SW Synthesis, Code Generation and Timing Analysis

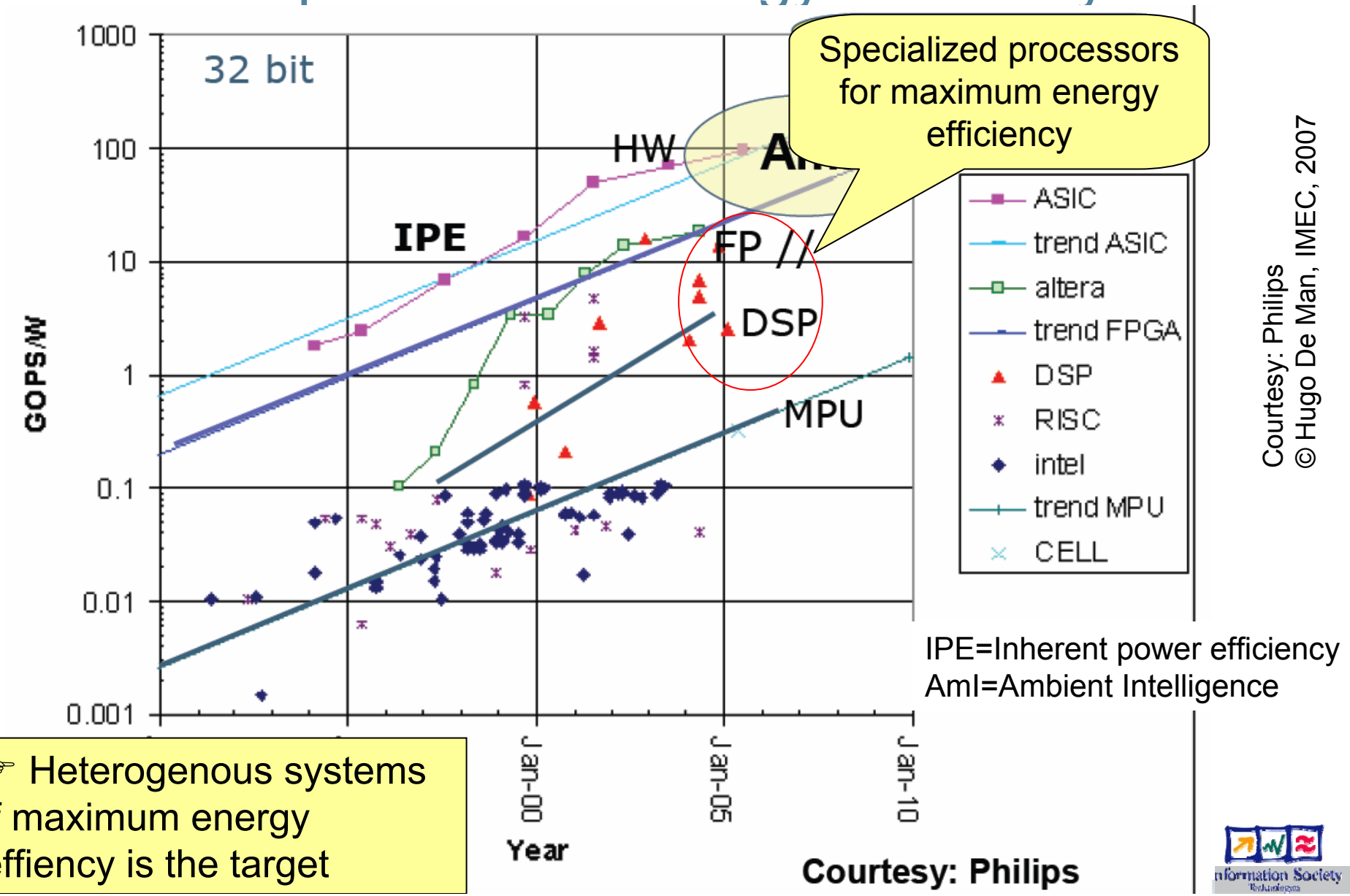
Cluster internal meeting

Jan. 29, 2008

Paris

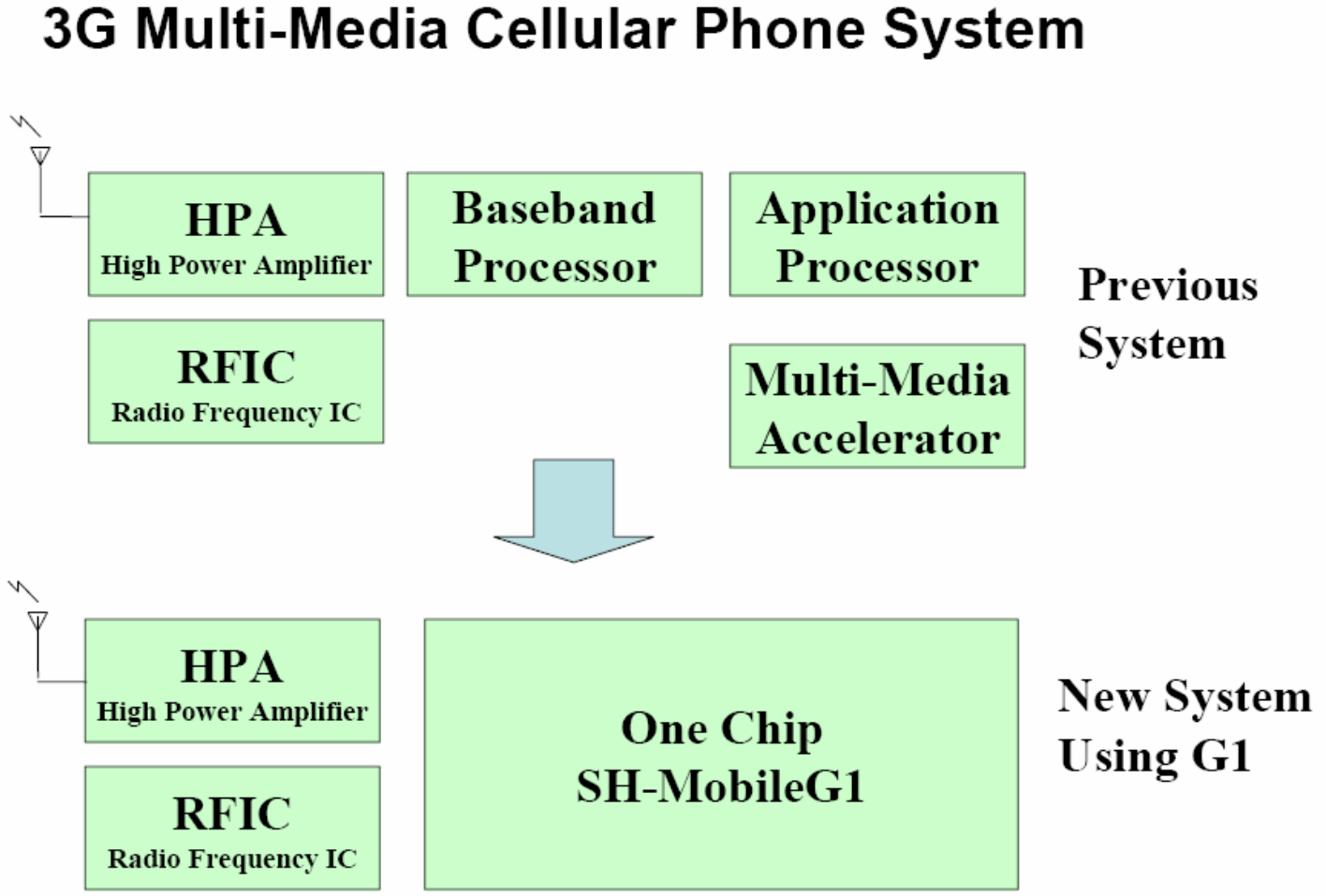
Peter Marwedel

Importance of Energy Efficiency



☞ Heterogenous systems if maximum energy effiency is the target

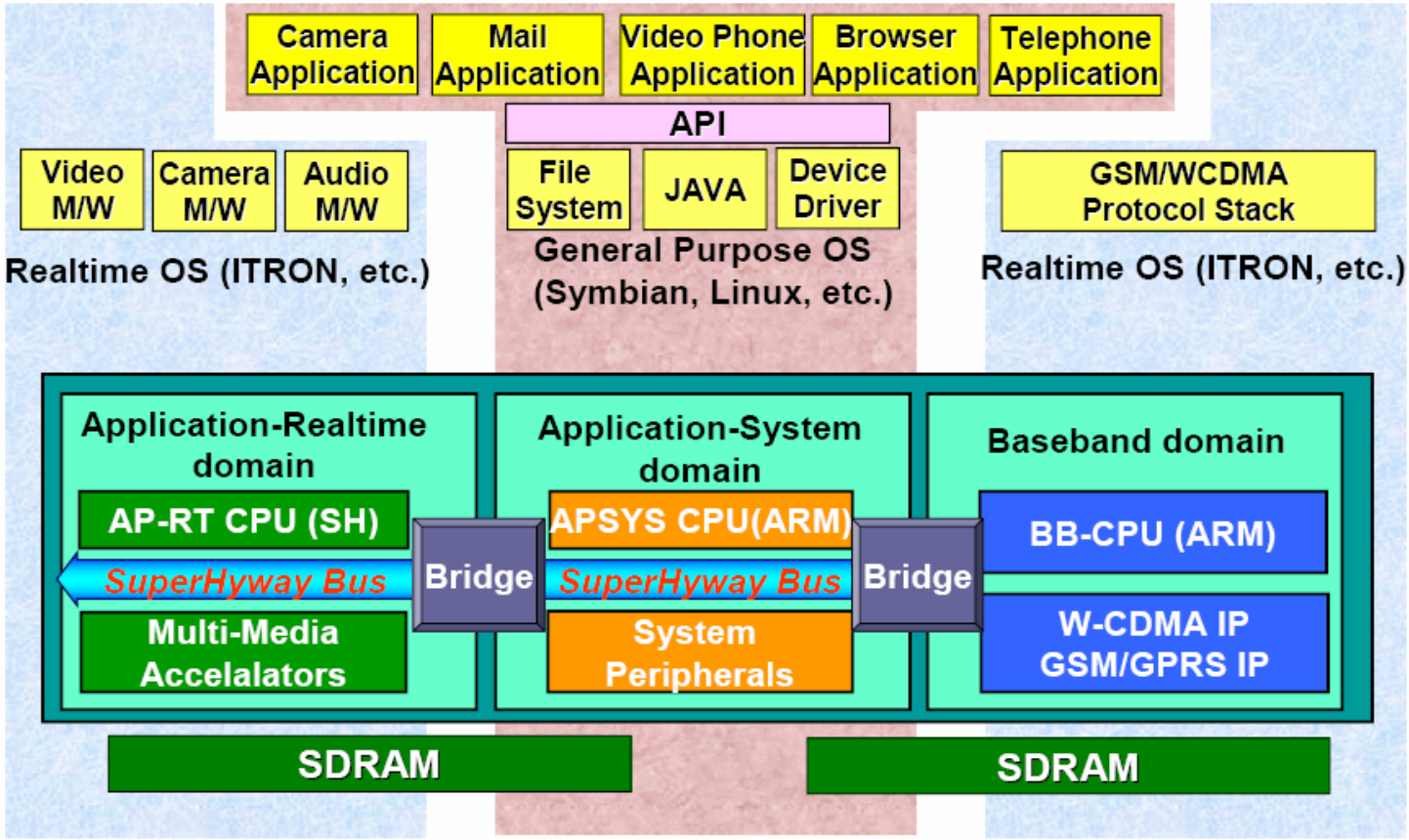
Trend: multiprocessor systems-on-a-chip (MPSoCs)



