High speed video de-interlacing with a programmable TriMedia VLIW core

Bram Riemens
Robert Jan Schutten, Kees Vissers

Philips Research Laboratories, Eindhoven, The Netherlands

Outline

Introduction

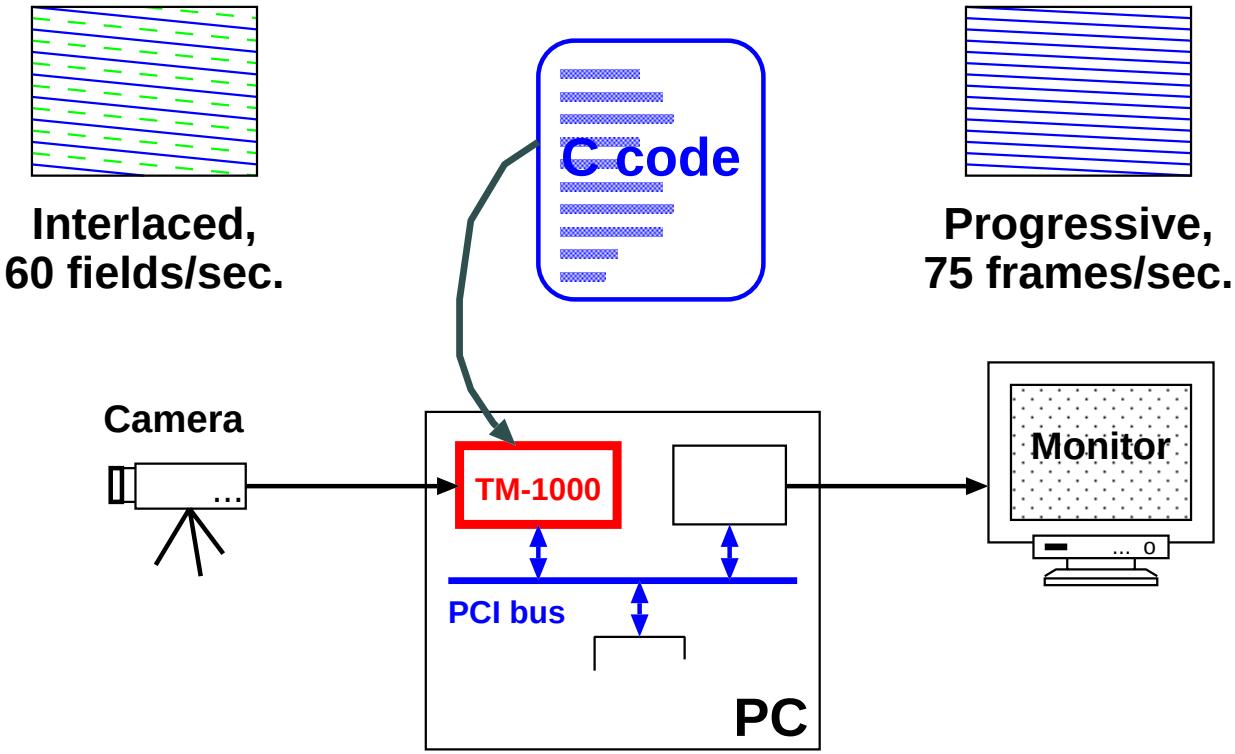
- Application and problem statement
- Architecture: TriMedia TM1000 processor
- Programming environment
- Algorithm: filter for video signals

