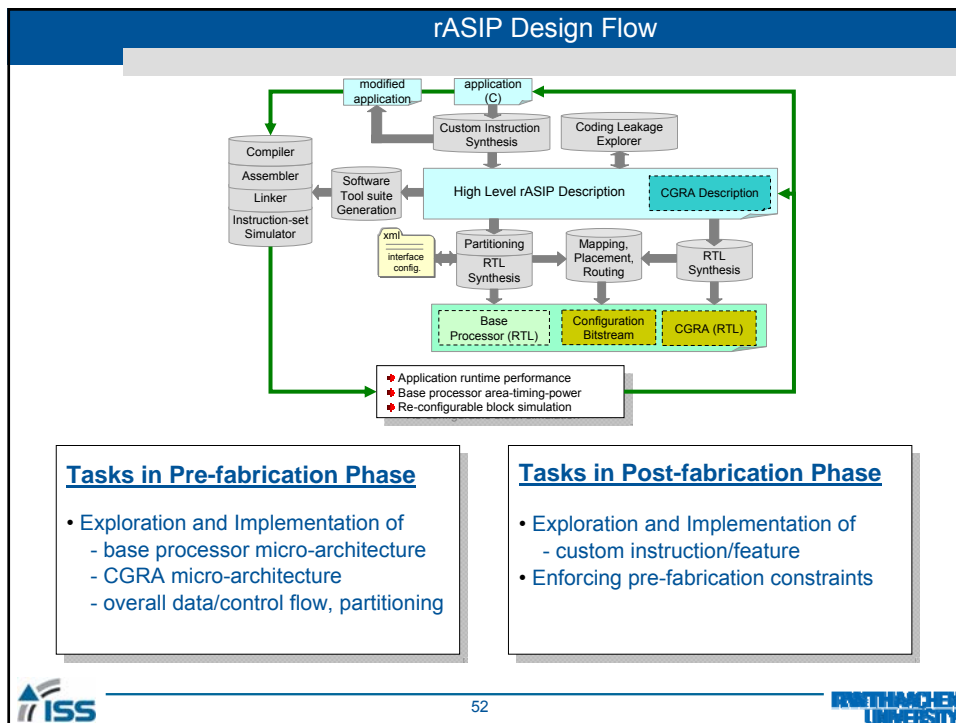
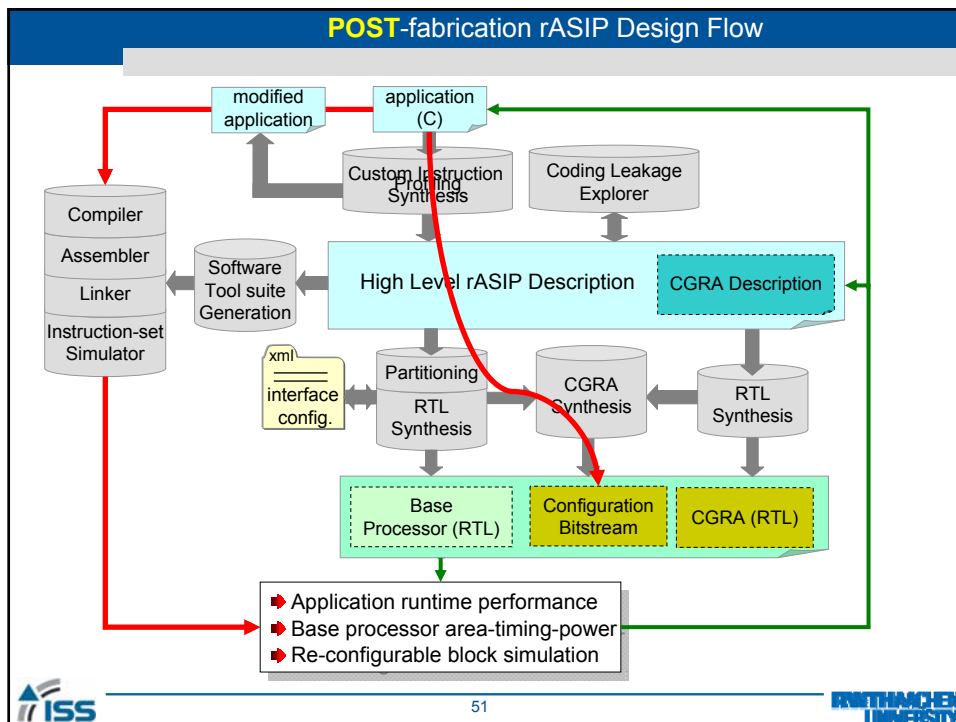
step 1: capturing the Instruction-Set Architecture (ISA)

