Multiprocessor SoCs for Video Processing

Wayne Wolf
MediaWorks Technology
and Princeton University

Outline

- Real-time environmental video processing.
- Architectural alternatives for media processing.
- Jason Fritts PhD work: Programmable VSPs.
- Hua Lin PhD work: loop optimizations and memory systems.
- Speculations on multiprocessor architectures.

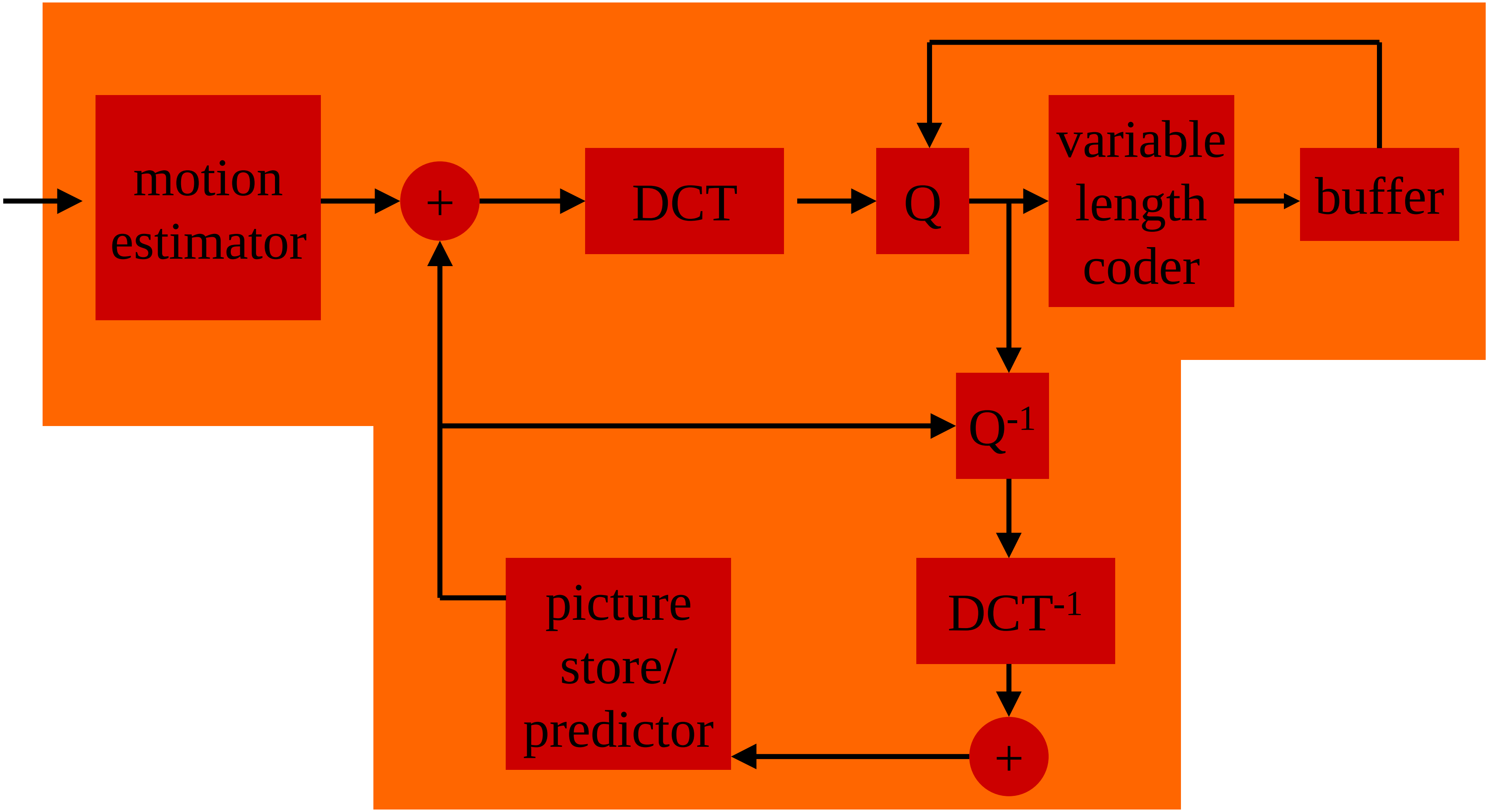
Architectural questions

- How much computational horsepower is required for interesting applications?
- How do we exploit levels of parallelism?
  - Instruction-level (static, dynamic);
  - Data-level;
  - Process-level.
- How do we estimate performance/power at each level?

Multimedia requirements

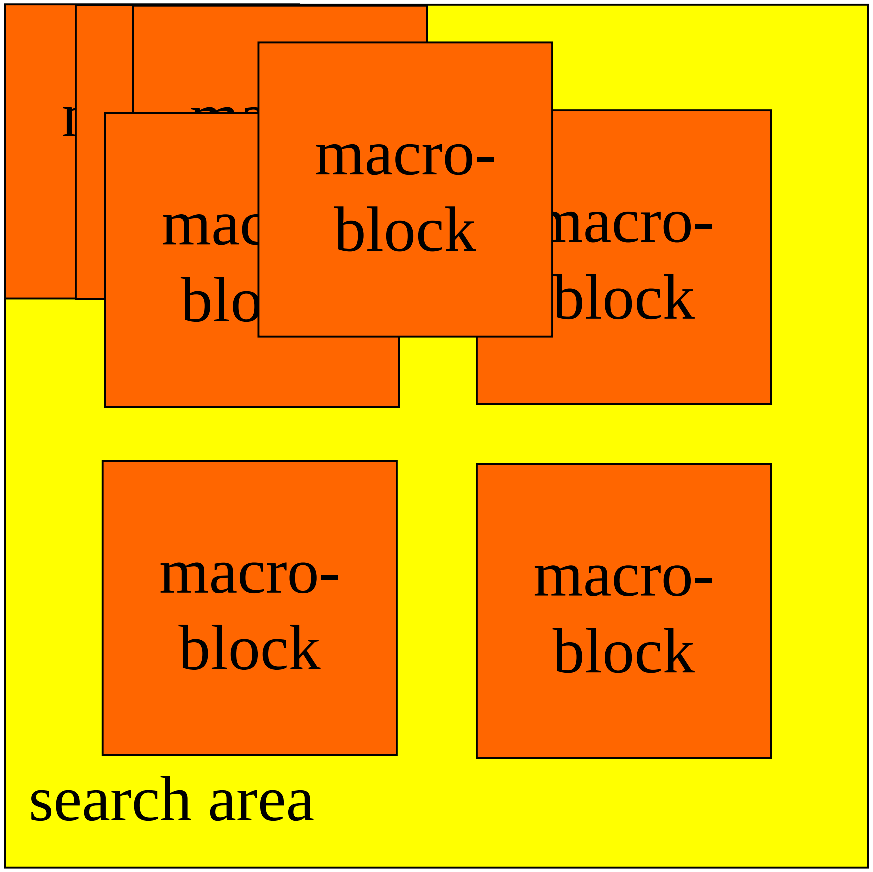
- Complex algorithms:
  - multiple phases;
  - data and control.
- Today’s applications: compression.
- Tomorrow’s applications: analysis.

MPEG-style compression engine



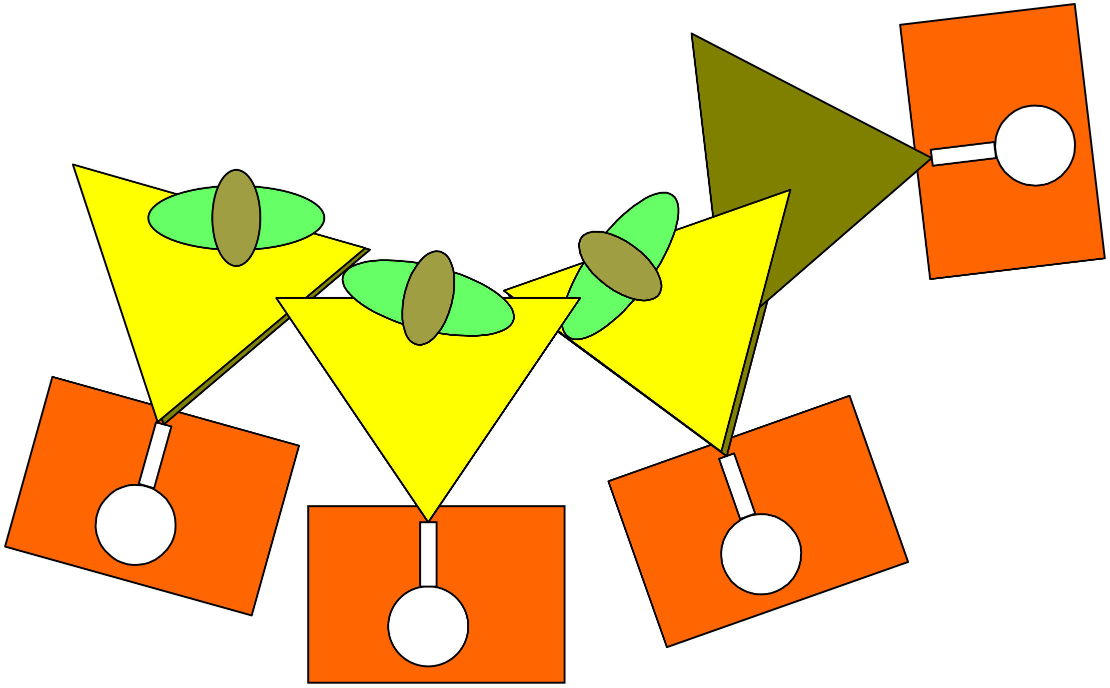
Block motion estimation

3-step search:

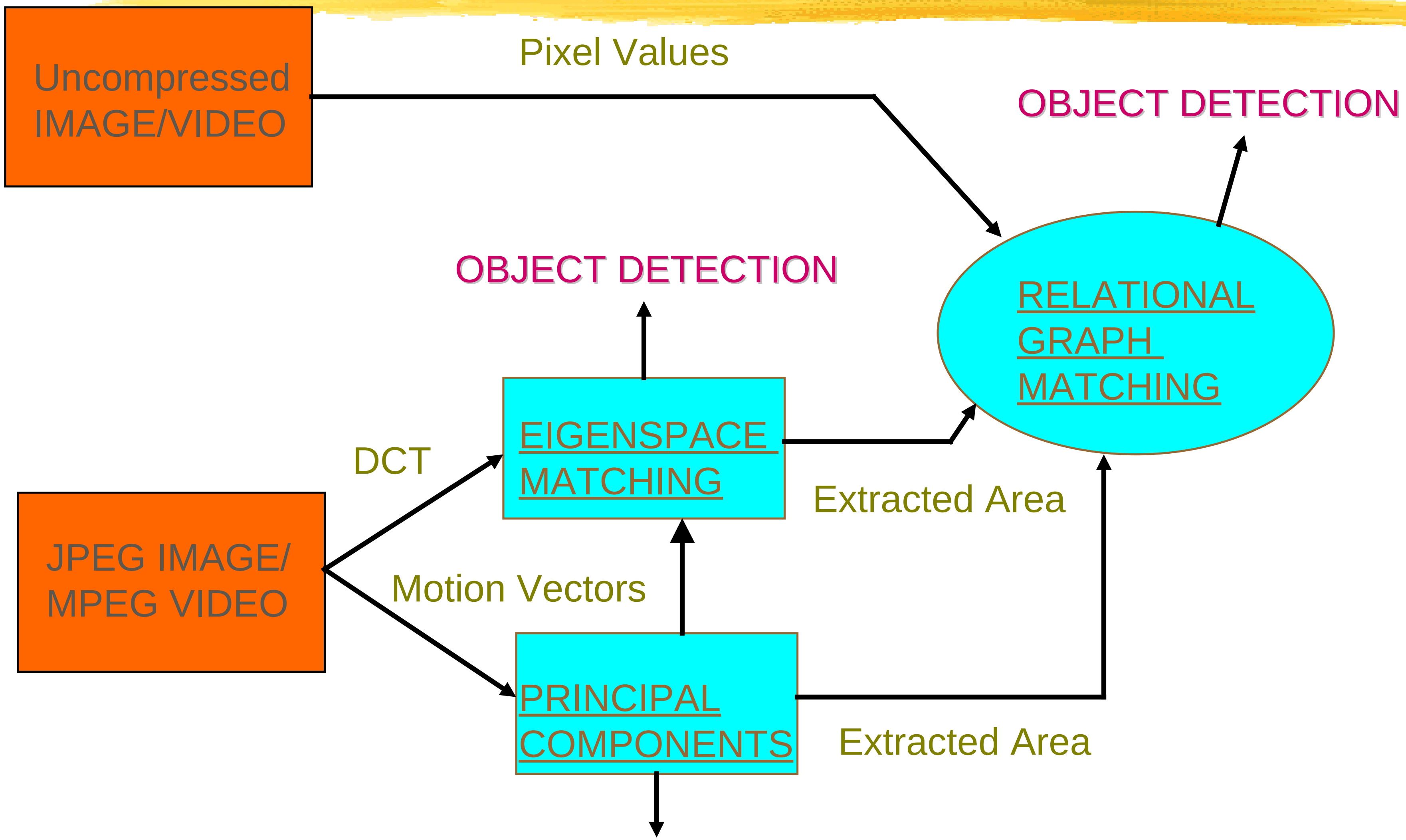


Smart cameras for smart rooms

- Coordinated cameras track subject:



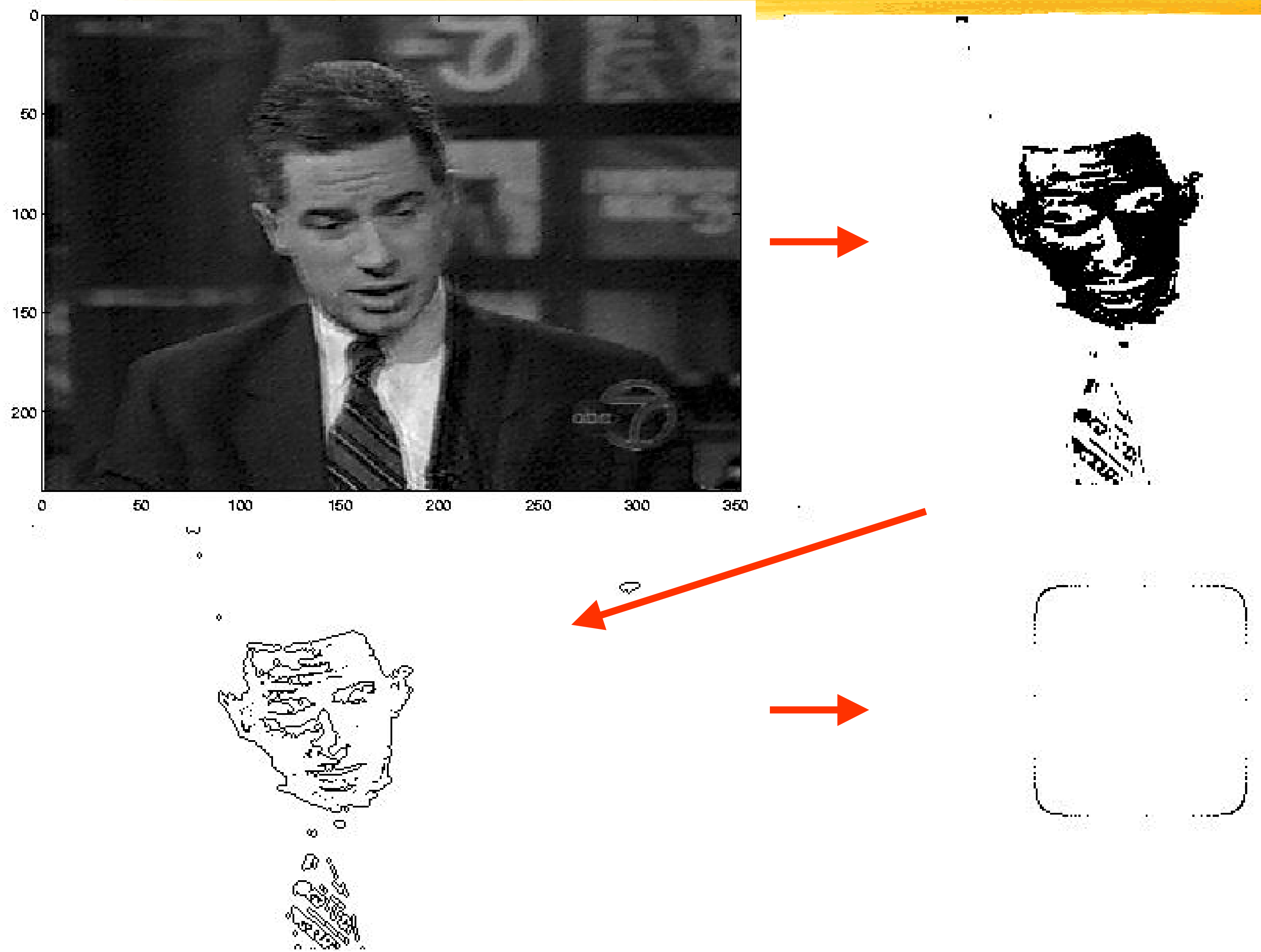
Ozer et al: human activity recognition



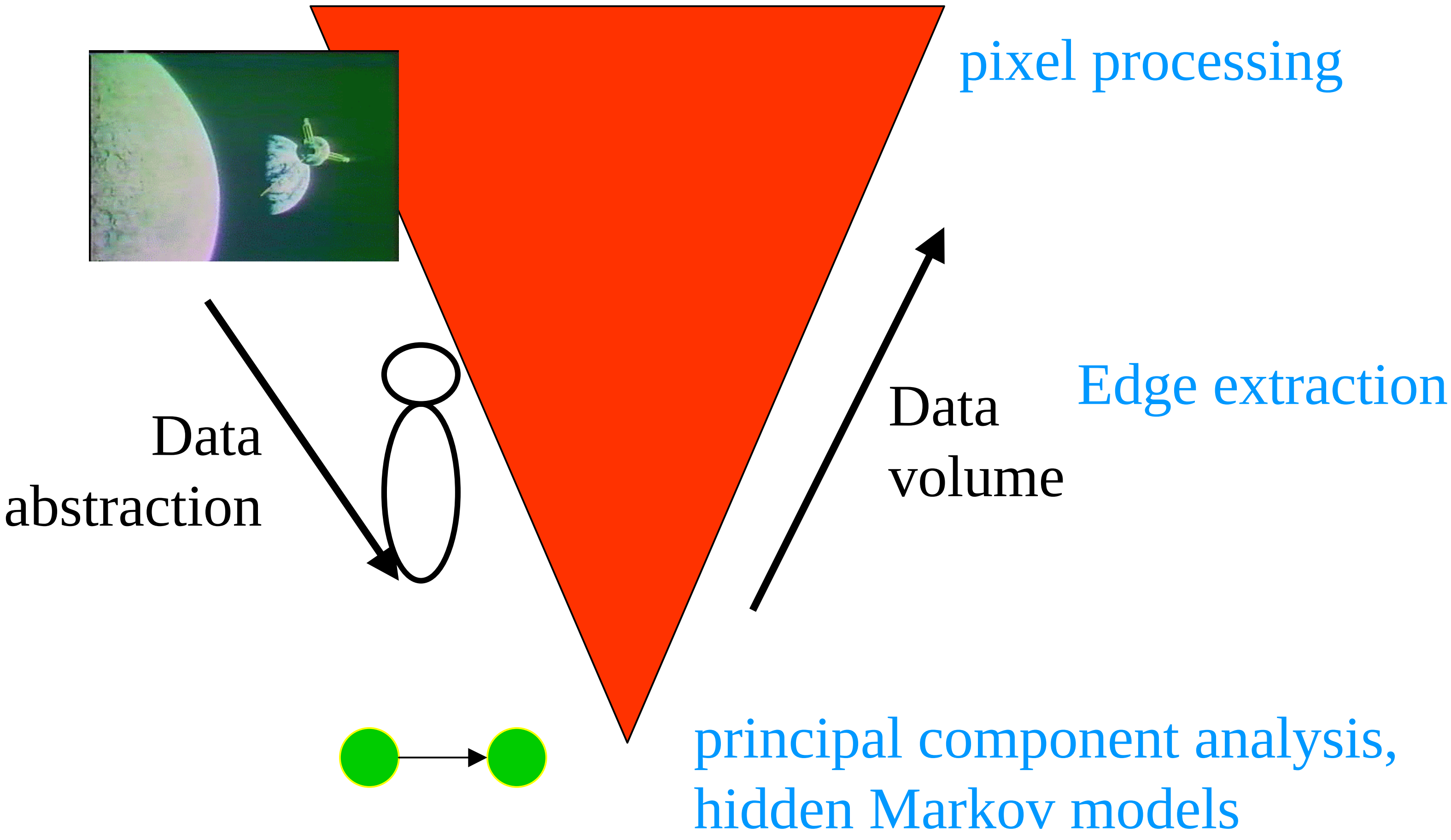
Our environmental video system

- Ozer/Wolf:
  - multiple Trimedia processors attached to PC;
  - plan to introduce multiple cameras.

Real-time analysis



The multimedia processing funnel



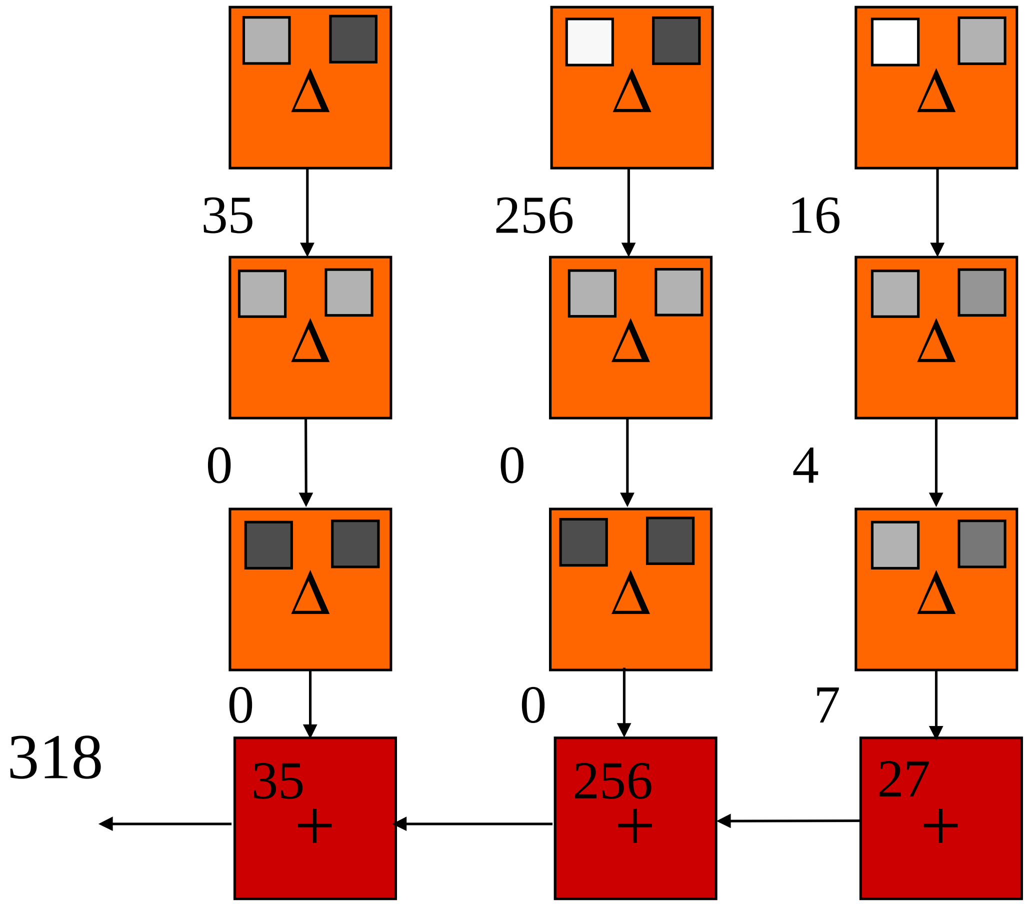
Architectural styles for video

- SIMD
- Heterogeneous.
- ISA extensions.
- VLIW.

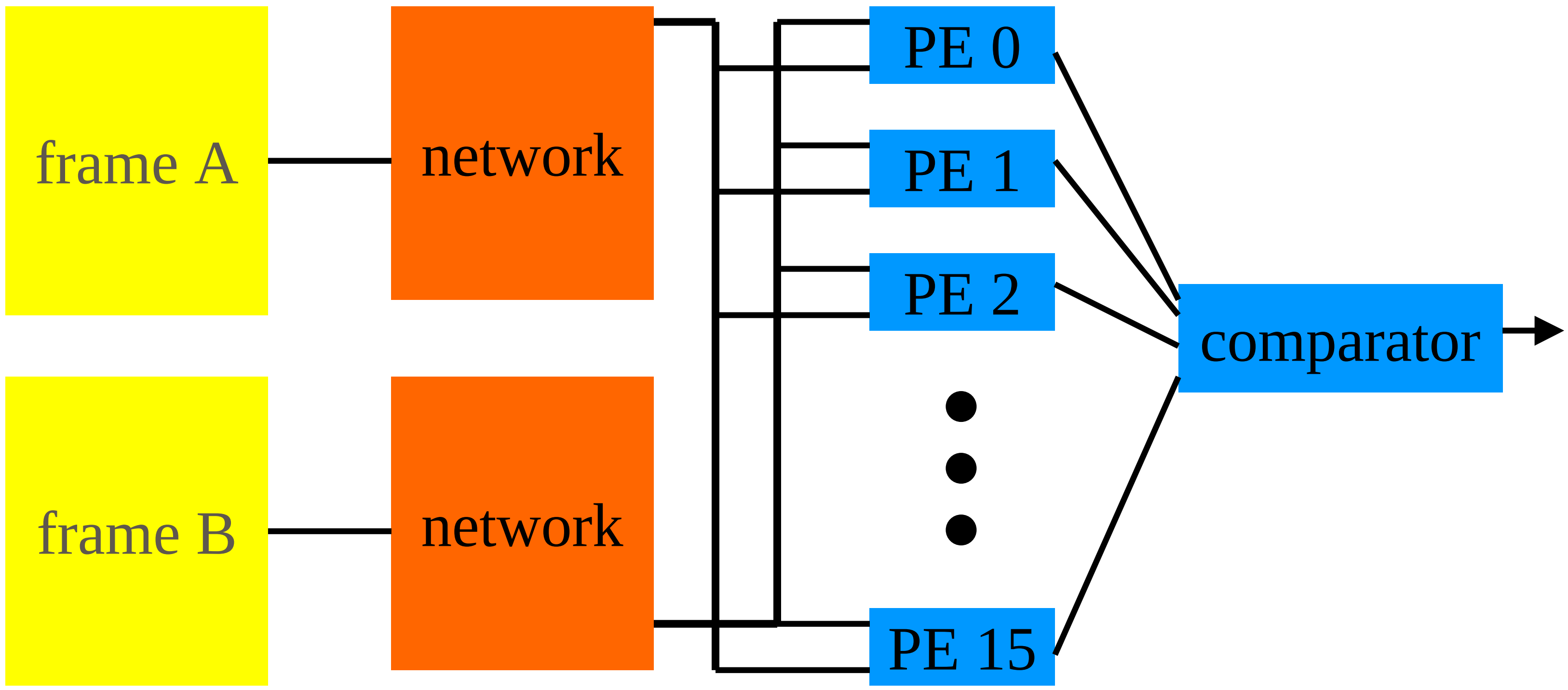
SIMD processing

- Broadcast operation to an array of processing elements, each of which has its own data.
- Well-suited to regular, data-oriented operations.

