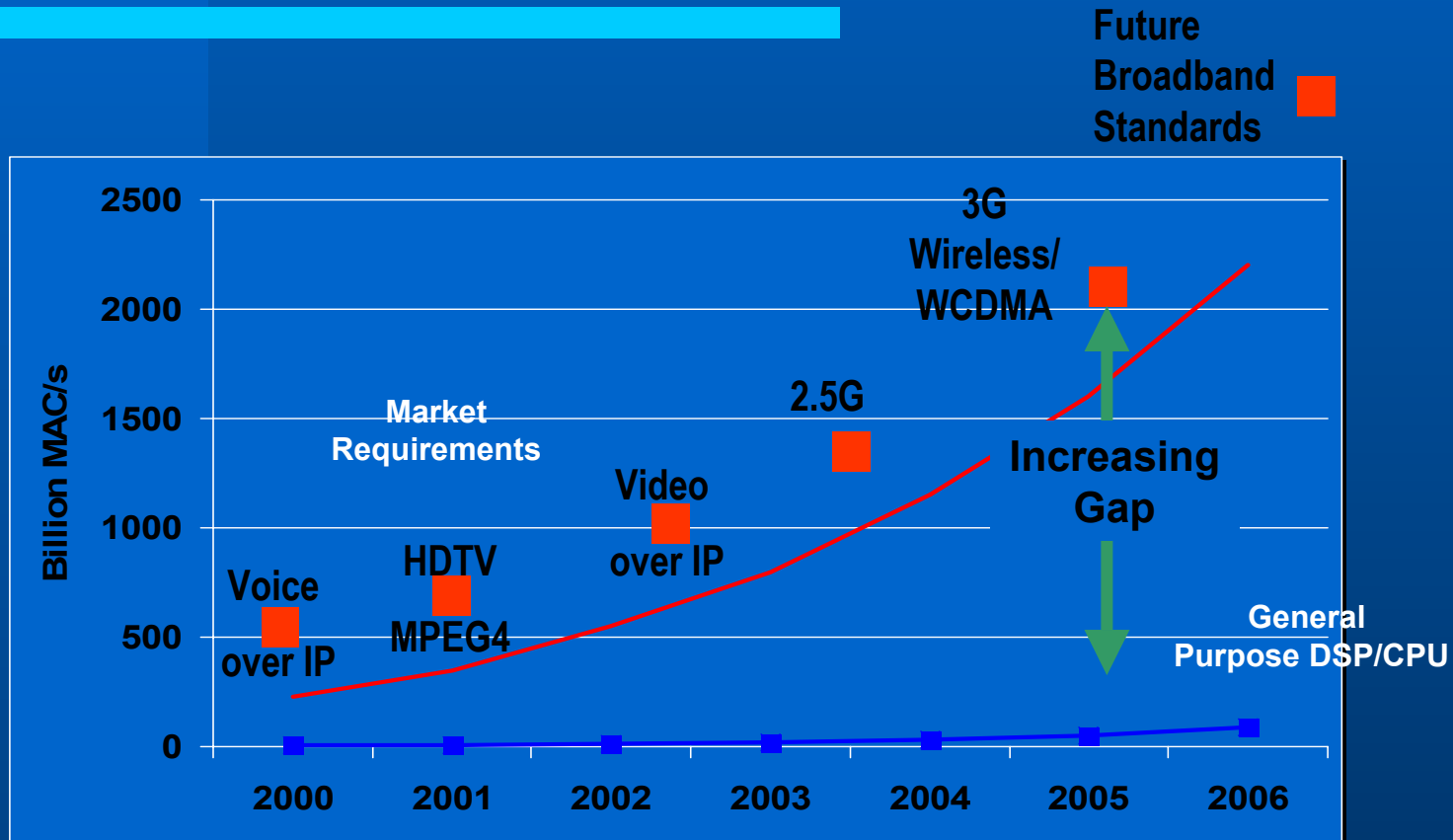


System Design Using Kahn Process Networks: The Compaan/Laura Approach

Bart Kienhuis
Assistant Professor
LIACS, Leiden University
The Netherlands

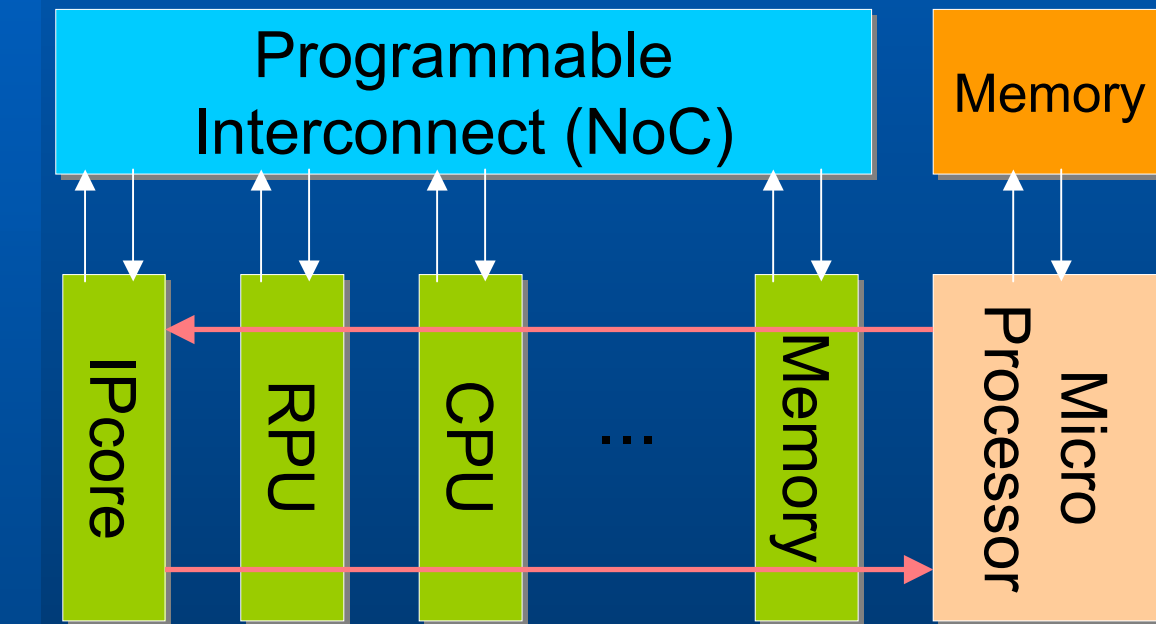
DSP Performance Requirements



Source: TI, Xilinx – 1 MAC = 8 bit Multiply-Accumulate

Applications have a ferocious appetite for more programmable compute power

Embedded DSP Architectures



Weakly coupled
Processing elements

CPU: A simple Microprocessor
RPU: Reconfigurable Processing Unit
IPcore; Dedicated Accelerator block
NoC: Network on a Chip

Programming Problem

EASY to specify

Sequential
Application Specification

```
for j = 1:1:N,  
  [x(j)] = Source1( );  
end  
for i = 1:1:K,  
  [y(i)] = Source2( );  
end  
for j = 1:1:N,  
  for i = 1:1:K,  
    [y(i), x(j)] = F( y(i), x(j) );  
  end  
end  
for i = 1:1:K,  
  [Out(i)] = Sink( y( i ) );  
end
```

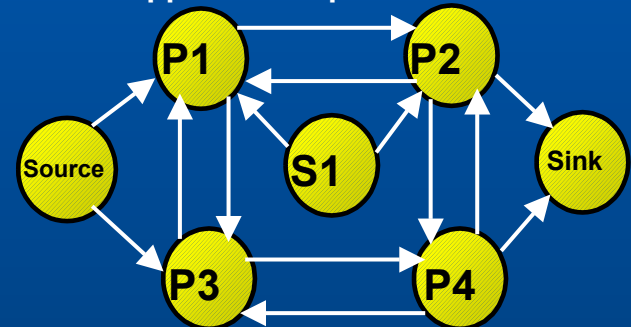
DIFFICULT to map

Application

Compaan

DIFFICULT to specify

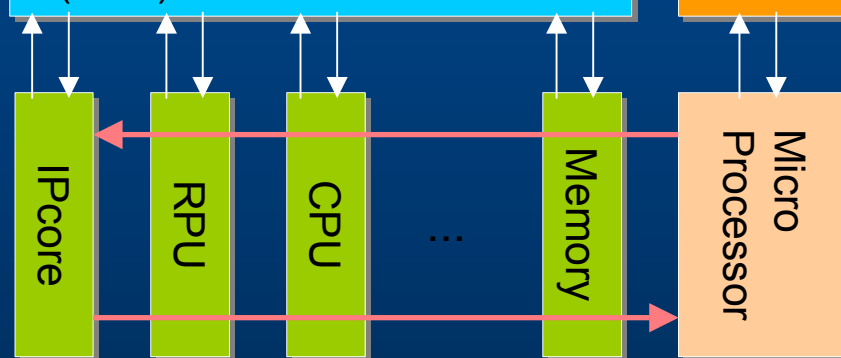
Parallel
Application Specification



EASY to map

Programmable Interconnect
(NoC)

Memory

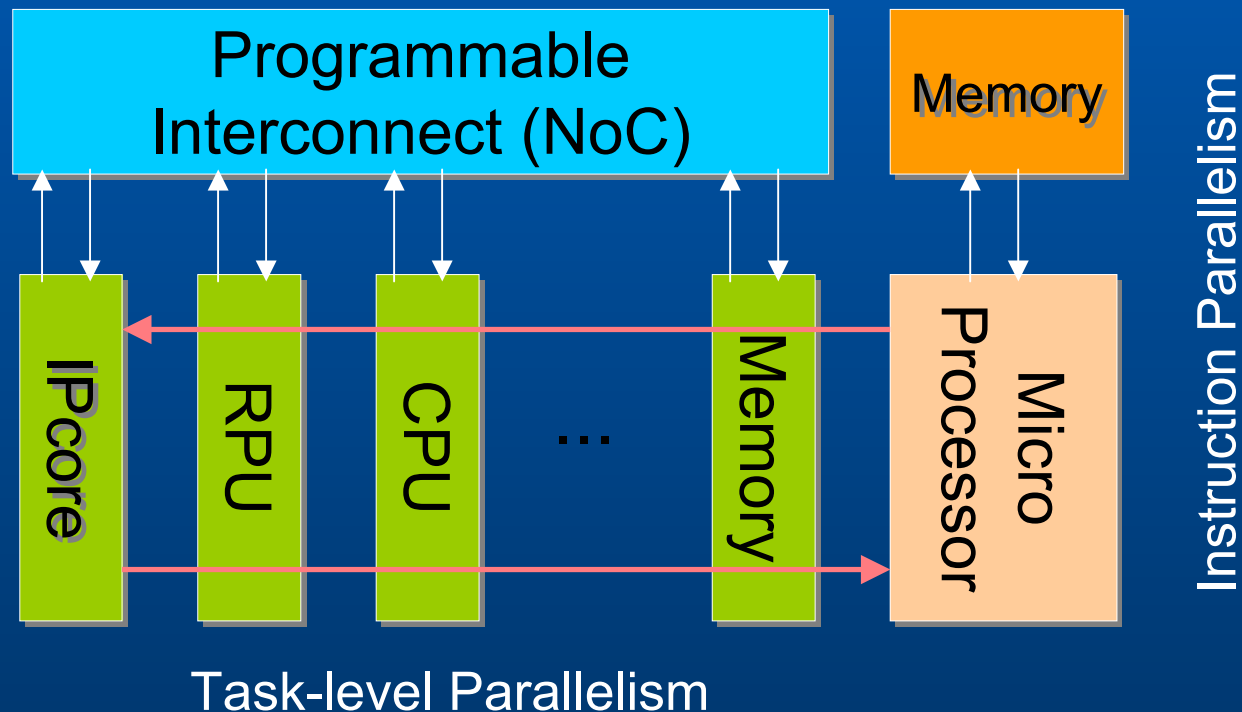


Laura

Outline

- The programming problem
- Kahn Process Networks
- System Design: Compaan/Laura Approach
- Case-study M-JPEG
- Conclusions

Embedded DSP Architectures



To satisfy the computational requirements, these architectures have to exploit:

- Distributed Control
- Distributed Memory

QR Algorithm (smart antennas)