step 2: structuring the ISA







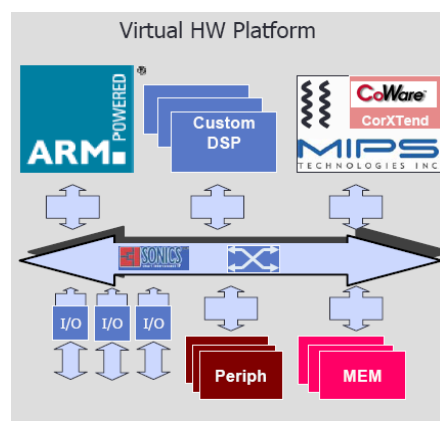


## Overview

- ASIP design methodologies
- C compiler generation for ASIPs
- Automated processor customization
- Design flow for reconfigurable ASIPs
- ➔ MPSoC virtual platforms
- MPSoC programming
- Summary

## What is a virtual platform?

- **A SW model of a HW SoC platform**
- **Enables...**
  - HW platform architecture exploration and optimization
  - SW development, debugging, and optimization
  - Concurrent HW/SW design („HW/SW codesign“)
- **Requirements**
  - High simulation speed
  - Speed/accuracy trade-off
  - Flexibility
  - Usability for non-HW-experts

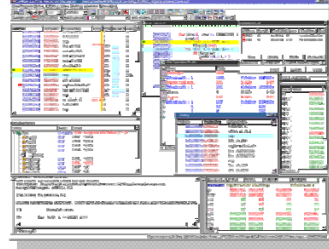


## Need for speed

- **Instruction set simulator (ISS) is at the heart of virtual platforms**

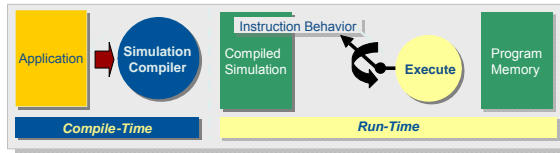
- **ISS speed evolution**

- Early interpretive: few KIPS
- Fast interpretive: ~100 KIPS
- Compiled: ~10 MIPS
- SW Sim. Cache: ~10 MIPS
- Binary translation: 50-100 MIPS



- **Challenge:**

- Instruction-accurate ISS technology has been pushed to its limits
- How to handle future many-core MPSoC?