A block correlation architecture



Block motion estimation architecture



Data flow in block motion estimation

time	PE0	PE1	PE2	...
0	a(0,0)-b(0,0)			
1	a(0,1)-b(0,1)	a(0,0)-b(0,1)		
2	a(0,2)-b(0,2)	a(0,1)-b(0,2)	a(0,0)-b(0,2)	
...				
16	a(1,0)-b(1,0)	a(0,15)-b(0,16)	a(0,14)-b(0,16)	
17	a(1,1)-b(1,1)	a(1,0)-b(0,1)	a(0,15)-b(0,17)	
18	a(1,2)-b(1,2)	a(1,1)-b(0,2)	a(1,0)-b(0,2)	

Hetereogeneous multiprocessor design

- Will need accelerators for quite some time to come:
  - power;
  - performance.
- Candidates for acceleration:
  - complex coding and error correction;
  - motion estimation.

Expensive operations

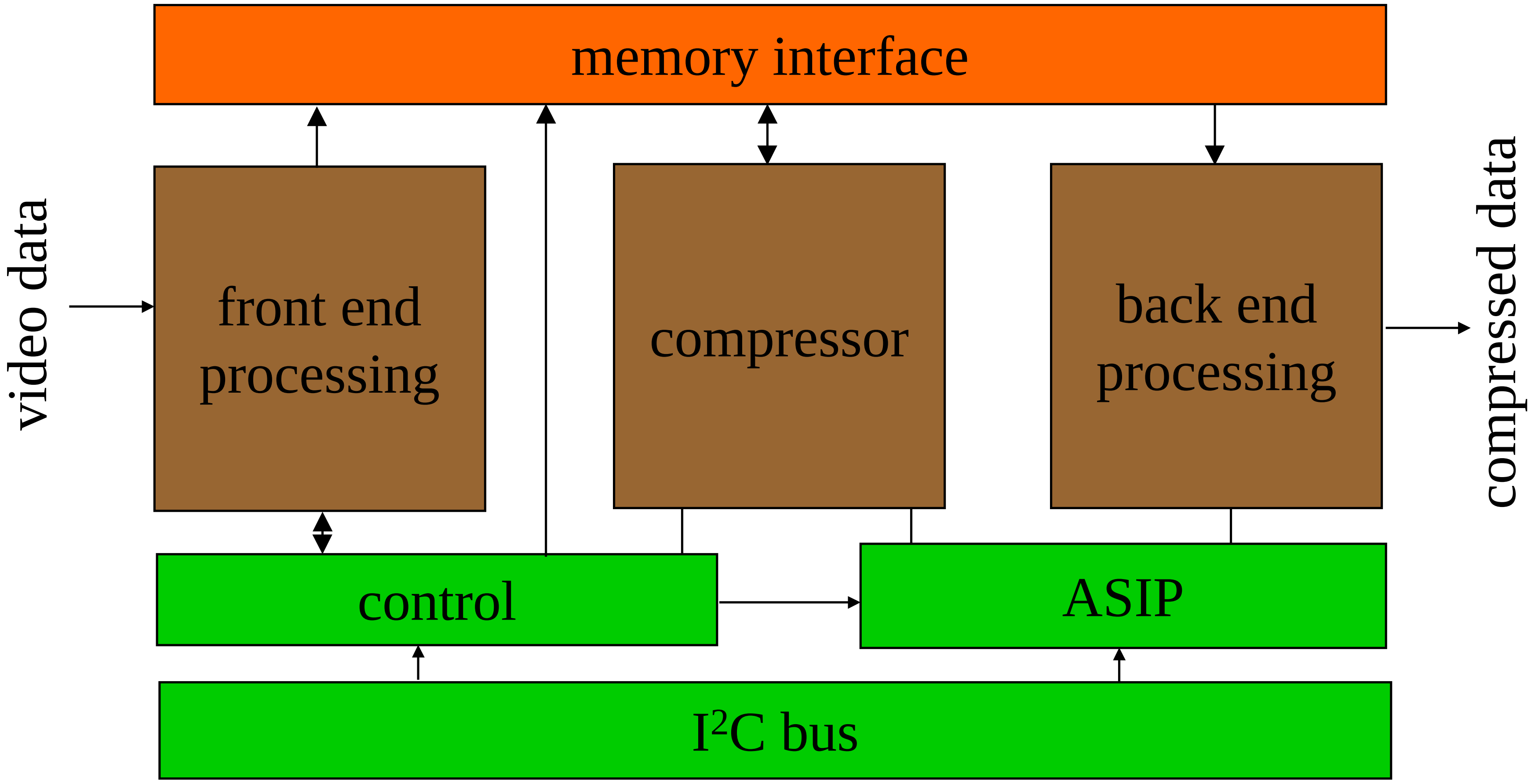
Expensive operations can be speeded up by special-purpose units:

- specialized memory accesses;
- specialized datapath operations.

Special-purpose units may be useful for only certain parameters:

- block size;
- search region size.

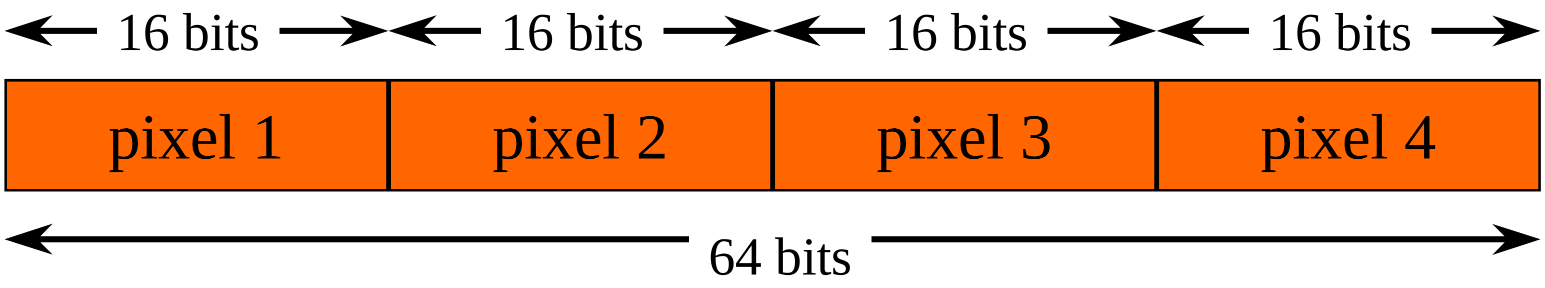
Philips MPEG2 encoder (ISSCC '97)



ISA extensions

Split data word into subwords to provide *single instruction multiple data* (*SIMD*) parallelism.

Assemble CPU word from pixels:

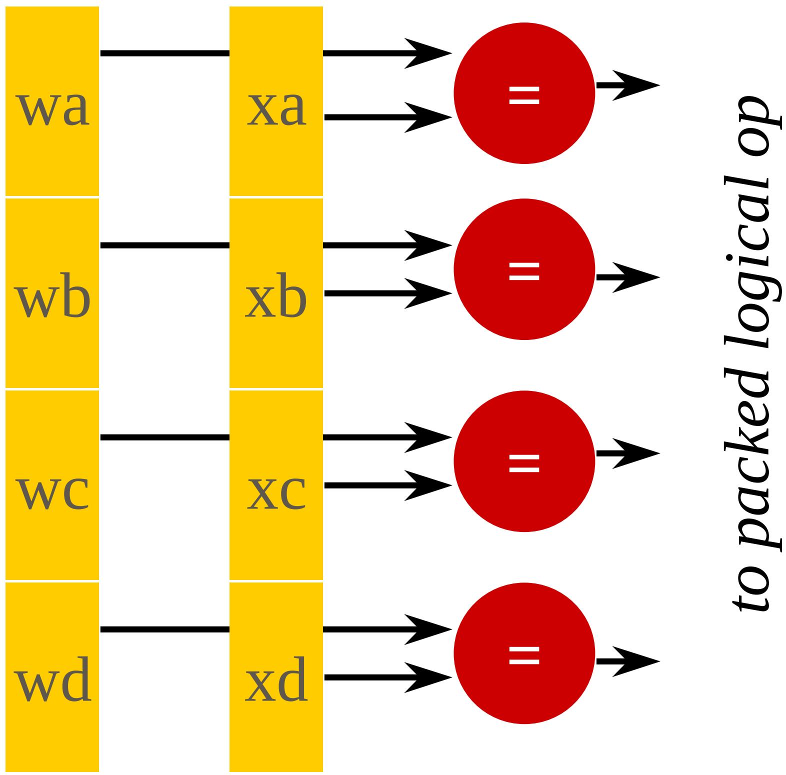


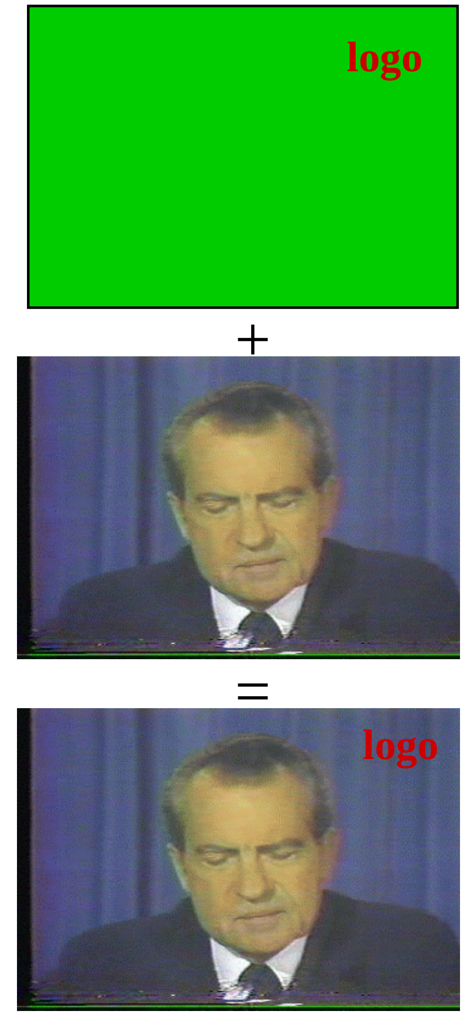
Why ISA extensions

- *Easy*: provide significant parallelism with small changes to architecture.
- *Cheap*: can be implemented with
- *Effective*: provide 2x-4x speedups.