.http://www.mpsoc-forum.org/2007/slides/Hattori.pdf

Multiprocessor systems-on-a-chip (MPSoCs) (2)

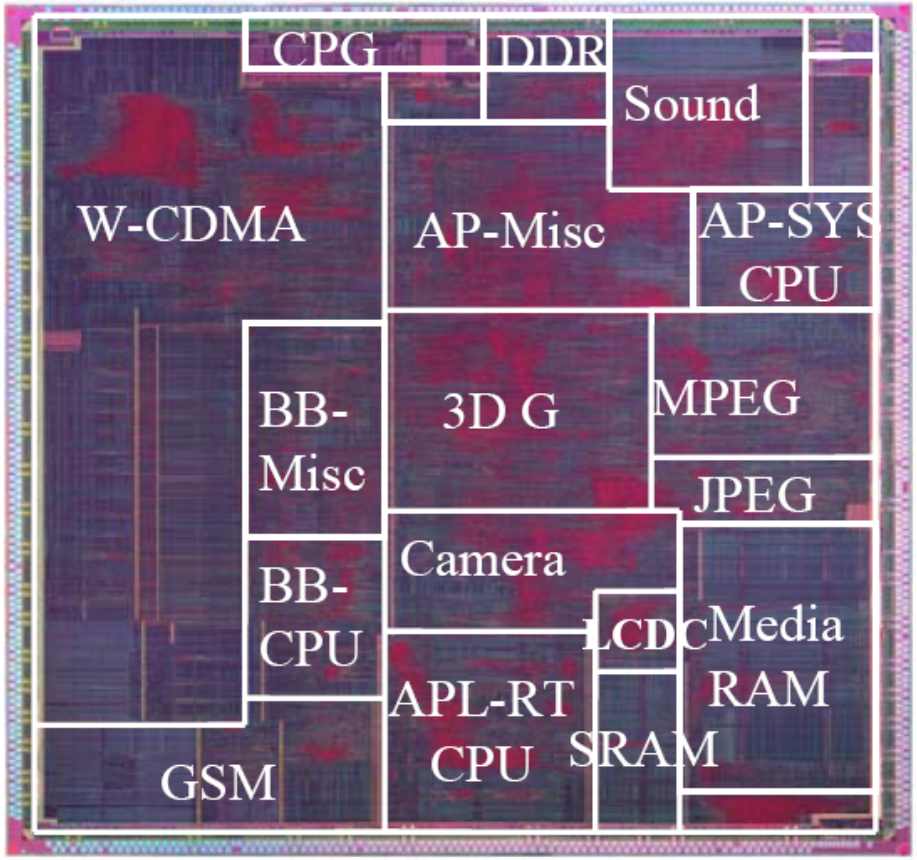
A Sample of System Architecture using G1



http://www.mpsoc-forum.org/2007/slides/Hattori.pdf

Multiprocessor systems-on-a-chip (MPSoCs) (3)

SH-MobileG1: Chip Overview

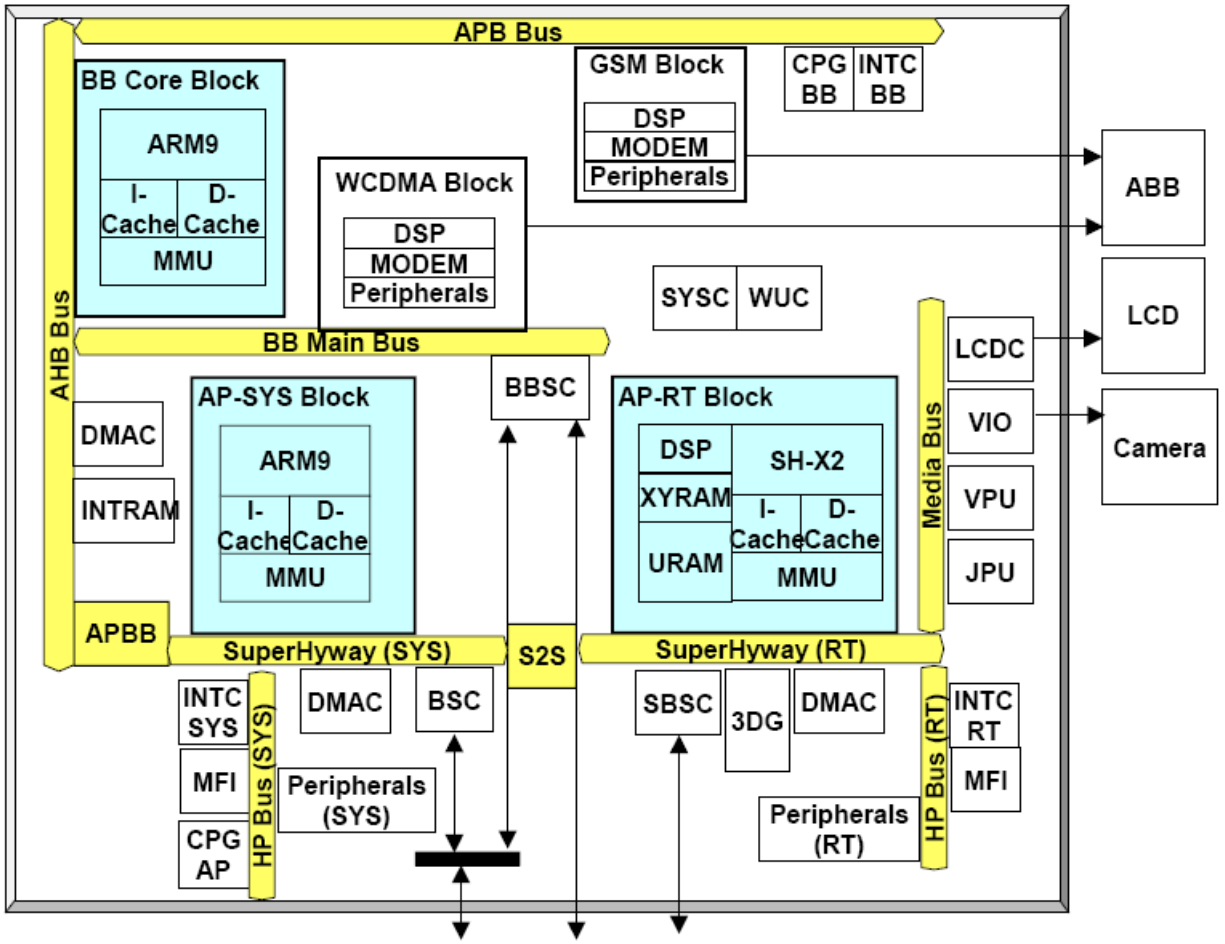


Die size	11.15mm x 11.15mm
Process	90nm LP 8M(7Cu+1Al) CMOS dual-Vth
Supply voltage	1.2V(internal), 1.8/2.5/3.3V(I/O)
# of TRs, gate, memory	181M TRs, 13.5M Gate 20.2 Mbit mem

http://www.mpsoc-forum.org/2007/slides/Hattori.pdf

Multiprocessor systems-on-a-chip (MPSoCs) (4)

G1 Module Diagram

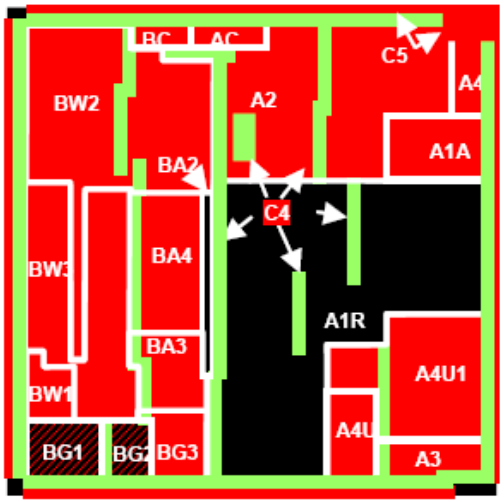


http://www.mpsoc-forum.org/2007/slides/Hattori.pdf

Multiprocessor systems-on-a-chip (MPSoCs) (5)

Leakage Current in Usage Scenes

(2)Telephony (W-CDMA)



