The Trimedia CPU64 VLIW Media Processor

Invited session at the ICCD ‘99 conference, 10-13 October, Austin, TX

Chair: Kees Vissers, Philips Research

TriMedia CPU64 Application Domain and Benchmark Suite

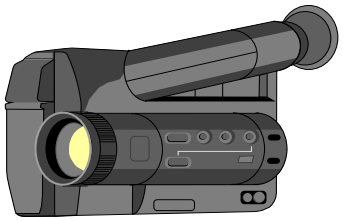
A.K. Riemens
Philips Research
Eindhoven, The Netherlands

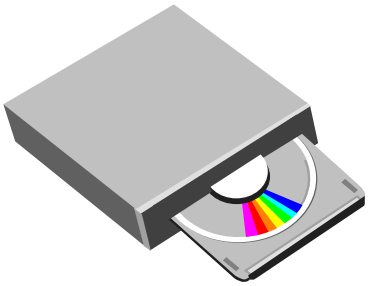
ICCD 1999/application-2

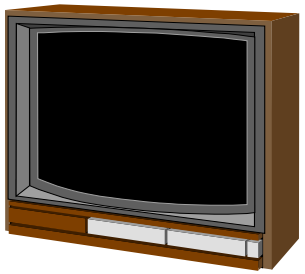
Outline

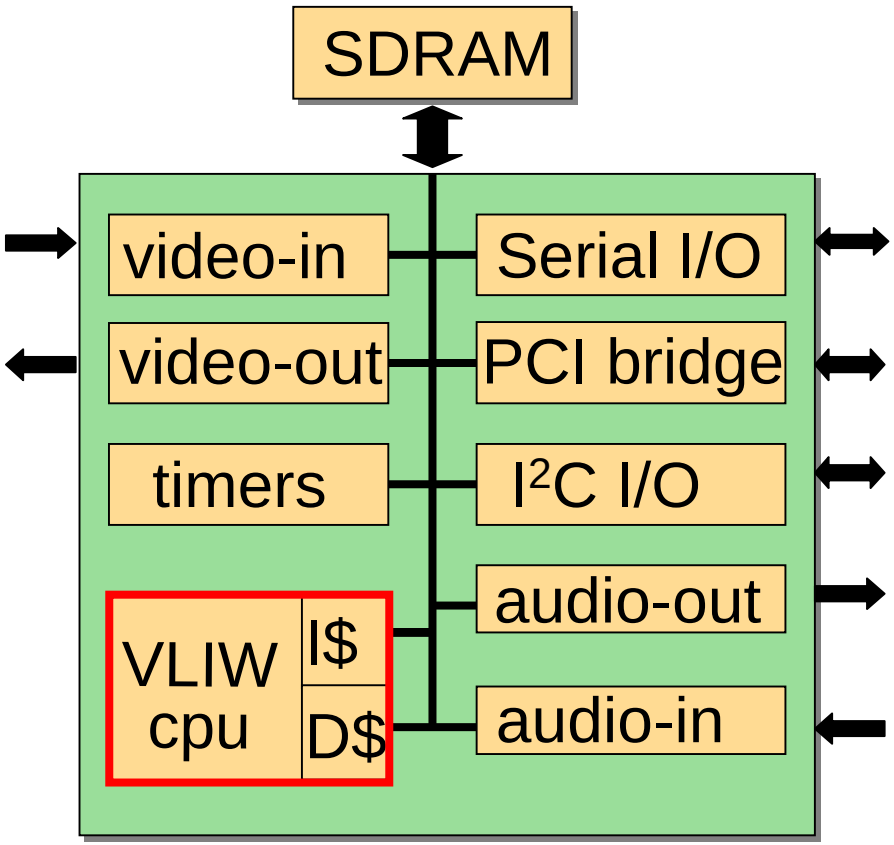
- Introduction
- Approach
- Benchmark suite
- Example
- Results

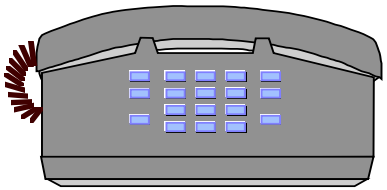
Design Problem

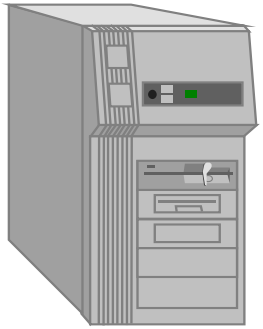












Application Domain

- High volume consumer electronics products future TV, home theatre, set-top box, etc.
- Embedded core
- Media processing:
  audio, video, graphics, communication

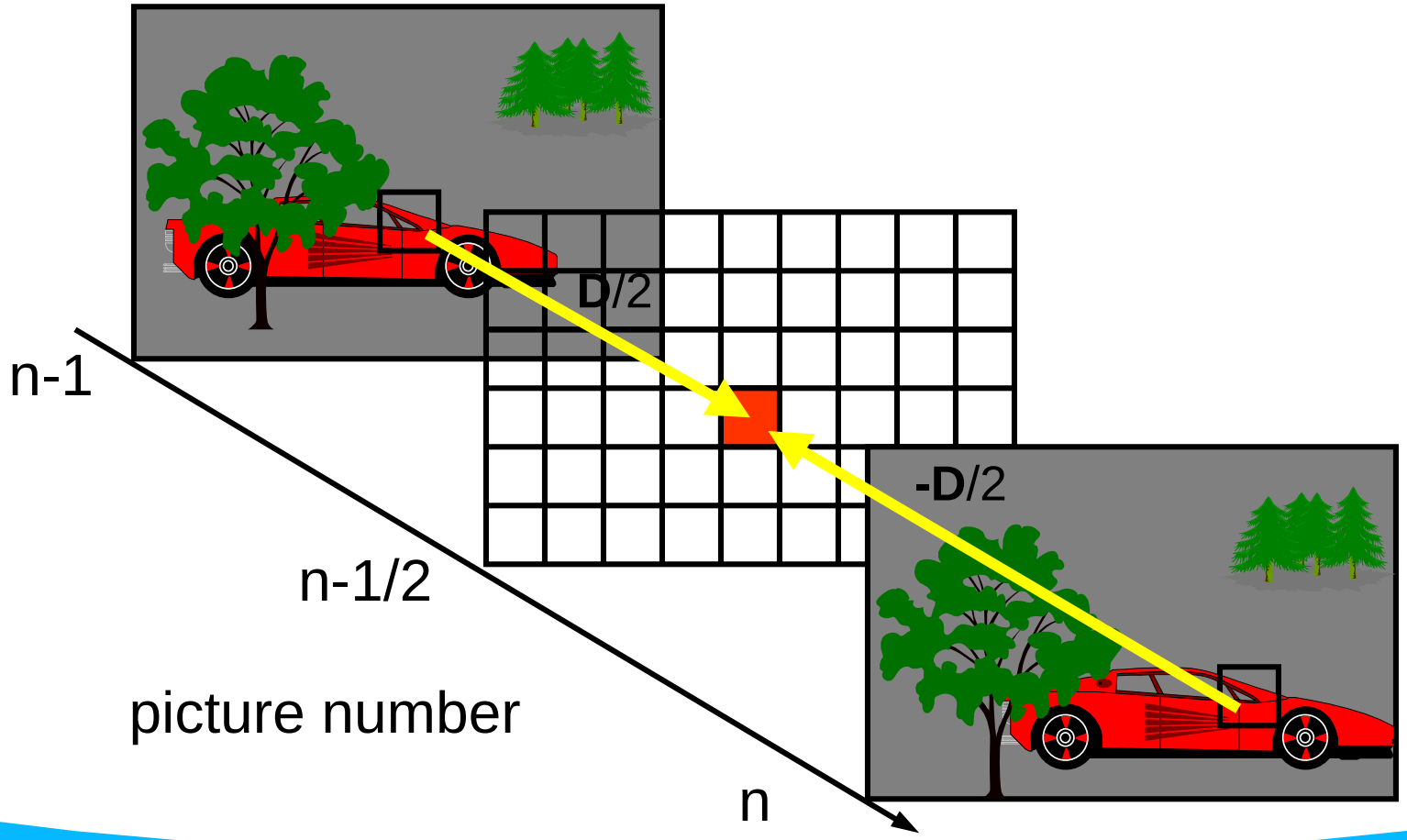
Media Processing CPU

Design considerations:

- Cost (embedded in high volume products)
- Performance for the application domain
- Ease of use (programming model)