Matrices are located in
Big Global Memory

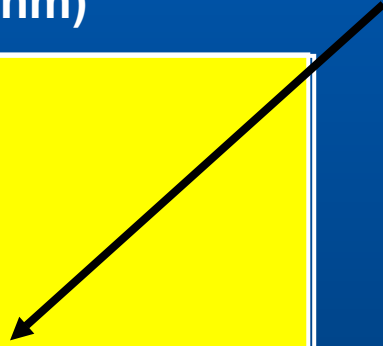
Matlab Code (QR Algorithm)

Sequentially Ordered



```
%parameter N 8 16;
%parameter K 100 1000;

for k = 1:1:K,
    for j = 1:1:N,
        [ r(j,j), x(k,j), t ]=Vectorize( r(j,j), x(k,j) );
        for i = j+1:1:N,
            [ r(j,i), x(k,i), t]=Rotate( r(j,i), x(k,i), t );
        end
    end
end
```



QR simple program: but keeps your CPU very busy

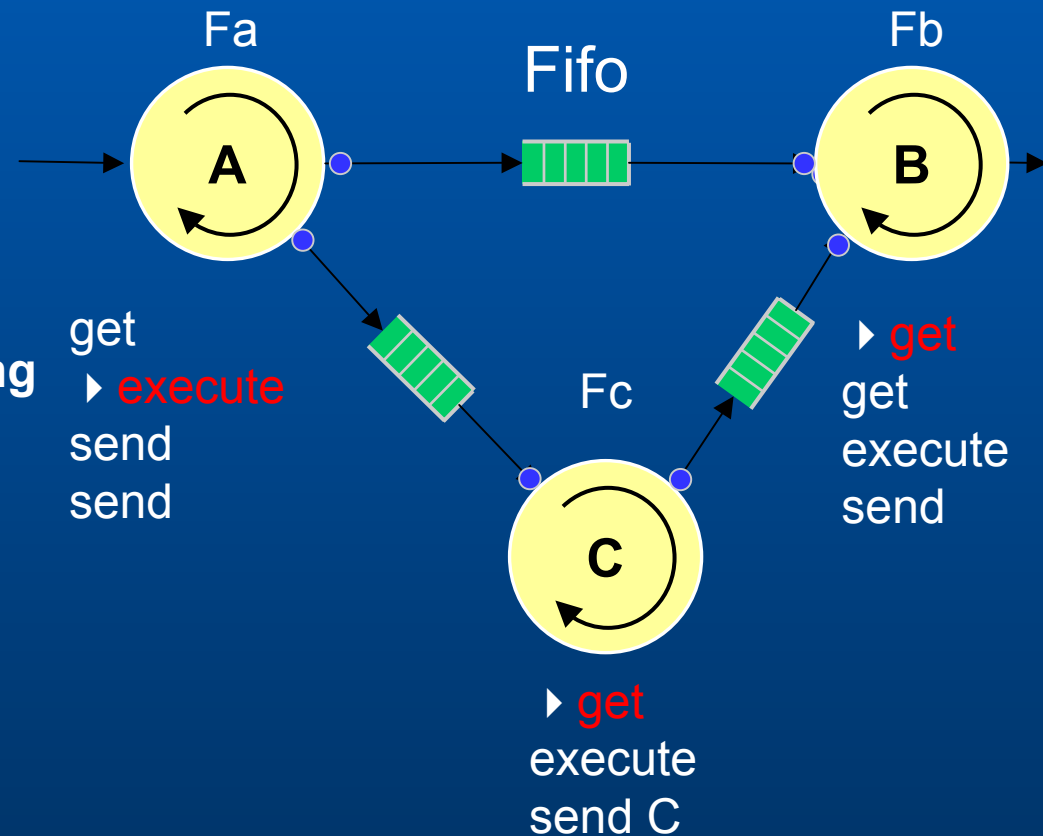
Solution

- Change the model of computation in such a way that it better fits the model of architecture.
- Make sure the data-type is of precisely the format that fits the architecture (e.g. Streams)
- What model of Computation would fit this description, when looking at Digital Signal Processing (DSP) applications, Imaging and Multi Media?

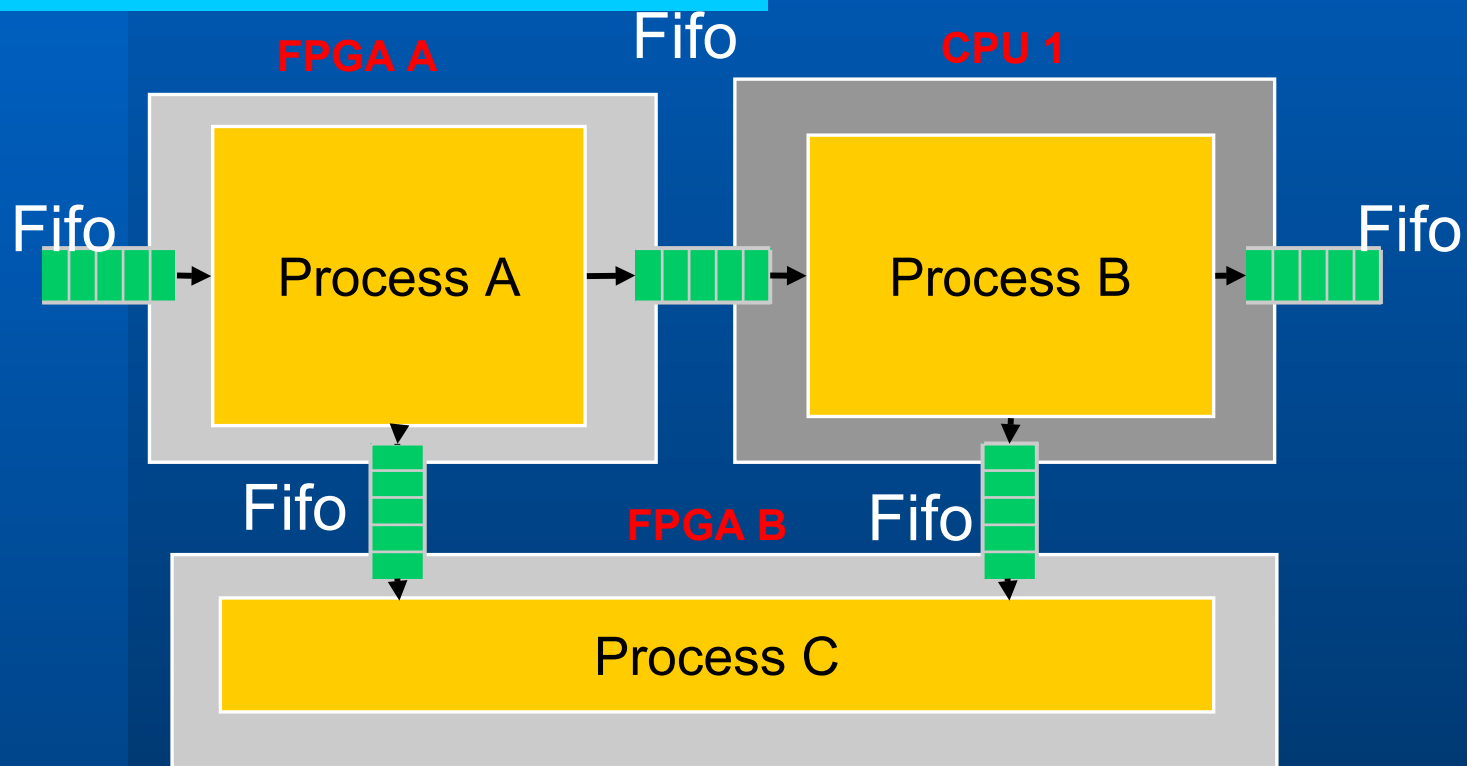
Kahn Process Networks

Kahn Process Network (KPN)

- **Kahn Process Networks**
[Kahn 1974][Parks&Lee 95]
 - Processes run autonomously
 - Communicate via unbounded FIFOs
 - Synchronize via blocking read
- **Process is either**
 - executing (**execute**)
 - communicating (**send/get**)
- **Deterministic**
- **Distributed Control**
- **Distributed Memory**



Kahn Process Network (KPN)



- Autonomously operating Processes; no global schedule needed
- Blocking Read simple realize in Hardware
- Buffer Sizes of the FIFOs are quite often very small

The Compaan Tool Chain

Matlab Application

MatParser

Data Dependency
Analysis

SAC

DgParser

Polyhedral Reduce Dependence Graph
(PRDG)

PRDG

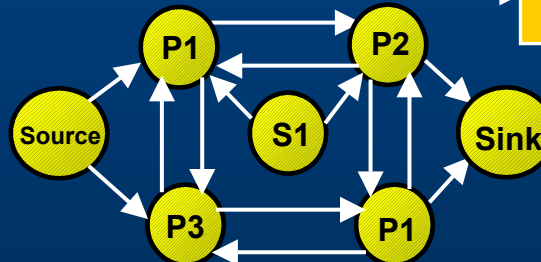
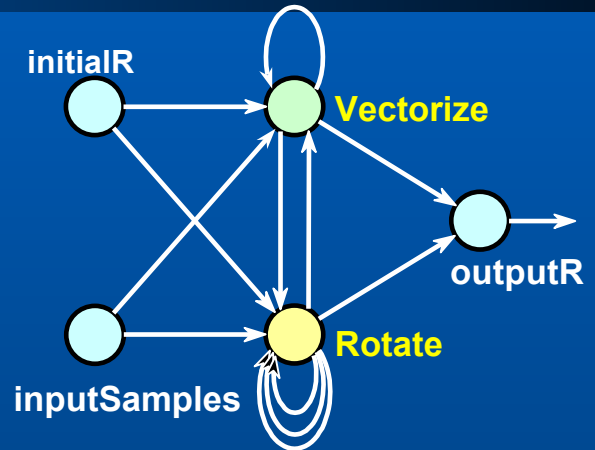
Panda

Linearization

Kahn Process
Network

Matlab Program

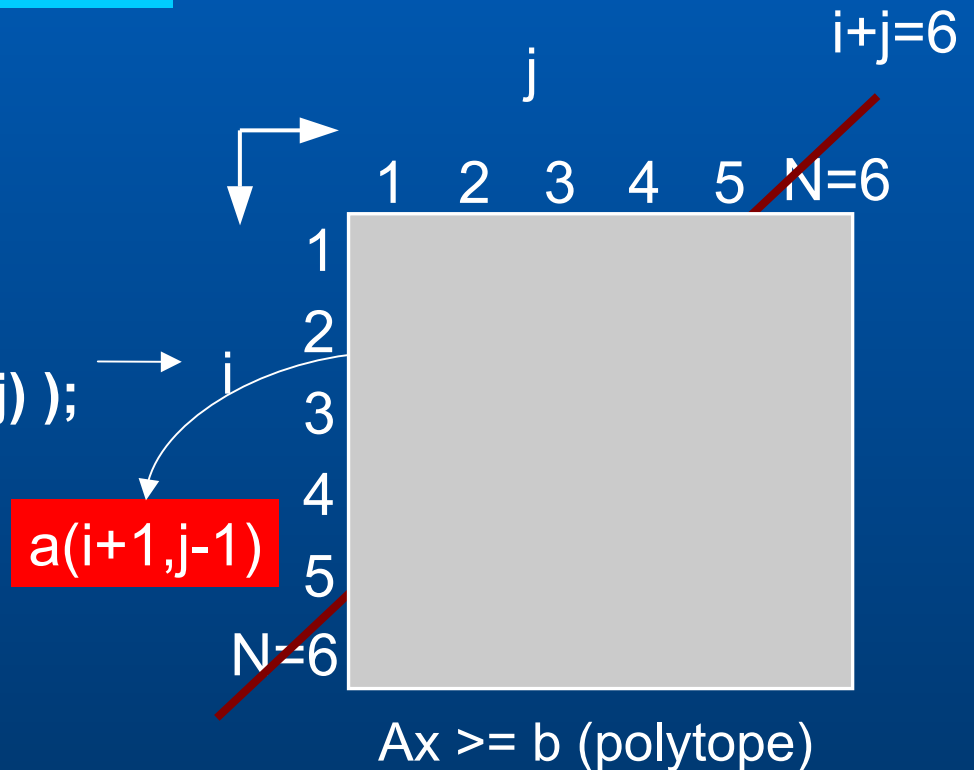
```
%parameter N 8 16;  
%parameter K 100 1000;  
  
for k = 1:1:K,  
  for j = 1:1:N,  
    [r(j,j), x(k,j), t]=Vectorize( r(j,j), x(k,j) );  
    for i = j+1:1:N,  
      [r(j,i), x(k,i), t]=Rotate( r(j,i), x(k,i), t );  
    end  
  end  
end
```