Power on

Power off

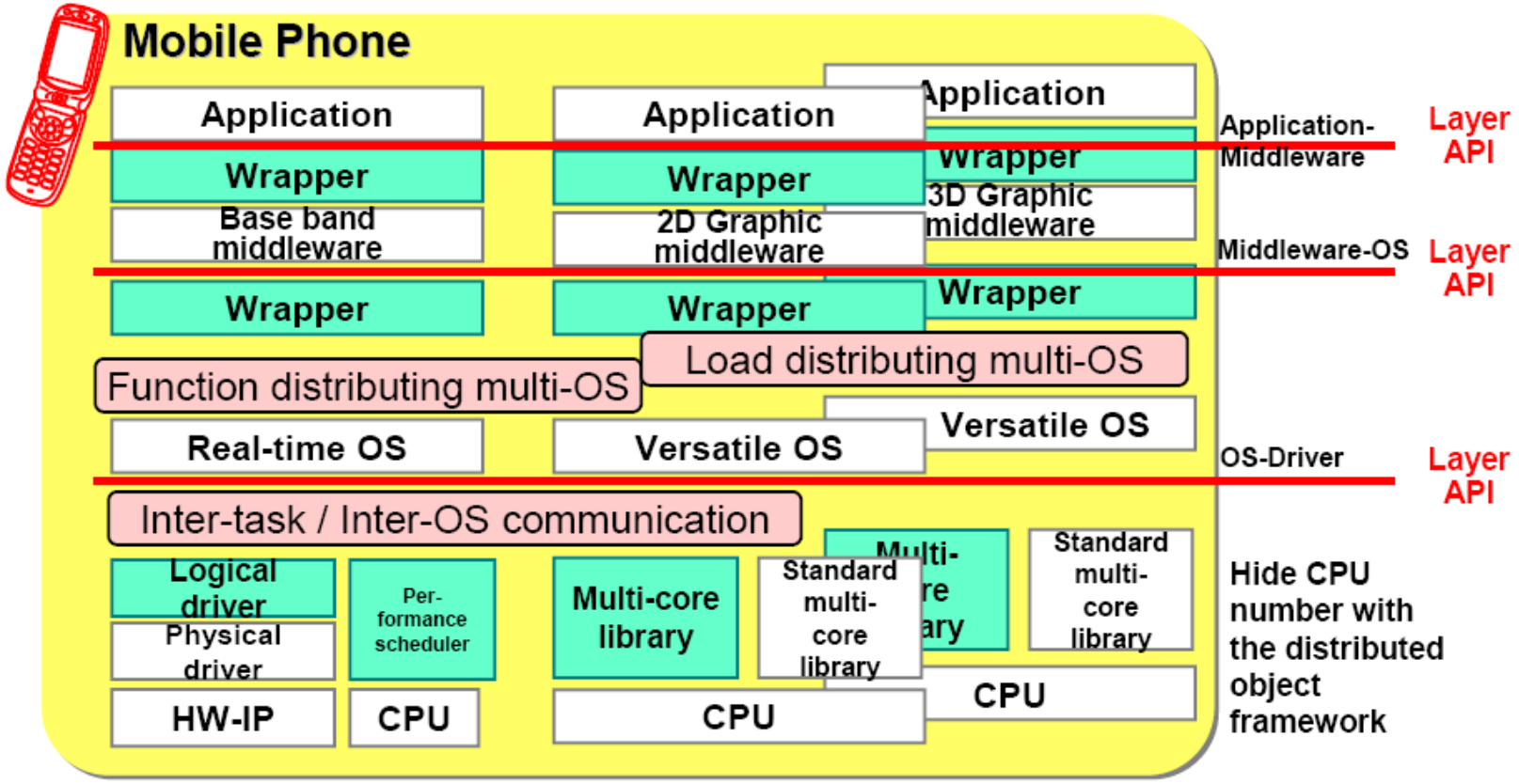
Baseband part	Control	ON
	W-CDMA	ON
	GSM	ON / OFF
Application part	System-domain	ON
	Realtime-domain	OFF
Measured Leakage Current (@ Room Temp, 1.2V)		407 µA

.http://www.mpsoc-forum.org/2007/slides/Hattori.pdf

Multiprocessor systems-on-a-chip (MPSoCs) (6)

EXREAL Platform™ Software Interconnect

- Promote reuse of software assets through Wrapper and standardized layer API
- Control operating frequency and power on/off through the performance scheduler

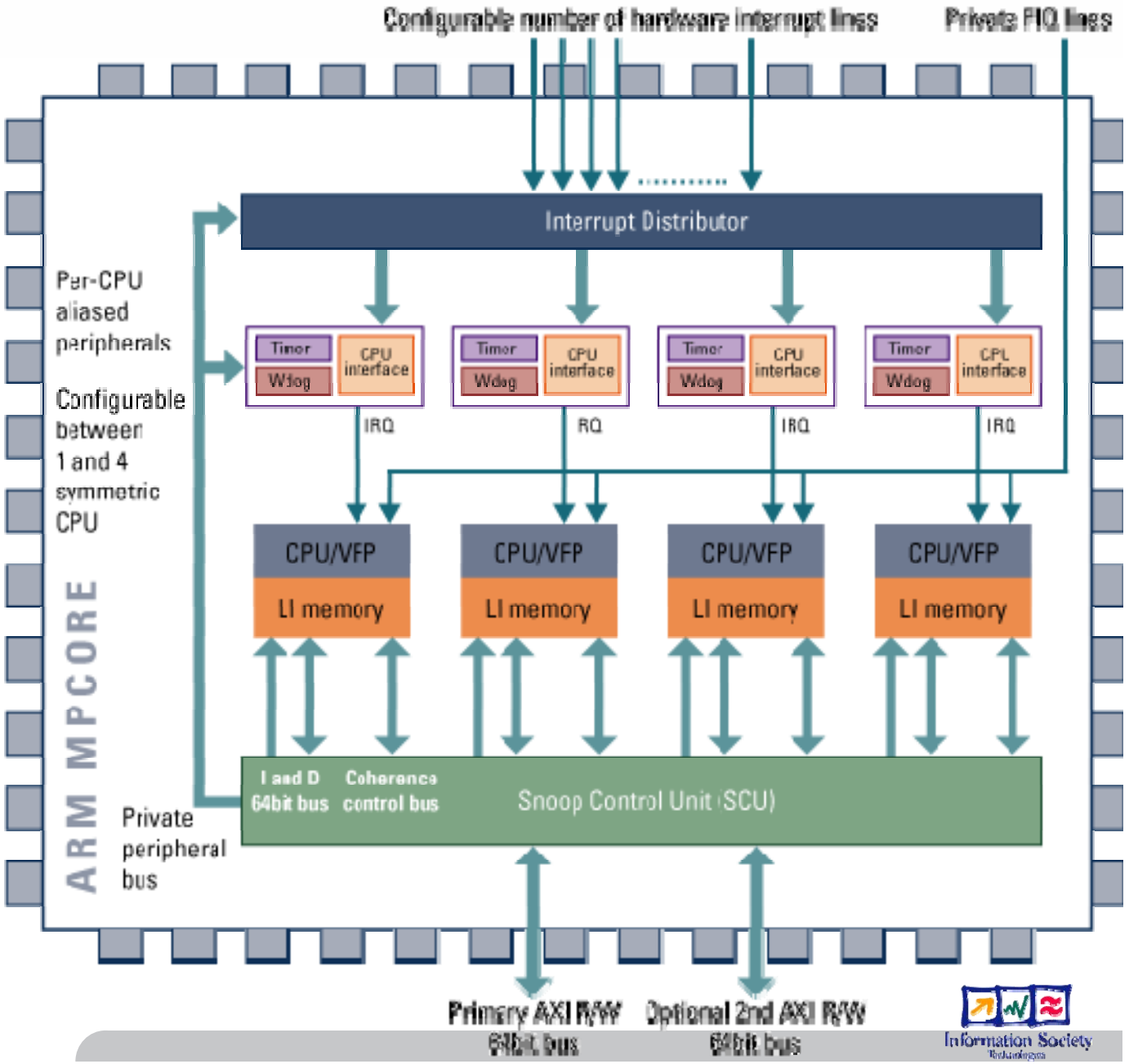


.http://www.mpsoc-forum.org/2007/slides/Hattori.pdf

Homogeneous Multiprocessors

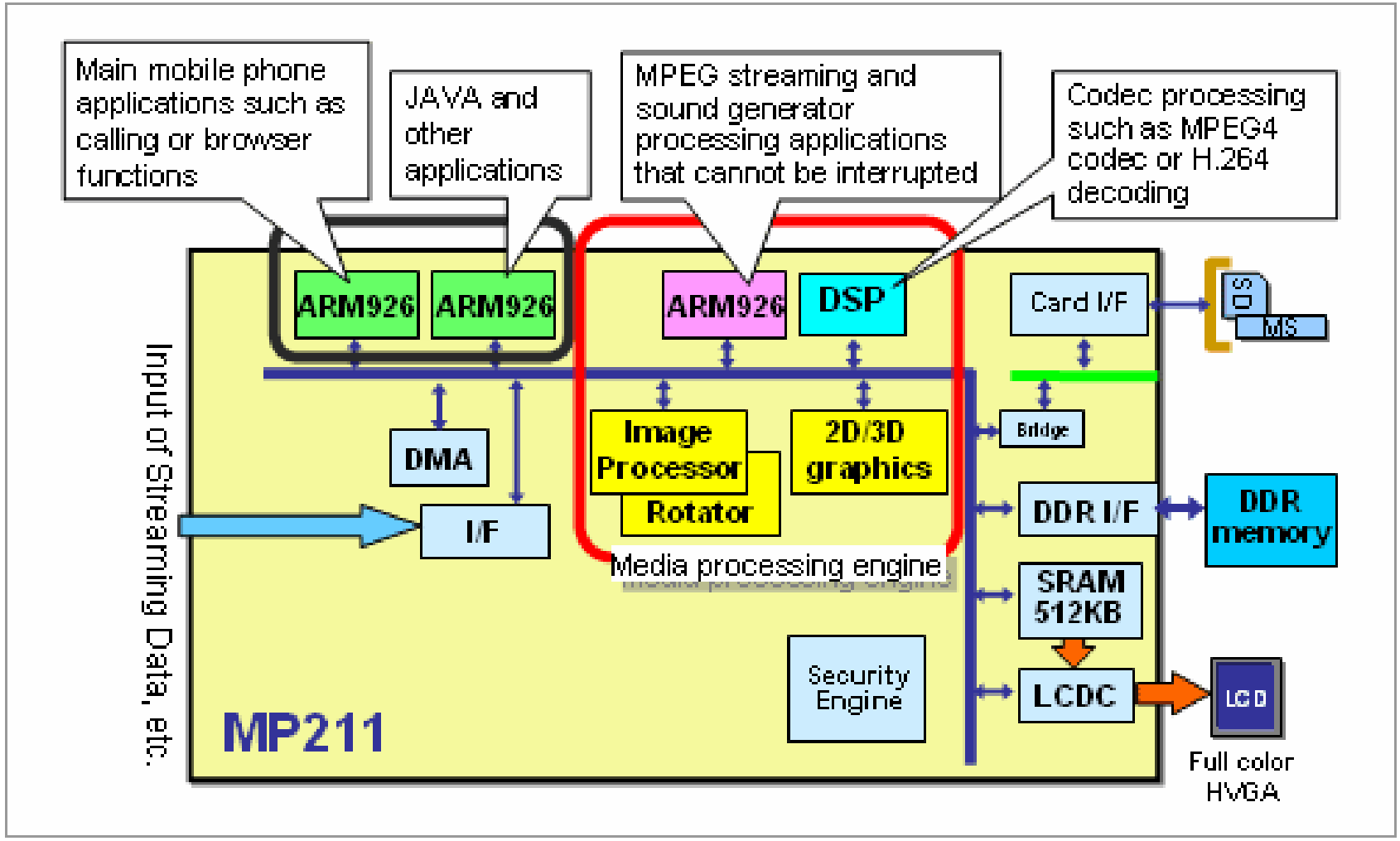
- Can be used if ultimate power efficiency is not a constraint
- Simplify dynamic migration
- Simplify fault tolerance
- May be easier to use in automotive/avionics than in high-speed portable appliances