⇒ Benchmark suite needed

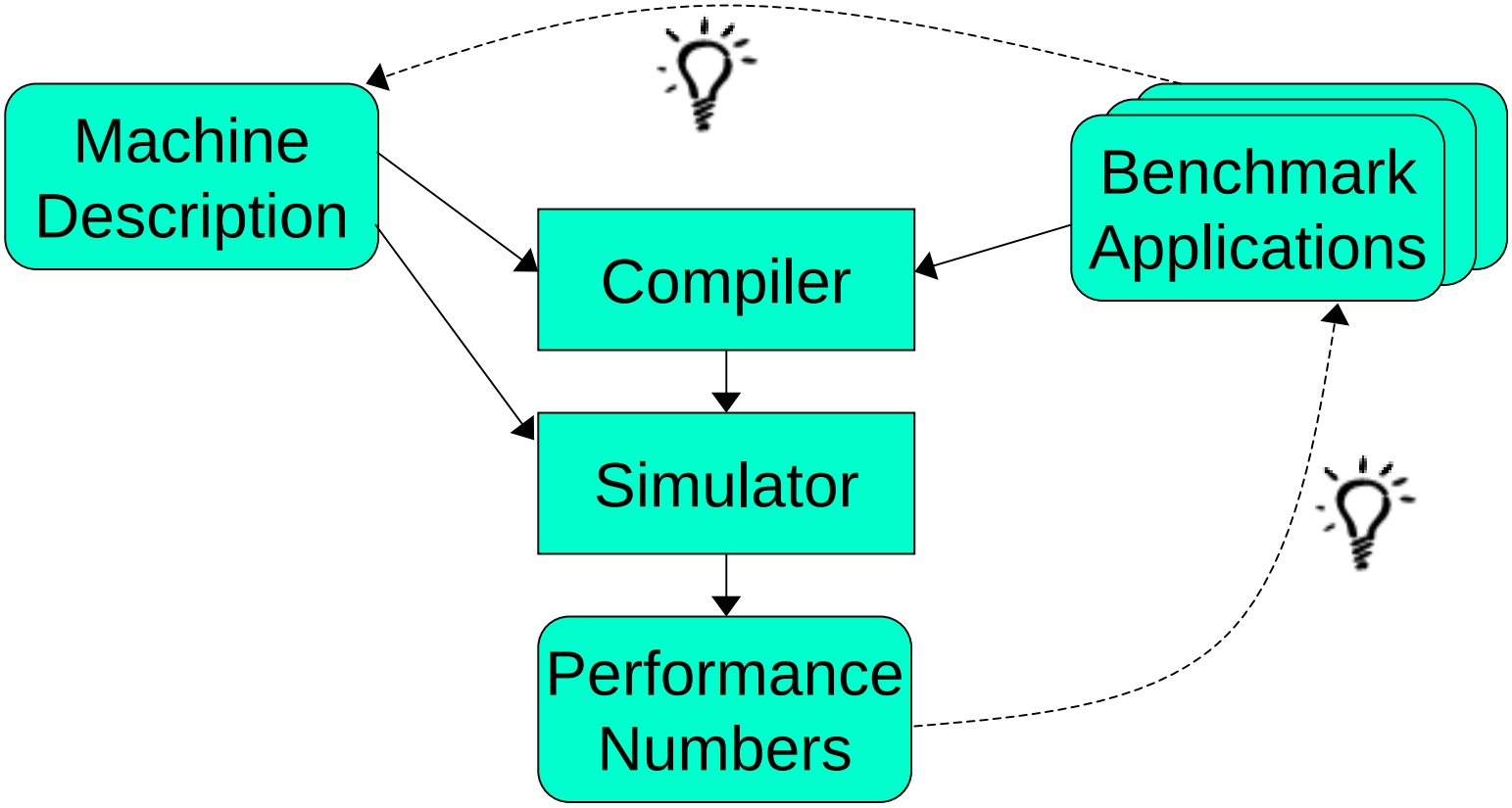
Application: Natural Motion



Outline

- Introduction
- Approach
- Benchmark suite
- Example
- Results

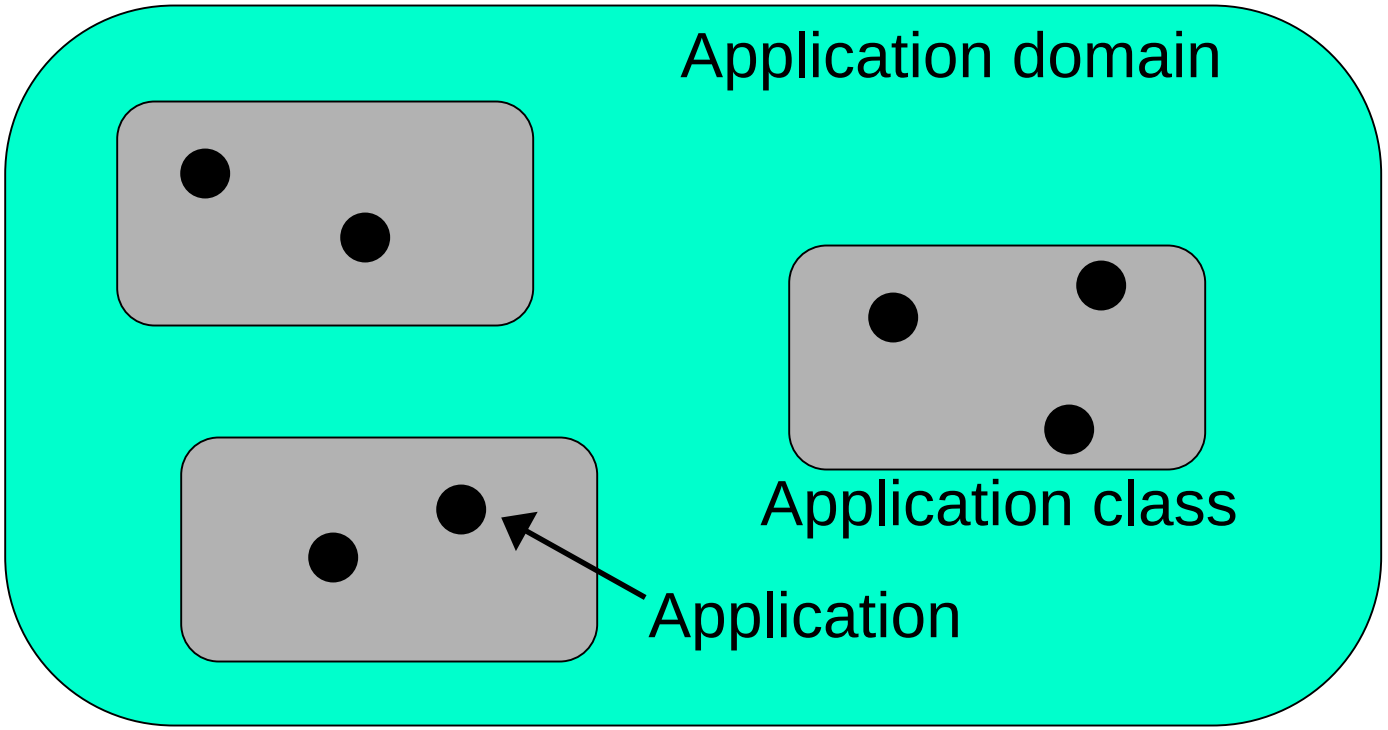
Project Approach: Y-chart



Outline

- Introduction
- Approach
- Benchmark suite
- Example
- Results

Terminology



Benchmark suite: set of all applications

Benchmark Suite Characteristics

- Each application: typical for a class of applications within application domain
- The set covers a significant area of the application domain
- Each benchmark is sufficiently well tuned to the architecture to measure its performance

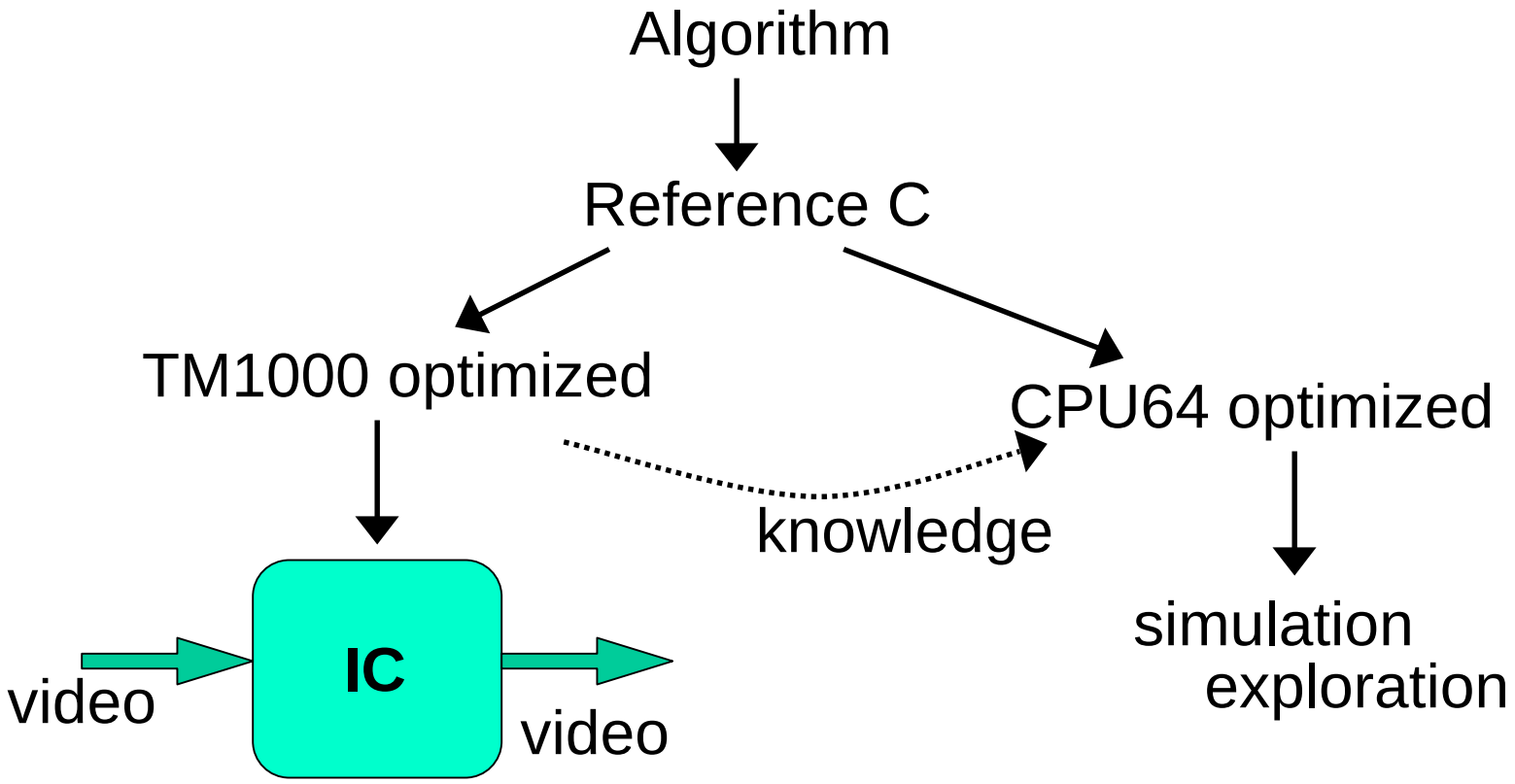
The Benchmark Suite

Data comm.	Viterbi decoding
Audio coding	AC3 decode
Video coding	MPEG2 encode DVC decode
Video proc.	Layered natural motion Dynamic noise reduction Peaking
Graphics	3D renderer library

Processing Characteristics

- Signal rates
  audio, video (at block and pixel rate)
- Basic data types
  byte (8), half-words (16), words (32), float (32)
- Data access patterns
  sample stream, bitstream, random access
- Data independent and dependent load
- Control processing and signal processing

Application Development



Initial Design Considerations

- Goal: 6-8 times TM1000 performance
- Standard ANSI-C, reuse of code
- Utilize instruction and data parallelism
- Limited complexity (embedded core)
- Compatibility through recompilation
- VLIW architecture

Initial Design Choices

- Double vector length: 32 → 64
- Enriched media instruction set

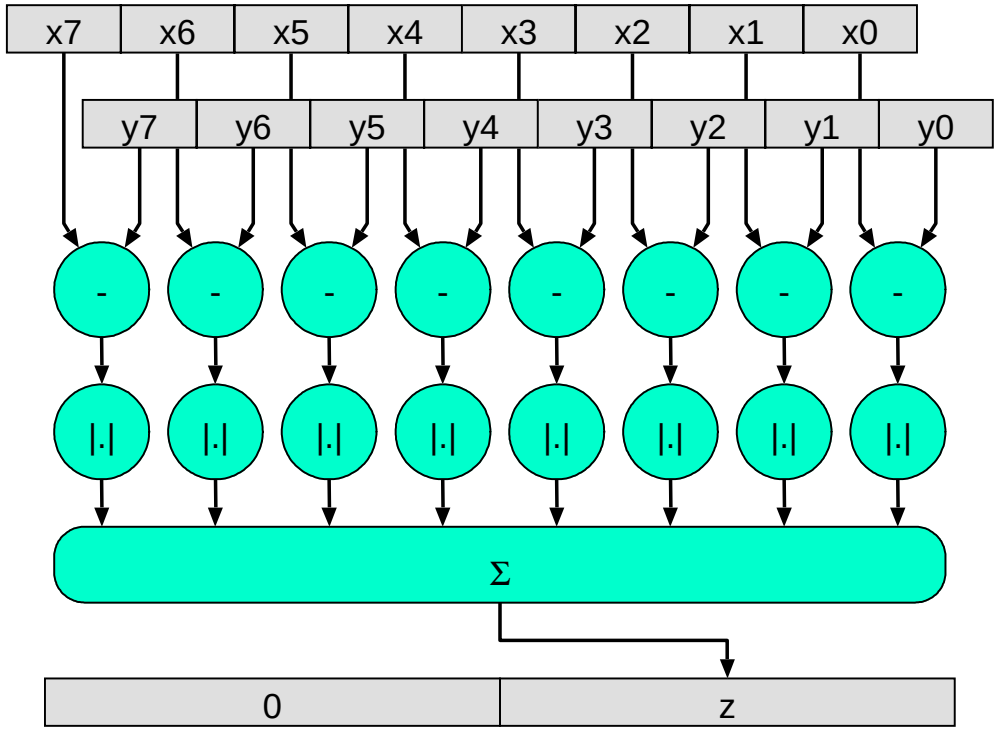
Instruction Set Considerations

- A machine operation must be sufficiently generic within the application domain
- Sufficiently powerful operations
- Limited number of operations
- Consistency and orthogonality

Outline

- Introduction
- Approach
- Benchmark suite
- Example
- Results

Vector Instruction Example



Sum of absolute differences (“ub_me”)