Process Network

Data Dependency Analysis

```
for i= 1 : 1 : N,  
  for j= 1 : 1 : N,  
    [ a(i+j) ] = funcA( a(i+j) );  
  end  
end
```



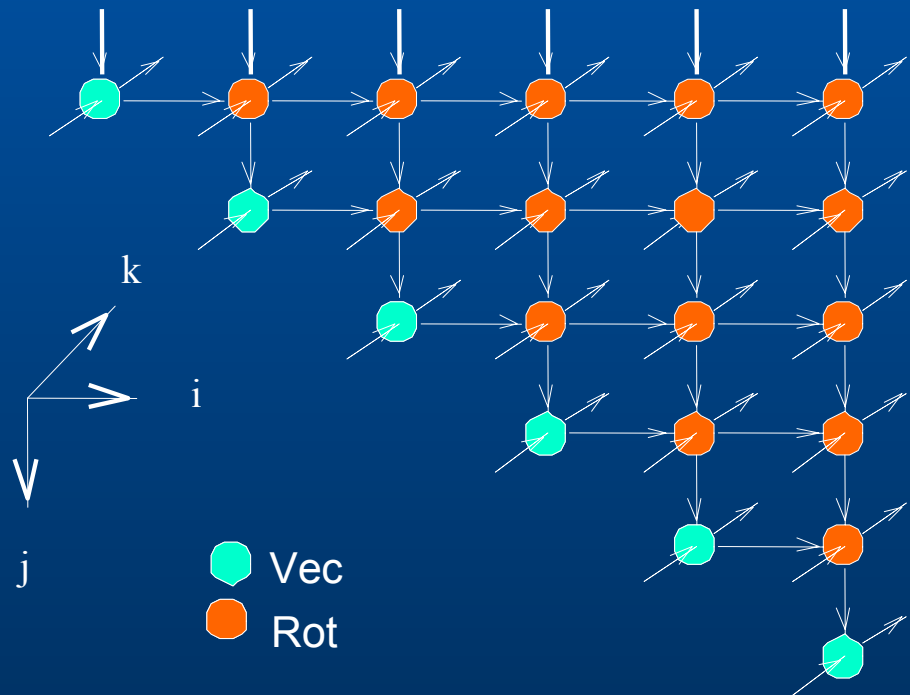
The for-next loops define an Iteration Domain

Polyhedral Reduced Dependence Graph

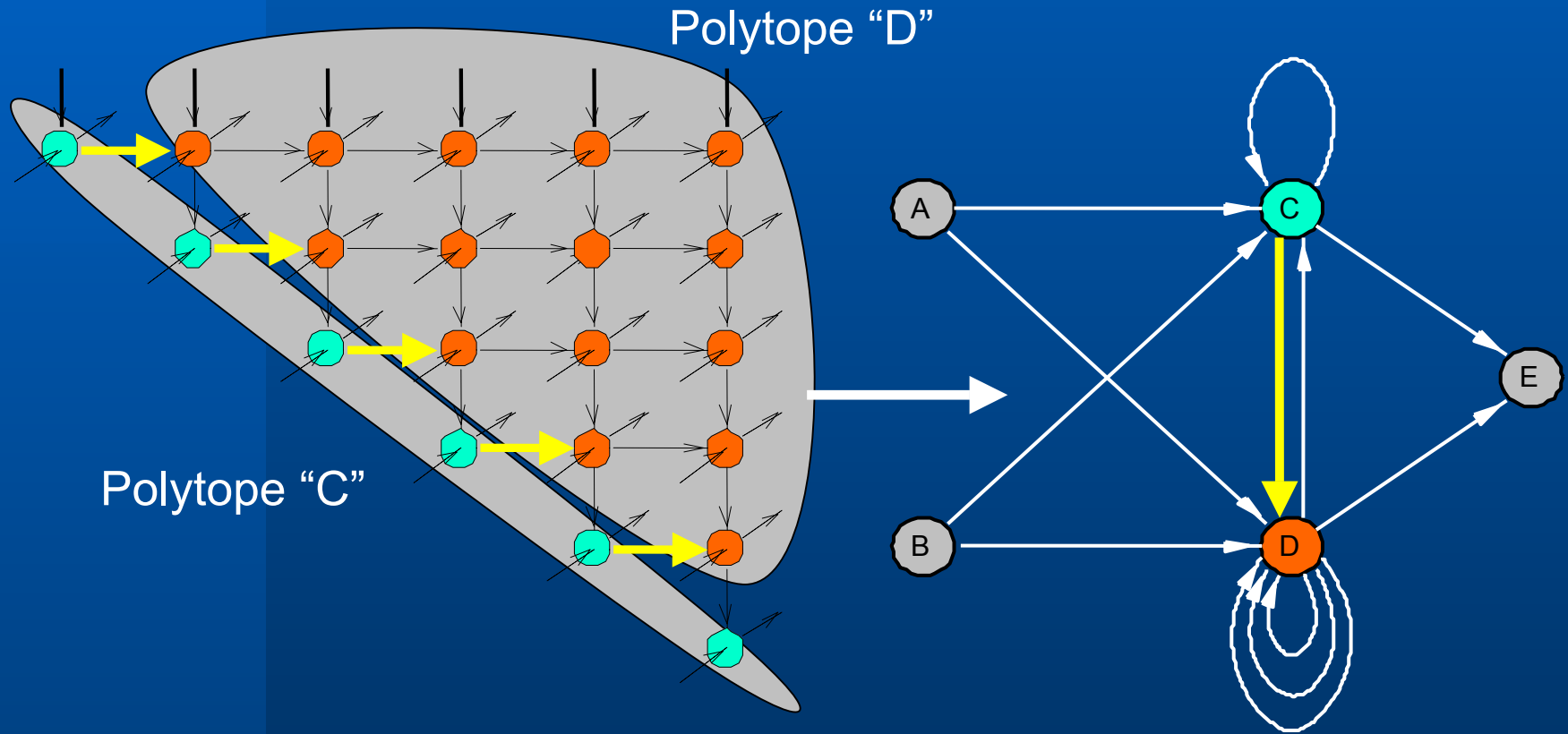
Matlab Code (QR Algorithm)

```
%parameter N 8 16;  
%parameter K 100 1000;  
  
for k = 1:1:K,  
    for j = 1:1:N,  
        [ r(j,j), x(k,j), t ]=Vectorize( r(j,j), x(k,j) );  
        for i = j+1:1:N,  
            [ r(j,i), x(k,i), t]=Rotate( r(j,i), x(k,i), t );  
        end  
    end  
end
```

Dependence Graph

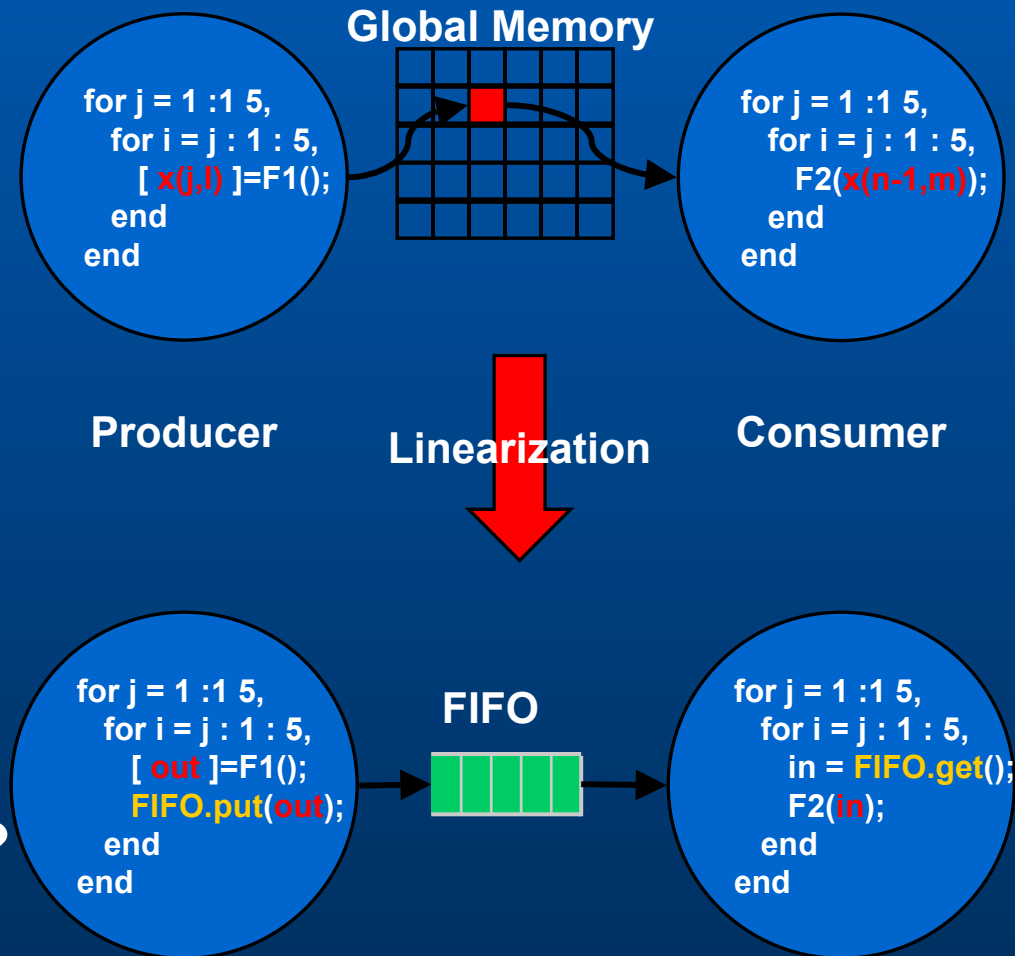


Polyhedral Reduced Dependence Graph



Linearization

- Linearization is the process of mapping high-order data-structures (e.g., Matrices) on a 1-D stream
- We replaced the indexing of the variable $x(j,l)$ and $x(n-1,m)$ by relative **put** and **get** operation on a FIFO buffer (unboxing)
- Is this always possible?