Programming the TM1000

- Methodology
- Step by step description

Results and conclusions

Application

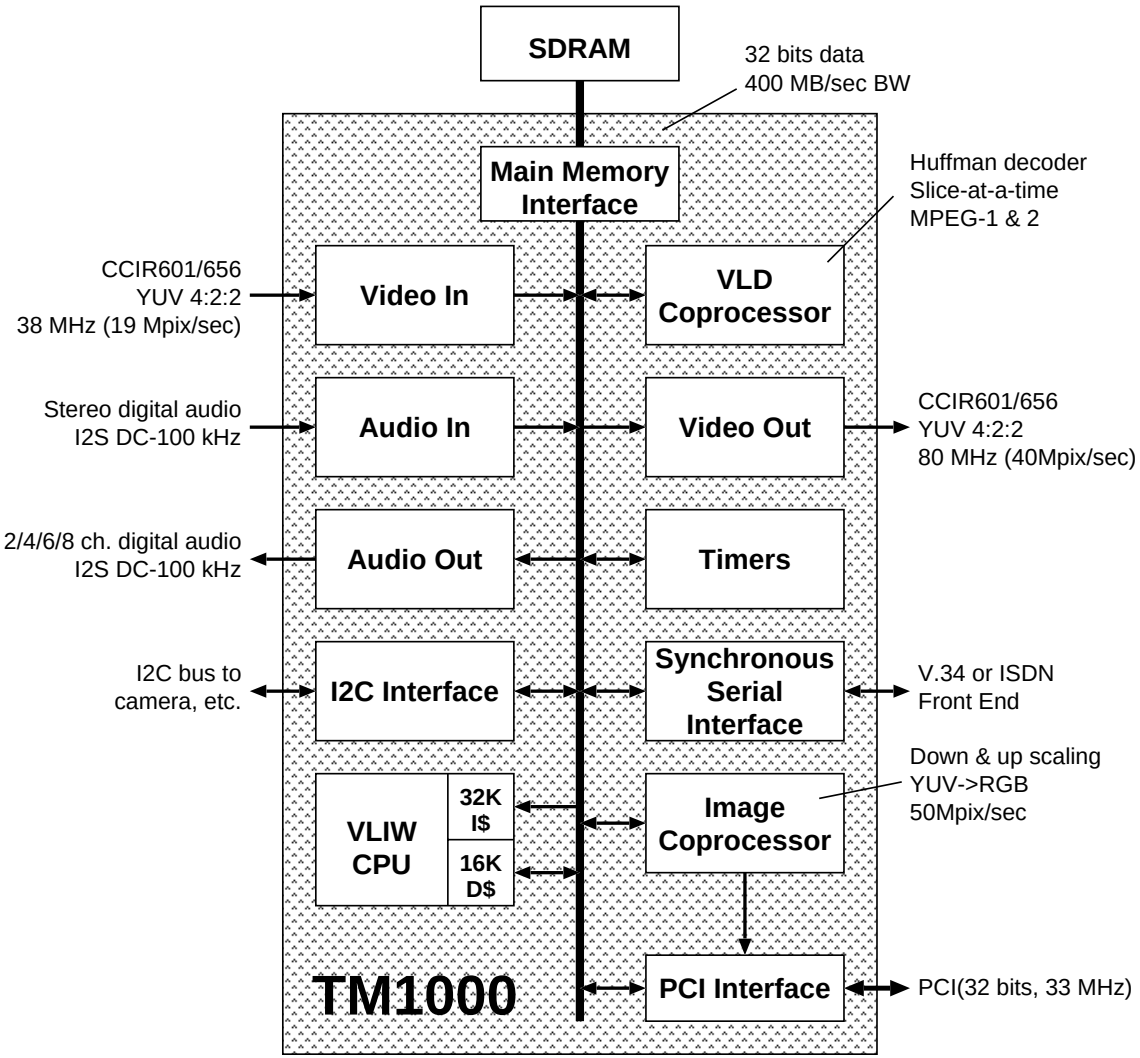


Software implementation on a TM1000 programmable processor.

TM1000 characteristics

Overview

- Media processor
- DMA peripherals for audio, video and graphics
- Stand-alone operation or PC related system