Application Code Example

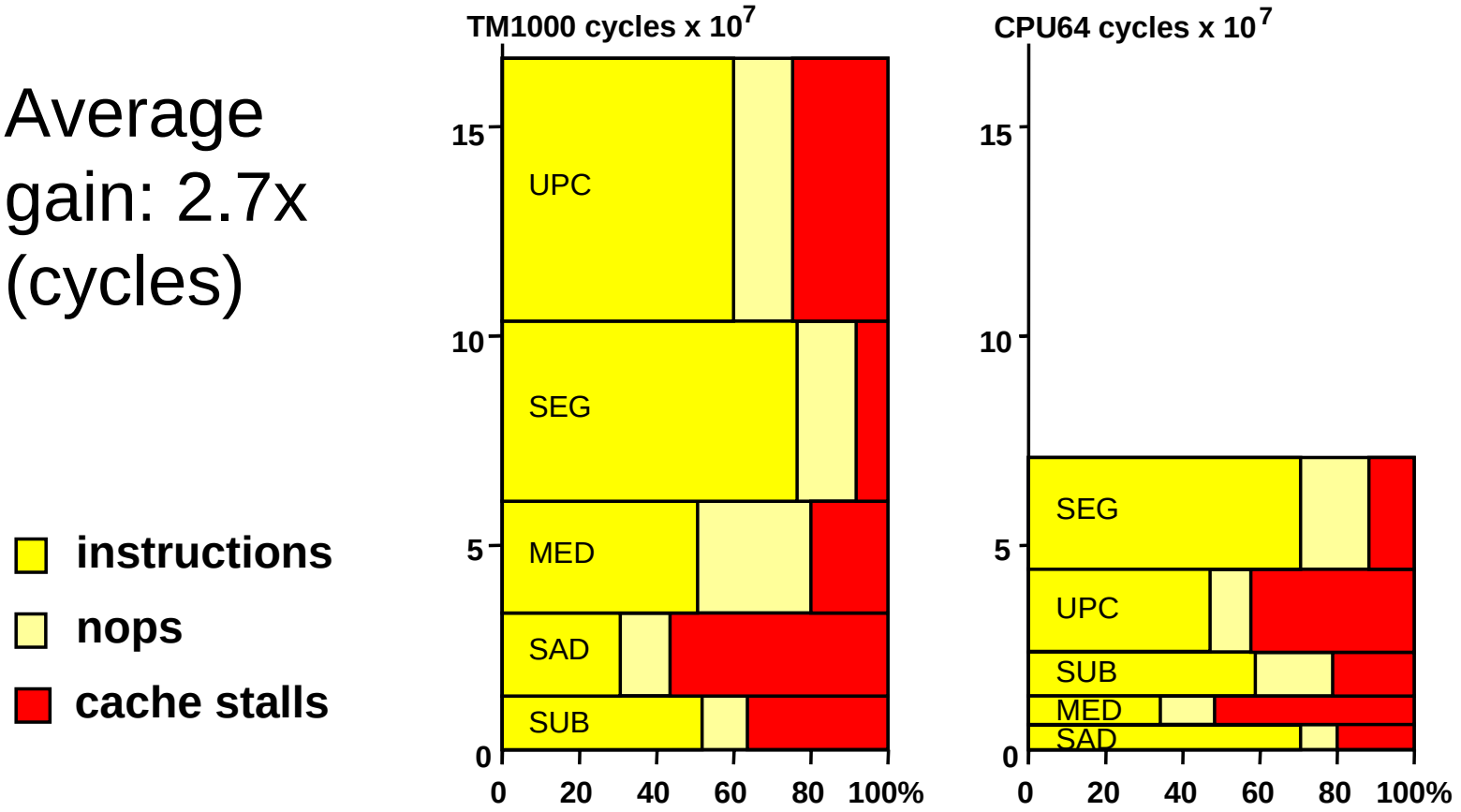
```
int calc_sad(vec64ub *prv, vec64ub *cur, int s)
{
    vec64ub left, right;
    int i; int sad = 0;
    for (i=0; i<8; i++)
    {   left = prv[i*s];   right = cur[i*s];
        sad += ub_me(left, right);
    }
    return(sad);
}
```

Outline

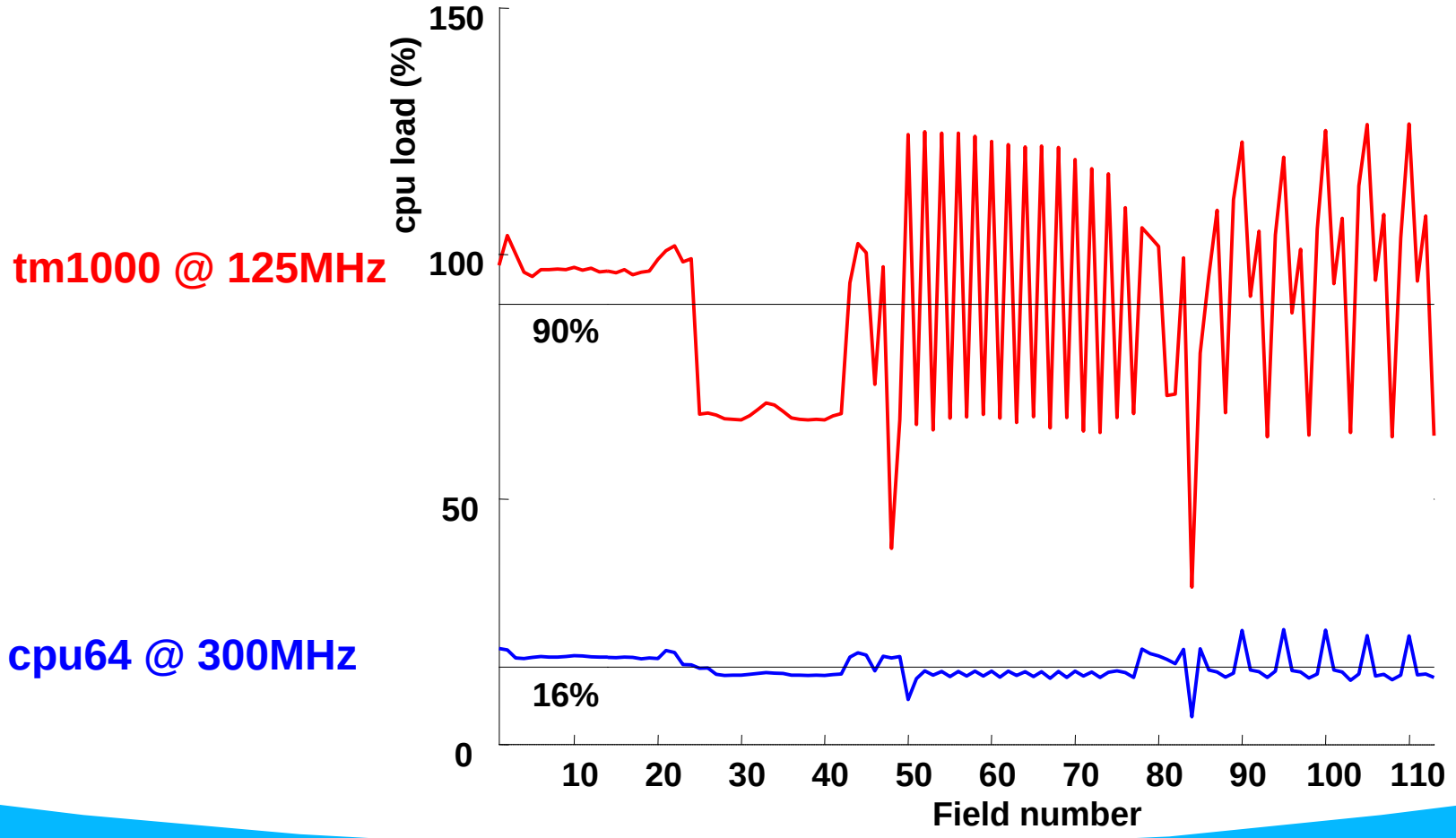
- Introduction
- Approach
- Benchmark suite
- Example
- Results

Results: Natural Motion

Average gain: 2.7x (cycles)



Natural Motion Dynamic Load



ICCD 1999/application-24

Summary

- Application domain:
  - Media processing for CE industry
- Benchmark suite:
  - Targeted to application domain
  - Optimized for class of processors
- Future products:
  - Gradual shift to software implementations of signal processing functions

TriMedia CPU64 Architecture

J. T. J. van Eijndhoven
Philips Research
Eindhoven, The Netherlands

ICCD 1999/application-26

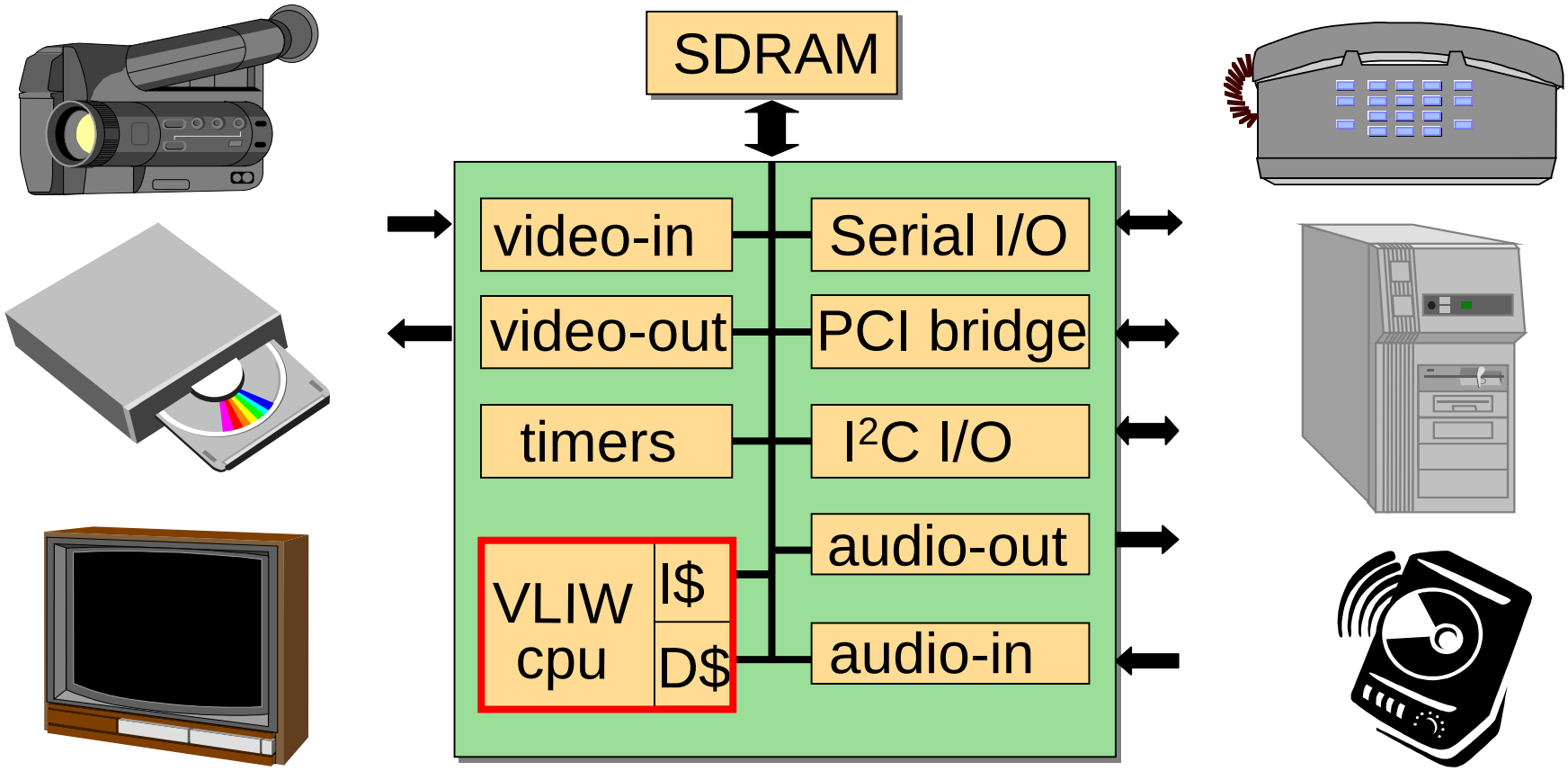
Outline

- Introduction
- VLIW architecture
- Instruction set
- IDCT example
- Status & conclusion

Application Target

- Processor core, to be embedded in different ICs and products
- Real time processing of media streams
- Cost-sensitive consumer electronics market

Embedded Application



Performance Target

Relative to TriMedia TM1000 (5-slot VLIW, 100MHz, 32-bit datapath, media operations):

- 6x to 8x performance increase, to process more or higher-resolution video streams
- Not more than double transistor count

Efficiency

Good performance/silicon cost ratio by:

- Optimize CPU architecture and media benchmark source code towards each other
- Solve resource conflicts at compile time