Packed compare instruction

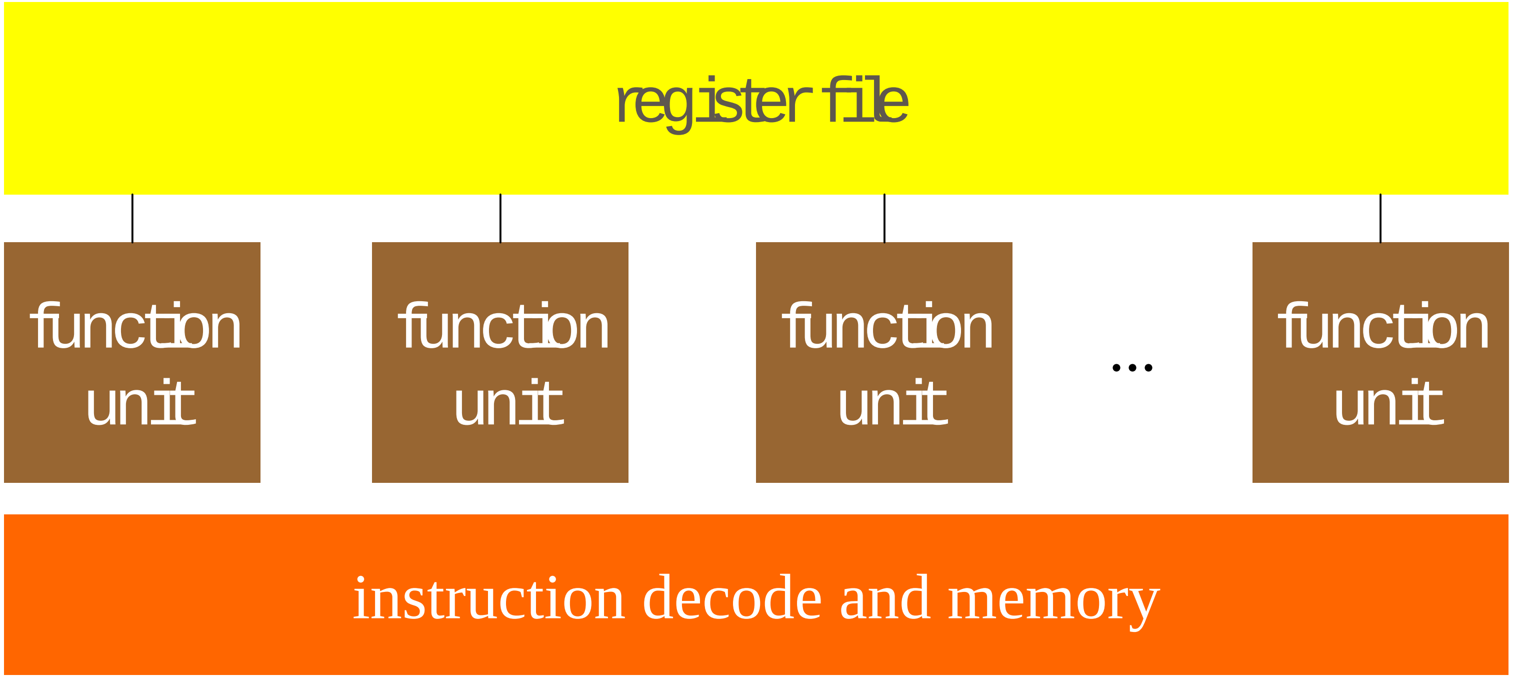
Used for chromakey:





VLIW architectures

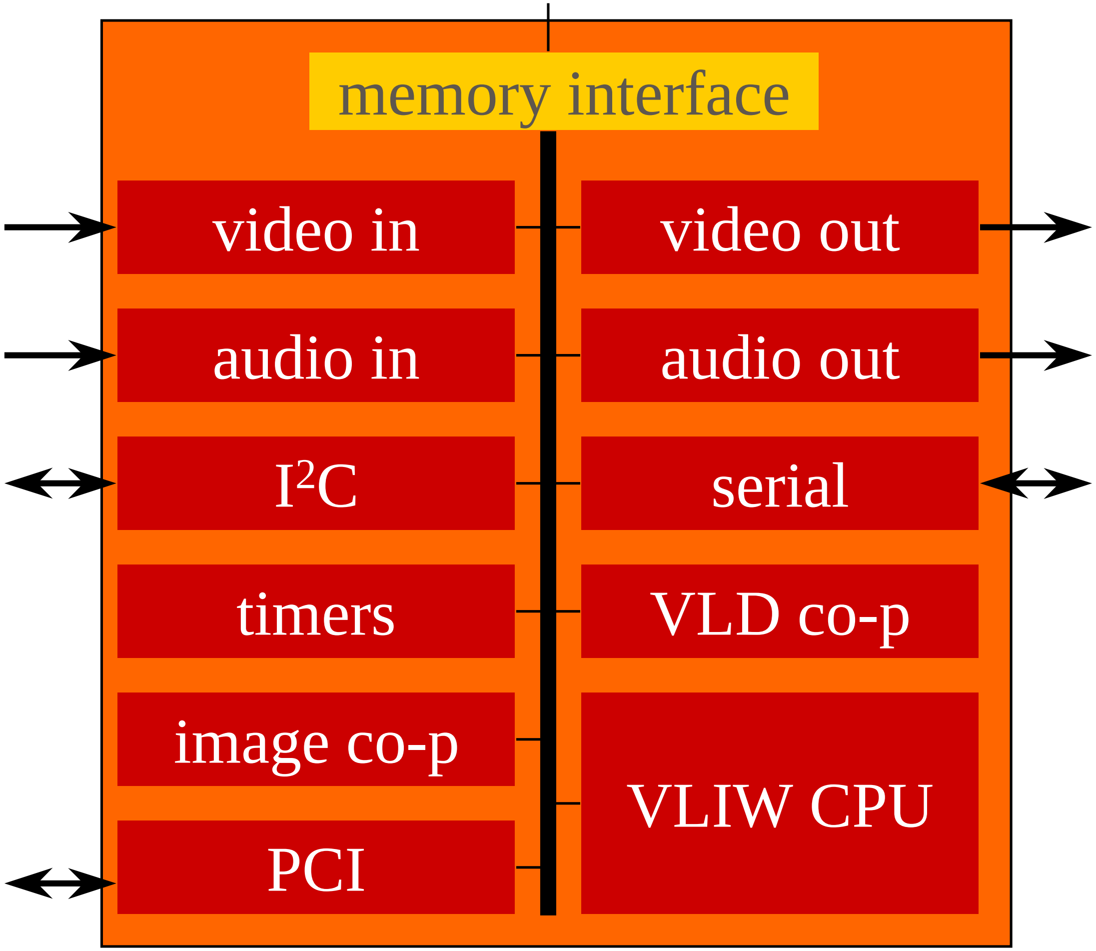
- Parallel function units, shared register file, static scheduling of operations:



VLIW's popularity

- Invented 20 years ago, popular today:
  - Good compiler technology.
  - Low control overhead.
  - Systems-on-silicon eliminates pinout problems.
- Advantages for video:
  - Embarrassing parallelism with static scheduling opportunities.
  - Less problem with code compatibility.

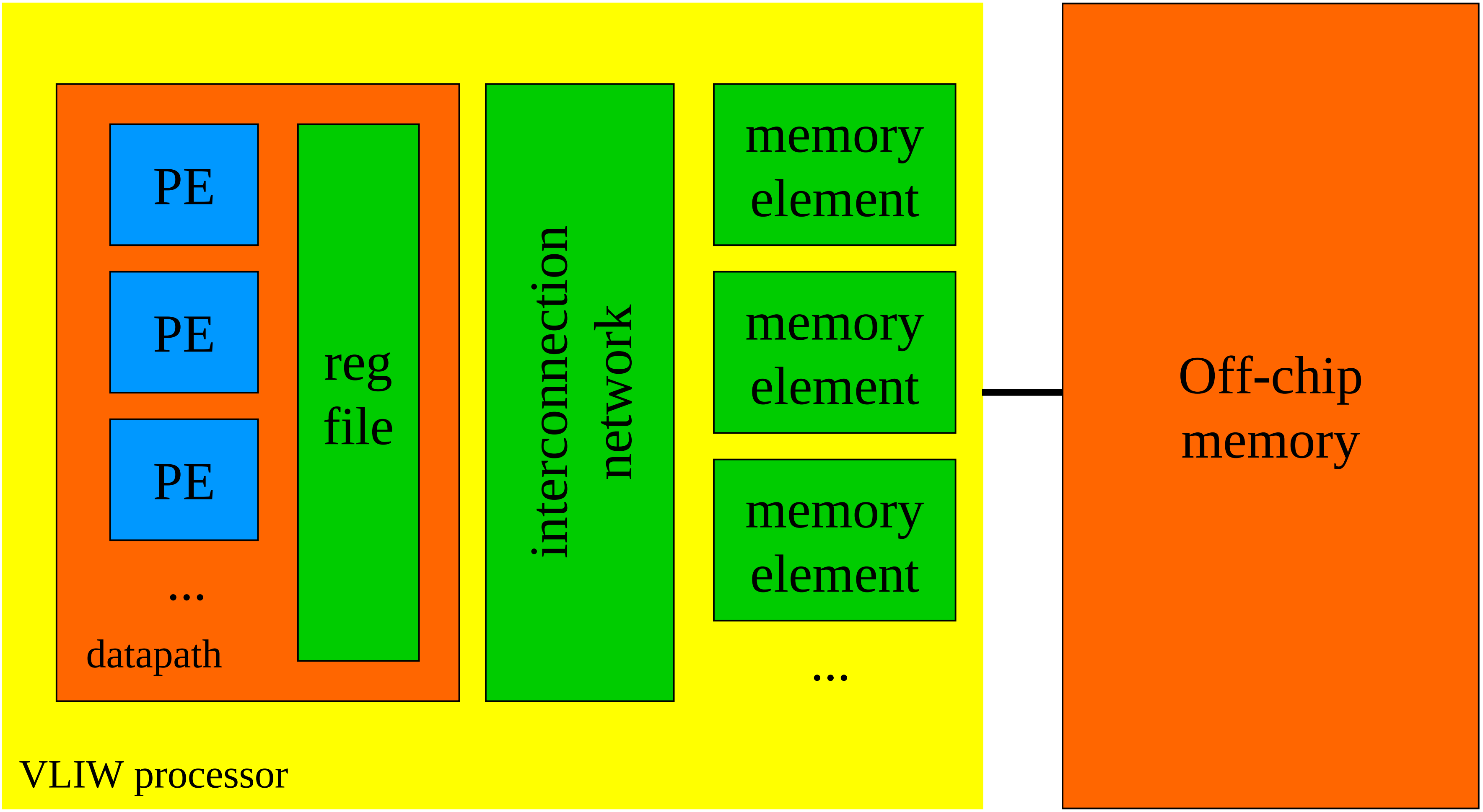
Trimedia TM-1



Architectural experiments

- Fritts/Wolf:
  - characterize applications;
  - compare architectural styles (VLIW, superscalar);
  - evaluate architectural parameters (clock rate, pipelining, etc.).

VLIW processor model



Workload characteristics experiments

- Goal: compare media workload characteristics to general-purpose load.
- Used MediaBench benchmarks.
- Compiled on Impact compiler, measured with with Impact simulator.

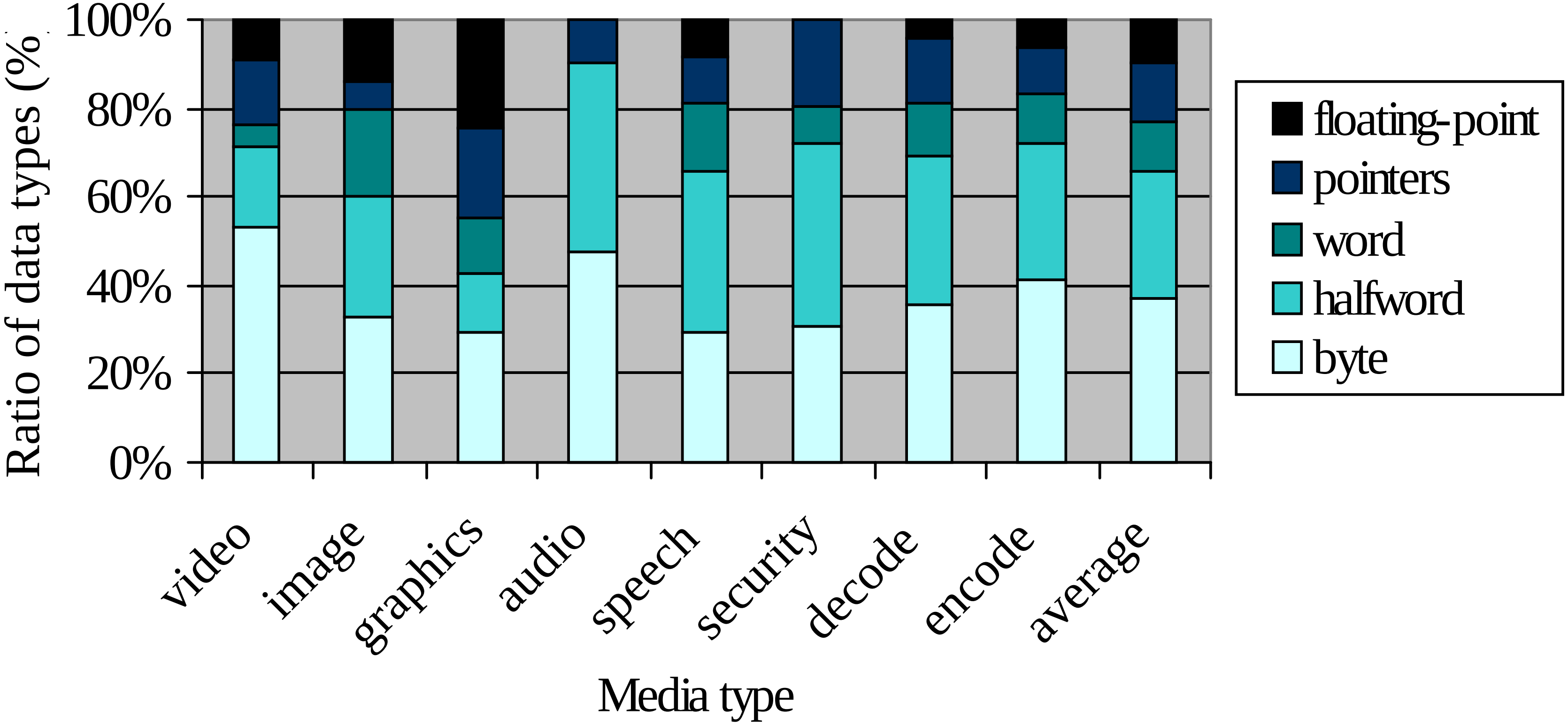
Basic characteristics

- Comparison of operation frequencies with SPEC
  - (ALU, mem, branch, shift, FP, mult) => (4, 2, 1, 1, 1, 1)
  - Lower frequency of memory and floating-point operations
  - More arithmetic operations
  - Larger variation in memory usage
- Basic block statistics
  - Average of 5.5 operations per basic block
  - Need global scheduling techniques to extract ILP