Homo-geneous …



http://www.arm.com/rximages/5309.gif

Partially homogeneous … (NEC)



http://www.necel.com/application_processor/en/product.html

Workshops

- Road mapping/brainstorming workshop in cooperation with hipeac2 and ACOTES June 16th/17th, Schloss Rheinfels, St. Goar (Germany) Ample time for discussions

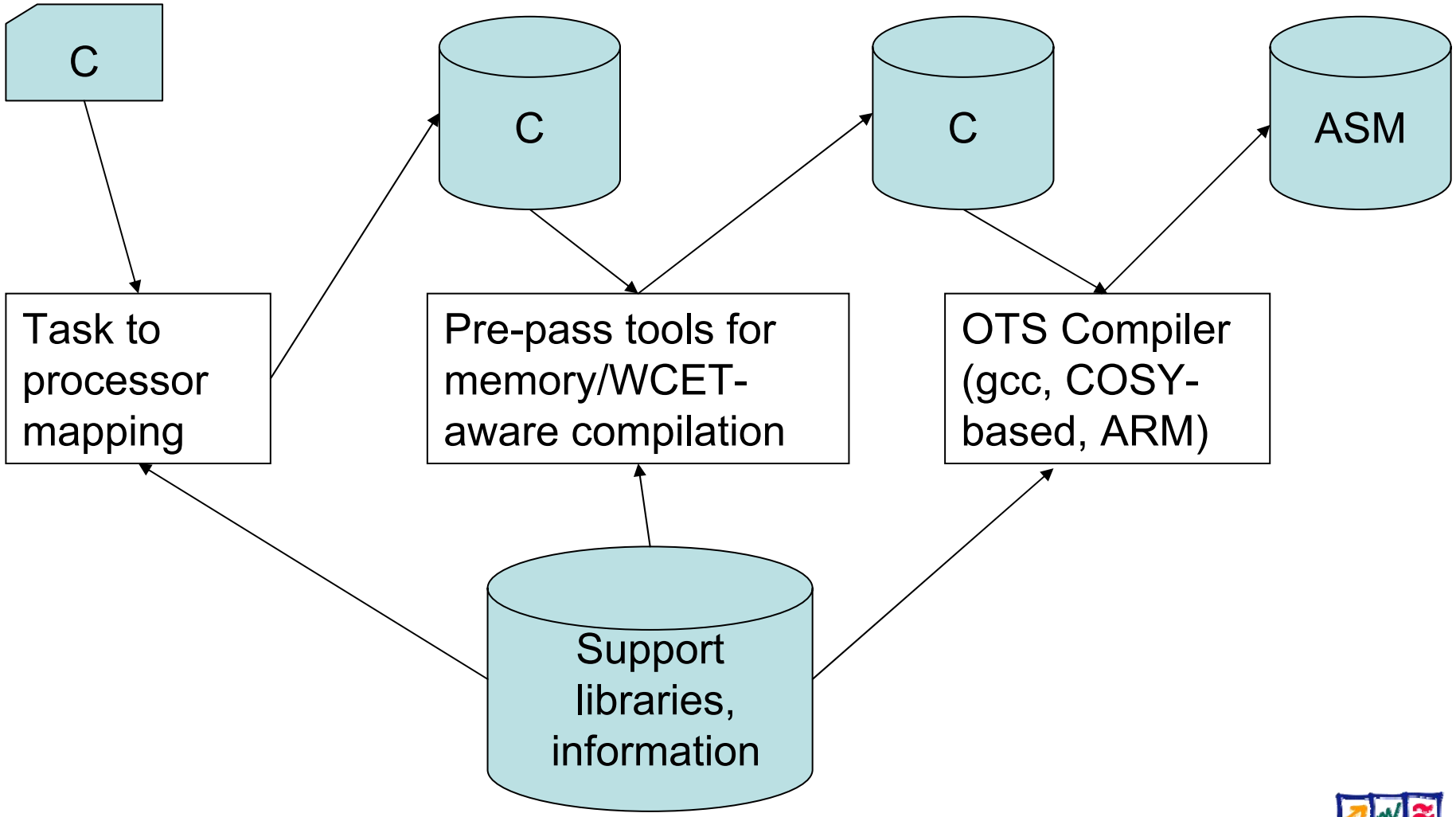
- SCOPES 2008 (Artist2, Munich, DATE Friday)
- likely: SCOPES 2009

MPSoC Compilation (MAPS), RWTH Aachen

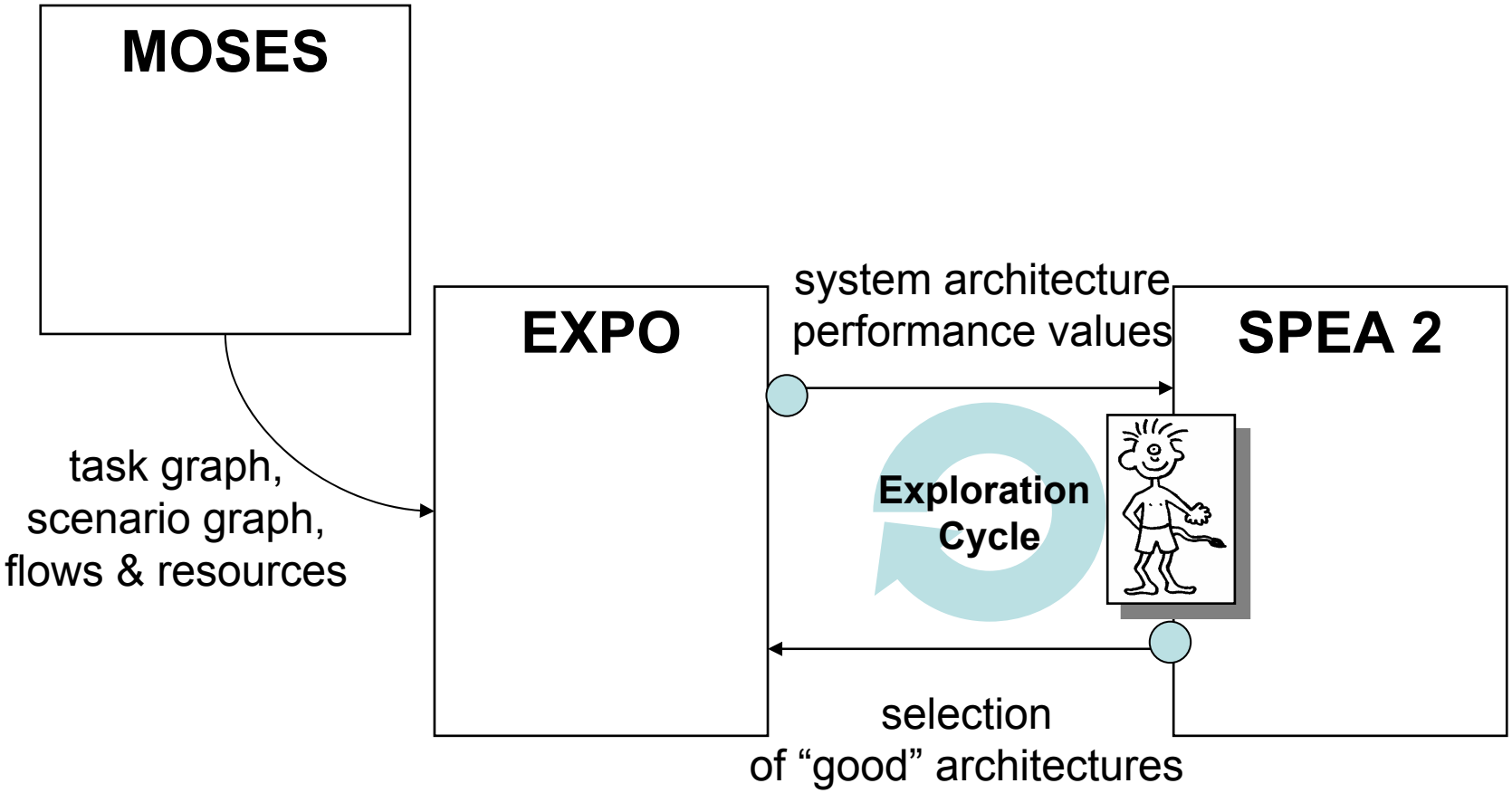
MAPS - MPSoC Application Programming Studio

- MAPS Concepts (Sequential code → Threaded code)
  - Application code analysis (static/dynamic)
  - Semi-automatic code parallelization
  - Task to processor mapping based on abstract MPSoC platform model
- First instance of MAPS backend - TCT
  - ISS cooperation with TokyoTech
  - TCT has tool support for threaded code mapping + simulators and chip prototype
- Need for static and dynamic task creation/scheduling
  - HW IP support