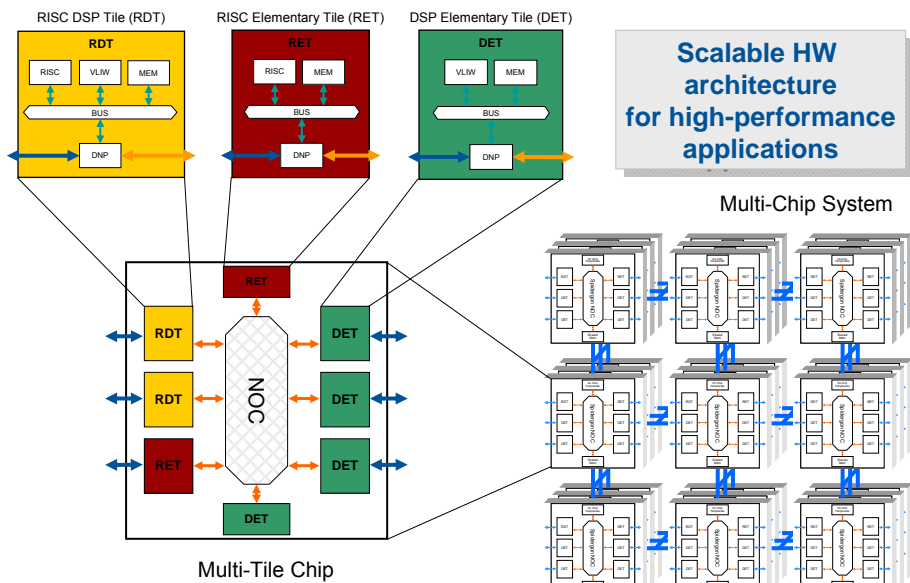


© 2008 R. Leupers

55



## Example: SHAPES platform



**Scalable HW architecture for high-performance applications**



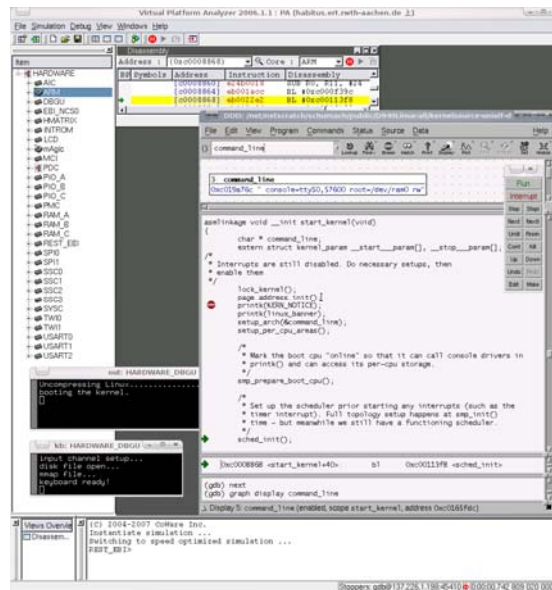
© 2008 R. Leupers

56



## Virtual SHAPES platform snapshot

- Register/memory inspection
- Breakpoints, watchpoints
- Program counter traces
- Log files
- Example snapshot: Linux OS booting on ARM with DDD debugger frontend inside VPA



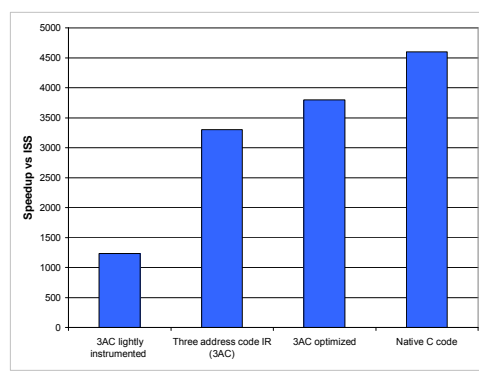
© 2008 R. Leupers

57



## What is left for accelerating simulation?

- Experiment with DES application and MIPS core
- Baseline 1: traditional IA ISS
  - Relatively slow
- Baseline 2: native C code execution
  - No target-specific information
- ~1000x speedup between ISS and native code theoretically available
- ~100x may be realistic
- Goal: hybrid simulator, toggling between
  - Fast native execution
  - Accurate target ASM simulation
  - Debugging enabled
  - Much faster than ISS



See also [Marques, Buels, et al., HPCA04]