Basic characteristics, cont'd

- Static branch prediction
  - Average of 89.5% static branch prediction on training input
  - Average of 85.9% static branch prediction on evaluation input
- Data types and sizes
  - Nearly 70% of all instructions require only 8 or 16 bit data types

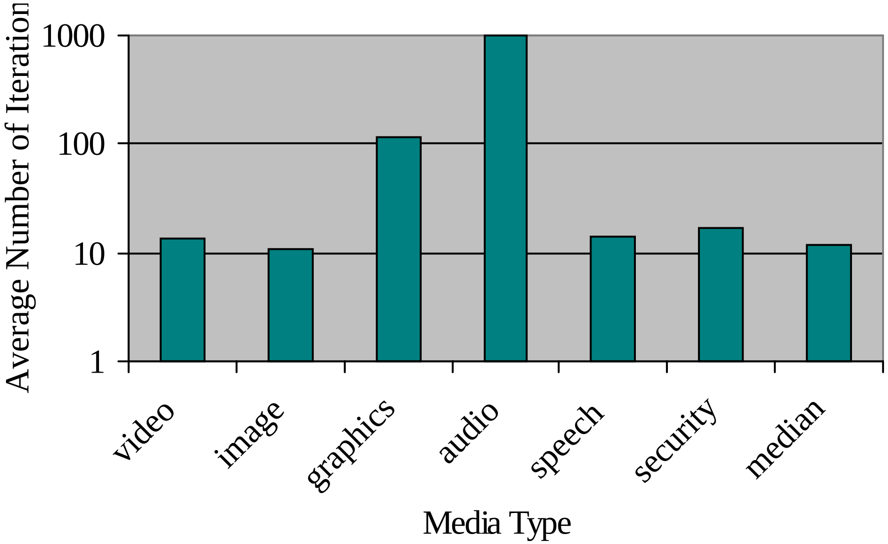
Breakdown of data types by media type



Multimedia looping characteristics

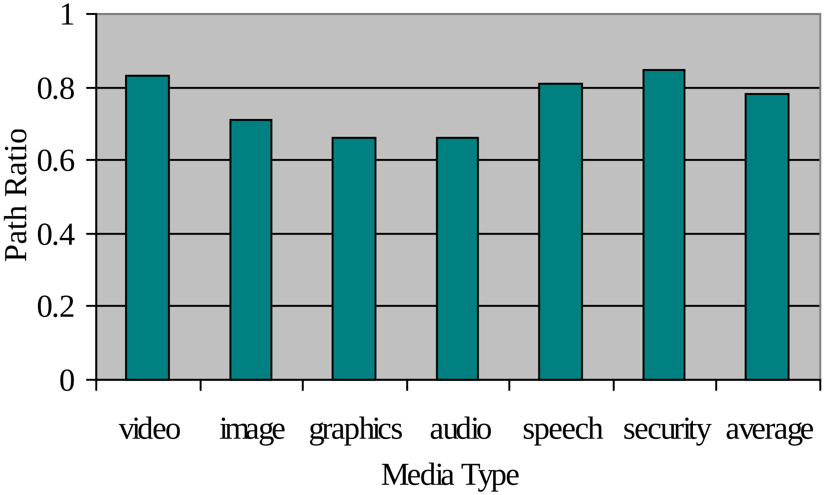
- Highly loop centric
  - 95% of CPU time in two innermost loop levels
  - Significant processing regularity
  - About 10 iterations per loop on average
- Complex loop control
  - = average # of instructions executed per loop invocation/total # of loop instructions
  - Average path ratio of 78%--high complexity

Average iterations per loop and path ratio



- average number of loop iterations

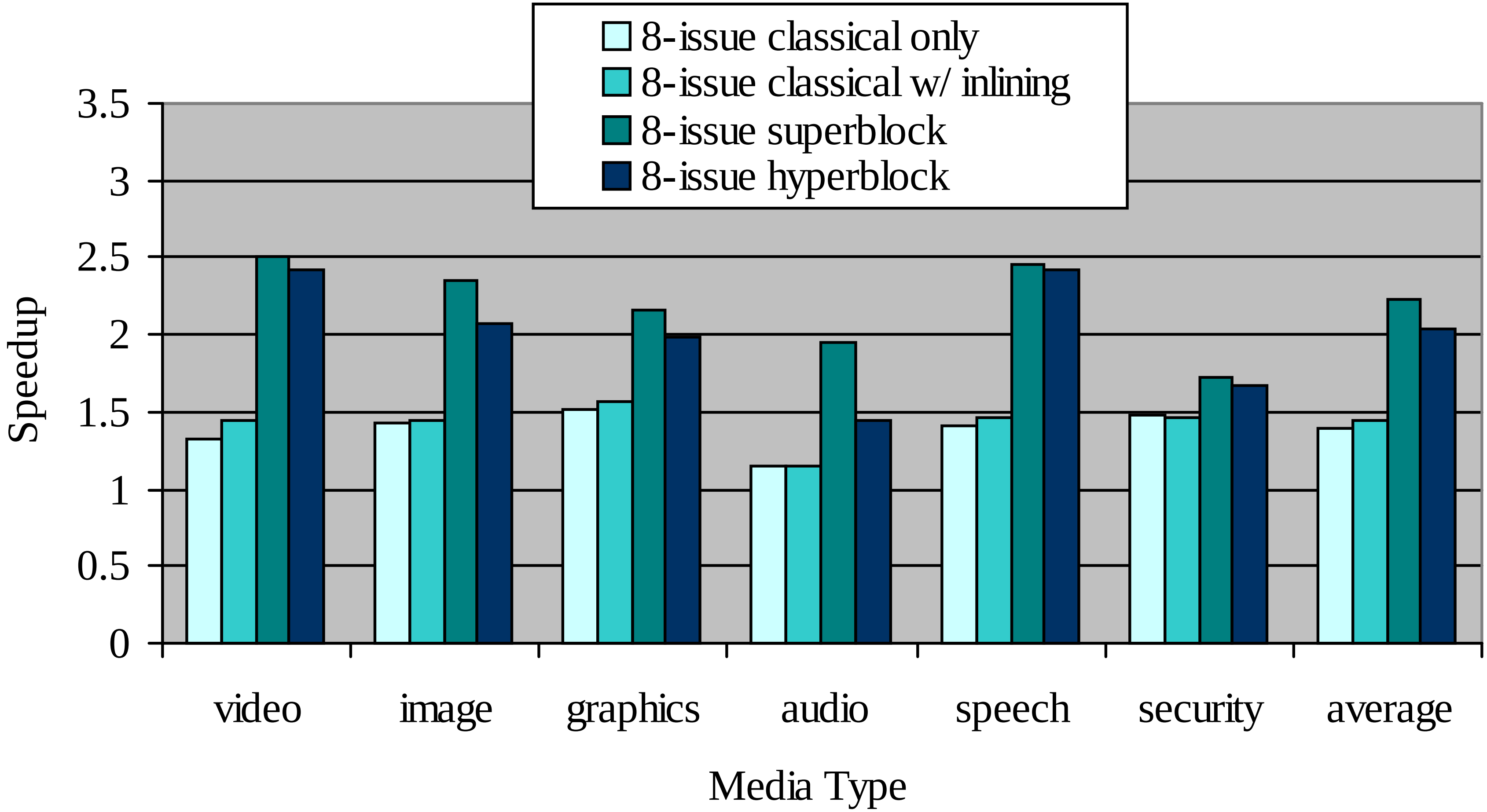
- average path ratio



Instruction level parallelism

- Instruction level parallelism
  - base model: single issue using classical optimizations only
  - parallel model:8-issue
- Explores only parallel scheduling performance
  - assumes an ideal processor model
  - no performance penalties from branches, cache misses, etc.

ILP results


8-issue classical only
8-issue classical w/ inlining
8-issue superblock
8-issue hyperblock
3.5
3
2.5
2
1.5
1
0.5
0
Speedup
video
image
graphics
audio
speech
security
average
Media Type

VSP architecture evaluation

- Determine fundamental architecture style
  - Statically Scheduled => Very Long Instruction Word (VLIW)
  - Dynamically Scheduled => Superscalar
- Examine variety of architecture parameters
  - Fundamental Architecture Style
  - Instruction Fetch Architecture
  - High Frequency Effects
  - Cache Memory Hierarchy

Fundamental architecture evaluation

- Major issues:
  - Static vs. dynamic scheduling
  - Issue width
- Focused on non-memory limited applications.

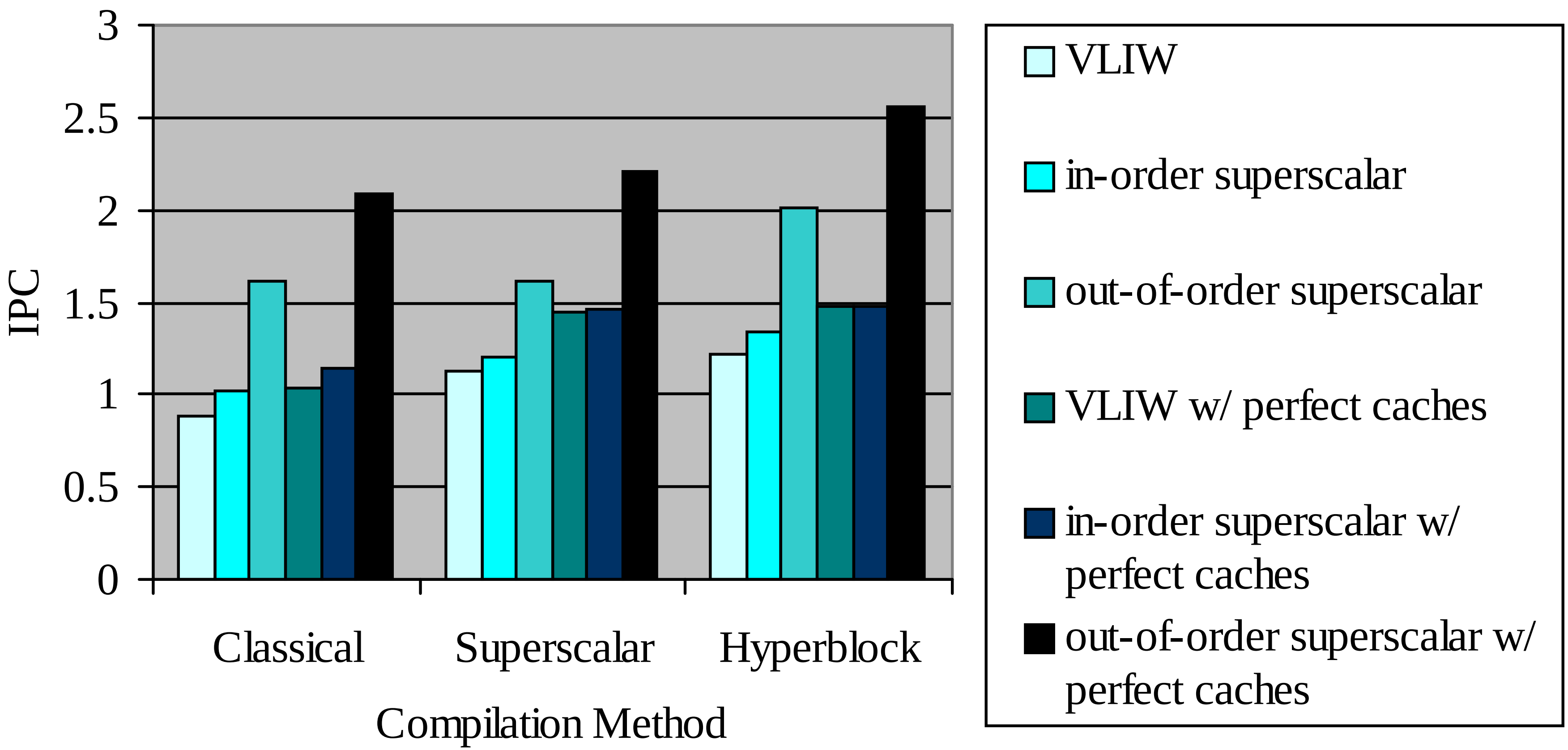
Architectural model

- 8-issue processor
- Operation latencies targeted for 500 MHz to 1 GHz
- 64 integer and floating-point registers
- Pipeline: 1 fetch, 2 decode, 1 write back, variable execute stages