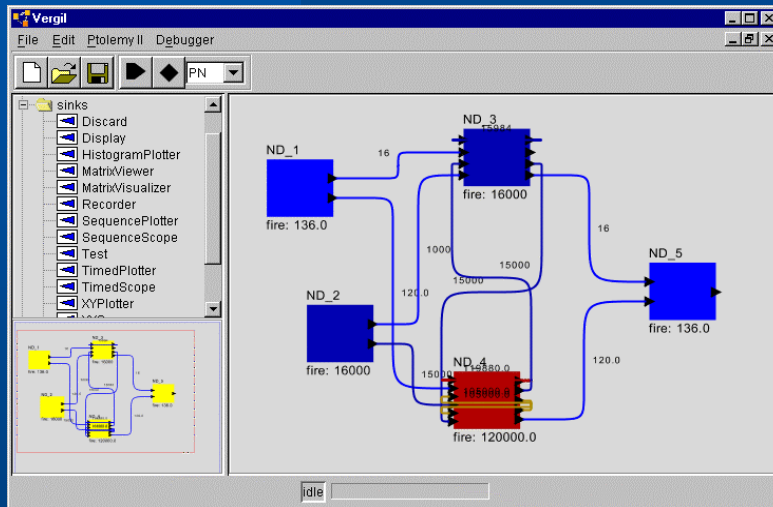


KPN Hand Off

Kahn Process Network

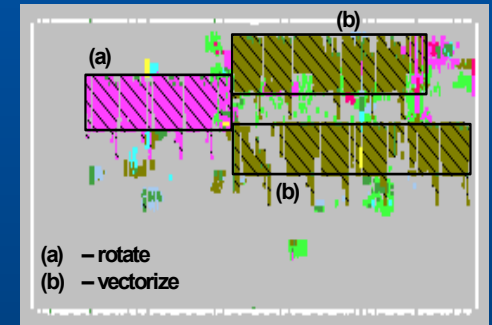
Ptolemy Actor
models in Java

C++/YAPI

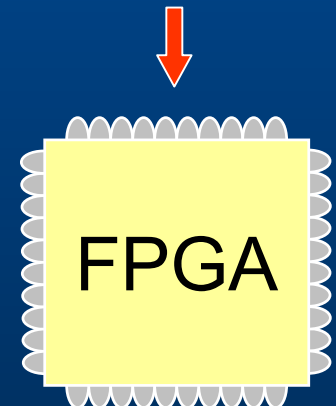


Functional Simulation in Ptolemy II

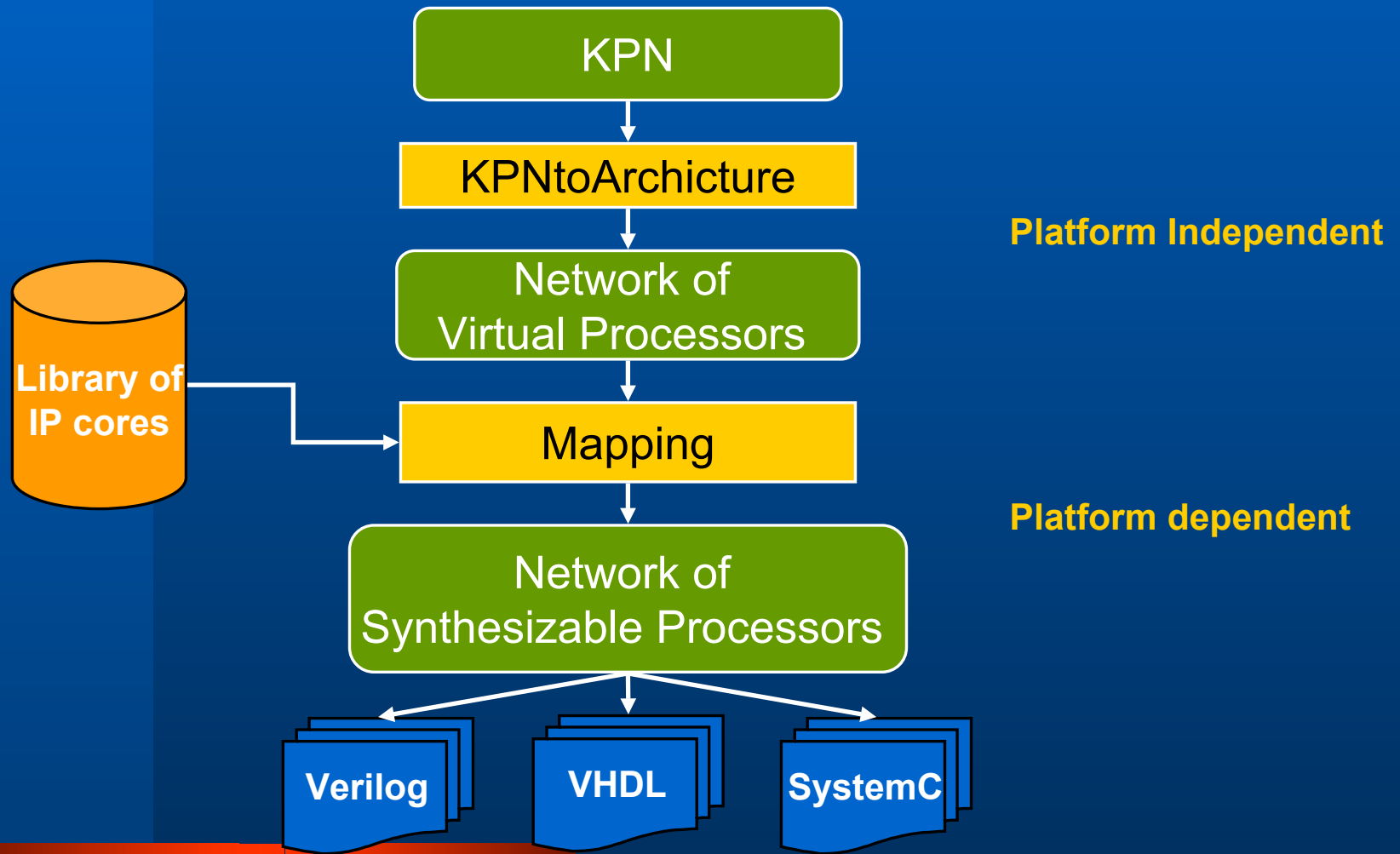
Synthesizable VHDL



Laura

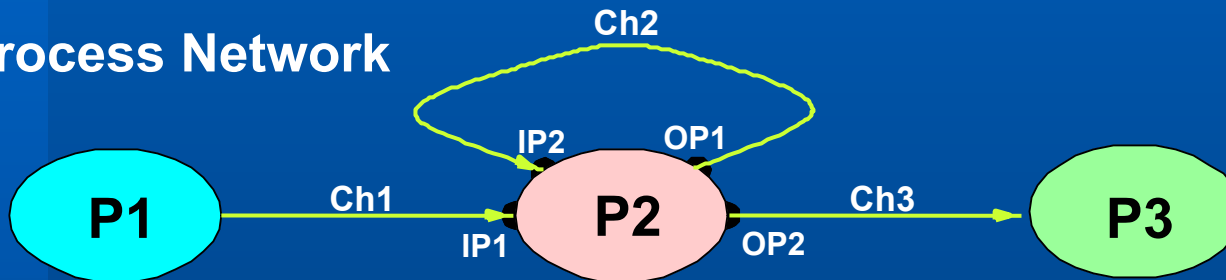


The Laura Tool



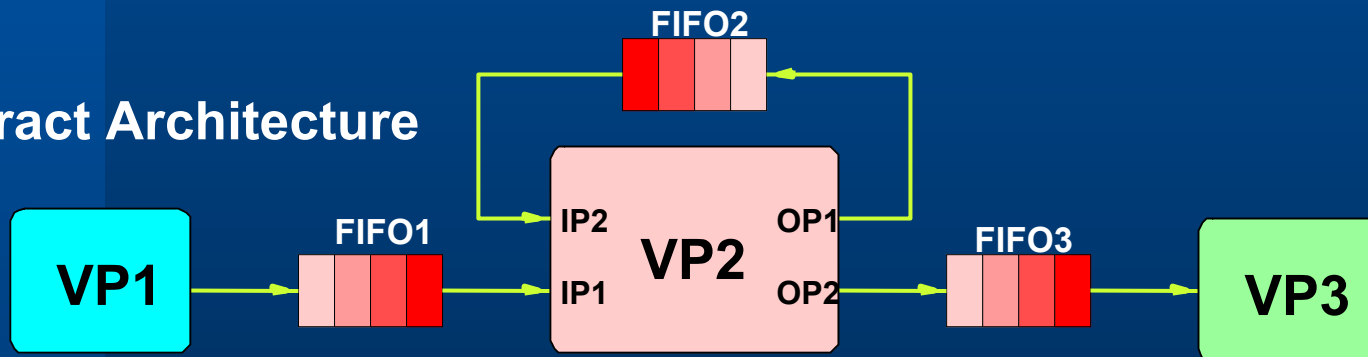
The Laura Tool

Kahn Process Network

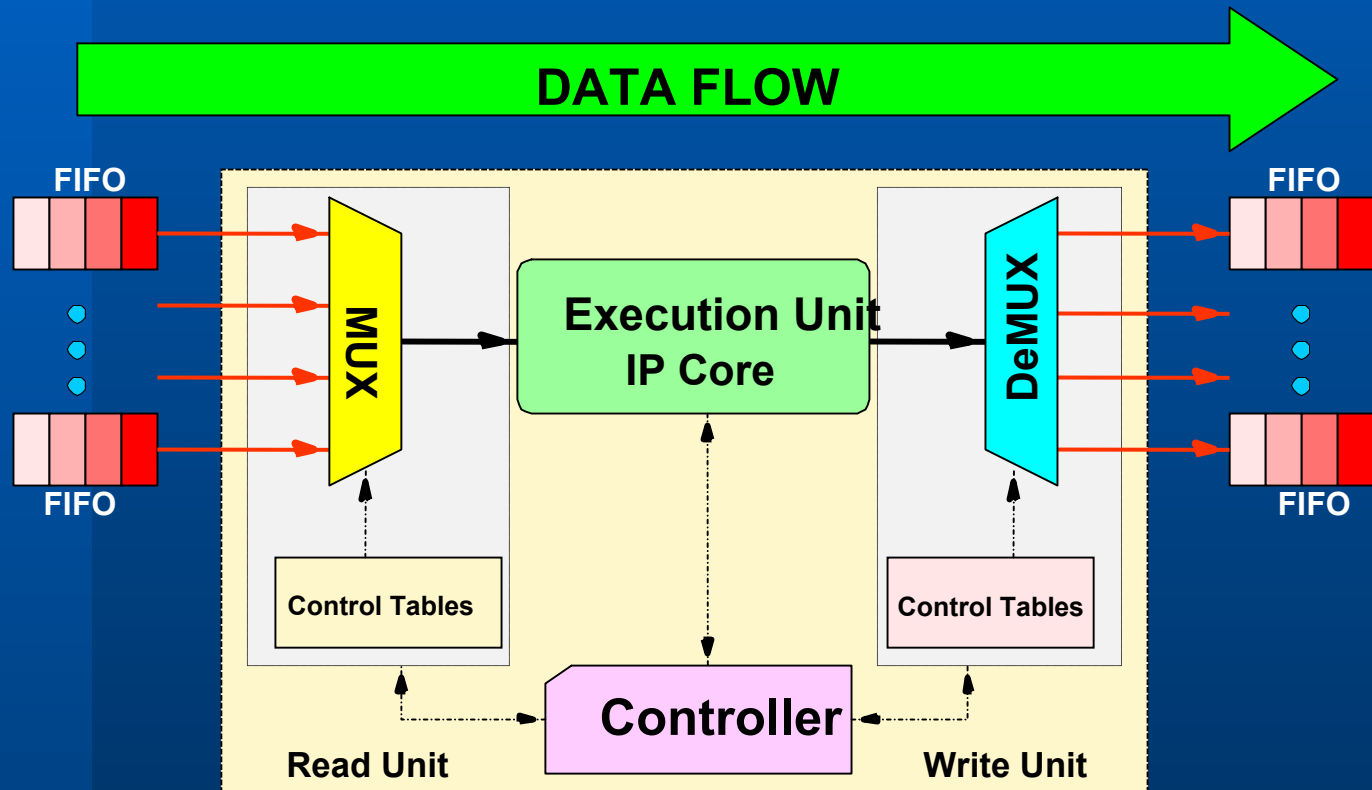


KPNToArchitecture

Abstract Architecture



The Laura Tool

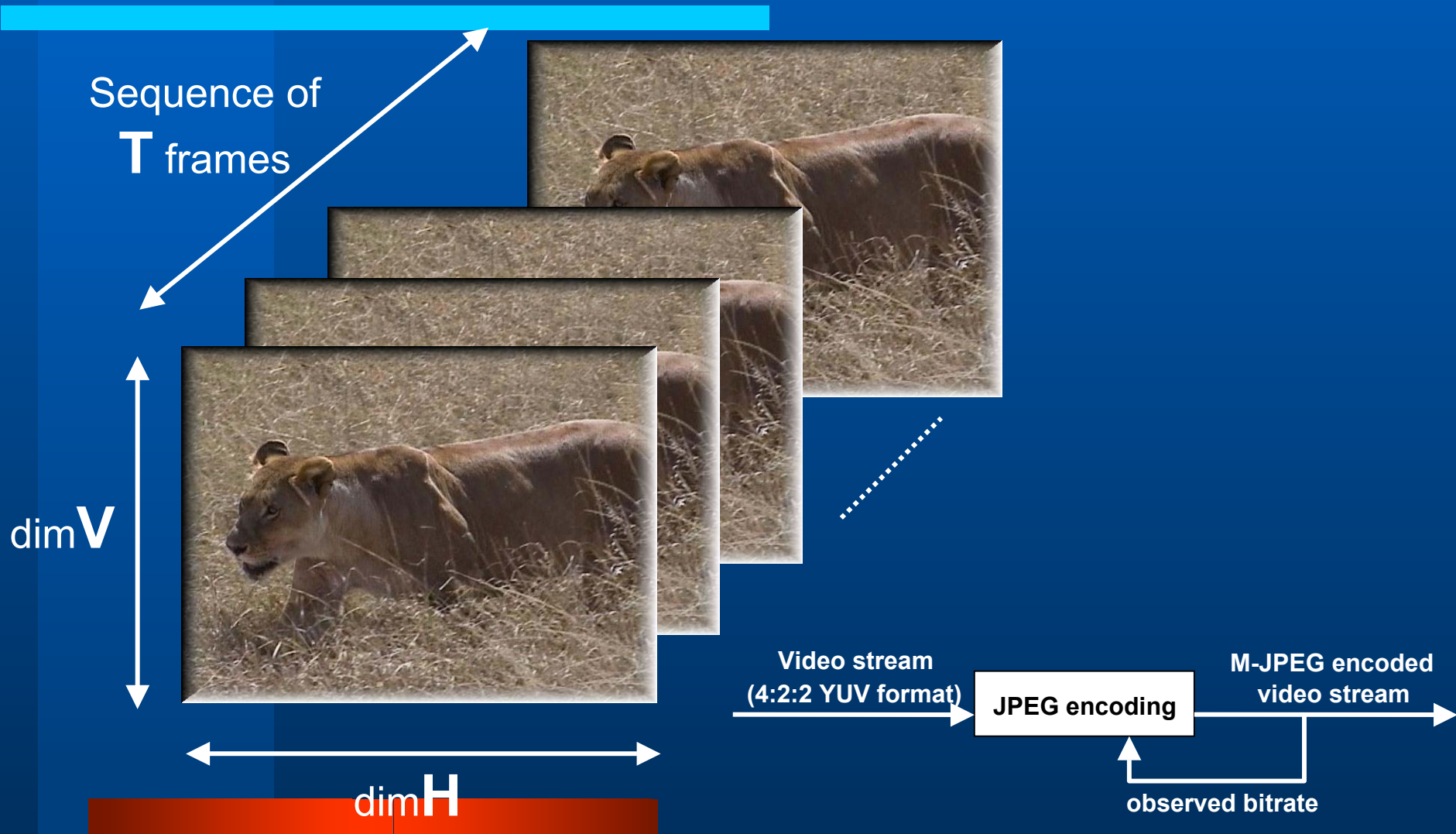


Structure of an individual processor