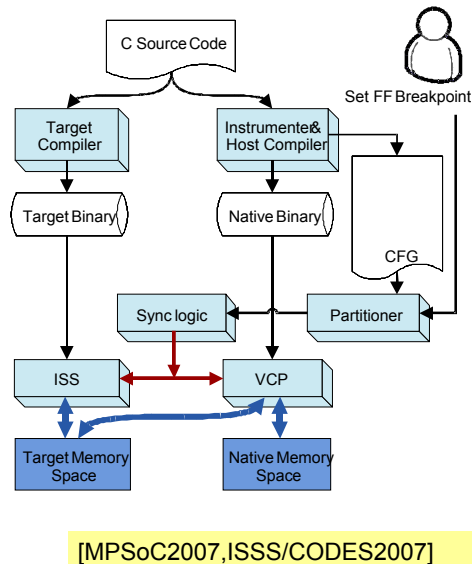
© 2008 R. Leupers

58



## HySim approach

- **Target ISS and C virtual co-processor (VCP) co-exist in simulation environment**
- **User has single graphical debug interface**
- **Use scenario 1:**
  - User sets breakpoint at program point of interest
  - HySim fast-forwards to that breakpoint
  - ISS continues as usual
- **Use scenario 2:**
  - FF breakpoint at end of program
  - Fast SW performance estimation
- **Major challenge:**
  - Synchronisation of processor state between ISS and VCP
  - Requires C code instrumentation

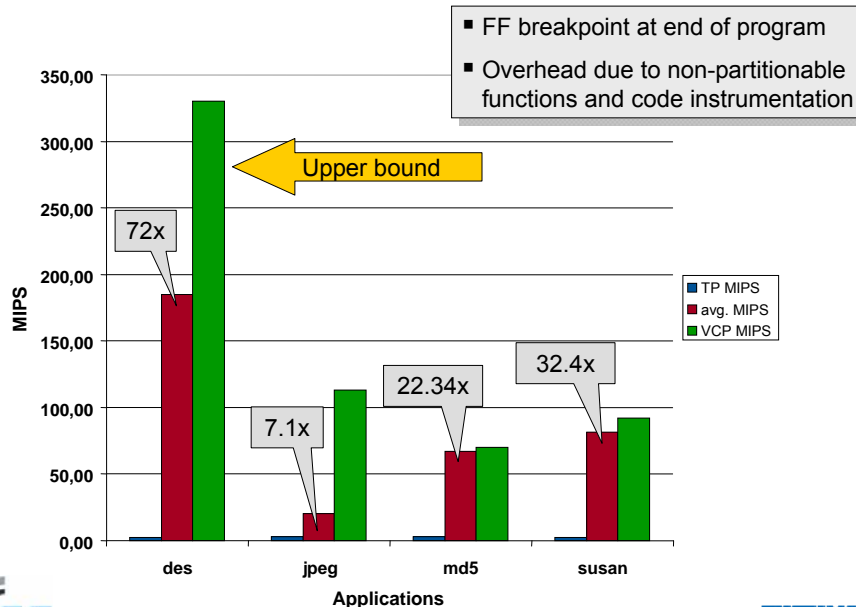


© 2008 R. Leupers

59



## Preliminary results (HySim for MIPS core)



© 2008 R. Leupers

60

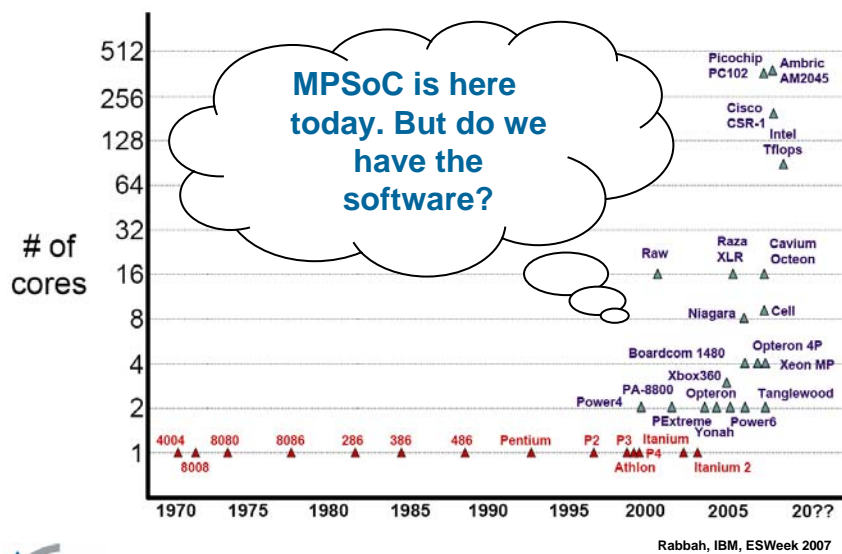


## Overview

- ASIP design methodologies
- C compiler generation for ASIPs
- Automated processor customization
- Design flow for reconfigurable ASIPs
- MPSoC virtual platforms
- ➔ MPSoC programming
- Summary