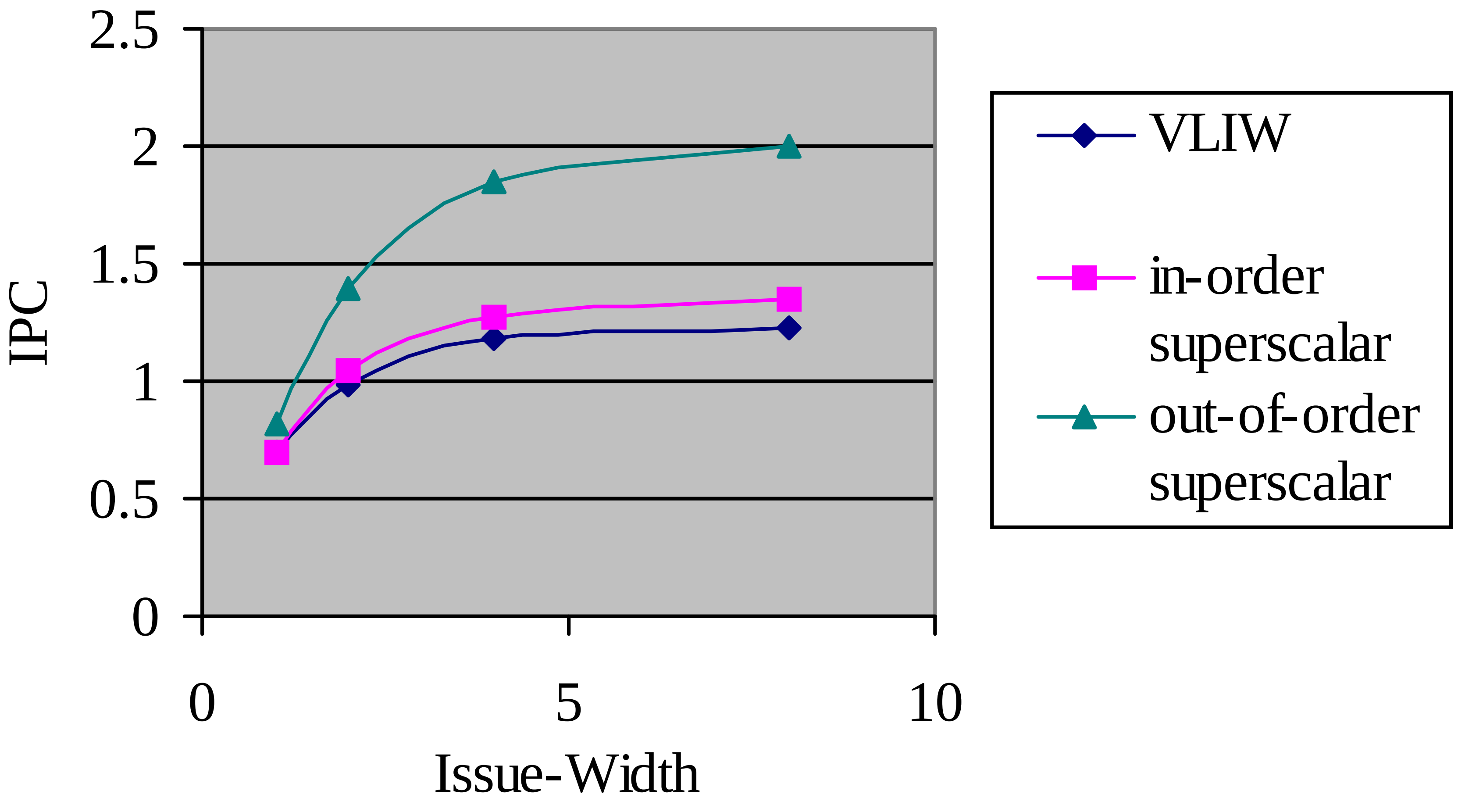
Architectural model, cont'd

- 32 KB direct-mapped L1 data cache with 64 byte lines
- 16 KB direct-mapped L1 instruction cache with 256 byte lines
- 256 KB 4-way set associate on-chip L2 cache
- 4:1 Processor to external bus frequency ratio

Static versus Dynamic Scheduling

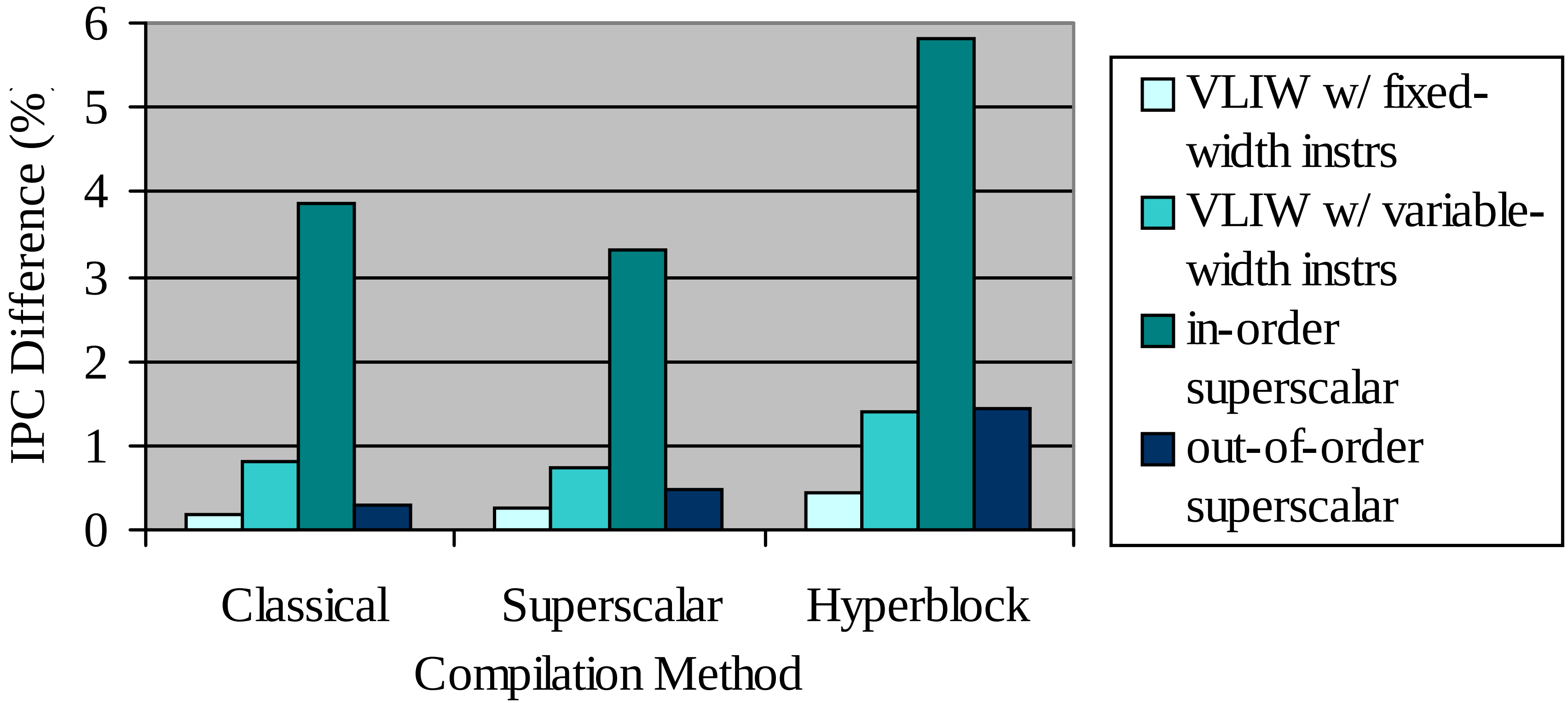


Increasing issue width



Instruction fetch architecture

- Unbuffered fetch vs. decoupled fetch:



Impact of higher processor frequencies

- Increased wire delay at higher frequencies may cause:
  - Longer operation latencies
  - Delayed bypassing

Processor frequency models

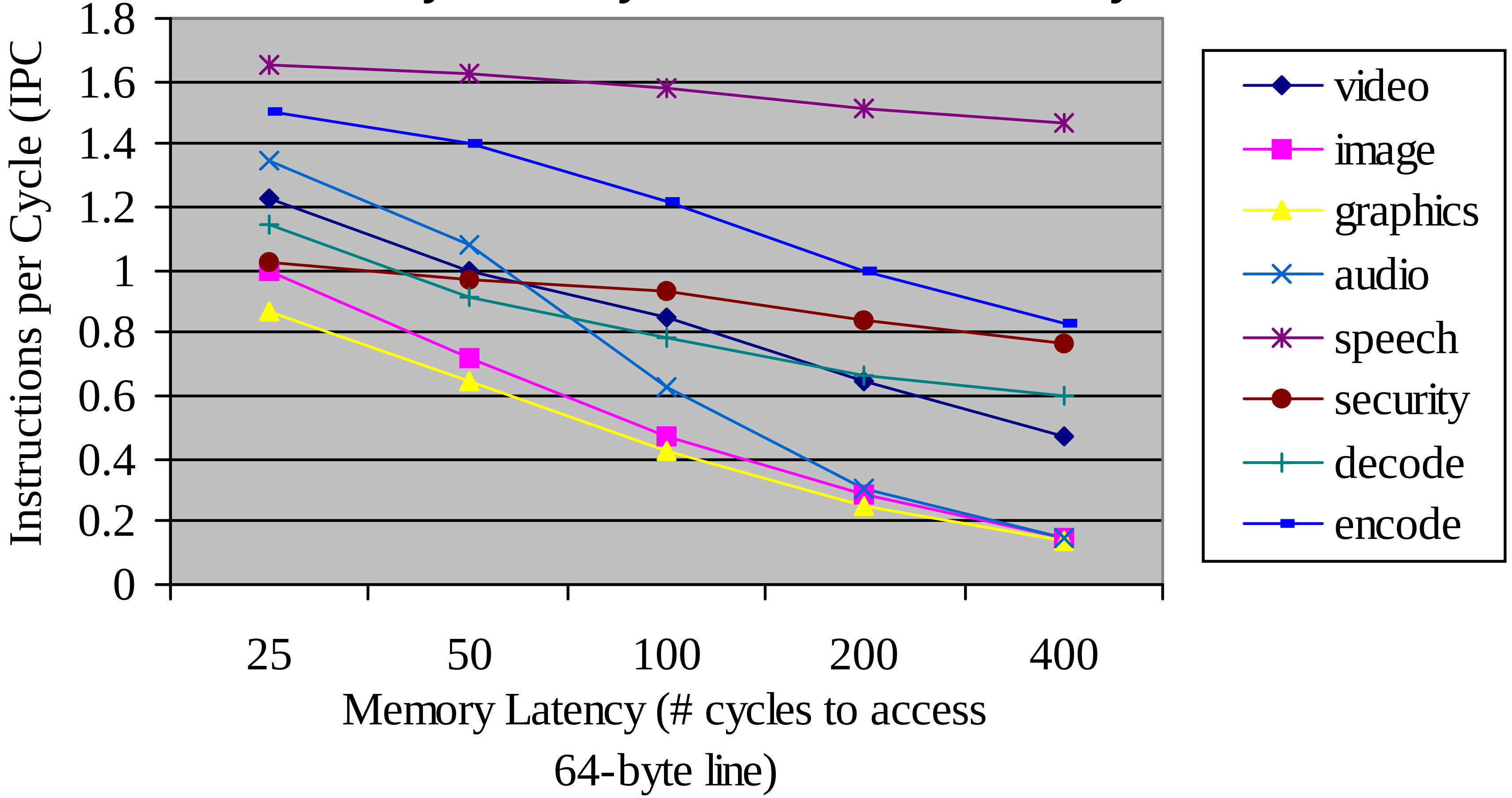
- Three processor models with different operation latencies
  - 250 MHz – 500 MHz: stores – 1, loads – 2, FP – 3, mult – 3, div – 10
  - 500 MHz – 1 GHz: stores – 2, loads – 3, FP – 4, mult – 5, div – 20
  - 1 GH – 2 GHz: stores – 3, loads – 4, FP – 5, mult – 7, div – 30

Processor frequency results

- 10% performance difference between processor models
- 35% performance degradation for delayed bypassing
- Out-of-order scheduling and superscalar compilation least susceptible to high frequency effects
  - 20-30% less performance degradation

Memory latency

- Effect of memory latency on access to 64-byte line on L2 miss:



More susceptable to memory latency than bandwidth.

Evaluation of cache memory hierarchy

- Conclusions
  - L2 cache has little impact on performance
    - useful for storing state during context switches
  - External memory miss latency is primary memory problem
    - Streaming data structures will help alleviate this
  - External memory bandwidth is second-most problem

Loop optimizations

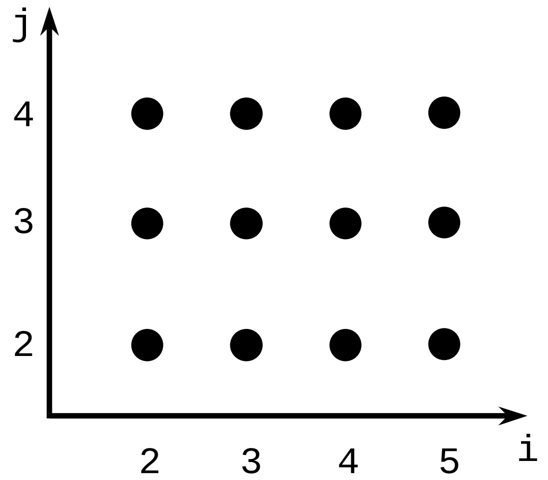
- Long-standing topic in compilers: identify and extract parallelism.
- Lin/Wolf: new twists for embedded systems:
  - develop more unified model for searching design space;
  - configure main memory, cache as well as optimize program.