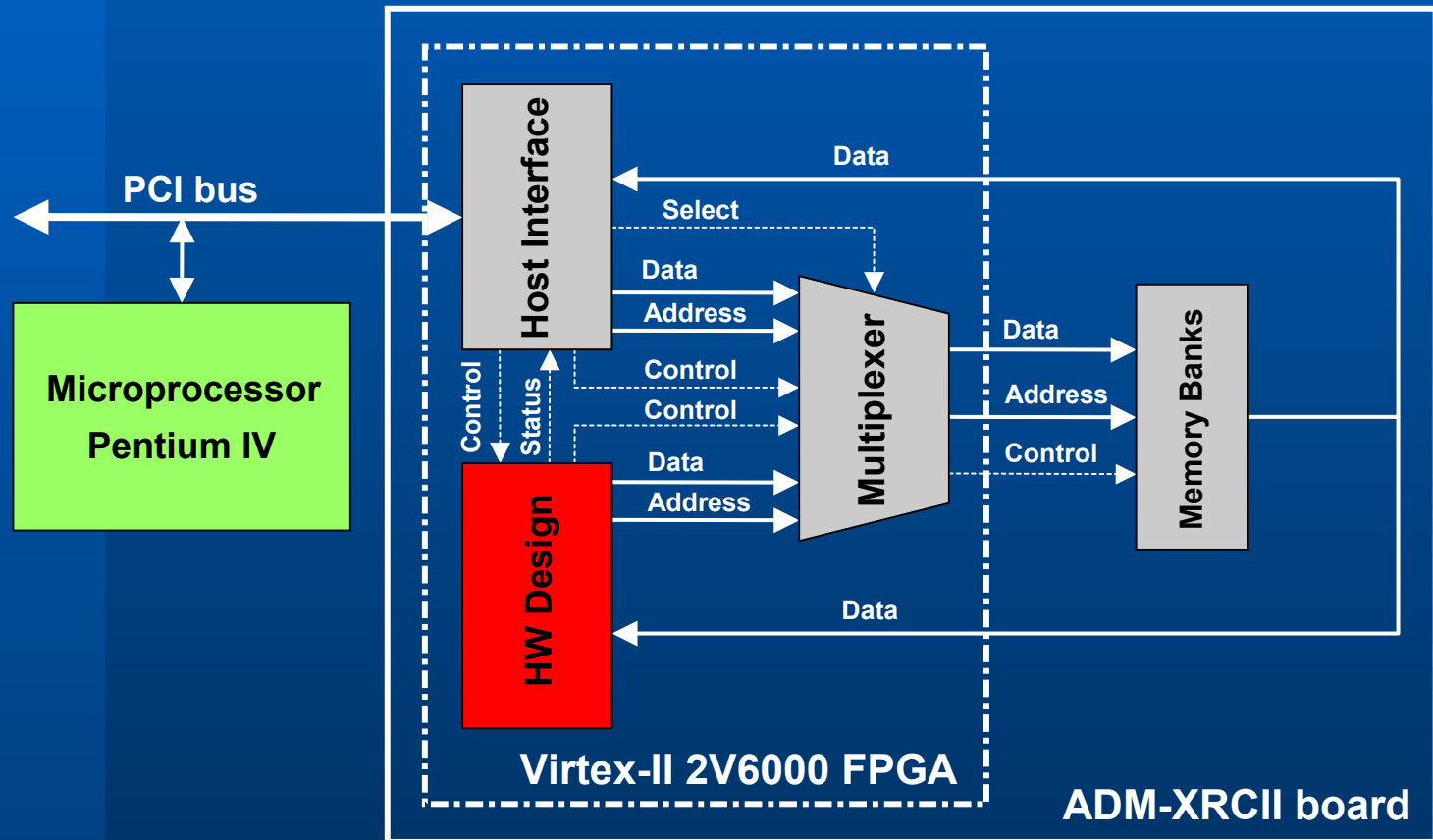
System Design Flow

- **The Tools in action**
 - **M-JPEG Example based on the original C-code of the Portable Video Research Group, Stanford University.**
 - **Simple Target Platform**
 - **Common PC platform**
 - **With FPGA board**

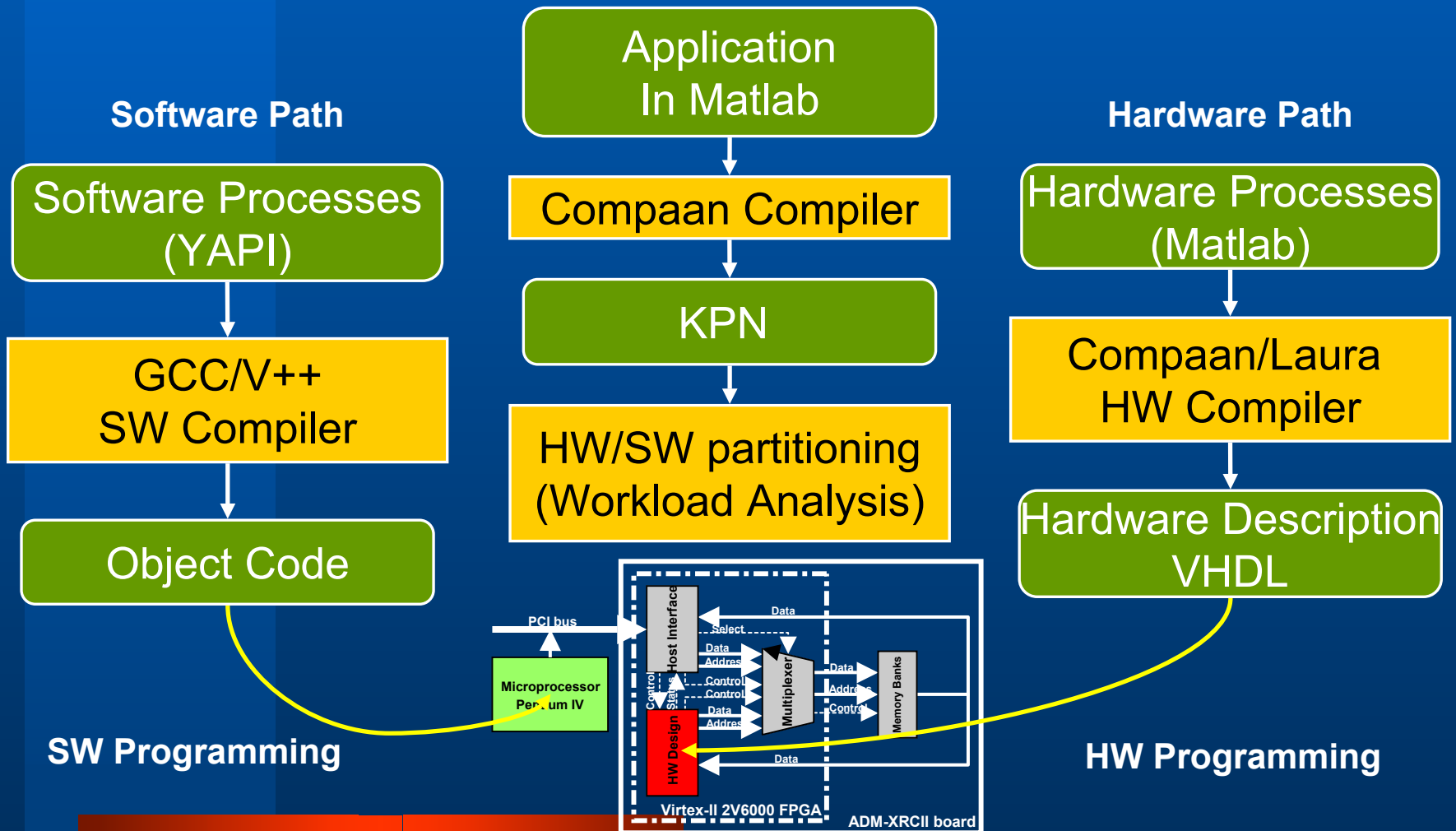
Motion JPEG encoder



Target Platform



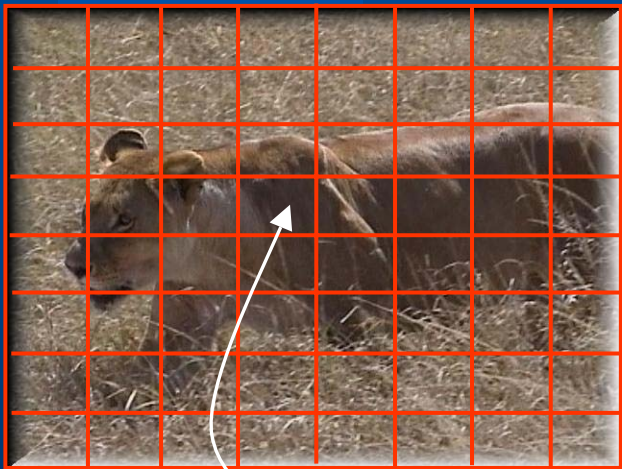
System Design Flow



M-JPEG Specification in Matlab

Parameterized

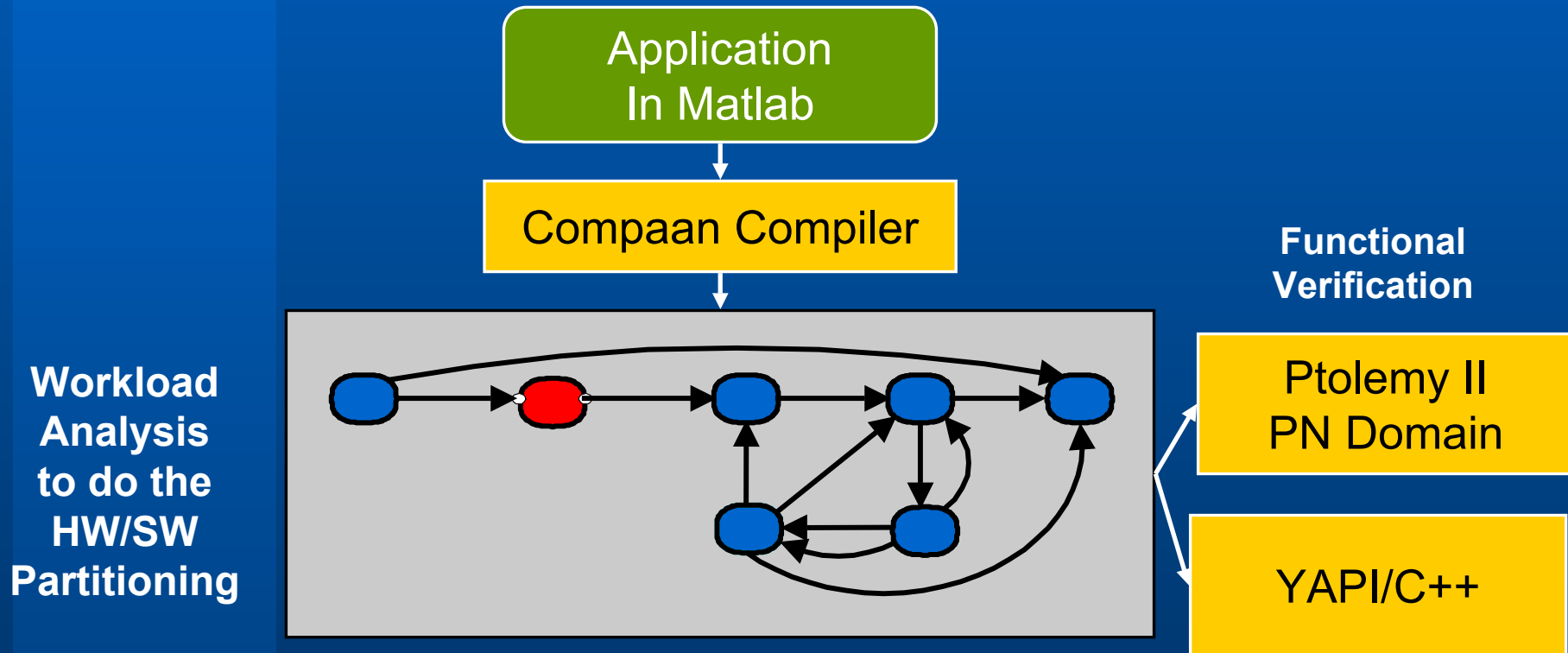
```
%parameter NumFrames 1 1000;  
%parameter VNumBlocks 16 256;  
%parameter HNumBlocks 8 256;
```