## MPSoC programming challenge



Rabbah, IBM, ESWeek 2007



## The von Neumann inheritance

- **Sequential programming of sequential machines**
  - Pascal, Modula-2, C, C++, Java, ...
- **Sequential programming of parallel machines?**
  - VLIW: handled by sophisticated compilers
  - SIMD: will be accomplished by compilers
  - HW multithreading: handled by OS and/or programmer
  - Does not scale to heterogeneous MPSoC with distributed control paths!
- **Parallel programming of parallel machines!**
  - „We need to move ... to parallel thinking and programming... We are standing at the very beginning... It's a huge area.“ (J. Gutknecht, ETH Zurich)
- **What to do in the meantime?**



© 2008 R. Leupers

63

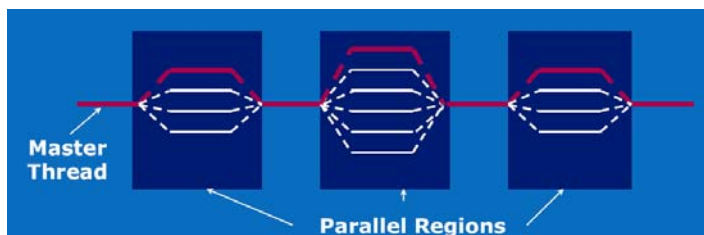


## New programming models

- E.g. OpenMP for Intel multi-core programming
- Enhancing standard languages (C/C++) with parallel programming pragmas
- **Problems:**
  - Need OpenMP enabled compiler
  - Tedious manual identification of potential parallelism
  - Not safe (e.g. thread race conditions, critical regions), need thread checker etc.

```

#pragma omp parallel
#pragma omp for
  for (i=0; i<N; i++){
    Do_Work(i);
  }
    
```



© 2008 R. Leupers

64



## Problem Statement

- **New parallel languages? C is hopeless?**

"85% of all embedded developers use C/C++. Any other language is a non-starter... I don't have much hope a new parallel language will get a foothold" (EETimes, 27.9.07)

D. Kleidermacher, CTO, Green Hills Software

- **Who takes Complexity of Parallelism?**

Hidden from programmers as far as possible

- **Push-button MPSoC programming framework?**

Workbench style preferred

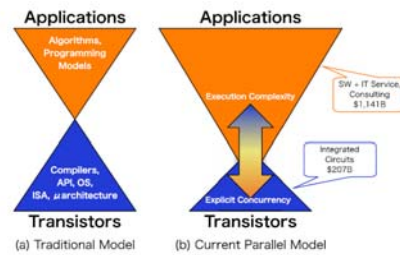
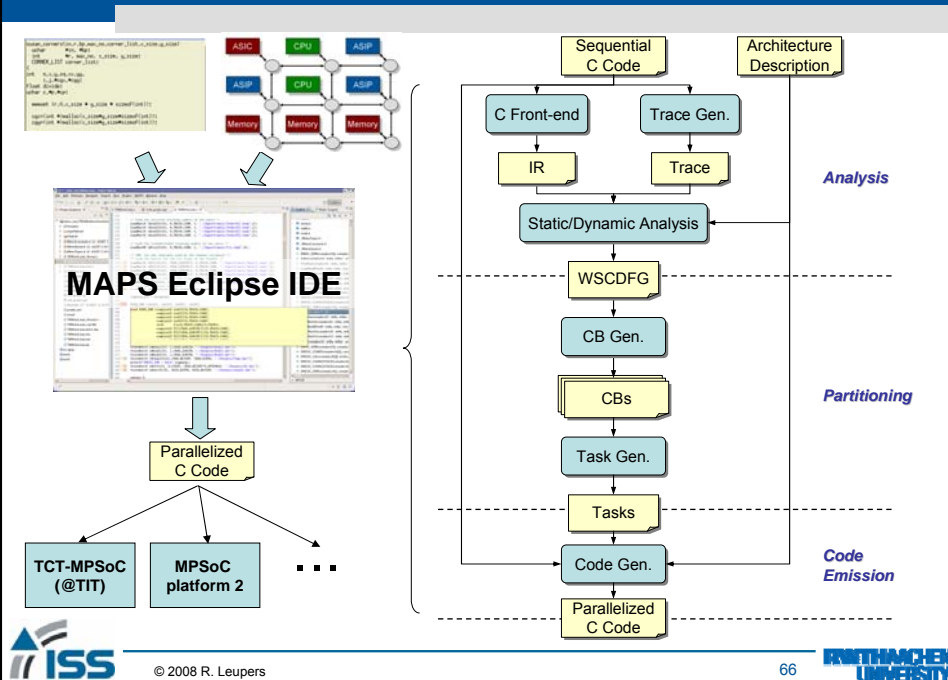