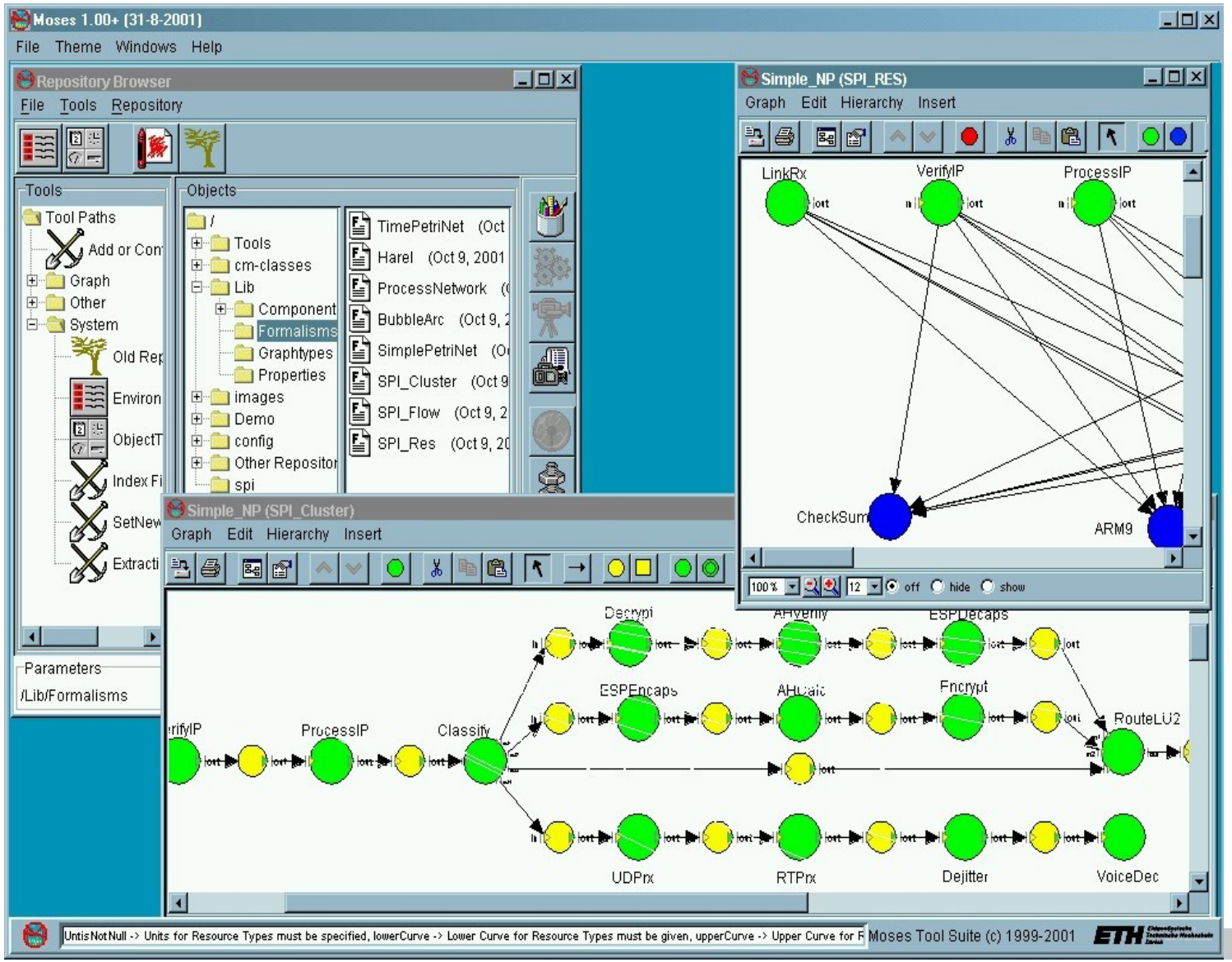
Mnemee/Predator Tool Suite @ Dortmund



EXPO – Tool architecture (1)

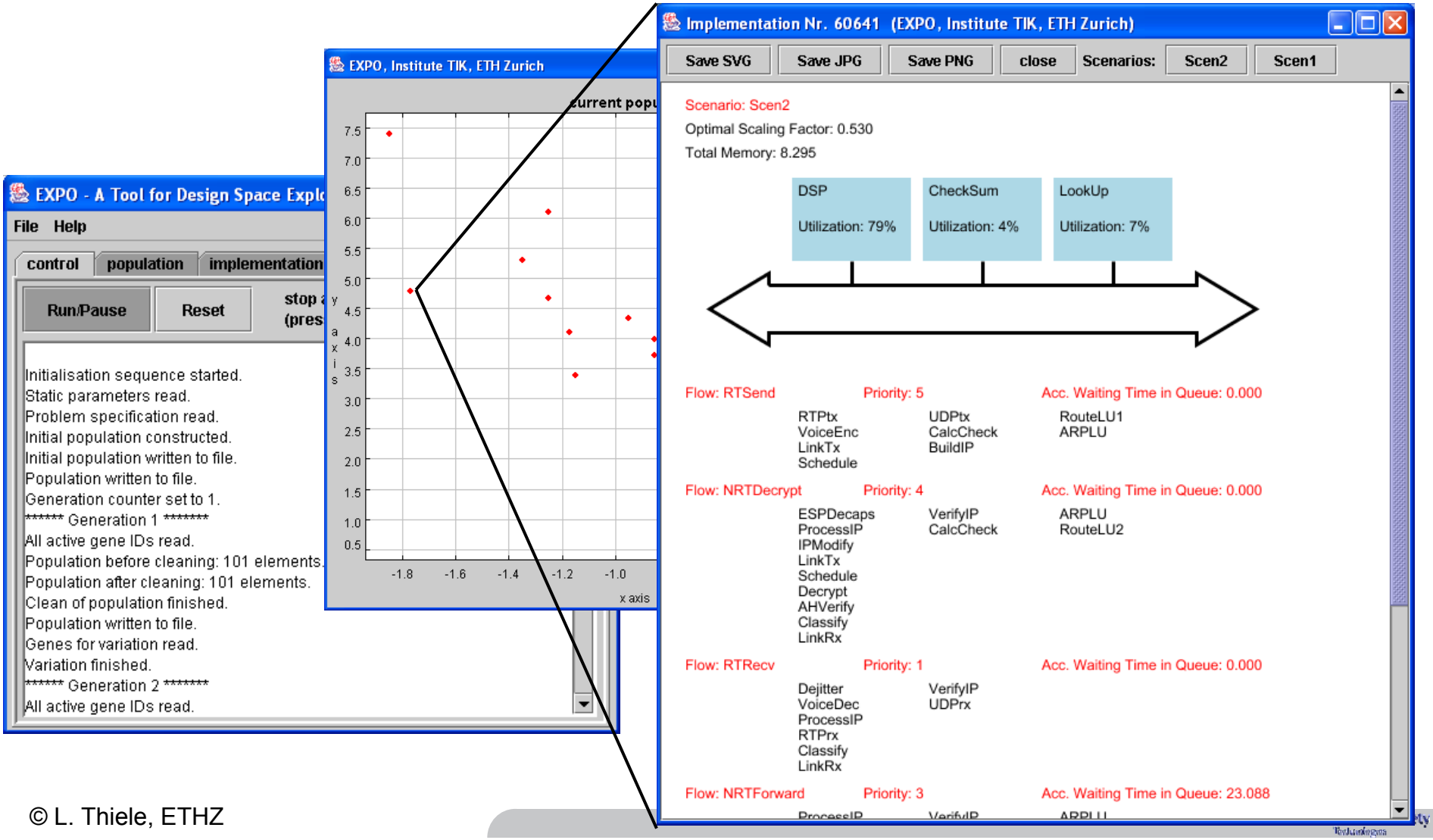


EXPO – Tool architecture (2)