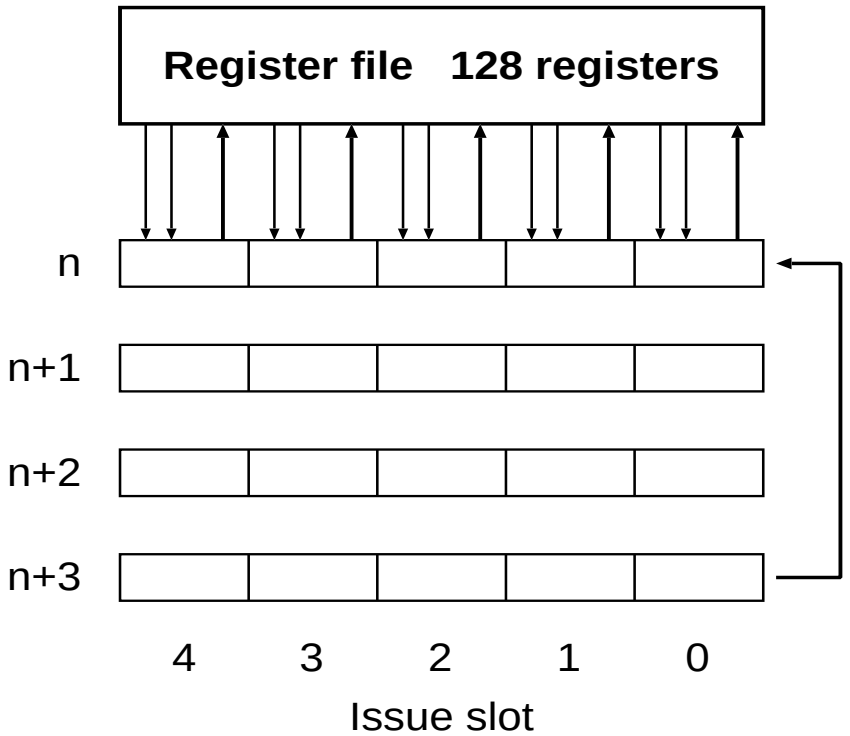


PHILIPS

TM1000 architecture

Processor core

- 32 bit architecture, 5 issue slot VLIW CPU
- Floating point and integer and DSP instructions
- 128 general purpose registers