Figure 2. Cost and Complexity Exposed to Programmers

Hwu et al, DAC 2007

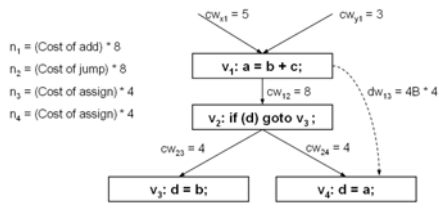


## MAPS Work-flow

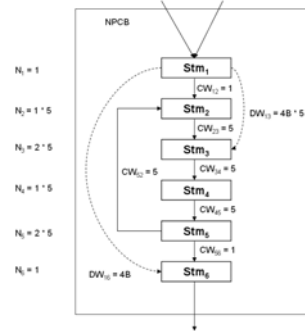


## Parallelism Discovery – CB

- Coarse-grained Task Parallelism in C
- Based on Compiler IR (Intermediate Representation)
- Suitable Granularity of Codes as Atomic code blocks  
→ **CB** (Coupled Block)
- CB Design Criteria
  - Tightly coupled by data-dependence
  - Schedulable



Weighted Statement Control/Data-flow Graph (WSCDFG)



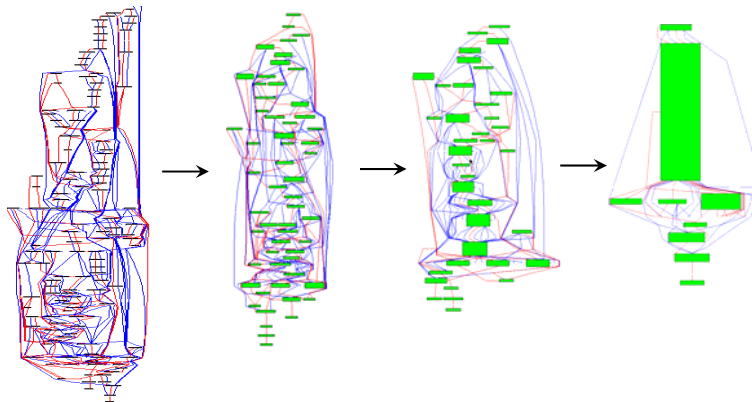
CB Example in WSCDFG

$$\bullet \forall v_i \in V', \frac{w_1 \cdot \sum_{c \in CW'} cw_{ij} + w_2 \cdot \sum_{d \in DW'} dw_{ij}}{D(v_i, V')} > T$$



## CB-to-task clustering

- CAHC: Constrained Agglomerative Hierarchical Clustering (inspired by *Agglomerative Clustering* in Data Mining)
- Enabled by advanced data flow analysis