Outline

- Introduction
- VLIW architecture
- Instruction set
- IDCT example
- Results

Architectural Speedup

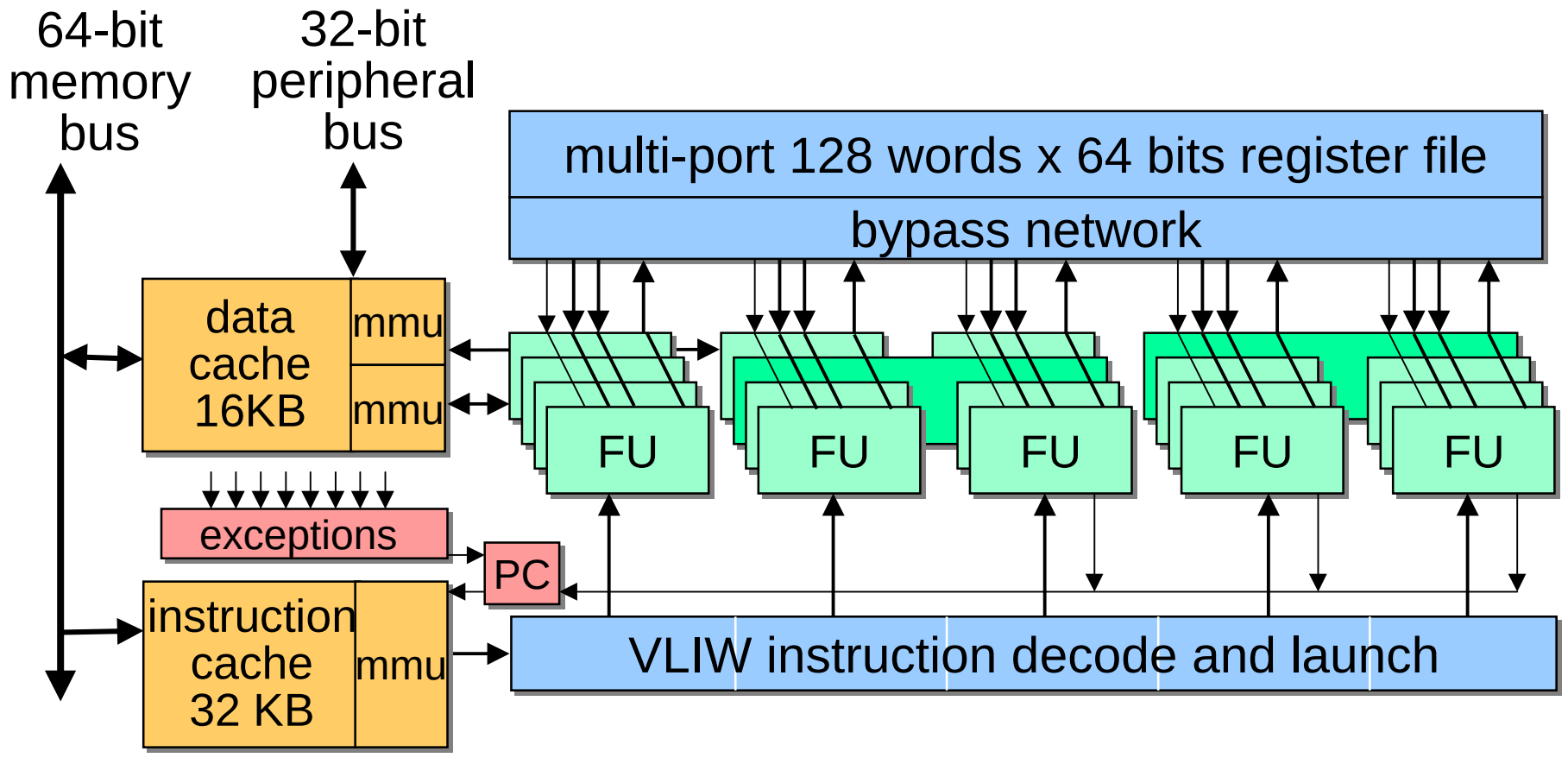
Not simply increasing the VLIW issue width:

- Diminishing gain of compiler-generated ILP
- Increasing implementation complexity (area, timing)

Architectural Speedup

- Extended SIMD: uniform 64-bit design, vectors of 1-, 2-, and 4-byte elements (data throughput per cycle)
- New, extensive, media instruction set (functionality per cycle)
- Improved cache control (prefetch, alloc)

CPU64 Architecture



Architecture VLIW+SIMD

- Issues a 5-slot instruction every cycle
- Each slot supports a selection of FUs
- All FUs support vectorized (SIMD) data
- Double-slot FU allows powerful multi-argument, multi-result operations
- All FUs are pipelined, latency 1 to 4 (except floating point divide and sqrt)

Outline

- Introduction
- VLIW architecture
- Instruction set
- IDCT example
- Results

Instruction Format

Per operation, per slot:

- Up to 2 arguments, up to 1 result
- Optionally an extra register for guarding (conditional execution)
- Immediate argument size can be 32 bit

Instructions have compressed variable length format, decompressed during decode

Operations

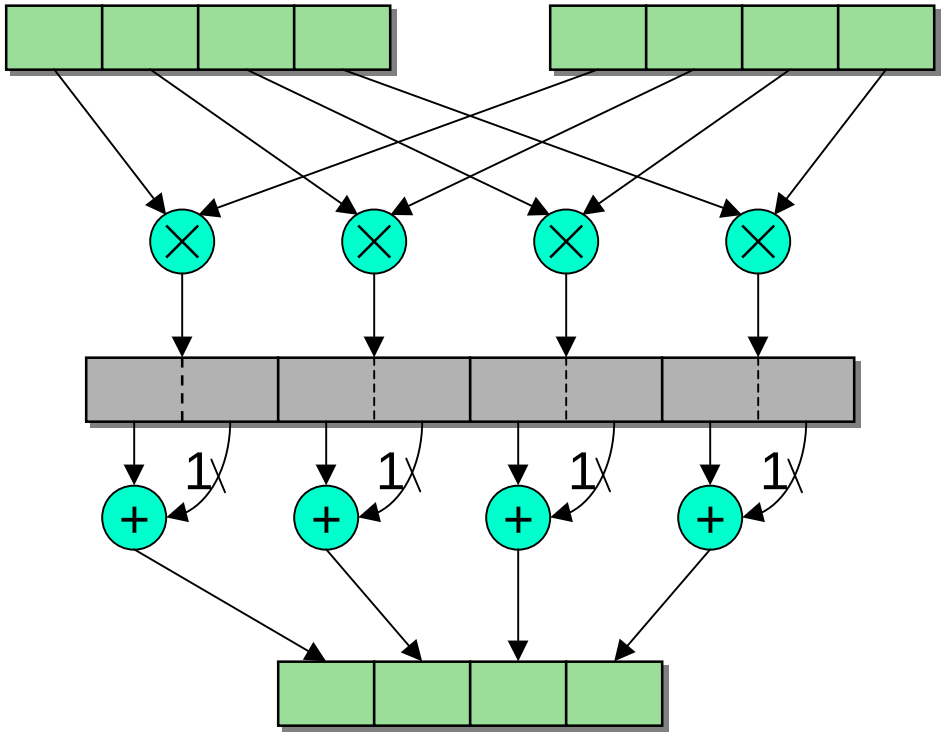
Intends to cover combinations of:

- 1-, 2-, 4-byte elements in an 8-byte vector
- Signed or unsigned type
- Clipped versus wrap-around arithmetic
- Set of basic functions
  (ld, st, add, mul, compare, shift, ...)

Instruction Set

Category	Cnt	Comment
Load/store ops	39	vectorized, endian-correct
Byte shuffles	67	vector type convert
Bit shifts	48	round, fields
Multiplies	54	round, clip, wrap around
Integer arith	104	clip, wrap around
Floating point	59	scalar and vectorized
Lookup table	6	vectorized
Special ops	23	MMU, cache, special regs
Branch	10	(un)interruptible, trap

Example: msp-multiply



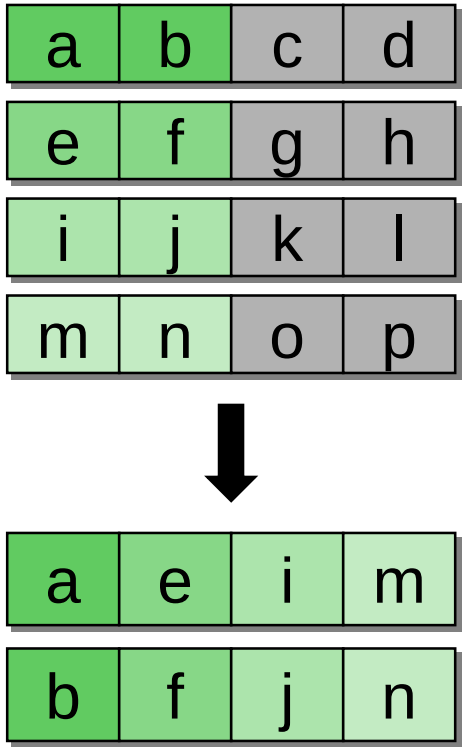
Each argument:
4-way 16-bit int

Multiply to
internal double
precision integer

Round lower half
into upper half

Return upper half

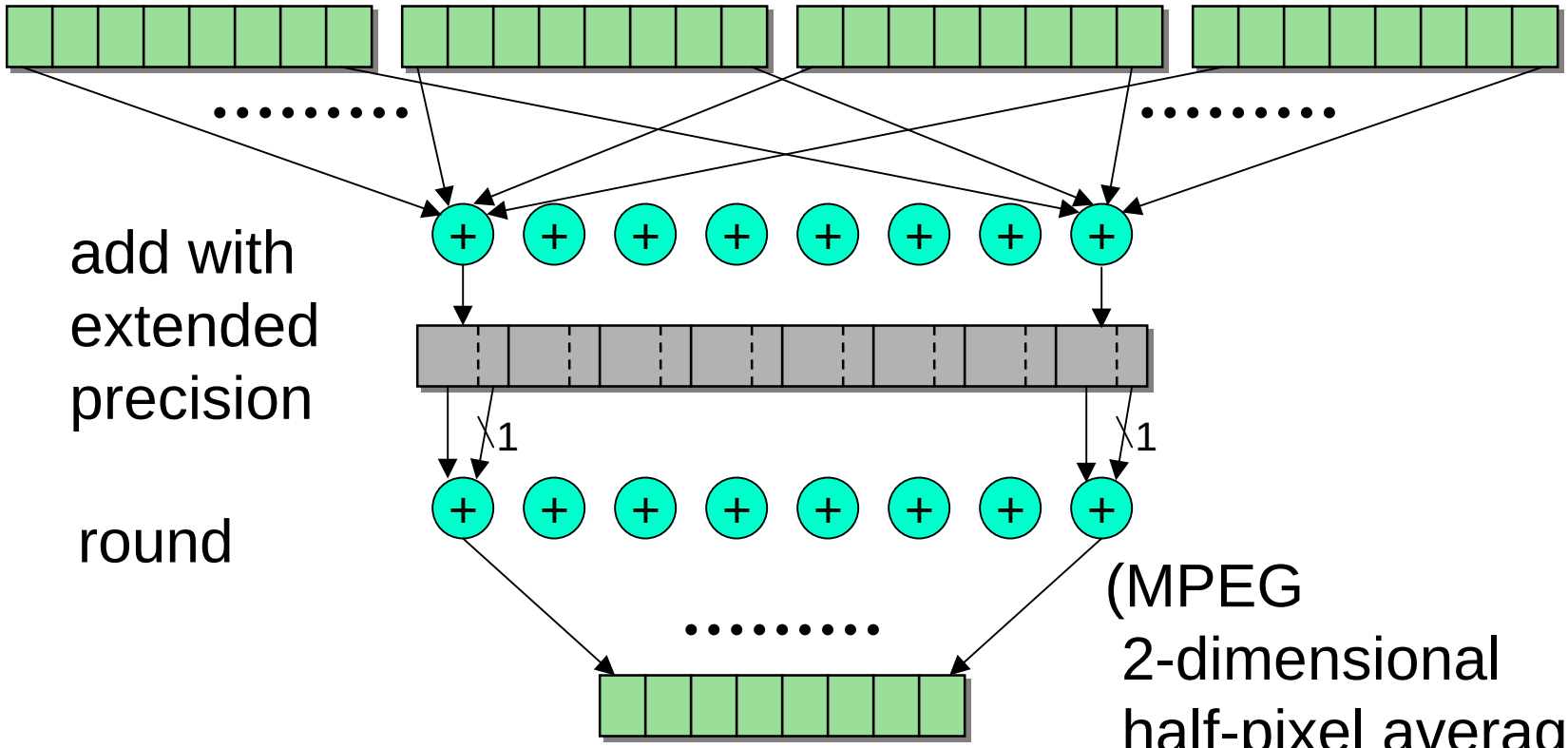
Example: Transpose



2-slot ‘super-op’:
takes 4 arguments,
shuffles bytes,
produces 2 results

(2-dimensional
filtering)

Example: 4-way average



Branches

- Branch operations have 3 branch delay slots
- Compiler & scheduler try to fill these (profiling, loop unrolling, inlining, guarding)
- Branch units are properly pipelined
- Up to 3 branch ops in 1 instruction
- No branch prediction hardware
- Branches are preferred moments for interrupt servicing

Memory Management Units

- Separate IMMU and dual-port DMMU
- Variable page size: 4K, 16K, … ,16M
- Indexed with 32-bit VA and 8-bit task ID
- Inter-task, X/W, and supervisor-mode protections
- TLBs are software managed through precise exceptions