PHILIPS

Programming environment

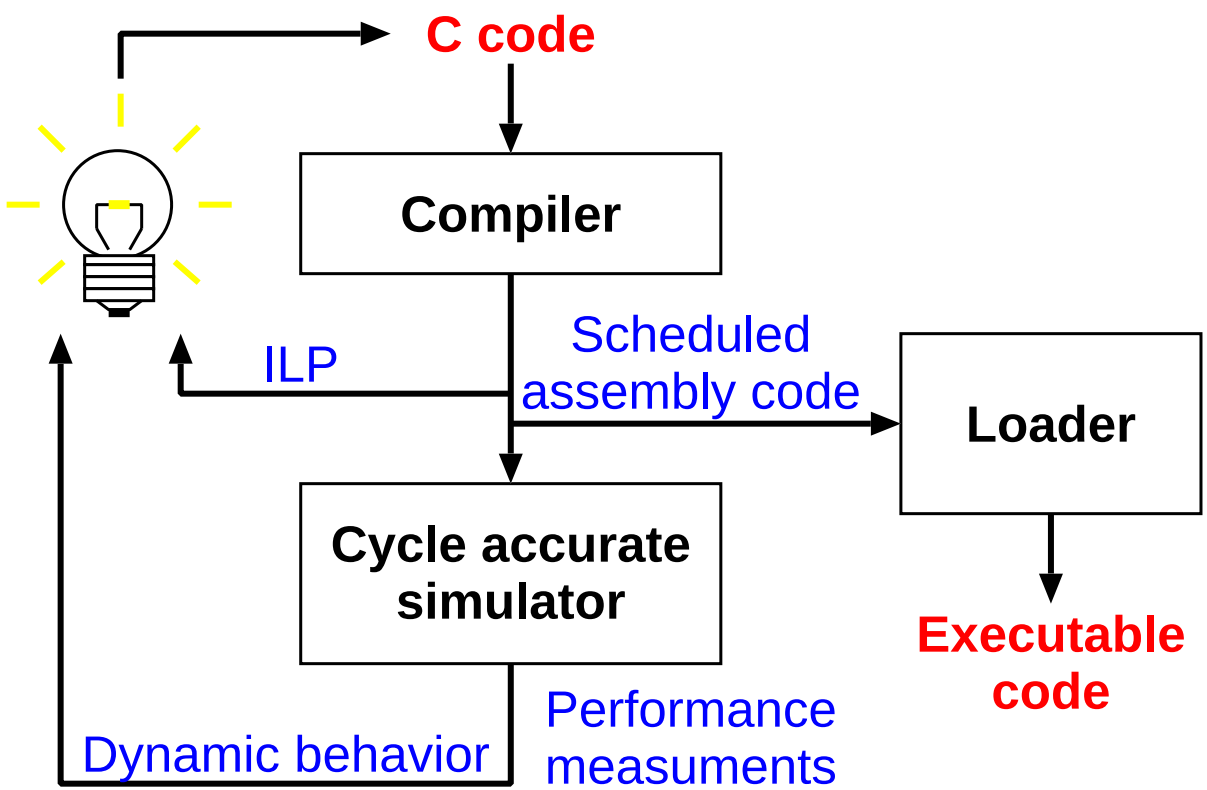
Structured approach

- High level C language, compiler and simulator tool set:

  Modify C –> Run simulator –> Analyze performance –>

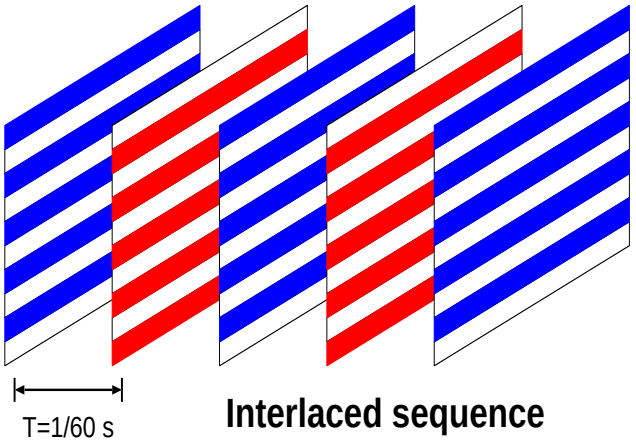
- Limited amount of performance indicators:
  - Total number of cycles for a test run (% of CPU load)
  - Instruction level parallelism (ILP) –> exploit VLIW core
  - Cache performance