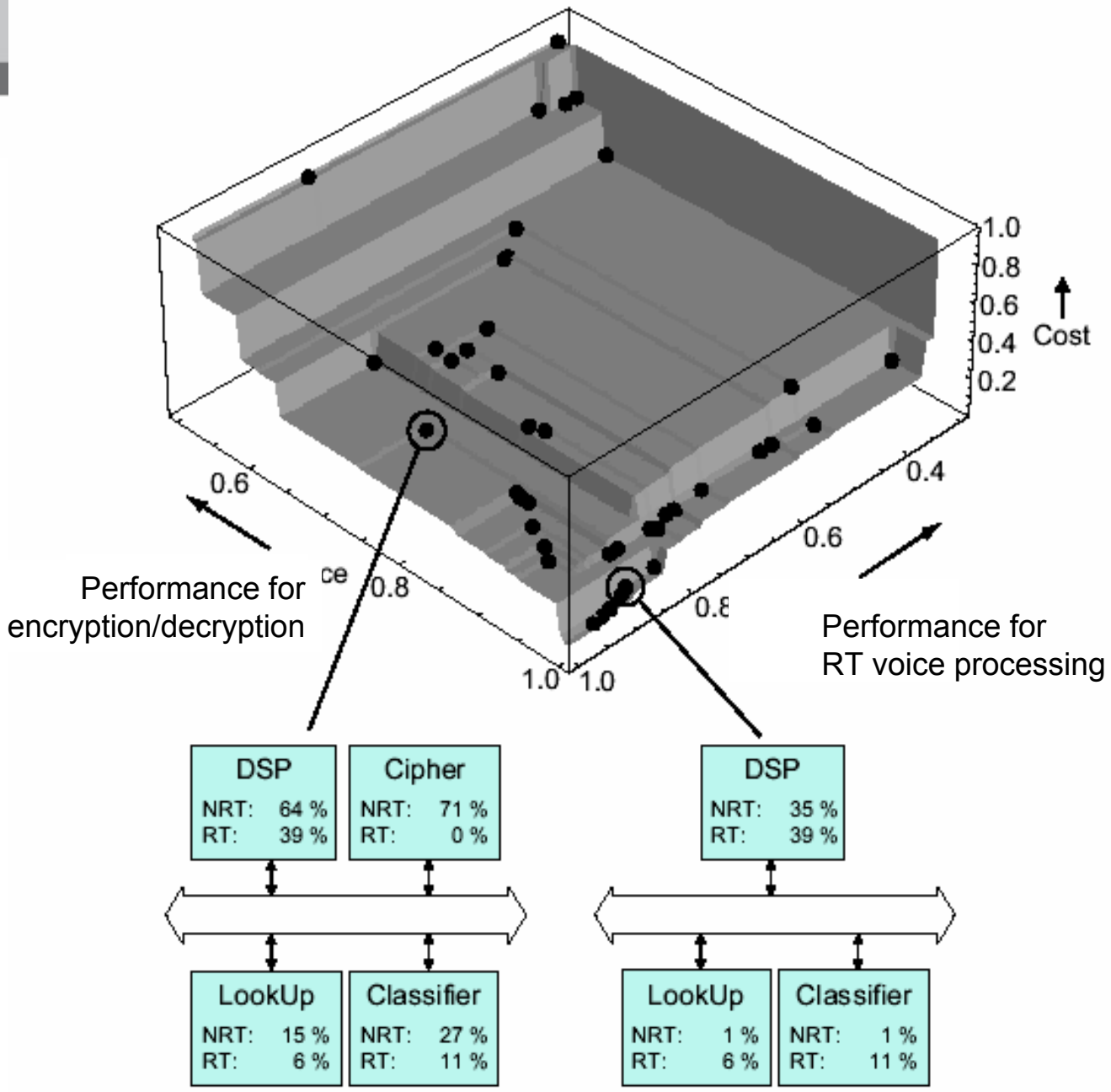


Tool available online: http://www.tik.ee.ethz.ch/expo/expo.html

EXPO – Tool (3)



Results



Proposed Solution: scope

- Application domain: streaming applications
  - Repeated computations on large datasets
  - Examples: video processing, SDR
  - Targets needs of European semiconductor industries
- Use a main stream language
- Target three architectures in one compiler
  - Cell (IBM)
  - xSTream (STMicroelectronics)
  - Ne-XVP (NXP)
- Application programmer is in control
  - No black box, Swiss-army-knife system

The ACOTES Project

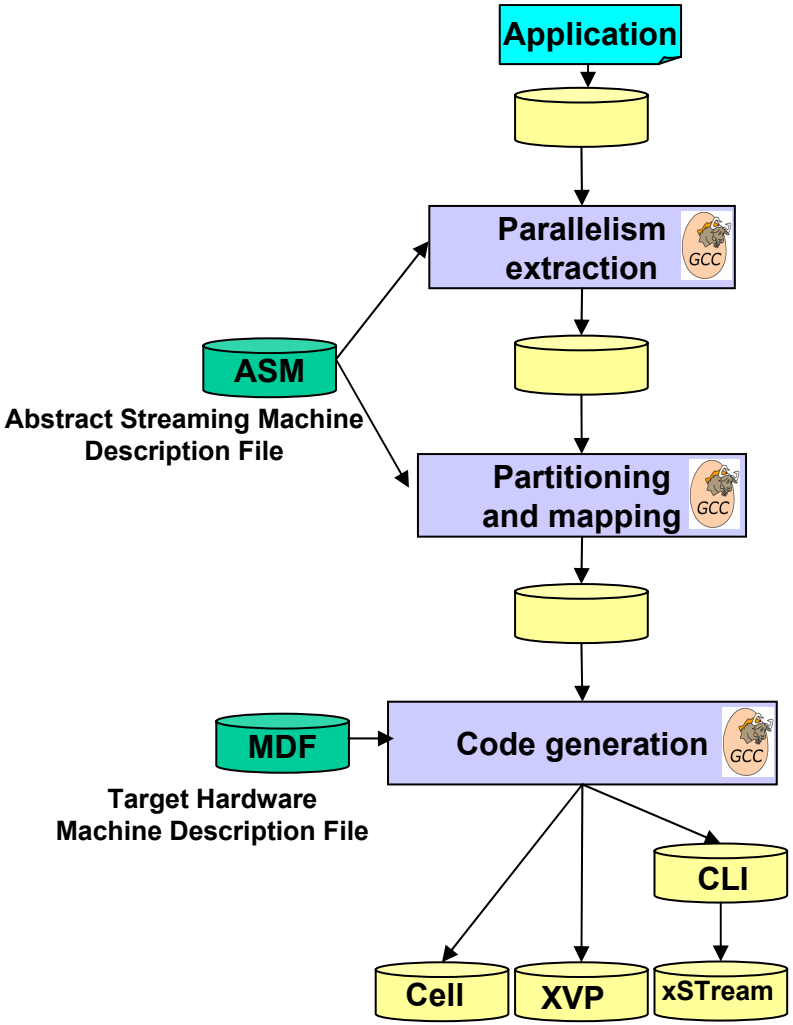
- Extensions to C (pragmas)
  - Expressing stream oriented features
- Stream-oriented optimisations
  - Task-level optimisations
  - Loop-nest optimisations
  - Code vectorisation
- Multi-ISA (Cell, xSTream, Ne-XVP) compilation
  - Same source
- Partitioning and mapping efficiency analysis
  - Supported by ASM

The ACOTES Project

- GCC4 as compiler platform
  - Industrial strength
  - Widely accepted
  - Community-based evolution and maintenance

- Applications and products targeted:
  - SDR-based telecom
  - Video processing
    - Multi media applications
    - Enhancement
    - 3D from 2D
  - Other application areas
    - SIMD intensive scientific computing, e.g. weather forecasting

The ACOTES Project



WP2

SPM and ASM

- Define SPM and ASM
- Implement infrastructure for SPM in GCC
  - Prototypes in Mercurium (Source-to-source)
- Implement infrastructure for ASM in GCC

The ACOTES Project

Partner	Expertise
NXP, IBM, STMicroelectronics	target architecture (Cell/BE, Ne-XVP, xSTream)
NXP, NOKIA, INRIA, IBM, UPC, STMicroelectronics, Silicon-Hive	streaming applications
NXP, INRIA-Futurs, UPC	concurrency representations
NXP, IBM, INRIA-Futurs, STMicroelectronics, Silicon-Hive	compiler construction
NXP, IBM, STMicroelectronics, Silicon-Hive, UPC	High performance embedded architectures

Nokia Compiler Workshop

A Complete Compiler Approach to Auto-Parallelizing C Programs for Multi-DSP Systems

Björn Franke & Michael O'Boyle
University of Edinburgh
School of Informatics

Düsseldorf, June 1st 2006

Overview

- Motivation
- Multi-DSP memory model
- Parallelization
  - Parallelization model
  - Parallelism detection & program recovery
  - Partitioning & mapping
  - Access Localization
  - Data Transfer Optimization
- Experimental results
- Conclusion & future work

Motivation

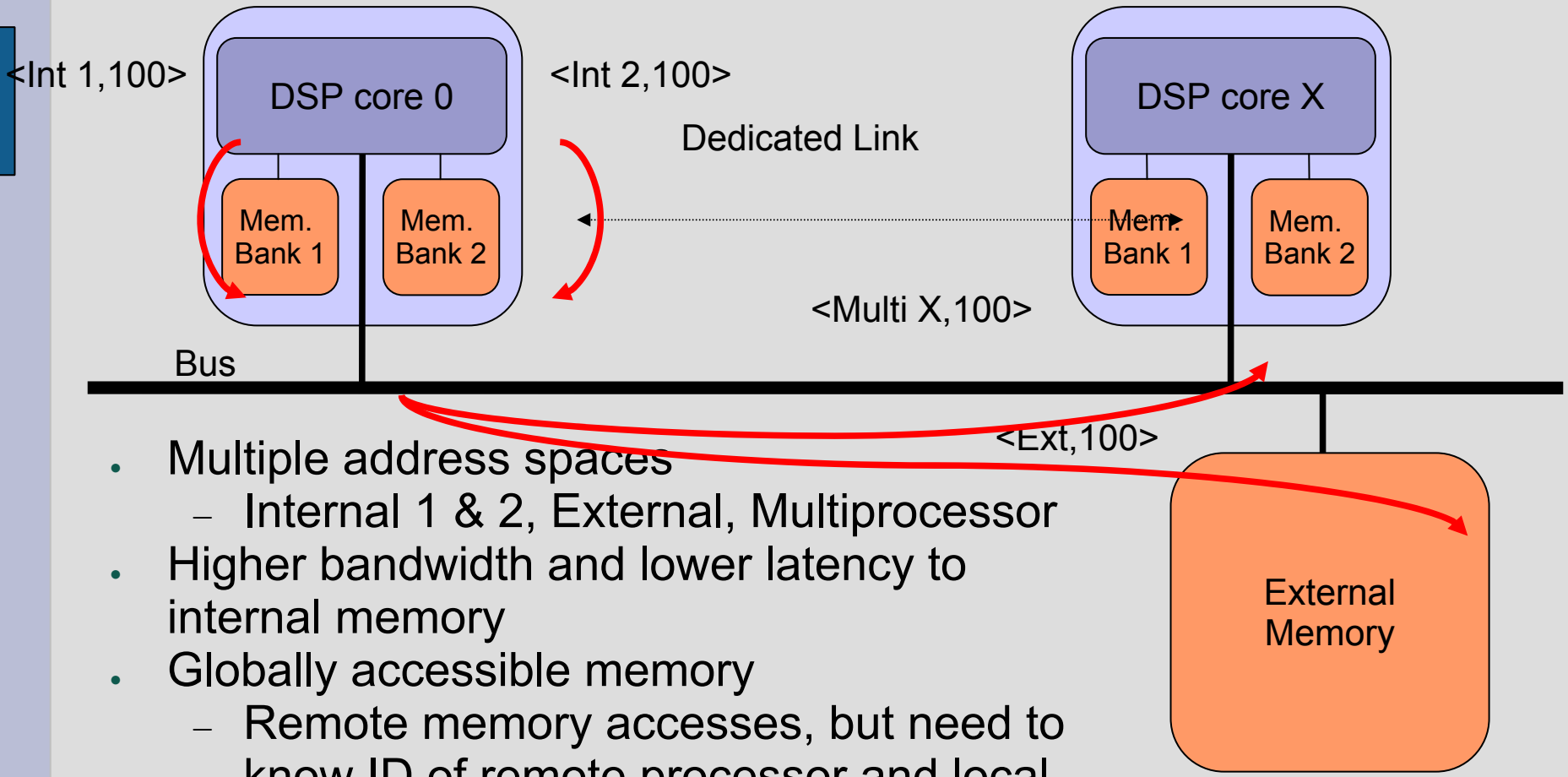
- Performance requirements exceed single processor capabilities
  - Radar & sonar, medical imaging, HDTV, ...
- Multi-DSP can provide enough performance, but
  - Little HW support for parallel execution
  - Even less tool support for parallel programming
  - Low level programming style in legacy codes

Parallelizing Compilers

- Successful in Scientific Computing
  - Over 25 years of research experience
  - Routinely parallelize standard Fortran codes
  - Not optimal, but sufficient parallelization quality
- Depend on compiler-friendly memory model
  - Shared memory
    - Single address space
    - Physical memory distributed or central
    - Distributed caches, but HW cache coherence
  - Multi-DSP do not provide this!