Outline

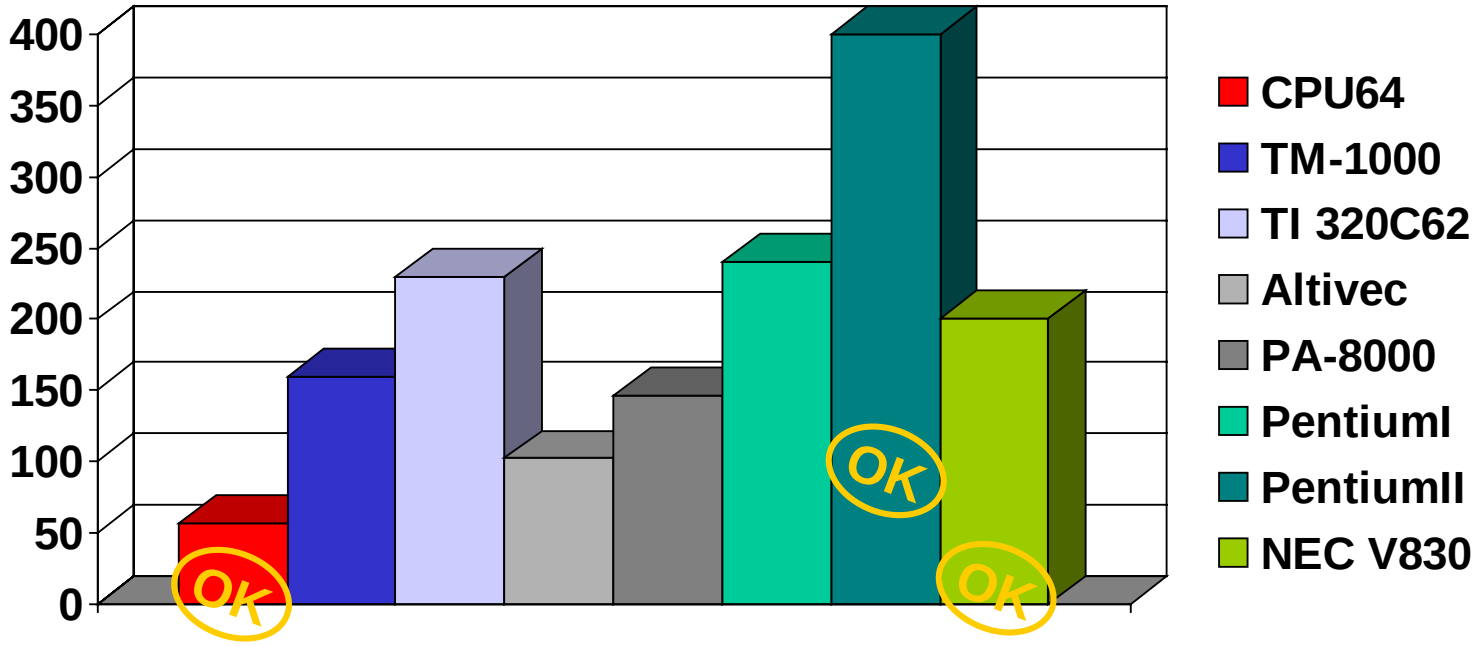
- Introduction
- VLIW architecture
- Instruction set
- IDCT example
- Results

2D-IDCT Example

- IDCT code was tested early in the project, also for studying the programming model
- Mapped through our experimental C compiler and instruction scheduler
- Operates entirely on (vectors of) 16-bit data elements
- Simulation showed IEEE 1180 accuracy compliancy

2D-IDCT Instructions

Execution time in cycles, excluding data cache misses



OK = accuracy compliant

ICCD 1999/application-48

Outline

- Introduction
- VLIW architecture
- Instruction set
- IDCT example
- Status & conclusion

Status

Now in construction at Philips Semiconductors, Sunnyvale

- 7M transistors (target)
- 300Mhz clock (target)
- 0.18µ technology
- 1.8 volt

Conclusion

- The 5-slot 64-bit VLIW provides a powerful architecture for media processing
- Performance gain of 6x to 8x on TM1000
- Rich and regular media instruction set
- Powerful multi-argument, multi-result ‘super-op’ naturally fits VLIW architecture
- Embedded DSP supports robust multi-tasking through MMU

TriMedia CPU64 Application Development Environment

E.J.D. Pol
Philips Research
Eindhoven, The Netherlands

ICCD 1999/application-52

Outline

- Introduction
- Machine description
- Vector models
- Compilation trajectories
- Optimization
- Conclusion

Basic architecture

- CPU64 is based on TM1000 VLIW DSPCPU
- Application domain: digital media processing
- Register-rich
- Custom operations
- Applications show several levels of parallelism

Parallelism

- MIMD
  - Implicit in program (C)
  - Compile-time scheduling (ILP)
- SIMD
  - Explicit in program (vector types)
  - Mixes well with MIMD
- Task level parallelism

Instruction Set

- General-purpose (scalar) operations for control
- Special-purpose (custom, vector) operations for media processing
- 5 scalar types (8,16,32,64 bit ints, 32 bit floats)
- 7 vector types (8,16,32 bit (un)signed ints, 32 bit floats)

Goal

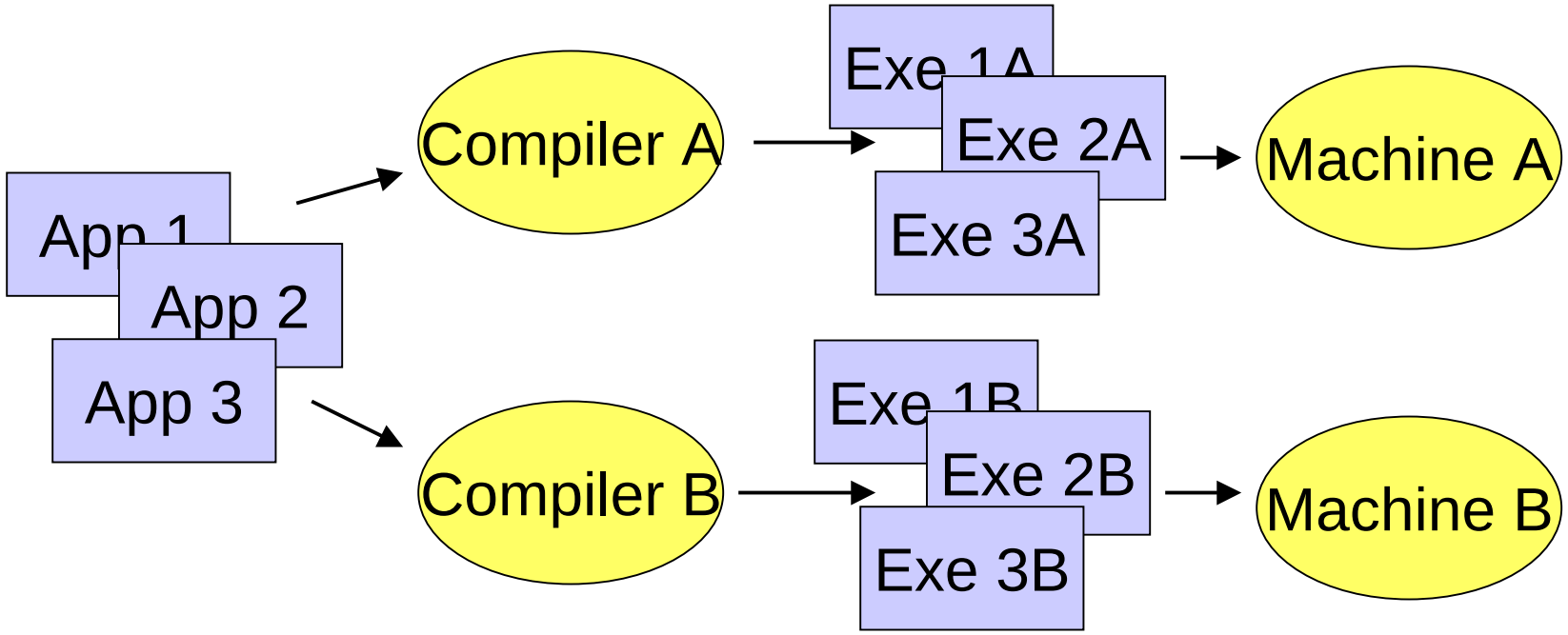
Make application development
as simple as possible!

- Higher levels of parallelism
- More special-purpose hardware
- Range of CPUs will be available

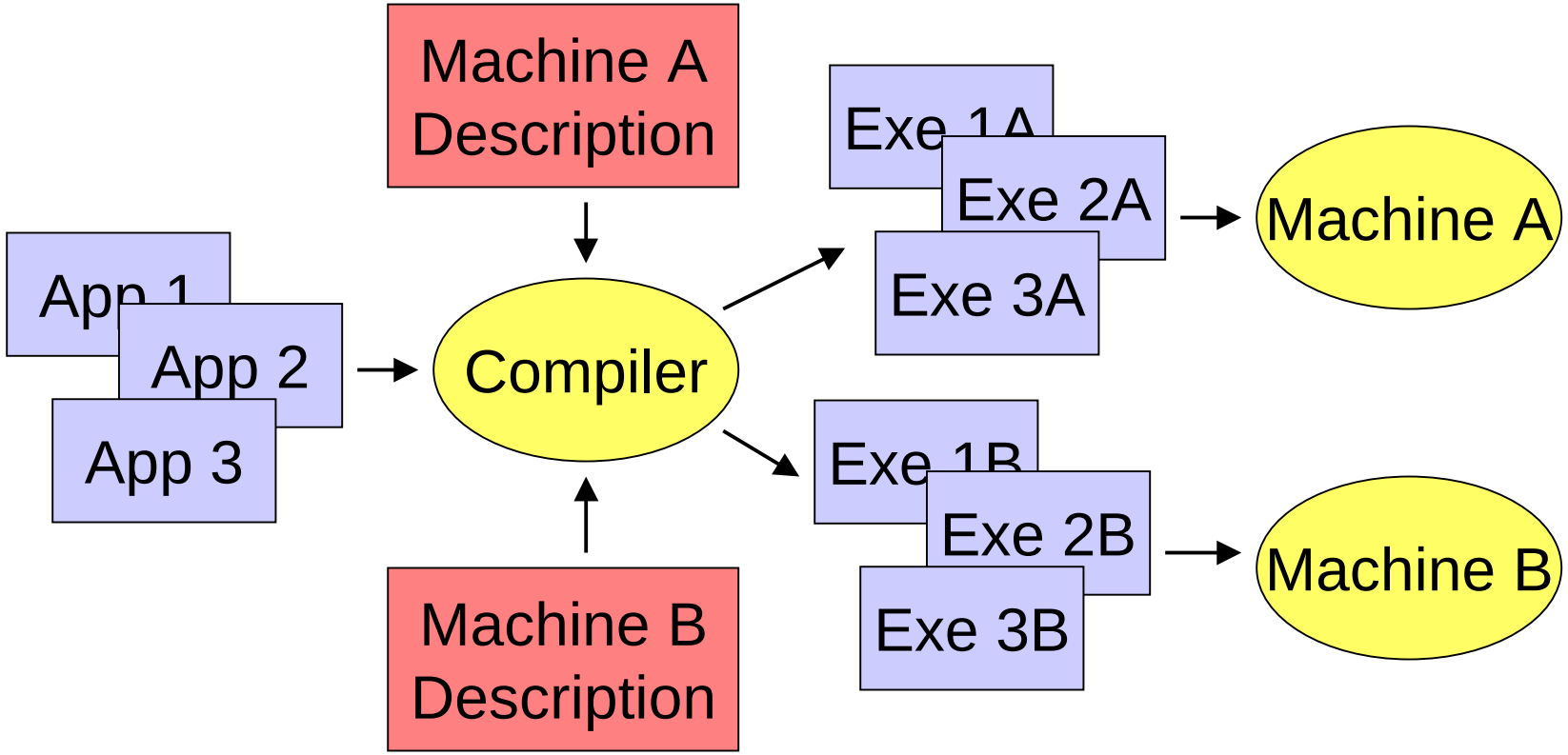
Outline

- Introduction
- Machine description
- Vector models
- Compilation trajectories
- Optimization
- Conclusion