# MAPS IDE snapshot

```

58 for (i = 0; i < 100; i++)
59 {
60     for (j = 0; j < 10; j++)
61     {
62         for (k = 0; k < 10; k++)
63         {
64             for (l = 0; l < 10; l++)
65             {
66                 // ...
67             }
68         }
69     }
70 }
71
72 for (i = 0; i < 10; i++)
73 {
74     for (j = 0; j < 10; j++)
75     {
76         for (k = 0; k < 10; k++)
77         {
78             // ...
79         }
80     }
81 }
82
83
84
85 for (j = 0; j < 10; j++)
86 {
87     for (k = 0; k < 10; k++)
88     {
89         for (l = 0; l < 10; l++)
90         {
91             // ...
92         }
93     }
94 }
95
96 for (j = 0; j < 10; j++)
97 {
98     for (k = 0; k < 10; k++)
99     {
100        // ...
101    }
102 }
103
104 for (k = 0; k < 10; k++)
105 {
106    // ...
107 }
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
    
```



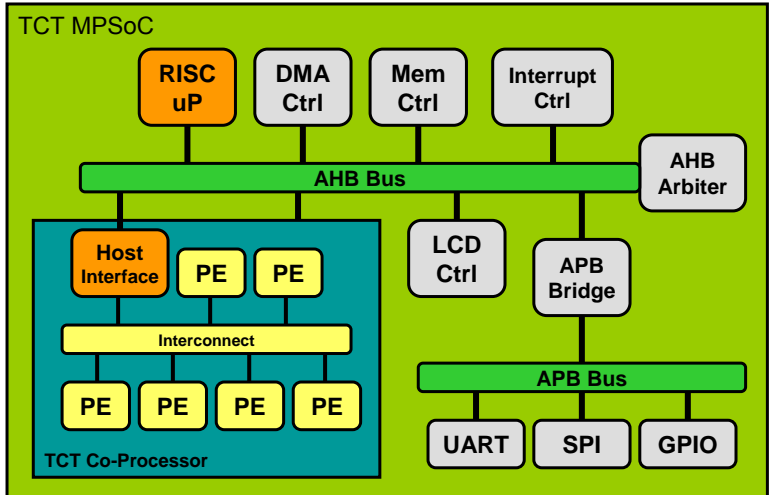
© 2008 R. Leupers

69



# TCT MPSoC

© Tokyo Institute of Technology



© 2008 R. Leupers

70



## TCT – Parallel Programming Model (Isshiki, TokyoTech)

- Drastically simple programming model

- Syntax:

**THREAD**(name) { statements }

thread scope header

thread scope

- Thread scope header** introduces a new *thread* labeled “name” in the program
- Thread scope** can have any compound statements in C, and also other thread scopes (*thread scopes can be nested*)
- Very simple coding style: seamless transition from sequential C codes
- Compiler support for automatic communication insertion