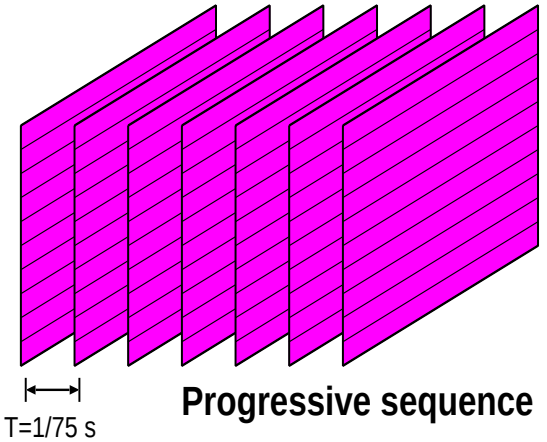
Tool chain

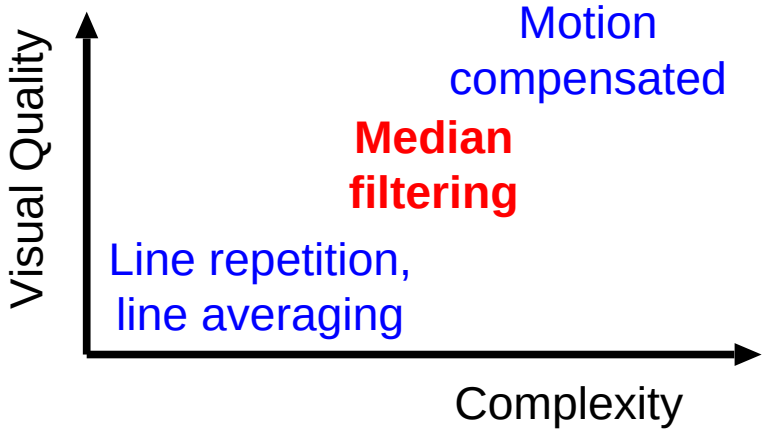


PHILIPS

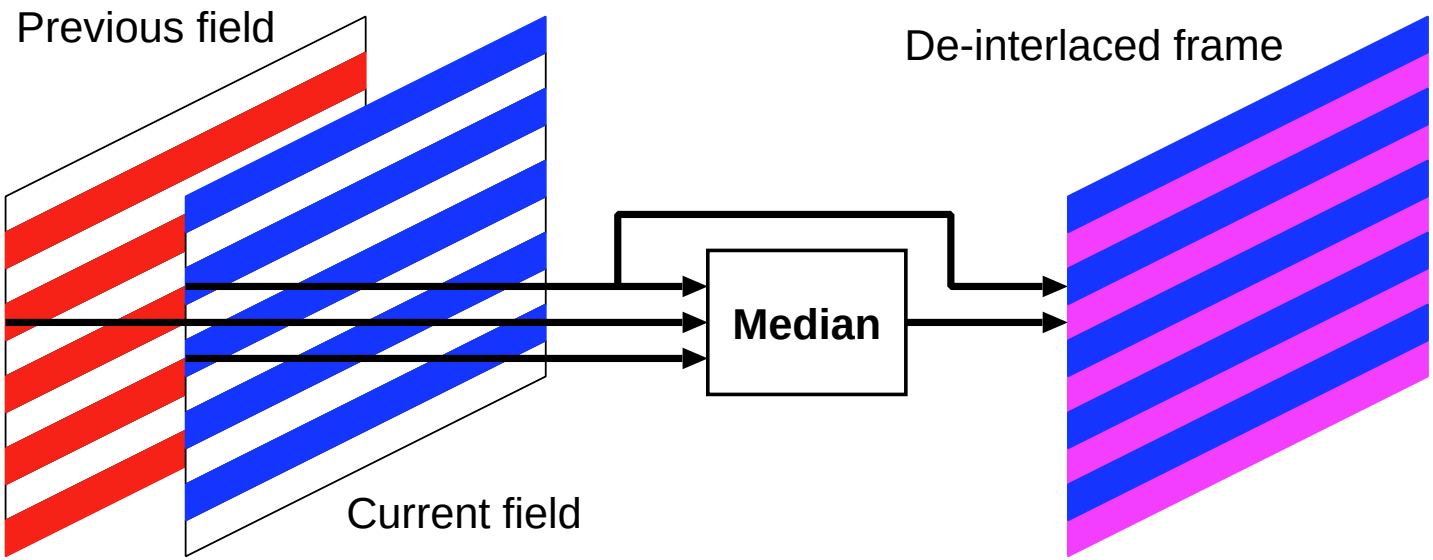
Scan rate conversion







Median filter de-interlacing algorithm



254 (lines) * 720 (samples) * 2 (Y/C) * 60 (fields) = 22 M med/sec

100MHz TM1000: 4.5 clock cycle/median

Functional reference code

```
pixel_t Median (pixel_t a, pixel_t b, pixel_t c)
{
    pixel_t   high, low;

    high = if (a>b) { if (a>c) a else c } else { if (b>c) b else c };
    low = . . . .
    return(a + b + c - high - low);
} /* end of Median */
```

```
for (lin = all lines) {
    for (pix = all pixels) {
        out_frm[(2*lin+1)*ppl+pix] = Median(cur_fld[lin*ppl+pix],
                                cur_fld[(lin+1)*ppl+pix], prv_fld[lin*ppl+pix]);
    }
}
```

Result: 594% of TM1000 performance

Reduce number of executed instructions

Step 1: Avoid function call and if-then-else branches

Alternative way of calculating median

Use “imin” and “imax” dedicated functions

Supported in the compilation chain;
translated into a single machine instruction

```
#define Median(a,b,c)      (imin(imax( imin(a, b), c), imax(a, b)))
```

Result: 319% of TM1000 performance

Reduce number of executed instructions

Step 2: Rewrite the inner loop body

- Address calculations outside the loop body
- Inform the compiler to avoid false data dependencies