→**No parallelizing compiler for Multi-DSP**

Multi-DSP Memory Model



- Multiple address spaces
  - Internal 1 & 2, External, Multiprocessor
- Higher bandwidth and lower latency to internal memory
- Globally accessible memory
  - Remote memory accesses, but need to know ID of remote processor and local offset

Parallelization Model

- Program Recovery
  - Eliminate “compiler unfriendly” constructs
- Parallelism Detection
  - Identify parallelizable loops
- Partitioning & Mapping
  - Minimize communications
  - Data partitioning, code follows later
- Access Localization
  - Minimize unnecessary remote accesses
- Data Transfer Optimization
  - Make use of DMA for bulk transfers

Running Example

```
/* Array Declarations */
int A[16],B[16],C[16],D[16];

/* Pointer Declaration & Initialization */
int *p_a = A, *p_b = &B[15], *p_c = C, *p_d = D;

/* Computational Loop */
for (i = 0; i < 16; i++)
   *p_d++ = *p_c++ + *p_a++ * *p_b--;
```

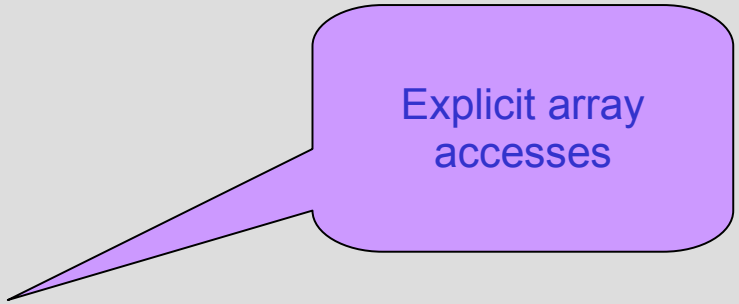
Next step: Eliminate pointer based accesses

Program Recovery
Pointer Conversion

```
/* Array Declarations */
int A[16],B[16],C[16],D[16];



/* Computational Loop */
for (i = 0; i < 16; i++)
   D[i] = C[i] + A[i] * B[15-i];
```



Next step: Partition data along first array index

Parallelisation
Data Partitioning

New array declarations

```
/* Array Declarations */
int A[2][8],B[2][8],C[2][8],D[2][8];
```

Non-affine indices

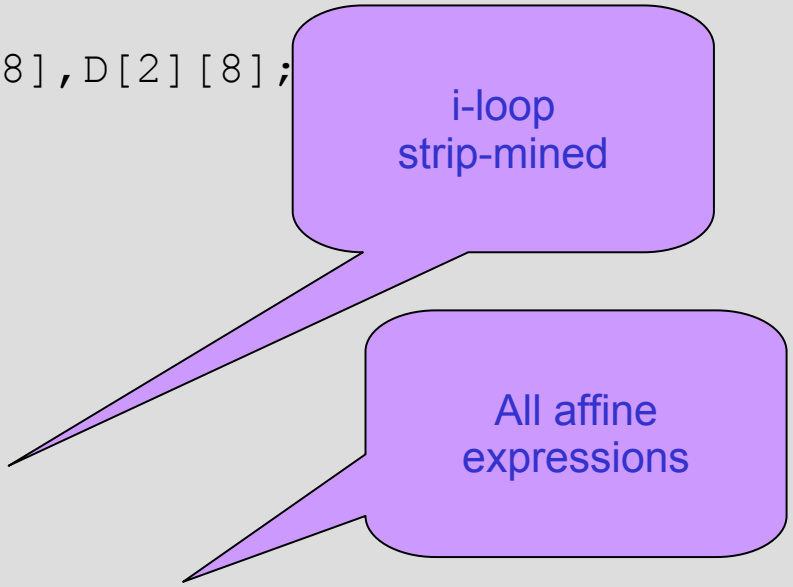
```
/* Computational Loop */
for (i = 0; i < 16; i++)
   D[i/2][i%8] = C[i/2][i%8] + A[i/2][i%8] * B[(15-i)/2][(15-
i)%8];
```

Next step: Eliminate modulo based array indexing

Parallelisation
Array Subscript Transformation

```
/* Array Declarations */
int A[2][8],B[2][8],C[2][8],D[2][8];

/* Computational Loop */
for (j = 0; j < 2; j++)
   for (i = 0; i < 8; i++)
      D[j][i] = C[j][i] + A[j][i] * B[1-j][7-i];
```



Next step: Parallel distribution of outer j loop

Parallelisation
Parallel Loop Mapping (for processor 0)

```
#define MYID  0

/* Array Declarations */
int A[2][8],B[2][8],C[2][8],D[2][8];

/* Computational Loop */
   for (i = 0; i < 8; i++)
      D[MYID][i] = C[MYID][i] + A[MYID][i] * B[1-MYID][7-
i];
```

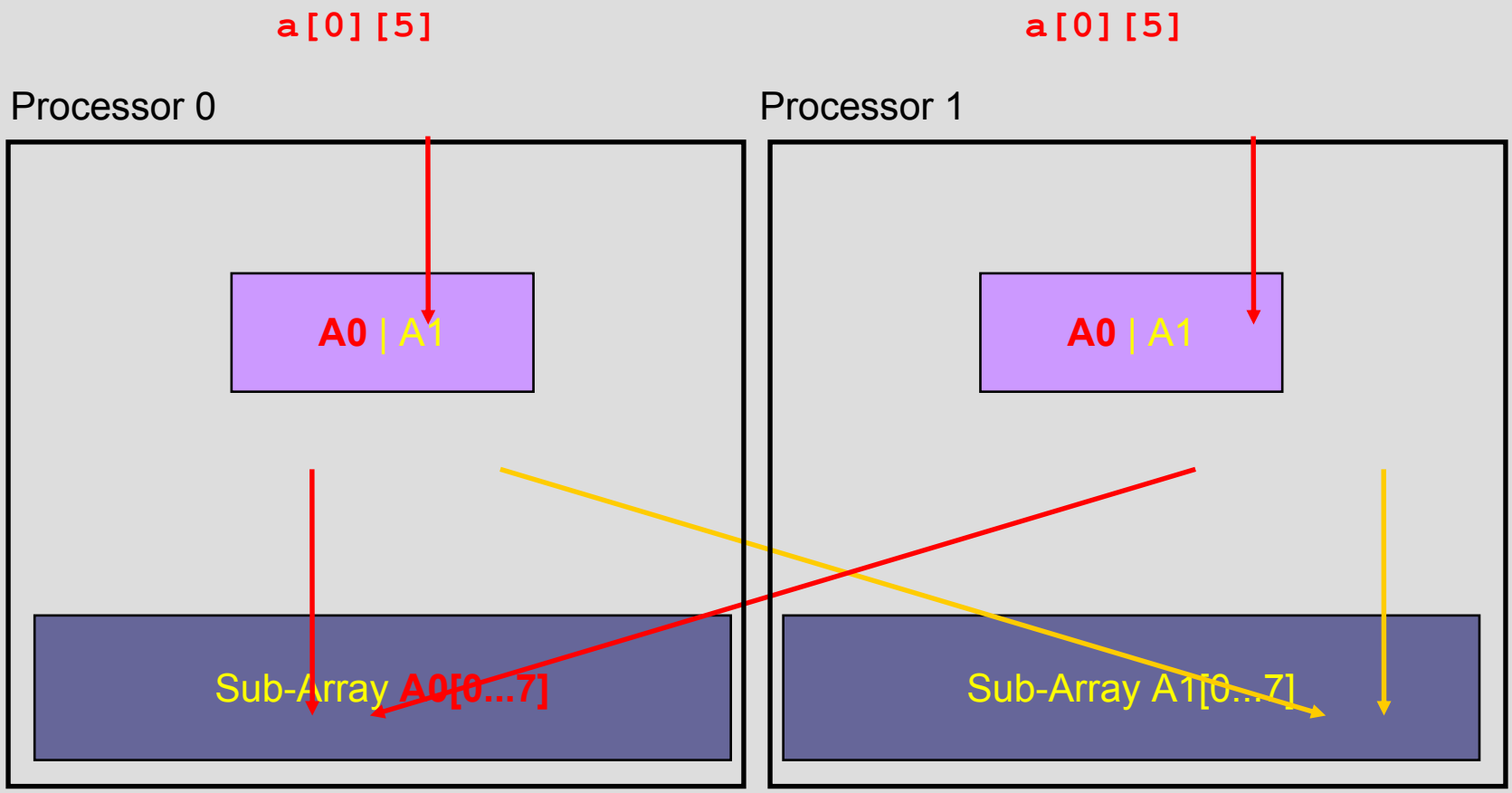
Constant processor ID

j-loop dropped

Explicit processor ID

Next step: Managing multiple address spaces

Parallelisation
Address Resolution & Descriptors



Parallelisation
Address Resolution

```
#define MYID 0


/* Array Declarations */
int A0[8]; extern int A1[8];
...
/* Descriptor Declarations */
int *A[2] = {A0,A1};
...
/* Computational Loop */
   for (i = 0; i < 8; i++)
      D[MYID][i] = C[MYID][i] + A[MYID][i] * B[1-MYID][7-
i];
```

Arrays distributed

Descriptors introduced

Accesses unchanged!