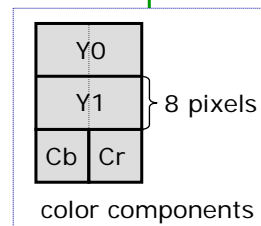


## TCT – JPEG Encoder Example

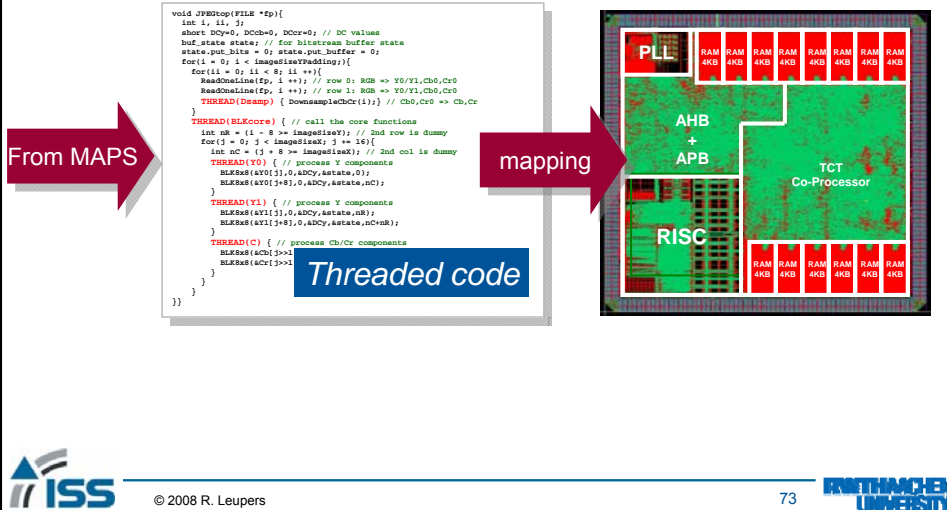
```

void JPEGtop(FILE *fp){
  int i, ii, j;
  short DCy=0, DCcb=0, DCCr=0; // DC values
  buf_state state; // for bitstream buffer state
  state.put_bits = 0; state.put_buffer = 0;
  for(i = 0; i < imageSizeYPadding;){
    for(ii = 0; ii < 8; ii ++){
      ReadOneLine(fp, i ++); // row 0: RGB => Y0/Y1,Cb0,Cr0
      ReadOneLine(fp, i ++); // row 1: RGB => Y0/Y1,Cb0,Cr0
      THREAD(Dsamp) { DownsampleCbCr(i); } // Cb0,Cr0 => Cb,Cr
    }
    THREAD(BLKcore) { // call the core functions
      int nR = (i - 8 >= imageSizeY); // 2nd row is dummy
      for(j = 0; j < imageSizeX; j += 16){
        int nC = (j + 8 >= imageSizeX); // 2nd col is dummy
        THREAD(Y0) { // process Y components
          BLK8x8(&Y0[j],0,&DCy,&state,0);
          BLK8x8(&Y0[j+8],0,&DCy,&state,nC);
        }
        THREAD(Y1) { // process Y components
          BLK8x8(&Y1[j],0,&DCy,&state,nR);
          BLK8x8(&Y1[j+8],0,&DCy,&state,nC+nR);
        }
        THREAD(C) { // process Cb/Cr components
          BLK8x8(&Cb[j>>1],1,&DCcb,&state,0);
          BLK8x8(&Cr[j>>1],1,&DCCr,&state,0);
        }
      }
    }
  }
}
    
```

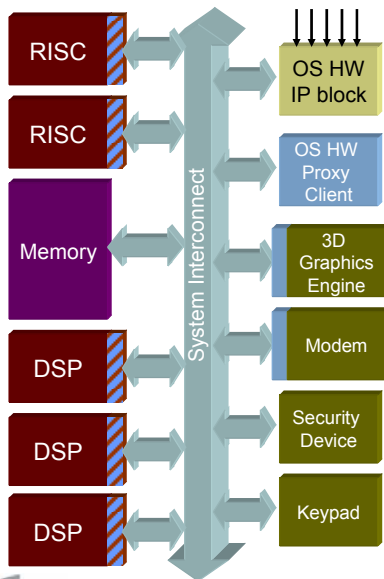
thread scopes



## MAPS-TCT backend



## Towards HW supported MPSoC multiprocessing



- Tasks need to be synchronized and need to communicate
- In general-purpose computing platforms: OS/RTOS supported
- Embedded MPSoC: OS overhead (performance/energy) is huge
- Research on supporting OS functionality with special HW block
- Must be configurable, programmable
- ASIP implementation ideal!

## References

- R. Leupers: *Code Optimization Techniques for Embedded Processors - Methods, Algorithms, and Tools*, Kluwer, 2000
- R. Leupers, P. Marwedel: *Retargetable Compiler Technology for Embedded Systems - Tools and Applications*, Kluwer, 2001
- A. Hoffmann, H. Meyr, R. Leupers: *Architecture Exploration for Embedded Processors with LISA*, Kluwer, 2002
- C. Rowen, S. Leibson: *Engineering the Complex SoC: Fast, Flexible Design with Configurable Processors*, Prentice Hall, 2004
- M. Gries, K. Keutzer, et al.: *Building ASIPs: The Mescal Methodology*, Springer, 2005
- P. lenne, R. Leupers (eds.): *Customizable and Configurable Embedded Processor Cores*, Morgan Kaufmann, 2006



© 2008 R. Leupers

75



Institute for  
Integrated Signal  
Processing Systems

**Thank you !**



**[www.iss.rwth-aachen.de](http://www.iss.rwth-aachen.de)**



Institute for Integrated Signal Processing Systems