  No aliasing for pointer dereferences, use *restricted pointers*

  Proposed new feature in ANSI-C

Result: 220% of TM1000 performance

Reduce number of executed instructions

Step 3: Use word oriented interface to background memory

Load/store 32 bit words, not bytes

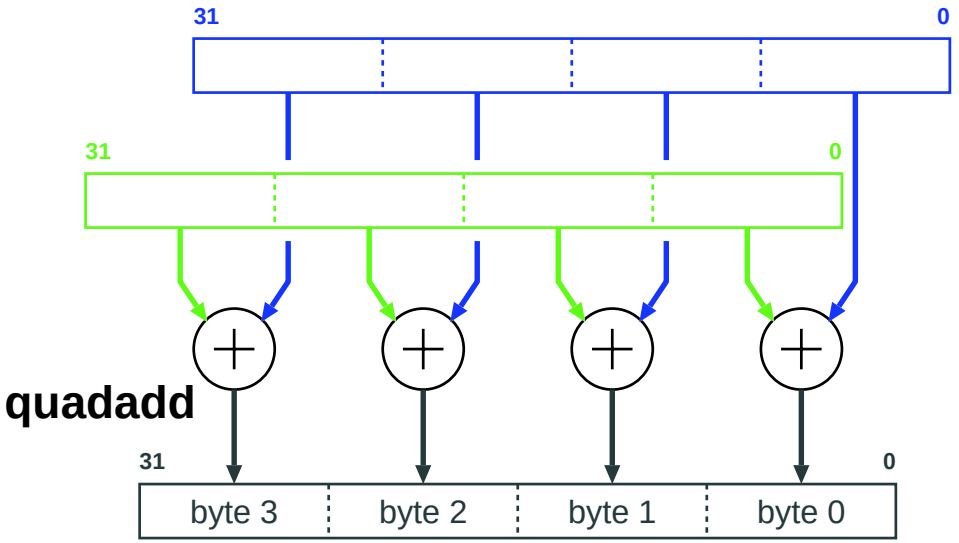
Also reduces loop overhead

```
#define Median_quad(a,b,c) (pack16lsb( packbytes( \
    Median(ubytesel(a, 3), ubytesel(b, 3), ubytesel(c, 3)), . . .\
packbytes( . . .))
```

Result: 93% of TM1000 performance

Suggestion

Add quadmin and quadmax SIMD instruction

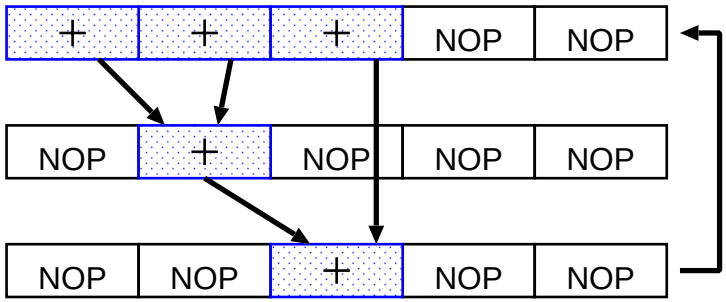


#define Median(a,b,c) (quadmin(quadmax(quadmin(a, b), c), quadmax(a, b)))

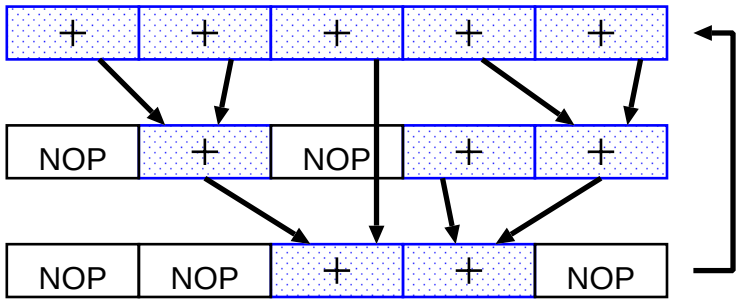
This would result in 30% of TM1000 performance