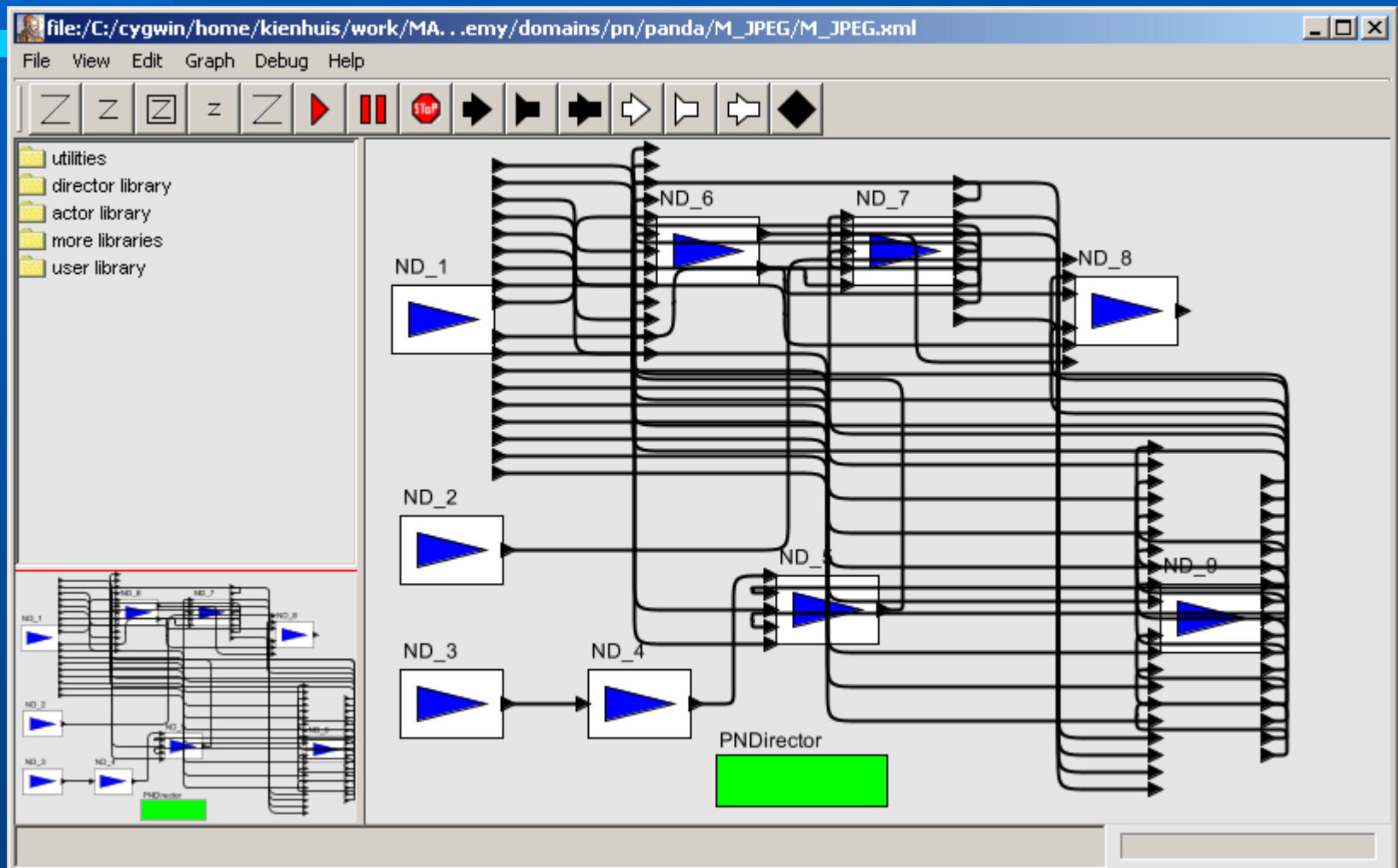
Block(j , i)

```
[ QTables, HuffTables, TablesInfo, EndOfFrame ] = P2_I_DefaultTables( );  
for k = 1:1:NumFrames,  
    [ HeaderInfo ] = P1_I_VideoInit( );  
    for j = 1:1:VNumBlocks,  
        for i = 1:1:HNumBlocks,  
            [ Block( j , i ) ] = P1_I_VideoInMain( );  
        end  
    end  
    for j = 1:1:VNumBlocks,  
        for i = 1:1:HNumBlocks,  
            [ Block( j , i ) ] = DCT( Block( j , i ) );  
        end  
    end  
    for j = 1:1:VNumBlocks,  
        for i = 1:1:HNumBlocks,  
            [ Block( j , i ) ] = Q( Block( j , i ), QTables );  
            [ Packets, StatisticsB ] = VLE( Block( j , i ), EndOfFrame, HuffTables );  
            [ BitRate, StatisticsF, EndOfFrame ] = CtrlIF1( StatisticsB );  
            [ ] = VideoOut( HeaderInfo, TablesInfo, Packets );  
        end  
    end  
    [ QTable, HuffTables, TablesInfo ] = P2_I_CtrlIF2( BitRate, StatisticsF,  
                                                    QTables, HuffTables, TablesInfo );  
end
```

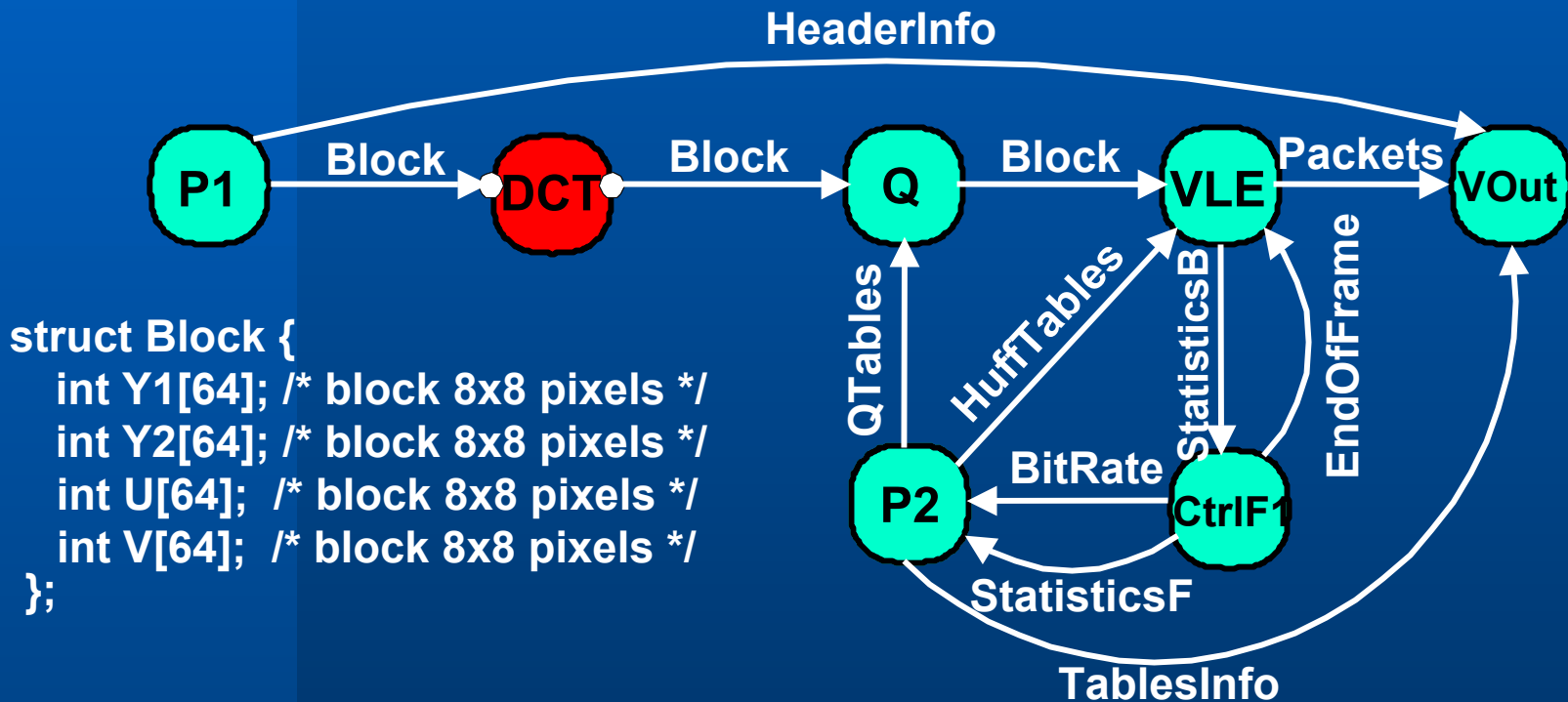

Deriving a KPN



Deriving a KPN



The KPN of M-JPEG



Interface Code DCT

- The DCT process is selected to move to Hardware.
- Interface code is needed to run with the Software Processes
- Observe that 'Blocks' are being moved to the FPGA and from the FPGA

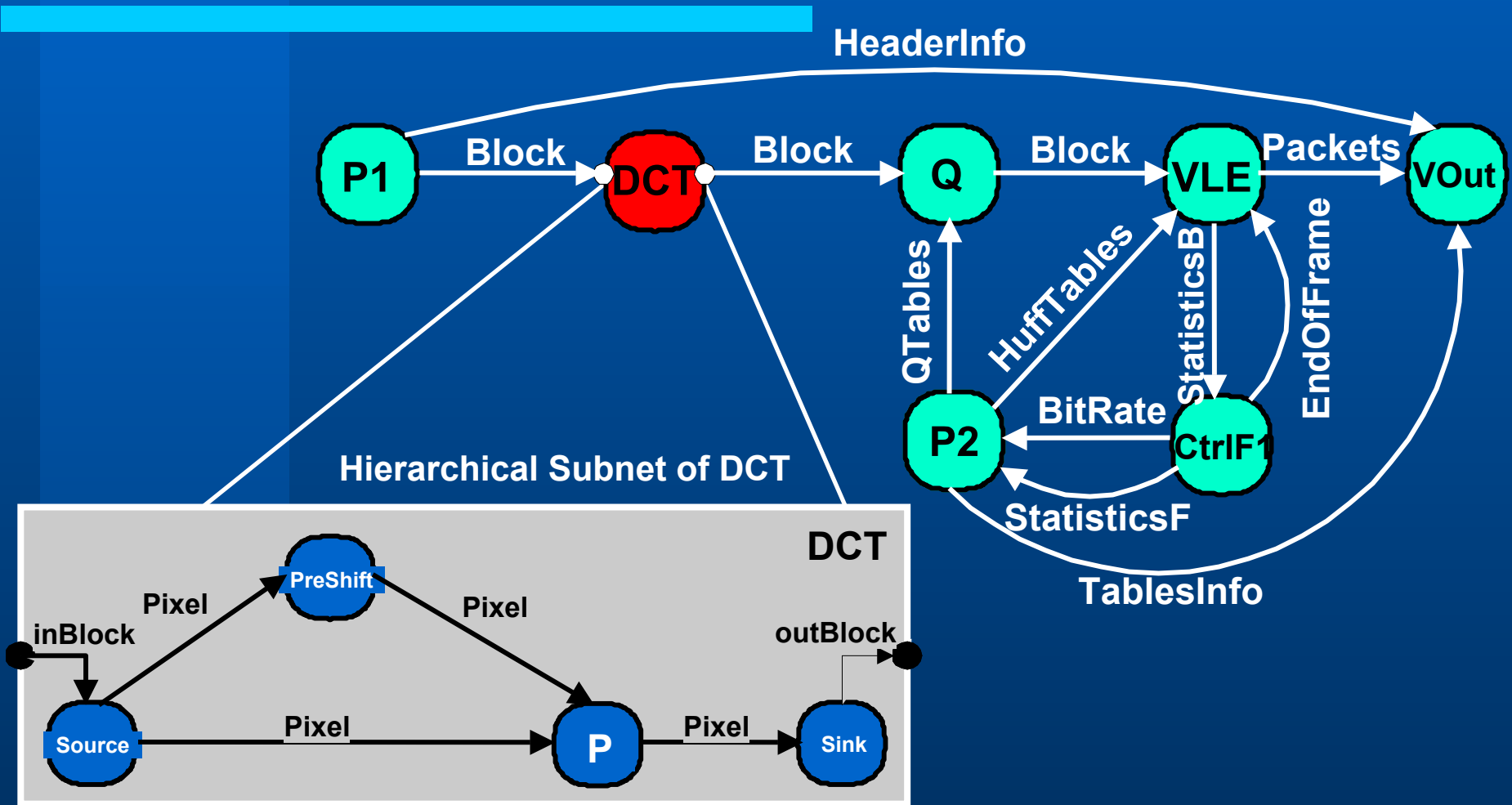
```
void DCT::main() {  
    // NumFrames = 100;  
    // VNumBlocks = 16;  
    // HNumBlocks = 8;  
  
    for ( int k=1; k <= NumFrames; k++ ) {  
        for ( int j=1; j <= VNumBlocks; j++ ) {  
            for ( int i=1; i <= HNumBlocks; i++ ) {  
                read( inPort, inBlock );  
                outBlock = DCT( inBlock );  
                write( outPort, outBlock );  
            }  
        }  
    }  
}
```