Previous Work

- Loop transformation
  - Banerjee, Wolfe, Wolf & Lam, McKinley, Cierniak&Li
- Data layout transformation
  - Kandemir&Ramanujam, O'Boyle&Knijnenburg, Panda&Dutt, Chatterjee
- [Cierniak&Li] Unifying data and control transformation for distributed shared-memory machine
  - Stride vector: $T^T v = L^T m$
- [Kandemir] Improving Cache locality by a combination of loop and data transformation
  - Consider the fastest changing dimension
  - Search for the transformation matrix

Affine Representation

```
for i = 2, n
  for j = 2, n
    y[i,j]=0.5*(x[i-1,j-1]+x[i-1,j])
end                               (1)
```



Two-dimension loop nest: $(\vec{e}_1, \vec{e}_2) = ([1,0],[0,1])$

N-dimension loop nest: $(\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_N)$

Non-singular loop transformation: T

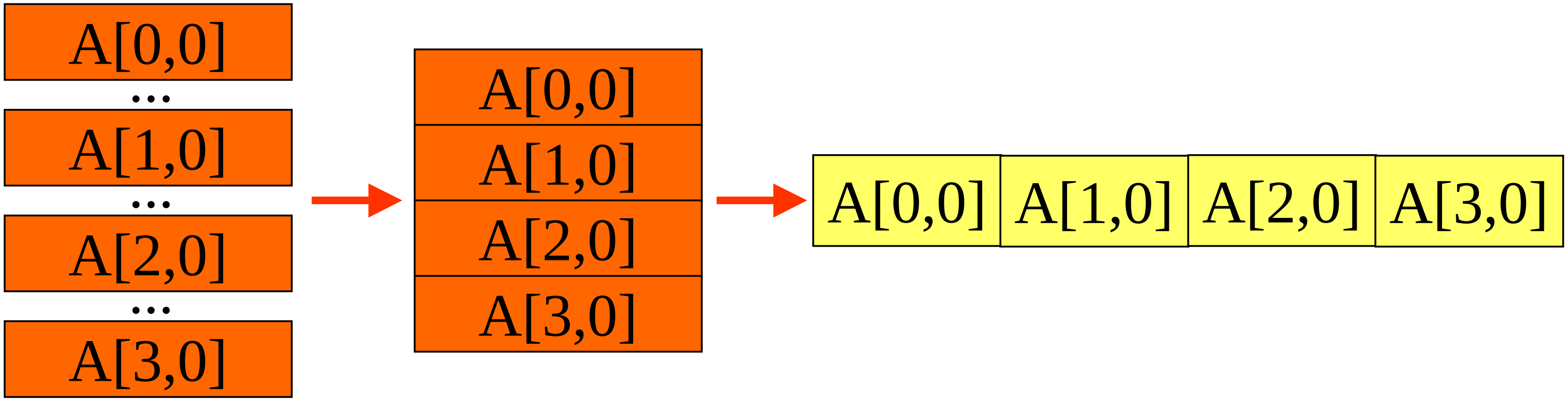
(1) $\begin{bmatrix} i-1 \\ j \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} + \begin{bmatrix} -1 \\ 0 \end{bmatrix} = A\vec{I} + c$ A: Reference Matrix

Varieties of Transformations

- Loop transformation:
  - Affect all array references in the transformed loop nest
  - Do not affect references in other loop nest
  - Data dependence vectors will change after transformation
- Data layout transformation:
  - Affect all references to the transformed array in the program
  - Do not affect references to other arrays, whether inside or not inside the same loop
  - Do not affect data dependence relationships

Data access locality

- Aspects of locality:
  - Spatially close: Elements of the fastest changing dimension of array
  - Temporarily close: Iterations of the innermost loops
- Can improve performance by putting local accesses in adjacent locations:



Our Work

- The starting point
  - Locality space instead of innermost loop
  - Integrated cache configuration and data locality optimization
  - Constructing the legal transformation matrix
  - Unified loop and data layout transformation
- Other initiatives
  - Dimensionality of the locality space and reuse vector space
  - Individual statement instead of loop body as the atomic unit of the iteration space
  - Locality space for arrays

Locality Space

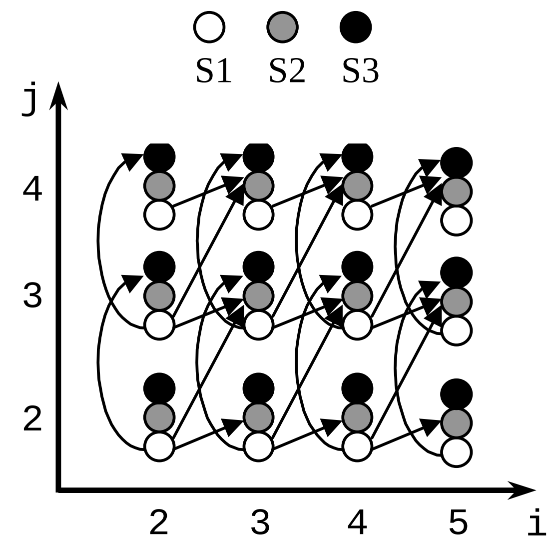
- Locality space $\mathrm{span}(\vec{e}_N, \vec{e}_{N-1}, \ldots, \vec{e}_{N-m-1})$ is defined by the *m* innermost loops.
  - Dimensionality of the locality space *m* is determined by the cache configuration and the number of iterations in the innermost loops.
  - Data locality optimization is to maximize the data reuse in the locality space.
  - Each level of memory hierarchy corresponds to one level of locality space s_i with $S_1 \subset S_2 \subset \cdots \subset S_H$ (H : level of the memory hierarchy)
- Cache configuration and data locality optimization are integrated with the concept of locality space

Compacting the Reuse Distance

- Compacting the Reuse Distance:
  - For a reuse vector $\vec{r}$ that is not in the locality space, i.e. $\vec{r} \notin \text{span}(\vec{e}_N, \vec{e}_{N-1}, \ldots, \vec{e}_{N-m-1})$, a non-singular transformation *T* can be applied s.t. $T\vec{r} \in \text{span}(\vec{e}_N, \vec{e}_{N-1}, \ldots, \vec{e}_{N-m-1})$
- For a set of reuse vectors:
  - The dimensionality of the reuse vector space
  - Choosing among reuse vectors for better locality: reuse quality, legal transformation
- Constructing legal transformation matrix
- To capture more reuse
  - Reducing the dimensionality of the reuse vector space
  - Increasing the dimensionality of the locality space

Reducing the Dimensionality of the Reuse Vector Space

```
for i = 2, n
  for j = 2, n
    x[i,j]=x[i,j]-128                      (S1)
    y[i,j]=0.5*(x[i-1,j-1]+x[i-1,j])       (S2)
    z[i,j]=x[i,j-1]                        (S3)
end
```



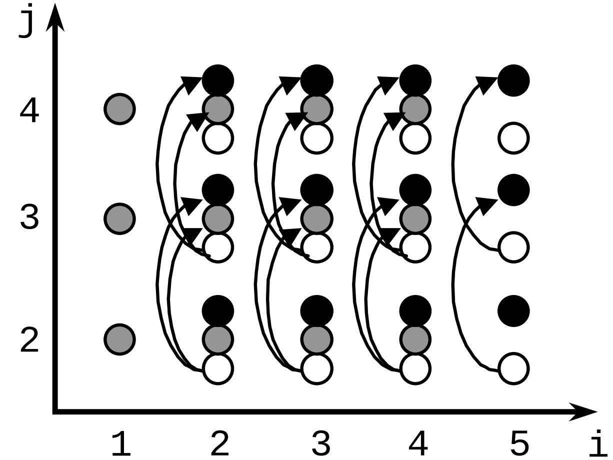
- Before loop alignment

  Spatial reuse (not shown in the figure):

  [0, 1]: S1→S1, [0, 1]: S2→S2, [0, 1]: S3→S3

  Temporal reuse

  [1, 0]: S1→S2, [1, 1]: S1→S2, [0, 1]: S1→S3



- After loop alignment [–1, 0]: S2

  Spatial reuse (not shown in the figure):

  [0, 1]: S1→S1, [0, 1]: S2→S2, [0, 1]: S3→S3

  Temporal reuse

  [0, 0]: S1→S2, [0, 1]: S1→S2, [0, 1]: S1→S3

Reducing the Dimensionality of the Reuse Vector Space (cont'd)

```
for i = 1, n
  for j = 2, n
    if i=1 then
      y[2,j]=0.5*(x[1,j-1]+x[1,j])
    else if i=n then
      x[n,j]=x[n,j]-128
      z[n,j]=x[n,j-1]
    else
      x[i,j]=x[i,j]-128
      y[i+1,j]=0.5*(x[i,j-1]+x[i,j])
      z[i,j]=x[i,j-1]
    endif
end
```

- Minimizing the Dimensionality of the Reuse Vector Space

Increasing the Dimensionality of the Locality Space

- Loop tiling increases dimensionality of locality space.
- Change the cache configuration:
  - Increasing the cache size
  - Reducing the line size if the program does not have good spatial locality
- The procedure:

  Loop alignment $\rightarrow$ non-singular transformation $\rightarrow$ loop tiling and cache configuration adjustment

More Spatial Reuse with Non-singular Transformation for Array Layout

- Locality space for arrays:

  span($\vec{e}_N, \vec{e}_{N-1}, \ldots, \vec{e}_{N-m-1}$) $\xrightarrow{A\vec{I}+c}$ span($\vec{a}_N, \vec{a}_{N-1}, \ldots, \vec{a}_{N-m-1}$)

  - ($\vec{a}_N, \vec{a}_{N-1}, \ldots, \vec{a}_{N-m-1}$) : the last m columns of the reference matrix A (or AT^{-1} with affine loop transformation T).
  - Let S = span($\vec{a}_N, \vec{a}_{N-1}, \ldots, \vec{a}_{N-m-1}$): locality space for the array
- Creating spatial reuse
  - If the fastest changing dimension of an array is not in its locality space, i.e. $\vec{e}_f \notin S$, a non-singular transformation T_A can be applied to the array layout s.t. $\vec{e}_f \in T_A S$
- Dimension interchange: $\{\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_A\} \cap S \neq \phi$

In-dimension Stride Vector

- *In-dimension Stride Vector* (ISV)
  - Distance vector between two iterations that access adjacent data within the same dimension of an array
- Why ISV?
  - Each dimension of the array has its own ISV
  - If the dimension is switched to be the fastest changing dimension, its ISV becomes the self-spatial reuse vector
  - Automate the unification of loop transformation and array dimension interchange
- How to compute ISV: $\mathrm{ISV}_j = \ker(A_j) - \ker(A)$

An Example

```
for i = 1, N
   for j = 1, N
      for k = 1, N
         c(i,j) = c(i,j) + a(i,k)*b(k,j)
end
```

Spatial locality

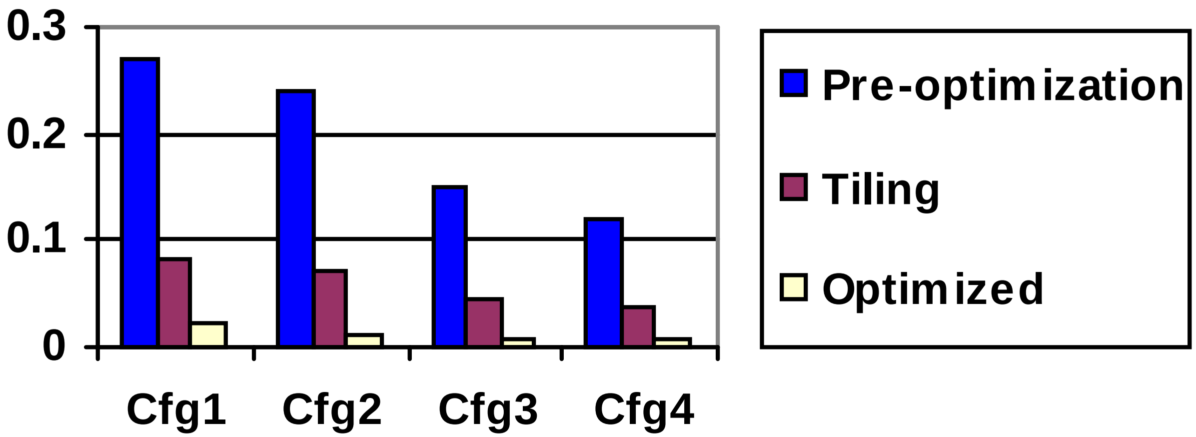
Temporal locality

ISV spaces:
Array c: 1st ISV space: *span*{(1, 0, 0)}, 2nd ISV space: *span*{(0, 1, 0)}
Array a: 1st ISV space: *span*{(1, 0, 0)}, 2nd ISV space: *span*{(0, 0, 1)}
Array b: 1st ISV space: *span*{(0, 0, 1)}, 2nd ISV space: *span*{(0, 1, 0)}

One possible transformation. Assuming row major, T : (1, 0, 0) $\Rightarrow$ (0, 0, 1)

```
for l = 1, N
   for m = 1, N
      for n = 1, N
         c(m,n) = c(m,n) + a(l,n)*b(m,l)
end
```

Experimental Result (Matrix Multiplication)