Improve instruction level parallelism

Step 4 & 5: Unroll the loop by 2 and then by 4



ILP = 5/3 = 1.67



ILP = 10/4 = 2.5

Supported by compiler directives

Result: 73% of TM1000 performance

Improve cache behavior

Step 6: Last fine tuning: remove data cache misses

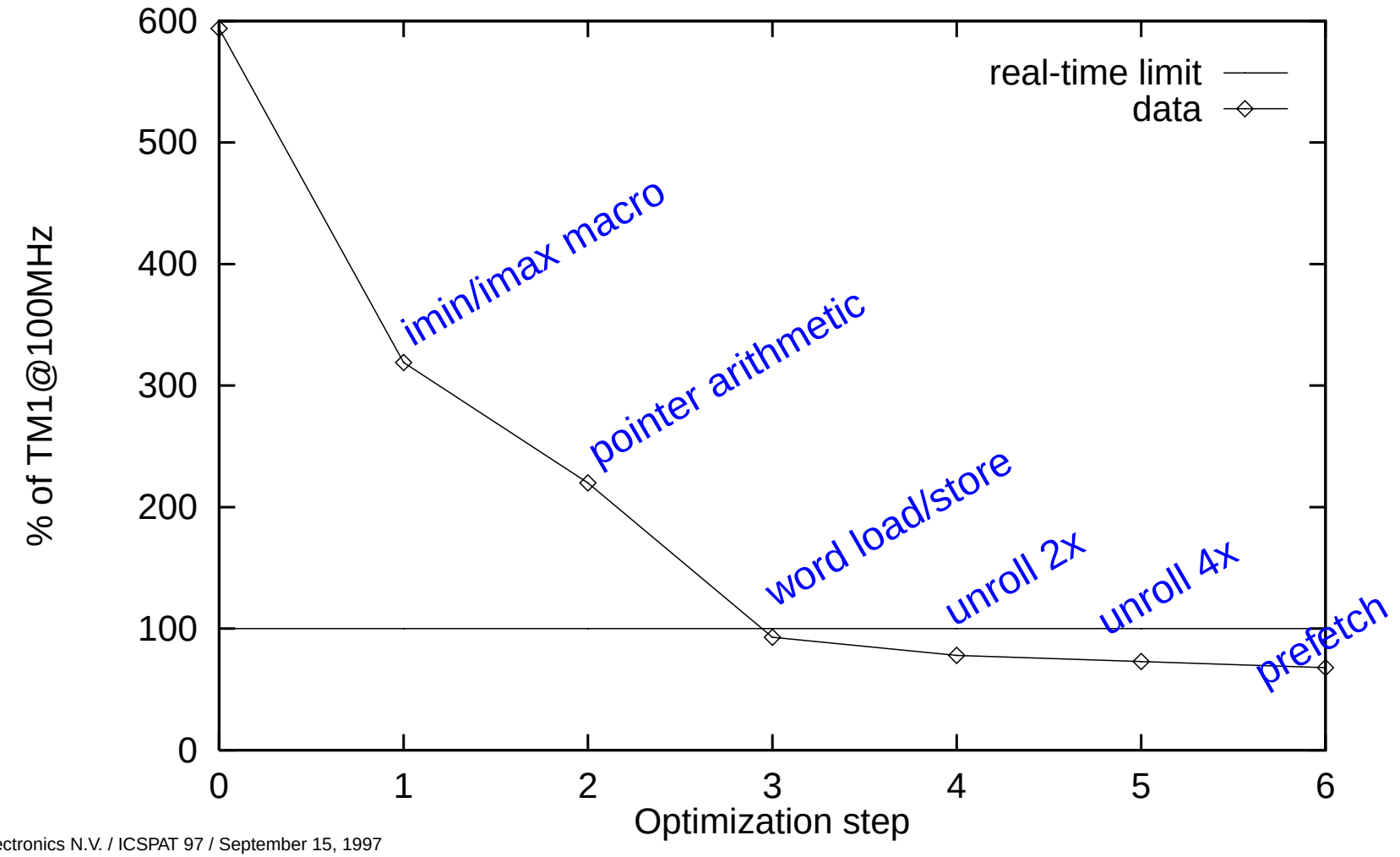
To reduce data cache misses, prefetch instructions are added.

They replace *nop* instructions, so:
the number of cycles for the loop body has remained the same.