Matlab of the DCT Process

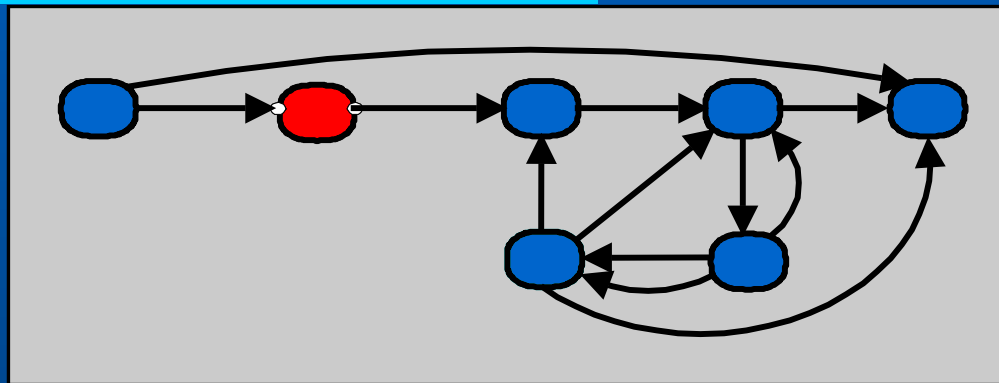
- Of the DCT process, we make a new Matlab program
 - This exposes more parallelism at a finer lever.
 - Automatic conversion from Blocks to Stream (*Linearization*)
- Compaan produces by default a process per function call.
 - However, using the Preamble 'P_1' we can group processes.

```
for k = 1:1:4,
    for j = 1:1:64,
        [ Pixel( k , j ) ] = Source( inBlock );
    end
end
for k = 1:1:4,
    if k <= 2,
        for j = 1:1:64,
            [ Pixel( k , j ) ] = PreShift( Pixel( k , j ) );
        end
    end
    for j = 1:1:64,
        [ Block ] = P_I_PixelsToBlock( Pixel( k , j ) );
    end
    [ Block ] = P_I_2D_dct( Block );
    for j = 1:1:64,
        [ Pixel( k , j ) ] = P_I_BlockToPixels( Block );
    end
end
for k = 1:1:4,
    for j = 1:1:64,
        [ outBlock ] = Sink( Pixel( k , j ) );
    end
end
```