Traditional Compilation



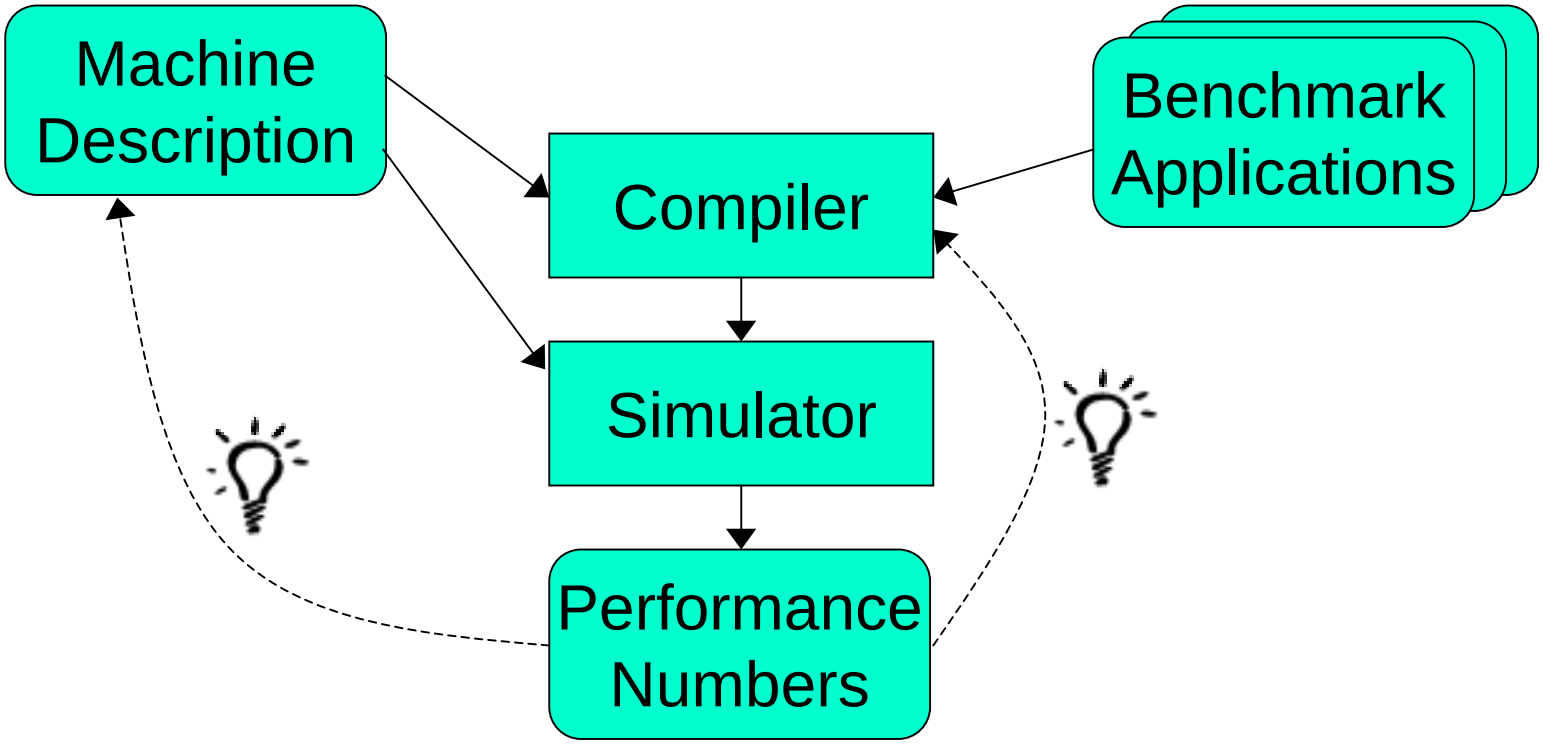
Retargetable Compilation



MD file contents

- MD describes instance from class of machines
- Contains information such as:
  - number & width of registers
  - number of issue slots
  - number & placement of function units
  - instruction latencies

Y-Chart



Outline

- Introduction
- Machine description
- Vector models
- Compilation trajectories
- Optimization
- Conclusion

Memory Endianness

- Endianness affects storage of scalars
- Programmability requires vectors to be subranges of scalar arrays
- C address arithmetic requires increasing address with increasing array index

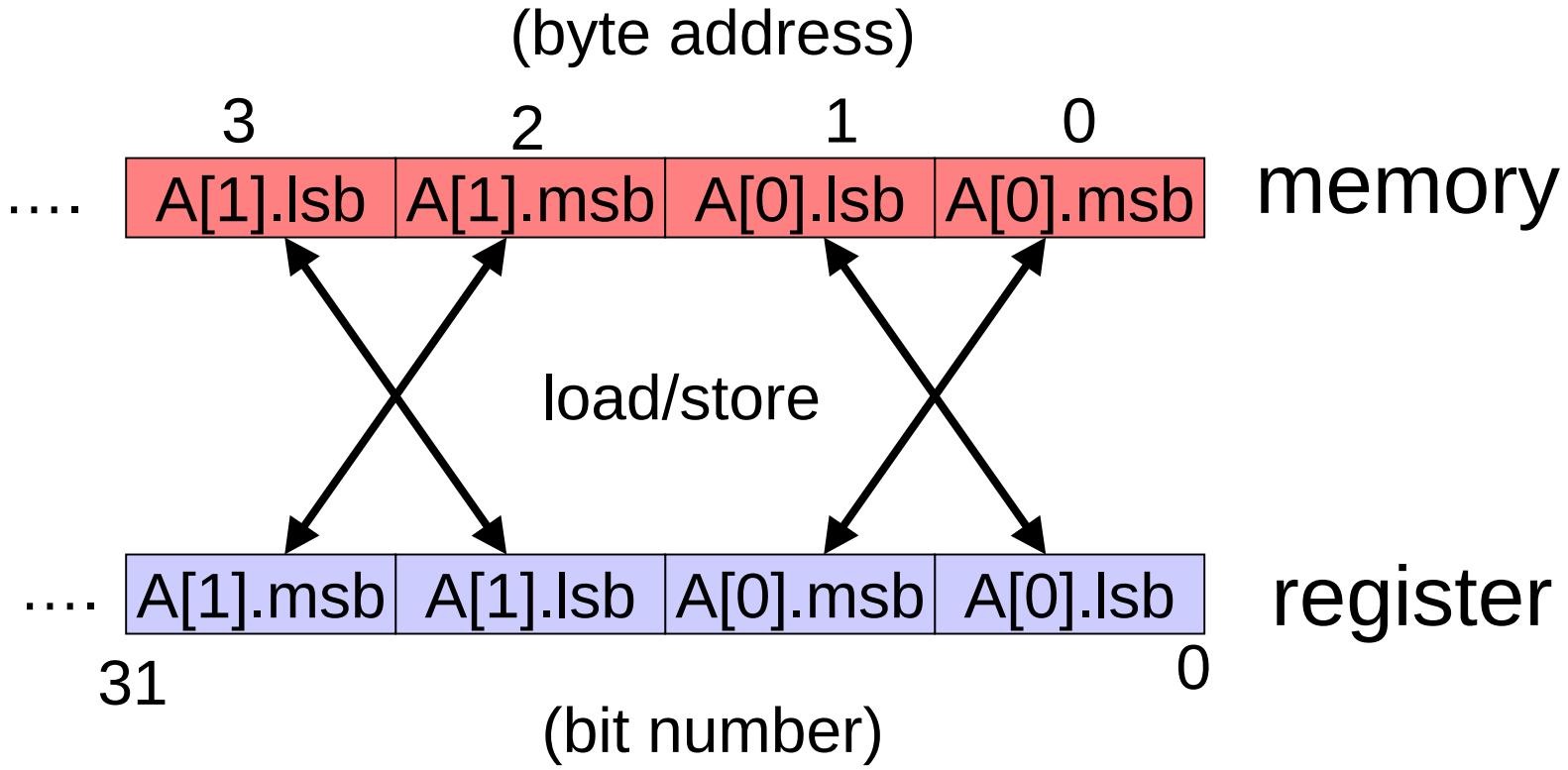
Register Endianness

- Endianness does not affect storage of scalars
- Specific vector elements are addressed by certain custom ops (if necessary)
- Endianness might affect ordering of vector elements in registers

Endianness design choices

- Memory endianness is fixed for intra-element order (C requirement)
- Register endianness is fixed for both inter- and intra-element order, but it is irrelevant which (programming ease)
- Load/Store operations translate inter-element order between registers and memory

Vector Load/Store operations



Outline

- Introduction
- Machine description
- Vector models
- Compilation trajectories
- Optimization
- Conclusion

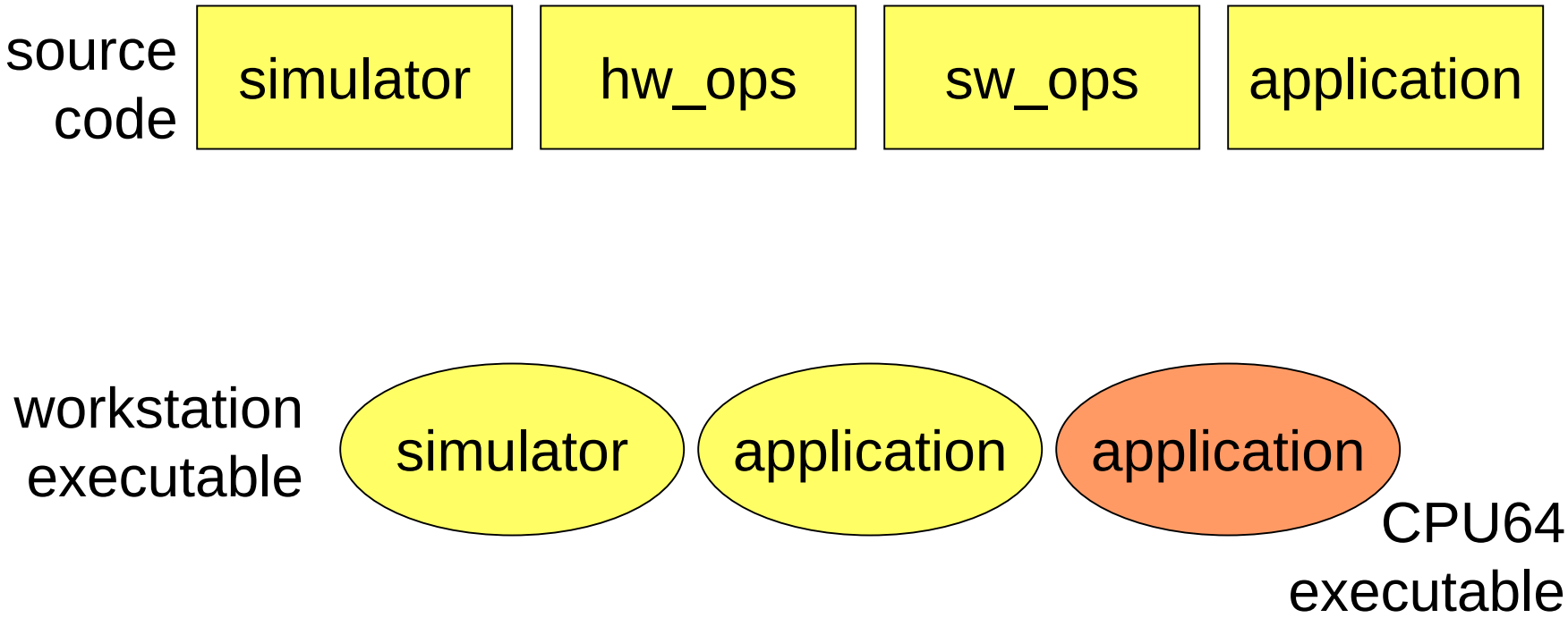
Software and Hardware Operations

- Hardware view of instruction set
  - Implement minimal, changeable set
- Software view of instruction set
  - provide regular, orthogonal, stable set
- Two instruction sets were defined for these purposes

Instruction set libraries

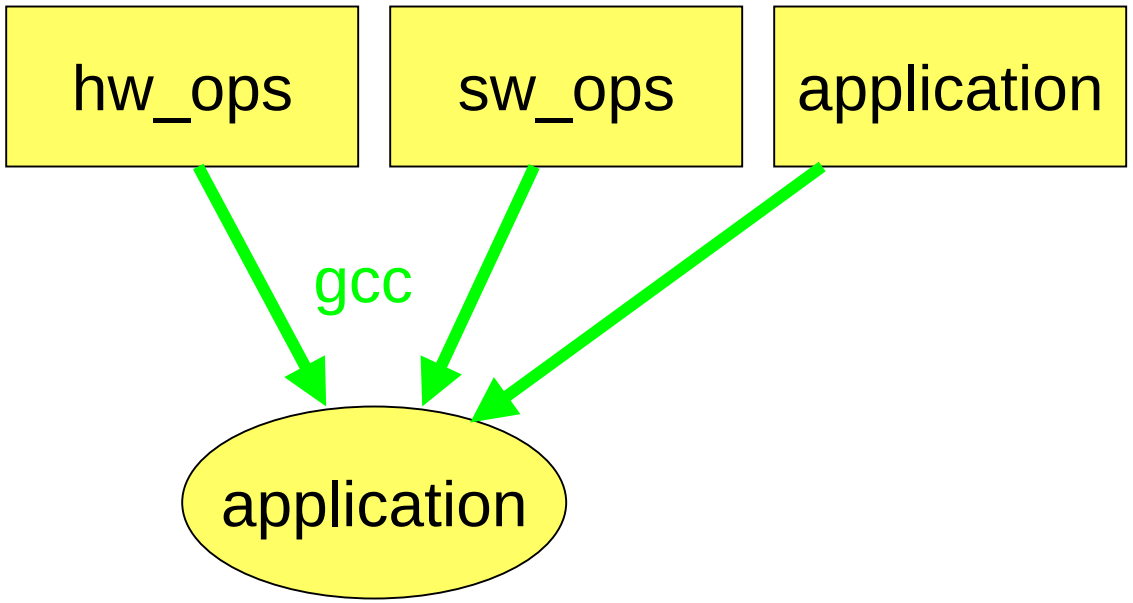
- Hardware operations library
  - untyped, minimal, changing
- Software operations library
  - vector-typed, orthogonal, stable
- Mapping in MD file and library