Cfg1: *n*=128, *l*=8, *a*=1, Cfg2: *n*=128, *l*=8, *a*=2

Cfg3: *n*=256, *l*=8, *a*=1, Cfg4: *n*=256, *l*=8, *a*=2

n : number of line sets, *l* : line size in words

a : degree of associativity

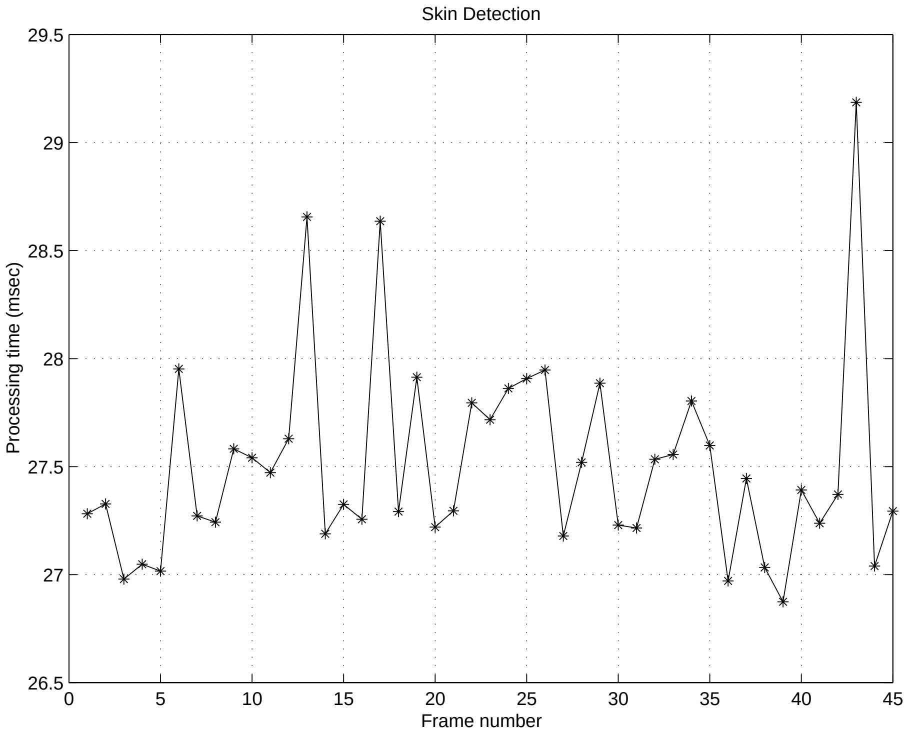
Figure: Miss rate for Matrix Multiplication

Multiprocessor architectures for video

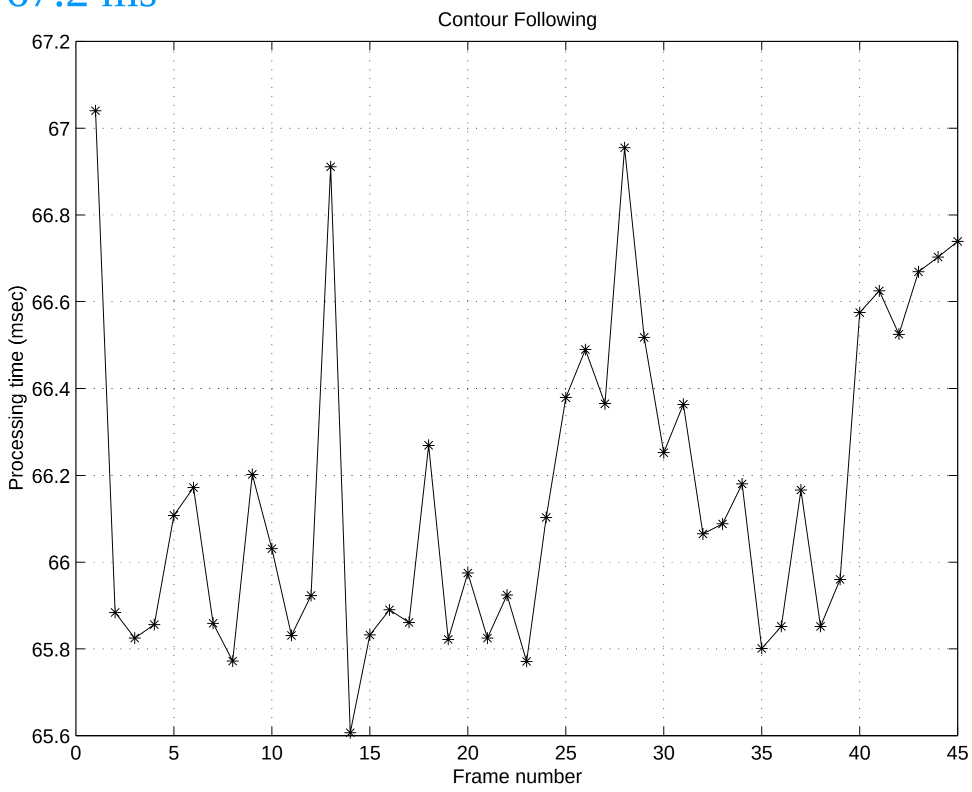
- One VLIW is not a good idea:
  - limited ability to extract parallelism from one process;
  - multiple processes are not easily described for instruction-level scheduling;
  - applications have natural decomposition.
- Symmetric multiprocessor is bad:
  - don’t want all shared memory space;
  - longer wires lead to more power.

Smart camera CPU times

29.5 ms



67.2 ms

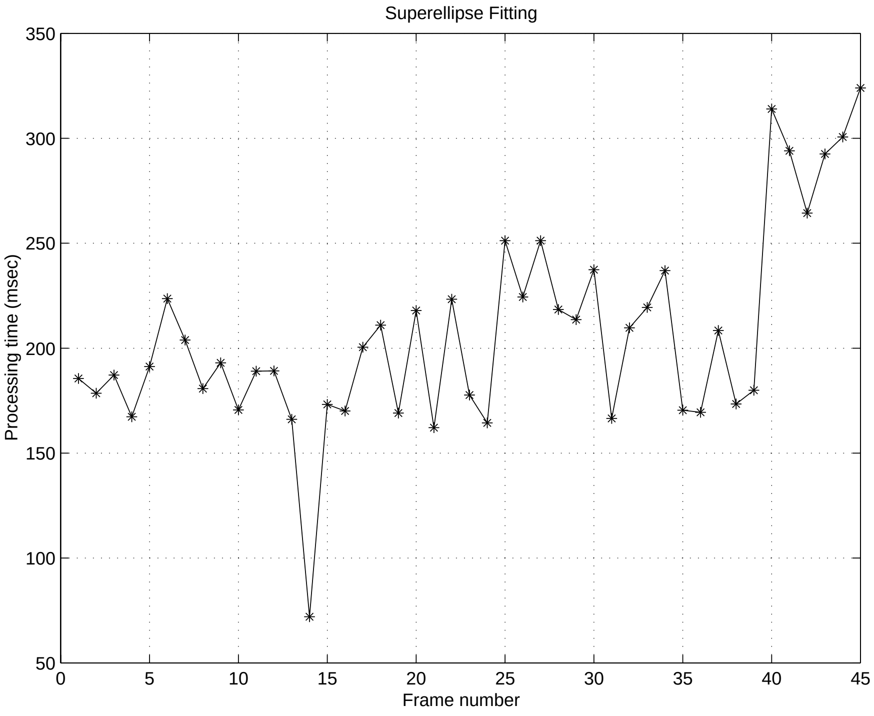


Skin detection

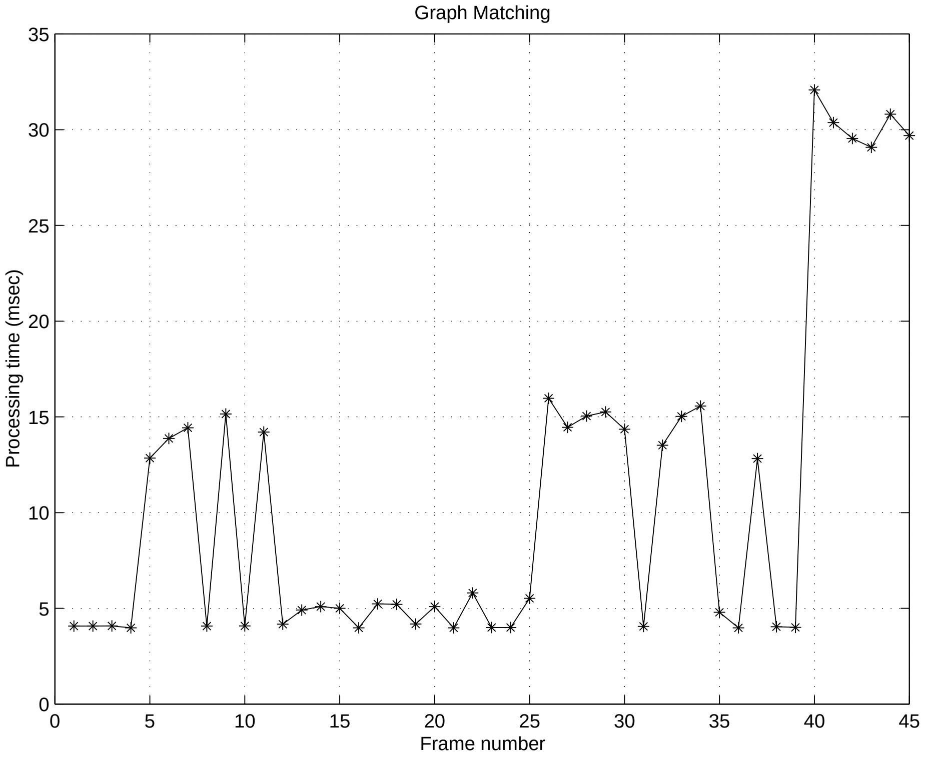
Contour detection

Smart camera CPU times, cont'd.

250 ms



35 ms



Superellipse fitting

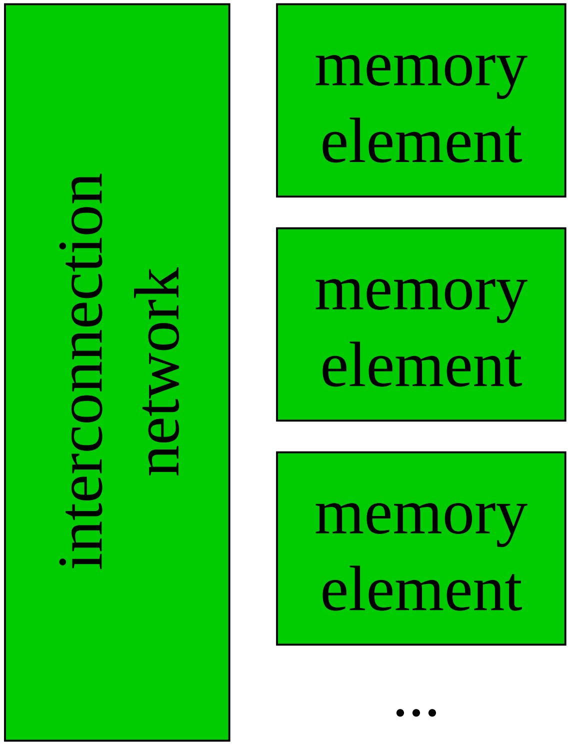
Graph matching

Observations on smart camera application

- Feed-forward communication.
- Somewhat unbalanced process-process CPU times.
- Significant variation in frame-to-frame CPU time.

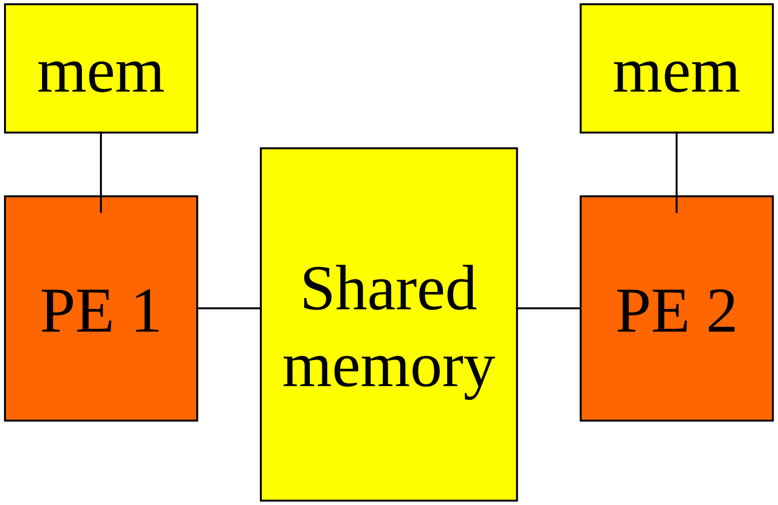
Problems with uniform shared memory

- Conflicts cause scheduling problems.
- Statically-scheduled compiler has problems with:
  - depth of scheduling;
  - non-deterministic conflicts.



Local shared memories

- Use locally shared memories to provide more predictable computation times.
- Provide API for interprocess communication.



Heterogeneous architectures

- Different phases have very different characteristics:
  - pixel-oriented;
  - line-oriented;
  - floating-point parameter matching.
- Different processing elements can be used for different stages.

Summary

- Multimedia applications are already more complex and will become more so:
  - multiple algorithms;
  - complex control and data.
- Instruction-level parallelism helps, but isn’t enough to handle complex applications.