Next step: Making local array accesses faster

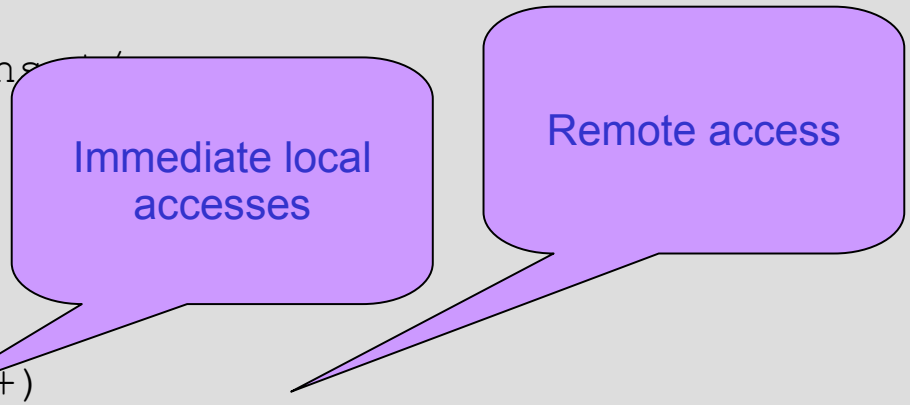
Parallelization
Access Localization

```
#define MYID 0

/* Array Declarations */
int A0[8]; extern int A1[8];

/* Descriptor Declarations */
int *A[2] = {A0,A1};

/* Computational Loop */
   for (i = 0; i < 8; i++)
      D0[i] = C0[i] + A0[i] * B[1-MYID][7-i];
```



Next step: Eliminate expensive element-wise remote accesses

Data Transfer Optimization

```
#define MYID 0

int temp[8];

/* DMA transfer */
DMA_get(temp,&(B[1][i]),8*sizeof(int));



/* Computational Loop */
   for (i = 0; i < 8; i++)
      D0[i] = C0[i] + A0[i] * temp[7-i];
```

Local buffer allocated

DMA block transfer to local buffer

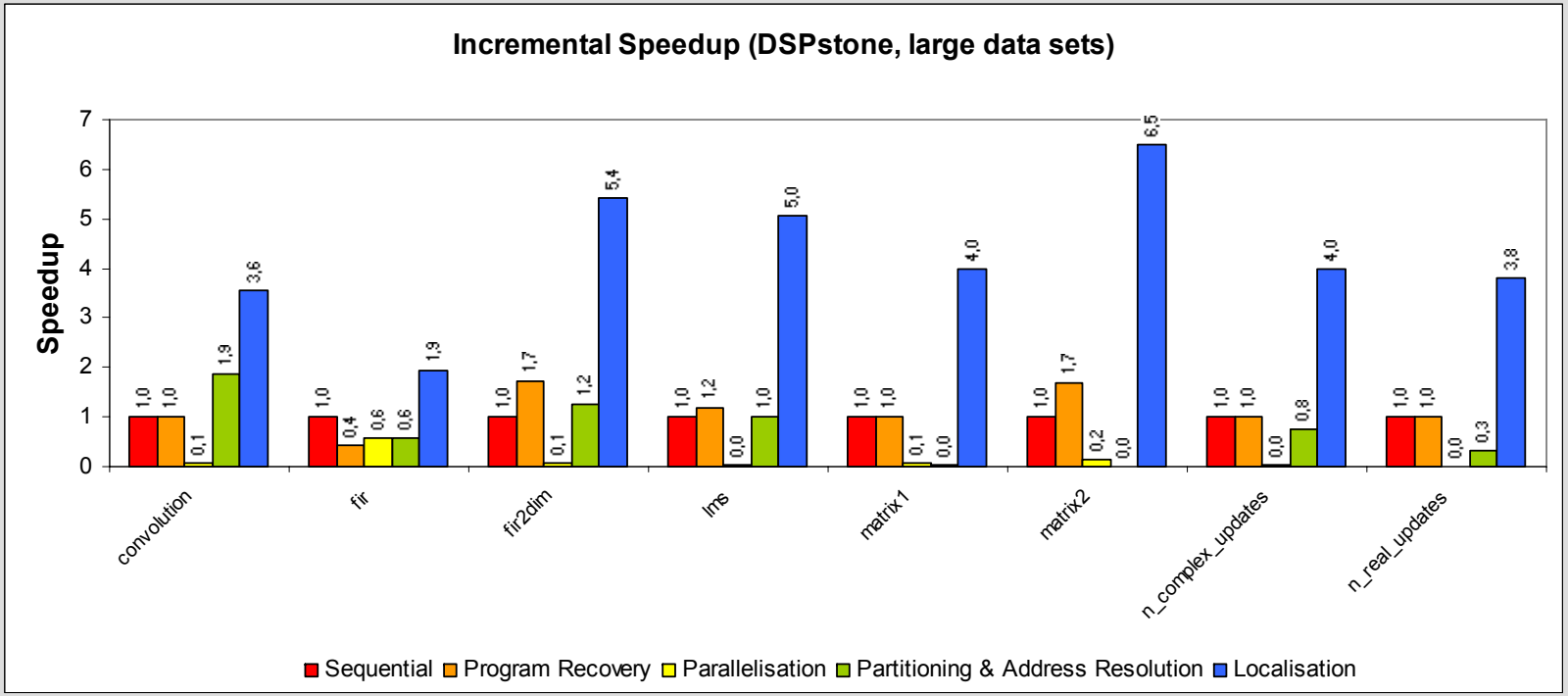
Access to local buffer

Done!

Empirical Evaluation

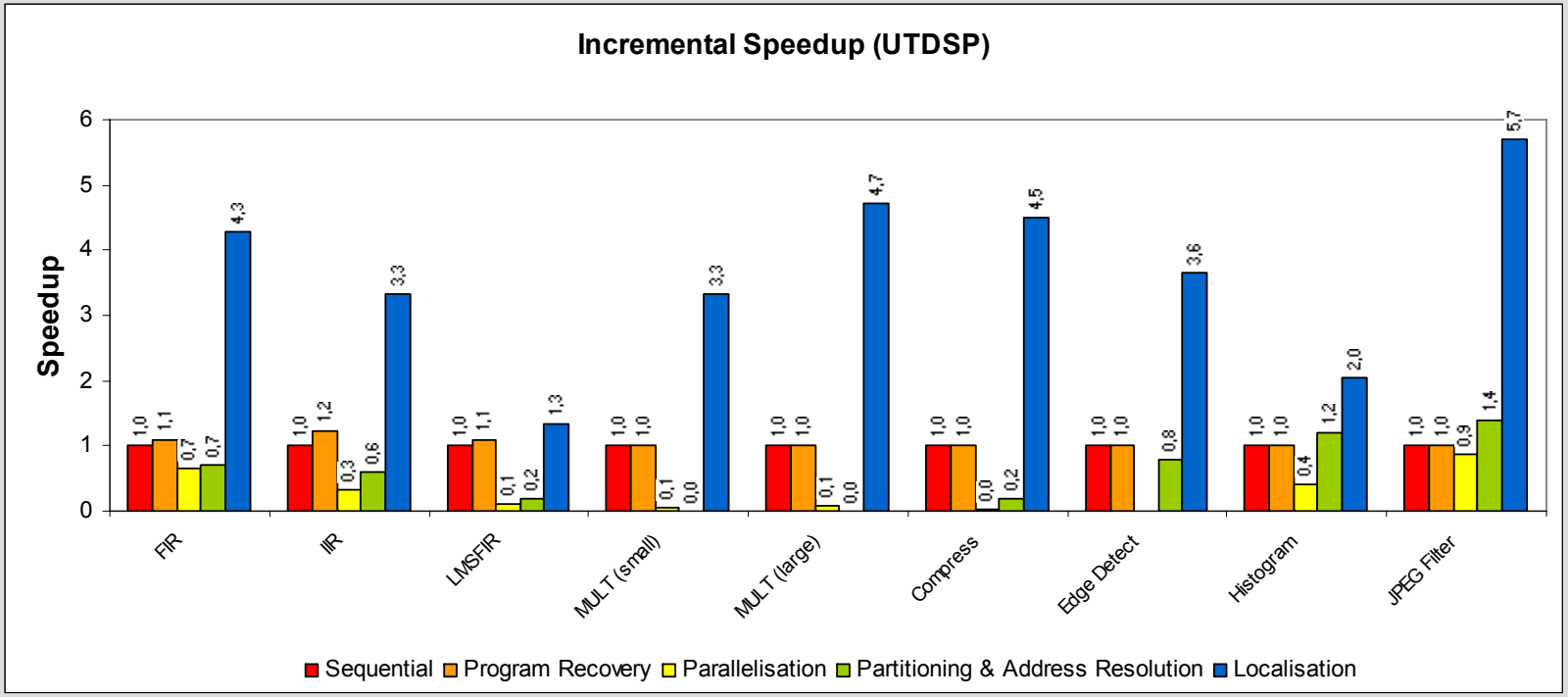
- Benchmarks
  - DSPstone & UTDSP
- Prototype implementation
  - Stanford SUIF1
  - Source-to-source
- Target machine & compiler
  - 4 processor Analog Devices TigerSHARC
  - Analog Devices VisualDSP++

Results
DSPstone



Average speedup on four processors: 4.28

Results
UTDSP



Average speedup on four processors: 3.65

Contributions

- Not a single method, but a combination of techniques
  - However, single algebraic framework
- Compiler based approach
  - Non-standard compilation model
  - Address resolution & data descriptors
- Performance optimization
  - Prove data locality for affine accesses
  - Partitioning/mapping makes processor ID explicit
  - Fast local memory accesses
  - Fast DMA bulk data transfers

Summary & Future Work

- Summary
  - Shared memory like parallel code
    - Multiple address spaces
    - Globally addressable memory
    - No message passing
  - Linear/superlinear speedup
    - Relevant benchmarks written in low-level C
- Future work
  - Parallelization for hetergeneous SoCs
    - Task graph extraction based on statistical models
    - MPSoC design space exploration