Library structure



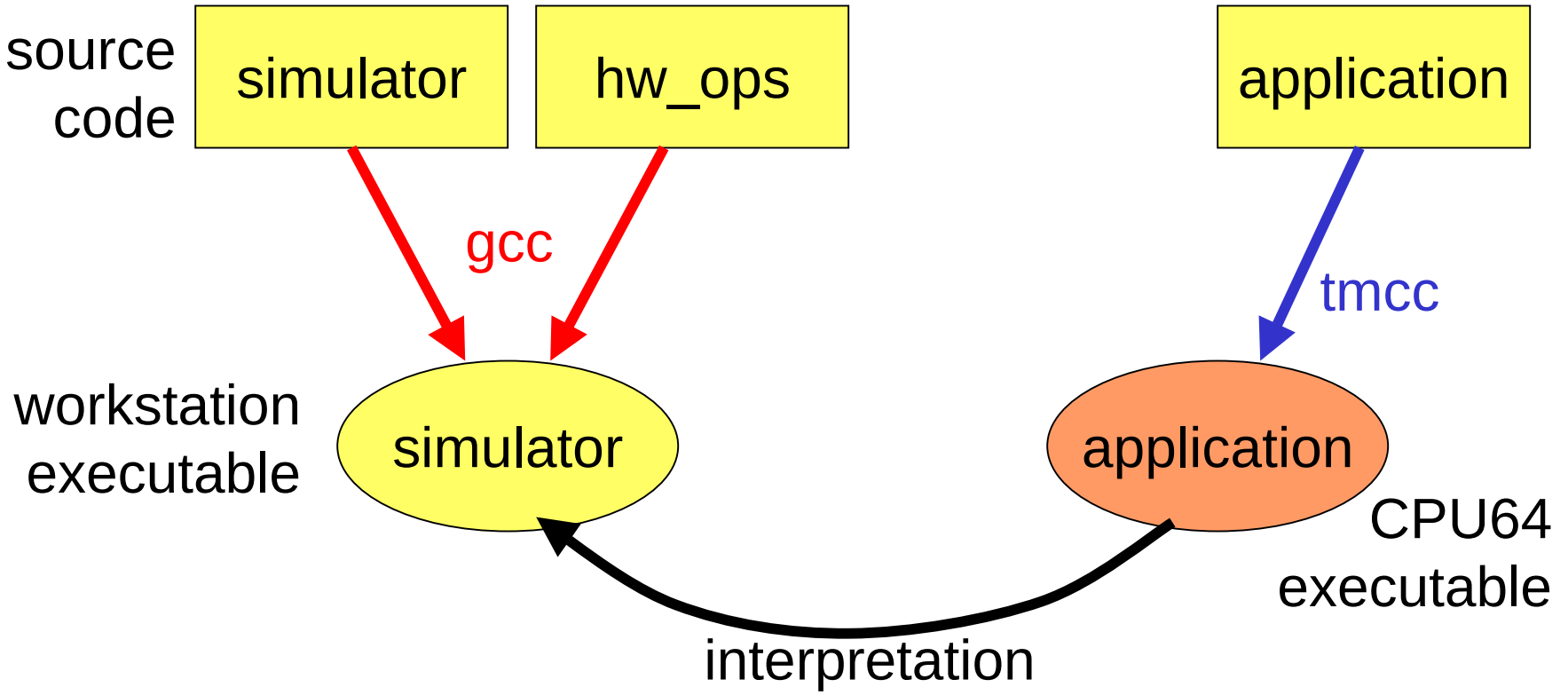
Functional Development

source code



workstation executable

Code Tuning



Outline

- Introduction
- Machine description
- Vector models
- Compilation trajectories
- Optimization
- Conclusion

General guidelines

- Design application architecture (data flow)
- Determine data formats (scalar/vector types)
- Arrange data locale (memory/cache/registers)
- Design control architecture (loop structure)
- Optimize computations (custom-ops)
- Fine tune cache behavior (prefetch/alloc)

Outline

- Introduction
- Machine description
- Vector models
- Compilation trajectories
- Optimization
- Conclusion

Conclusions

- Applications are written in C code only
- Machine Description supports changes in ISA
- Vector types keep code legible
- Endianness is not visible in application code
- Fast functional development track
- Accurate application tuning track

TriMedia CPU64 Design Space Exploration

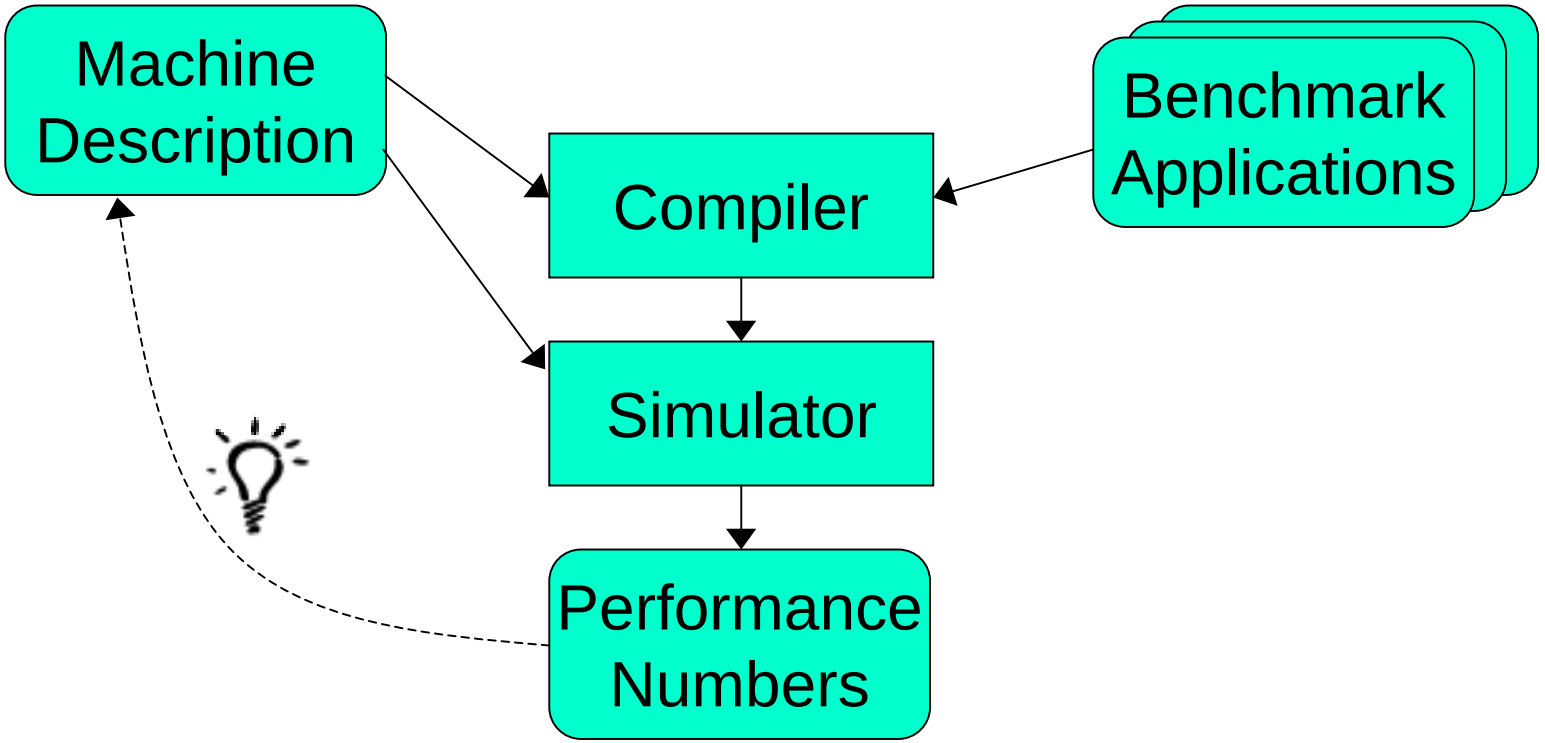
G.J. Hekstra
Philips Research
Eindhoven, The Netherlands

ICCD 1999/application-78

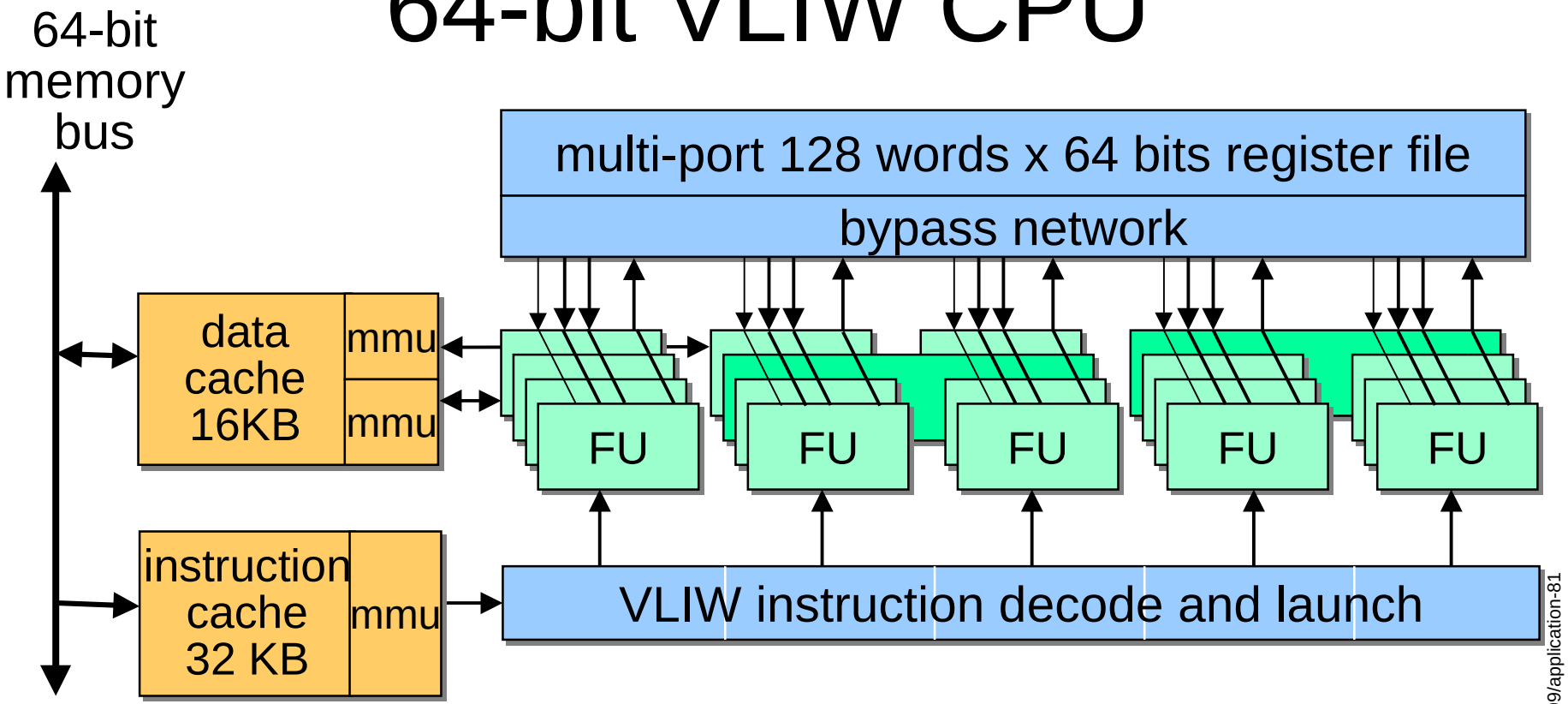
Outline

- Introduction
- Tools
- Exploration
- Real numbers
- Conclusions

Methodology: the Y-chart



64-bit VLIW CPU



- Many parameters, large design space(s)

ICCD 1999/application-81

Multimedia benchmark

- Nine multimedia applications from:
  - Data communication
  - Audio coding
  - Video coding
  - Video processing
  - Graphics
- Representative
  - Applications
  - Code
  - Datasets