Result: 68% of TM1000 performance

Percentage of TM1000 after each step

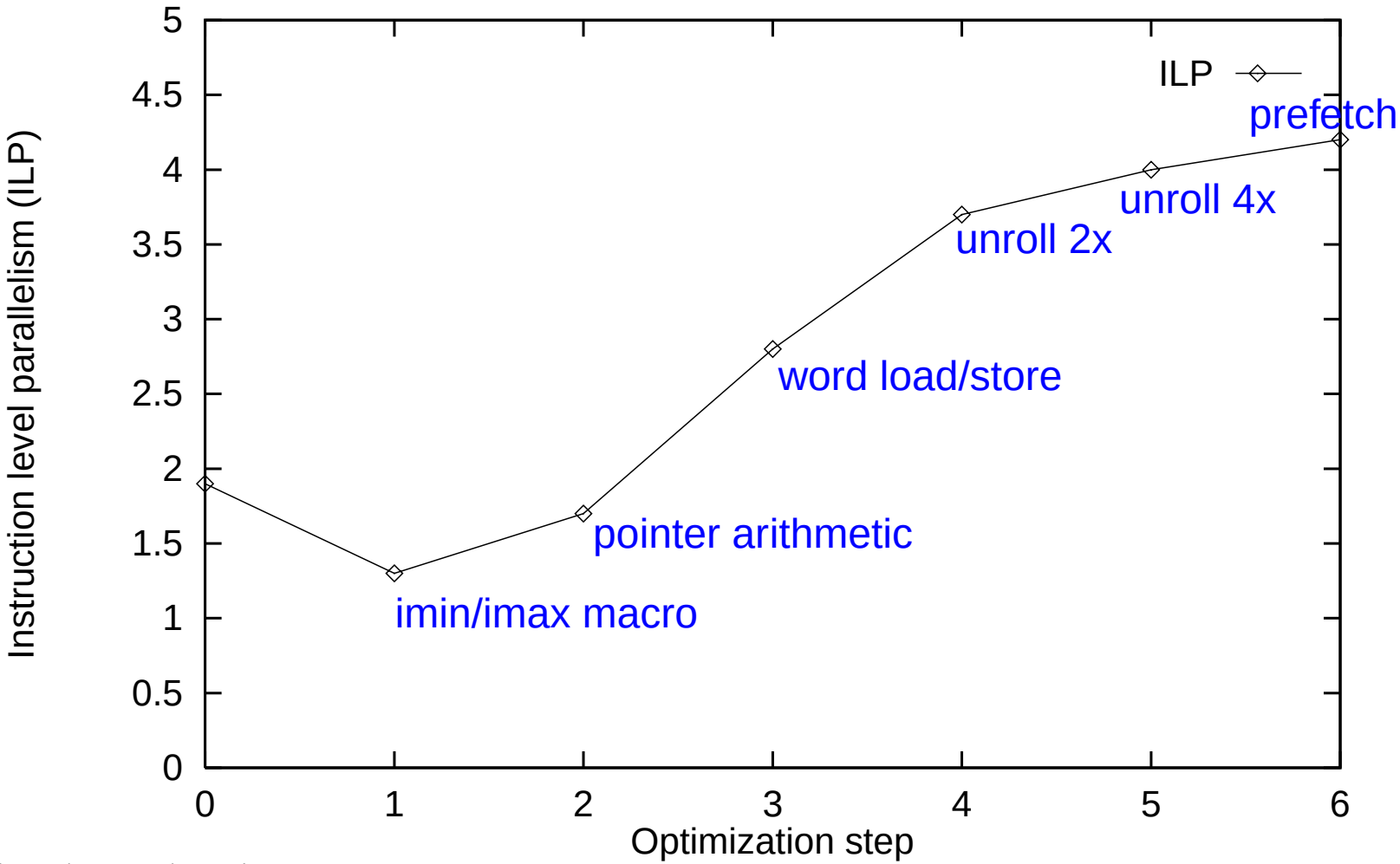
Total gain: 10 times! Result 3 cycles/median



PHILIPS

Instruction level parallelism (ILP) after each step

Final ILP: 4.2



Summary and conclusions

Structured method to increase the algorithm efficiency

1. Reduce the number of executed instructions
2. Utilize SIMD special instructions
3. Additional instructions will be beneficial
4. Increase the ILP
5. Tune the data cache behavior

Video processing with TM1000 CPU

One week programming in C

Demonstration

We can meet again at the

demo session: today 15:00 - 17:00.