Challenge for design space exploration

- Simulation time for the benchmark for a single machine is 18 hours
- The number of design points for functional unit configuration alone: $\approx 10^{15}$
- Results in 2000,000,000,000 years of computation time

Outline

- Introduction
- Tools
- Exploration
- Real numbers
- Conclusions

Essential tooling

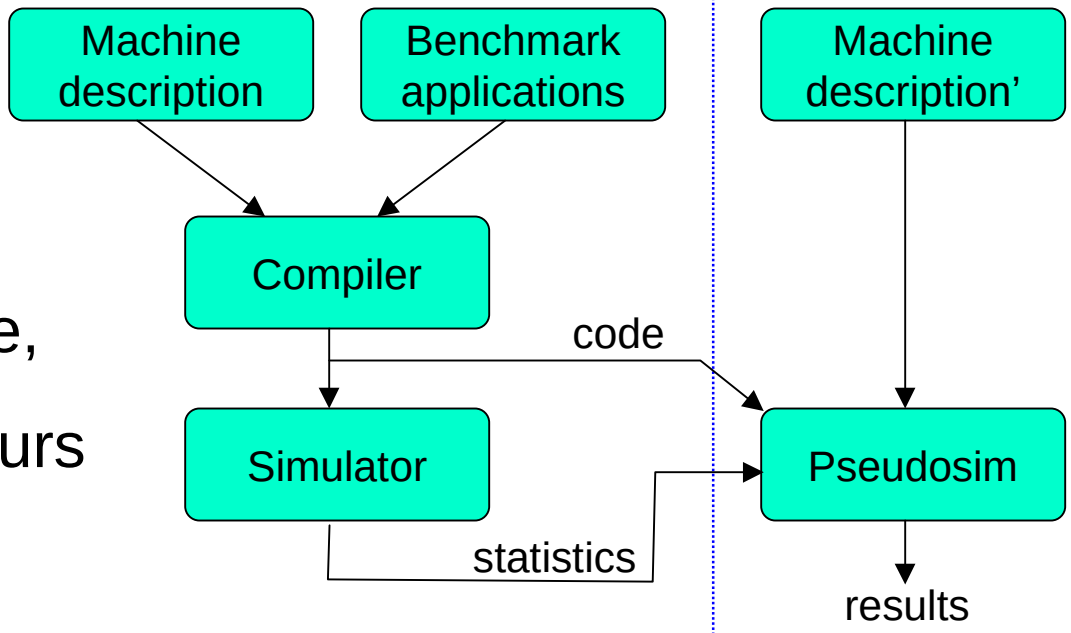
- Retargetable toolchain
  - Core compiler, scheduler, simulator
- Design and experiment management
- DSE support tools
  - Machine generation, pseudosim, analysis
- Glue software

Pseudo-simulation

Pseudosim: a retargetable pseudo-simulator

- Performs a *cycle-accurate calculation* of the amount of instruction cycles, without doing the actual machine simulation
- Gathers other machine statistics, such as utilisation of slots, functional units, operations
- Time for the whole benchmark is reduced from 18 hours to 4 minutes

Pseudo-simulation



once,

18 hours

many times,

4 minutes

ICCD 1999/application-87

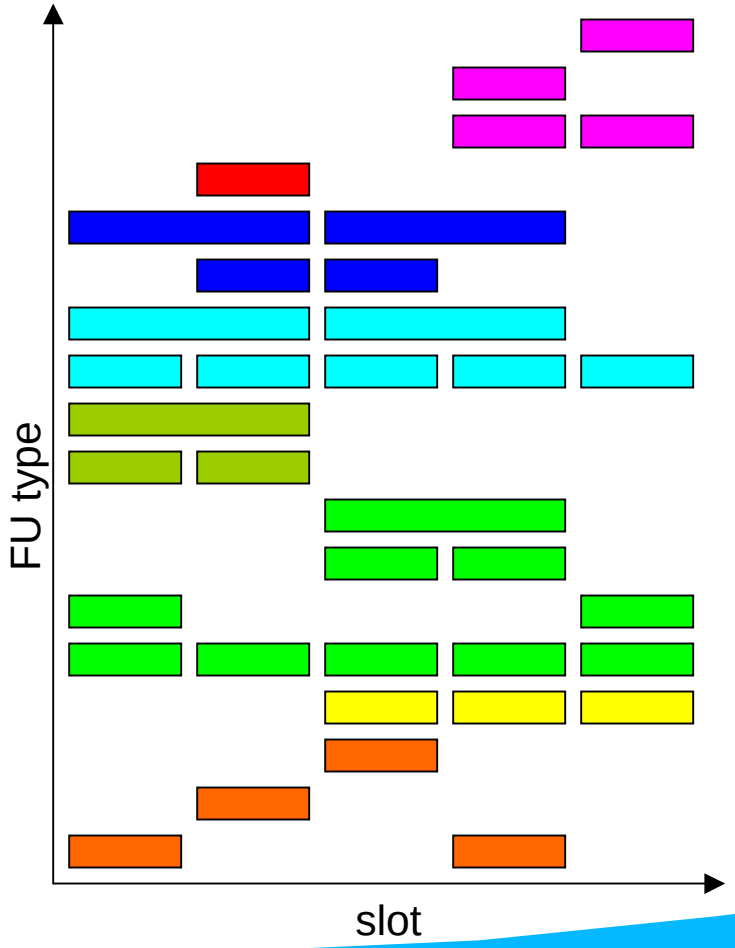
Outline

- Introduction
- Tools
- Exploration
- Real numbers
- Conclusions

Functional unit configuration

Problem:

- How many of each type of FU do I need?
- Where do I place them?
- How do I prevent simulating too many machine variations?



ICCD 1999/application-89

Naïve calculation of the design space size

- 24 types of FU's
  31 configurations per type
- 6 types of super-op FU's
  7 configurations per type

Size of design space: $31^{24} \times 7^{6} \approx 10^{40}$

Not so naïve calculation of the design space size

- Assume relationship between FU types
- Best-guess lower and upper bounds on number of FU's per type
- Reduce permutations

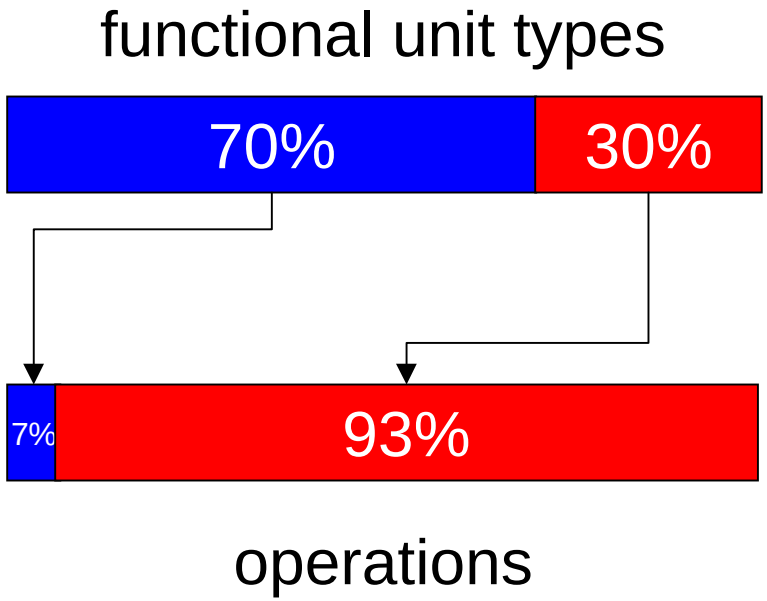
Size of design space: $\approx 10^{15}$

Systematic exploration

- It is not feasible to exhaustively compute all design points in the design space
- Therefore we probe, partition, and then explore the design space
- Careful set-up of experiments

Reduction of the design space

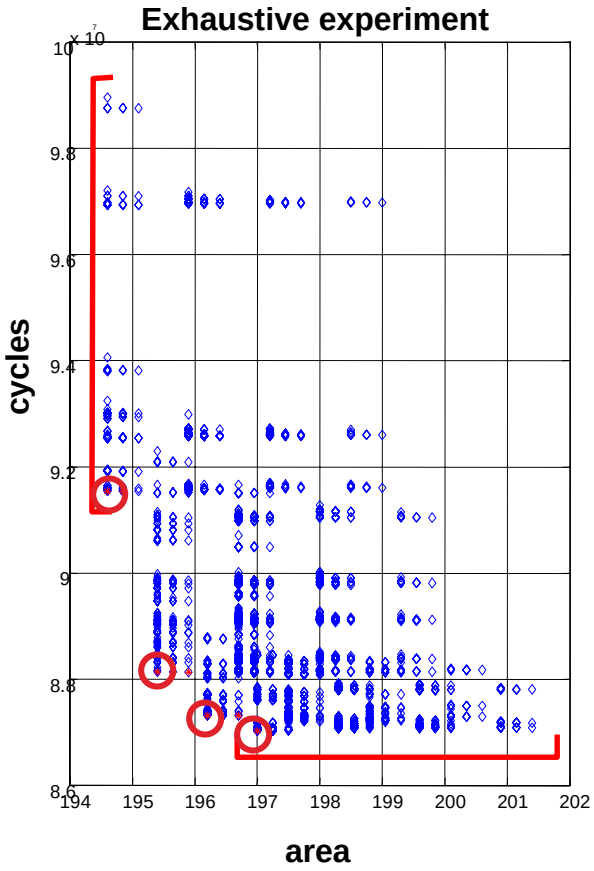
- Observation: over 93% of operations are done in 30% of FU types
- Action: partition space
  - Exhaustive exploration of important FU's
  - Greedy exploration of the remainder



ICCD 1999/application-93

Exhaustive experiment

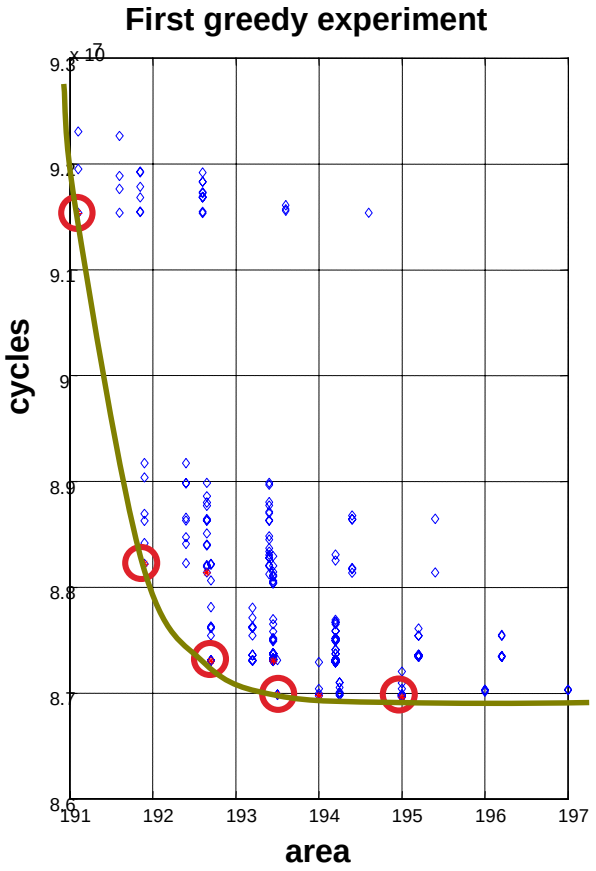
- Close to 3000 machine variations
- Only 4 machines survive
- Large performance variation due to placement
- Time taken = 200 hours



ICCD 1999/application-94

Greedy experiments

- Less machine variations per experiment
- More machines survive
- Less performance variation due to placement
- Close to another 3000 machine variations over all greedy experiments



Outline

- Introduction
- Tools
- Exploration
- Real numbers
- Conclusions

Simulated machines

Experiment	number
Probing	185
Exhaustive	3023
Greedy	2519
Total	5727

- 180 Gbyte data transferred
- 800 Mbyte compressed data stored
- 2T cycles simulated
- 10T operations issued

Outline

- Introduction
- Tools
- Exploration
- Real numbers
- Conclusions

Summary and conclusions

- A full-fledged design space exploration was done for the 64-bit TriMedia VLIW CPU core
- The use of both pseudo-simulation and a systematic exploration has made this feasible
- The outcome is:
  - A range of machines in A-T space
  - Rules for functional unit placement
  - A flexible, integrated environment for continuation of DSE

Summary and conclusions

- A large amount of time is spent in developing tools to enable the DSE
- The design of experiments and the analysis of results takes up more time than the execution
- Performing a DSE stretches the capabilities of the tools to the maximum
- The pseudo-simulator is a powerful tool that supports the full design space offered by the machine description