KPN Sub network DCT



Programming the CPU



M-JPEG specified
In YAPI

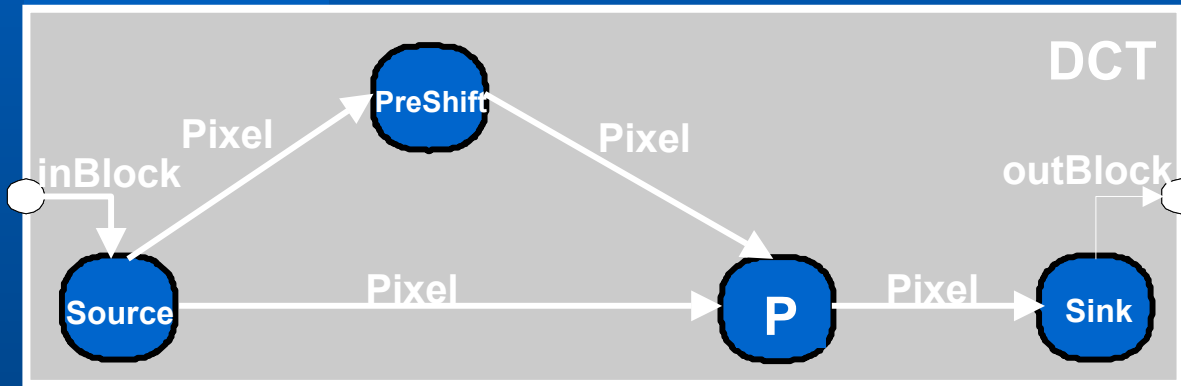
C++ Compiler

YAPI Executable

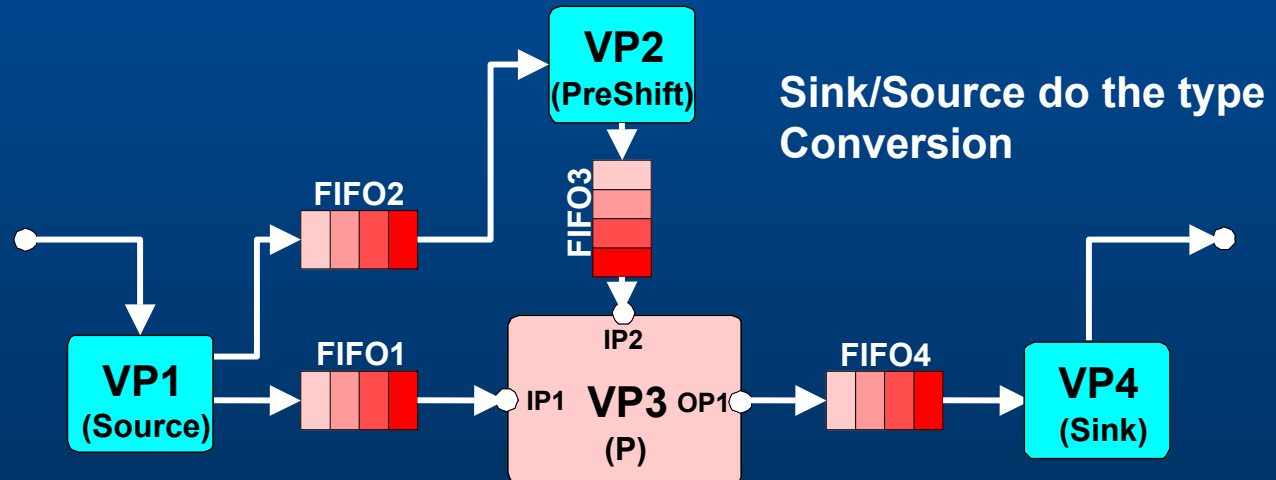
YAPI Multithreading
Environment

Pentium IV

Laura DCT Hardware Model



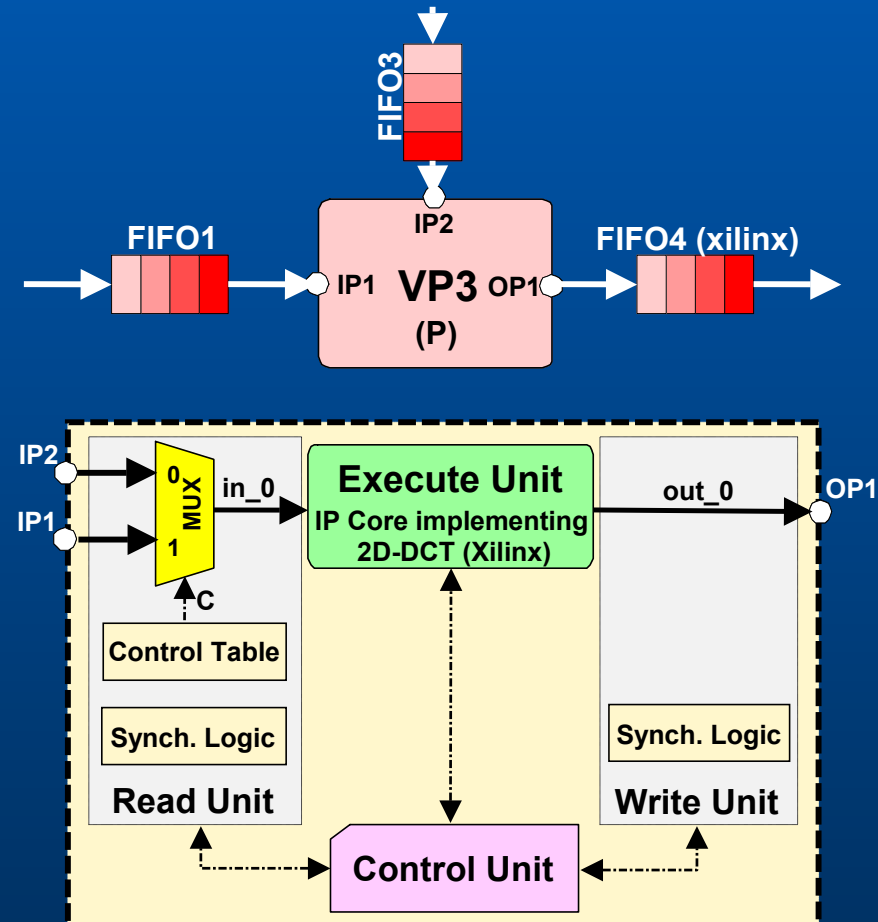
One-to-One
Mapping



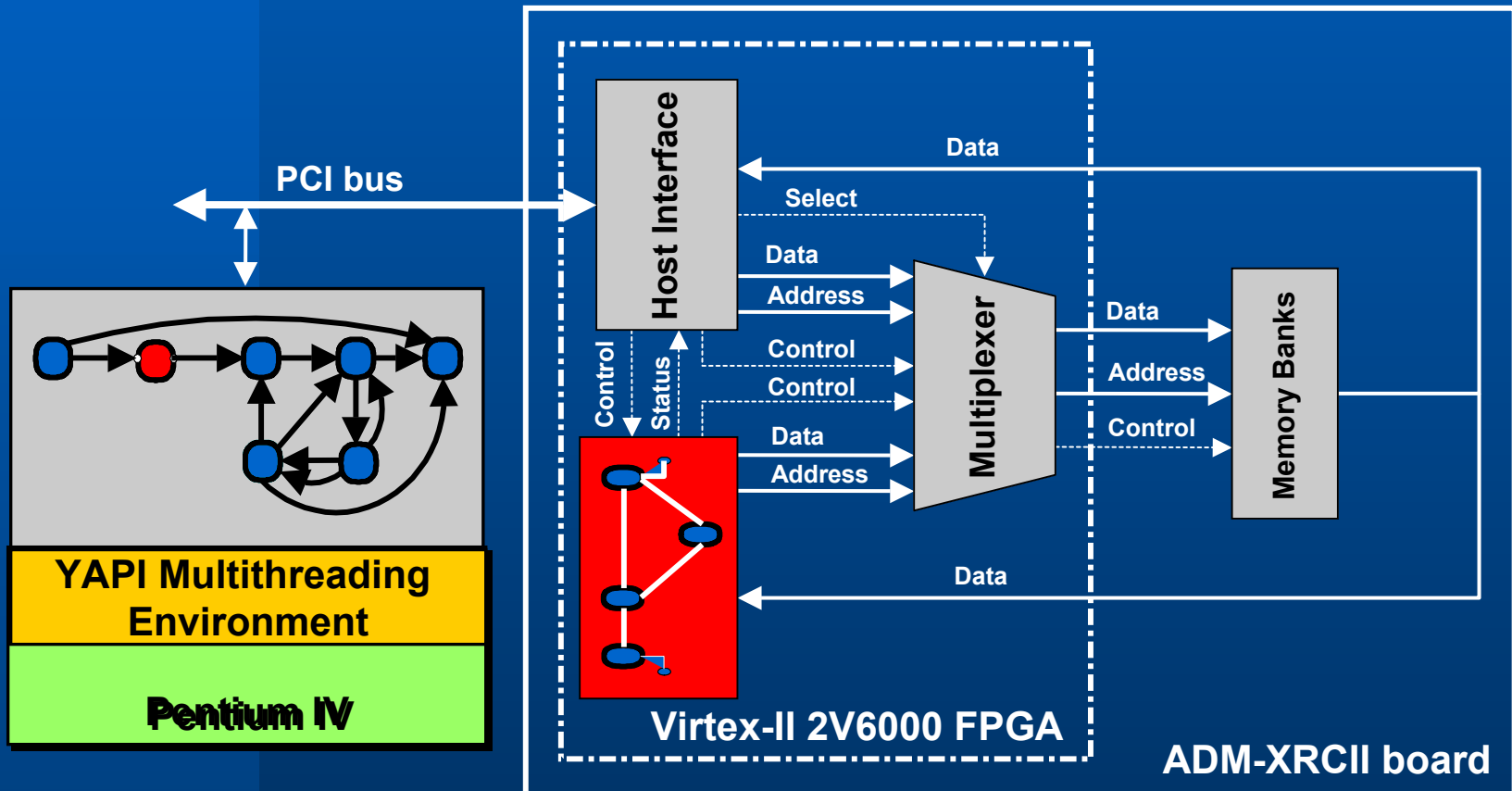
Abstract Hardware Model: Network of Virtual Processors

Laura DCT Hardware Model

- To get the functionality of the Virtual Processor, we integrated an IPcore.
- We have taken the Core (2D-DCT) from the Xilinx Website.
- Make Processor specific for a platform
 - Determine the Bit width
 - Determine the FIFO sizes
 - Take into account the Clock
 - Determine the Control tables for the switches



Hw/Sw Solution for M-JPEG



The way it is programmed; the CPU and FPGA run in parallel

Processing Time M-JPEG

	Compaan	Laura	Other tools	Manually	Total
M-JPEG -> KPN	00:00:22	--	--	00:30:00	00:30:22
Software Compilation	--	--	00:00:35	--	00:00:35
DCT Subnet Compilation	00:00:08	--	--	--	00:00:08
Laura	--	00:00:07	--	03:00:00	03:00:07
Synthesis to FPGA	--	--	00:13:10	--	00:13:10
Overall	00:00:30	00:00:07	00:13:45	03:30:00	03:44:22

Device Utilization for the DCT

FPGA resource	Utilization	%
Number of MULT18x18s	8 out of 144	5%
Number of RAMB16s	4 out of 144	2%
Number of SLICEs	2367 out of 33792	7%
Number of BUFGMUXs	2 out of 16	12%

Virtex-II 2V6000 FPGA
(taking 4% of the FPGA)

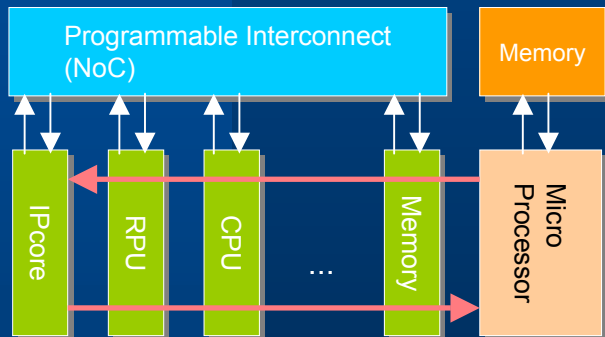
Real-time performance M-JPEG

- **Throughput of the system**
 - 10.5 CIF frame (128x128) per second
 - Running Windows 2000
 - Simple Compiler
 - Simple Multithreading architecture
- **Required is 25 frames per second**
 - Communication FPGA/CPU is too slow (PCI)
- **However,**
 - 64 bit PCI
 - Running at 66Mhz
 - 4 times increase in performance
- **Then 25 frames per second (128x128) not a problem**

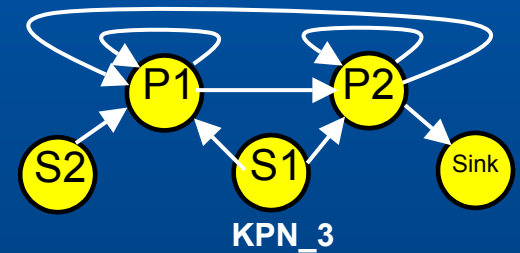
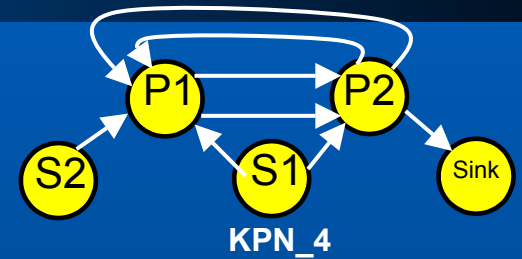
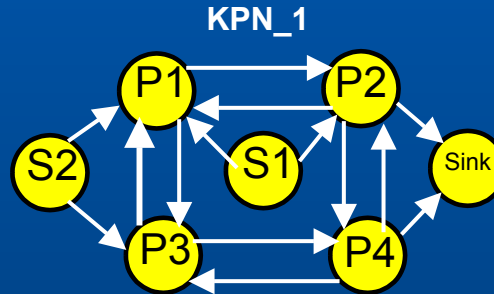
Exploration

```

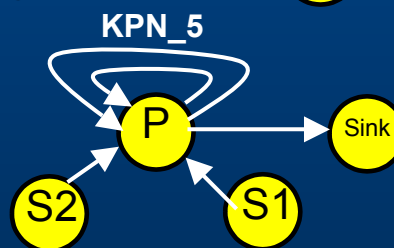
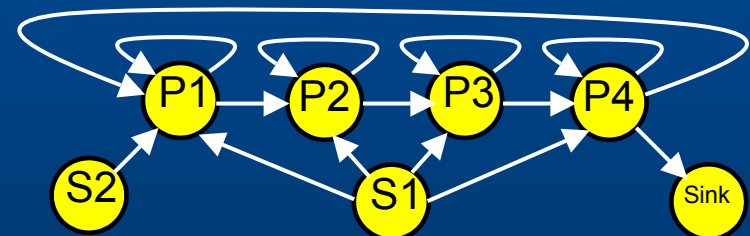
for j = 1:1:N,
  [x(j)] = Source1( );
end
for i = 1:1:K,
  [y(i)] = Source2( );
end
for j = 1:1:N,
  for i = 1:1:K,
    [y(i), x(j)] = F( y(i), x(j) );
  end
end
for i = 1:1:K,
  [Out(i)] = Sink( y(i) );
end
    
```



Generate



Map
Explore



Alternative Application Instances

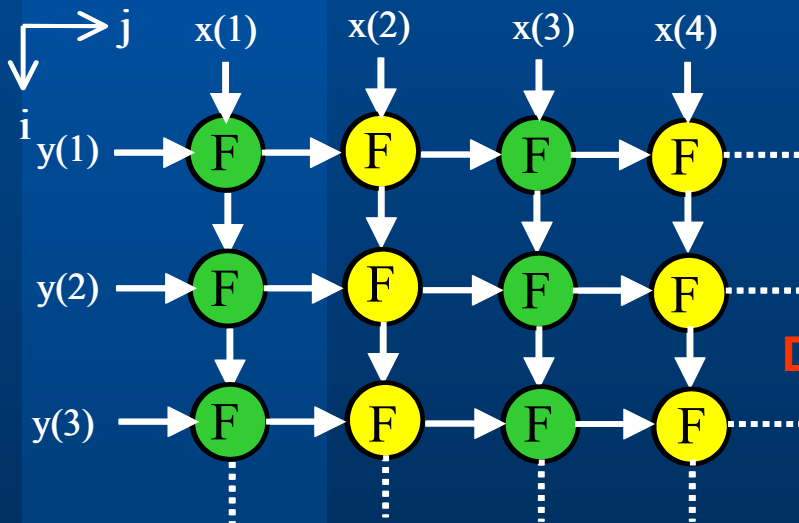
Unrolling/Unfolding

```
%parameter N 100 1000;
%parameter K 8 48;

for j = 1:1:N,
  for i = 1:1:K,
    [y(i), x(j)] = F(y(i), x(j));
  end
end
```

MatTransform

$U = [N, K]$

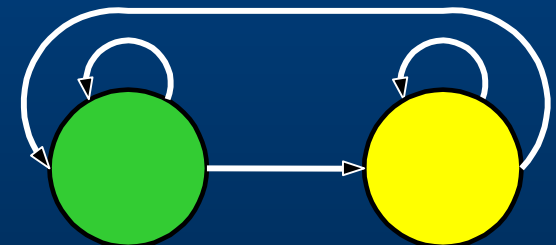


Difficult to derive

```
for j = 1:1:N,
  if mod(j, 2) == 1,
    for i = 1:1:K,
      [y(i), x(j)] = F(y(i), x(j));
    end
  end
end
```

```
if mod(j, 2) == 0,
  for i = 1:1:K,
    [y(i), x(j)] = F(y(i), x(j));
  end
end
end
```

Compaan



Retiming/Skewing

```
%parameter N 100 1000;
%parameter K 8 48;

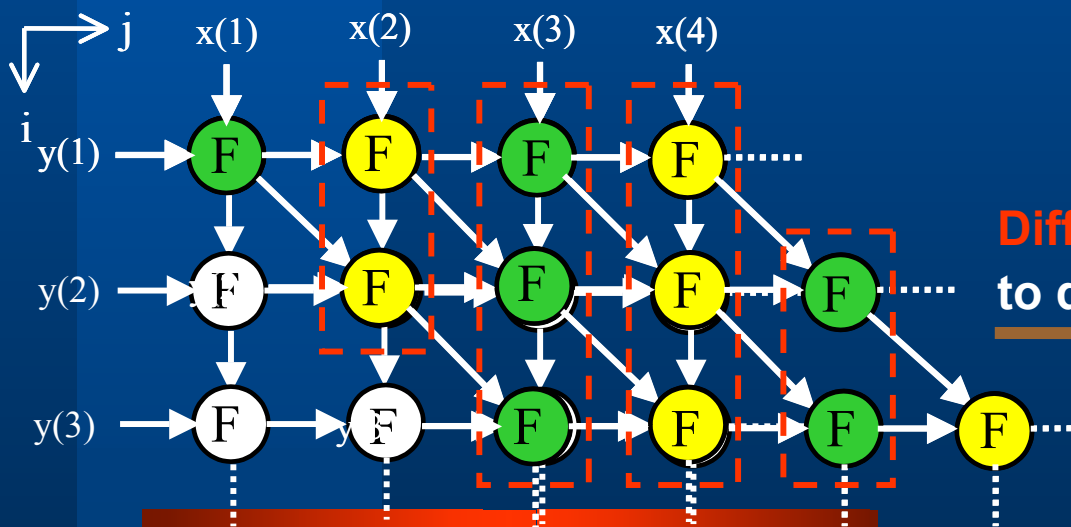
for j = 1:1:N,
  for i = 1:1:K,
    [y(i), x(j)] = F(y(i), x(j));
  end
end
```

Skewing matrix

$$M = \begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

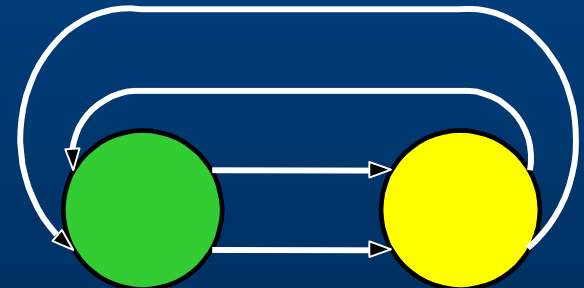
Unfolding vector

$$U = [u_1, u_2] = [2, 1]$$

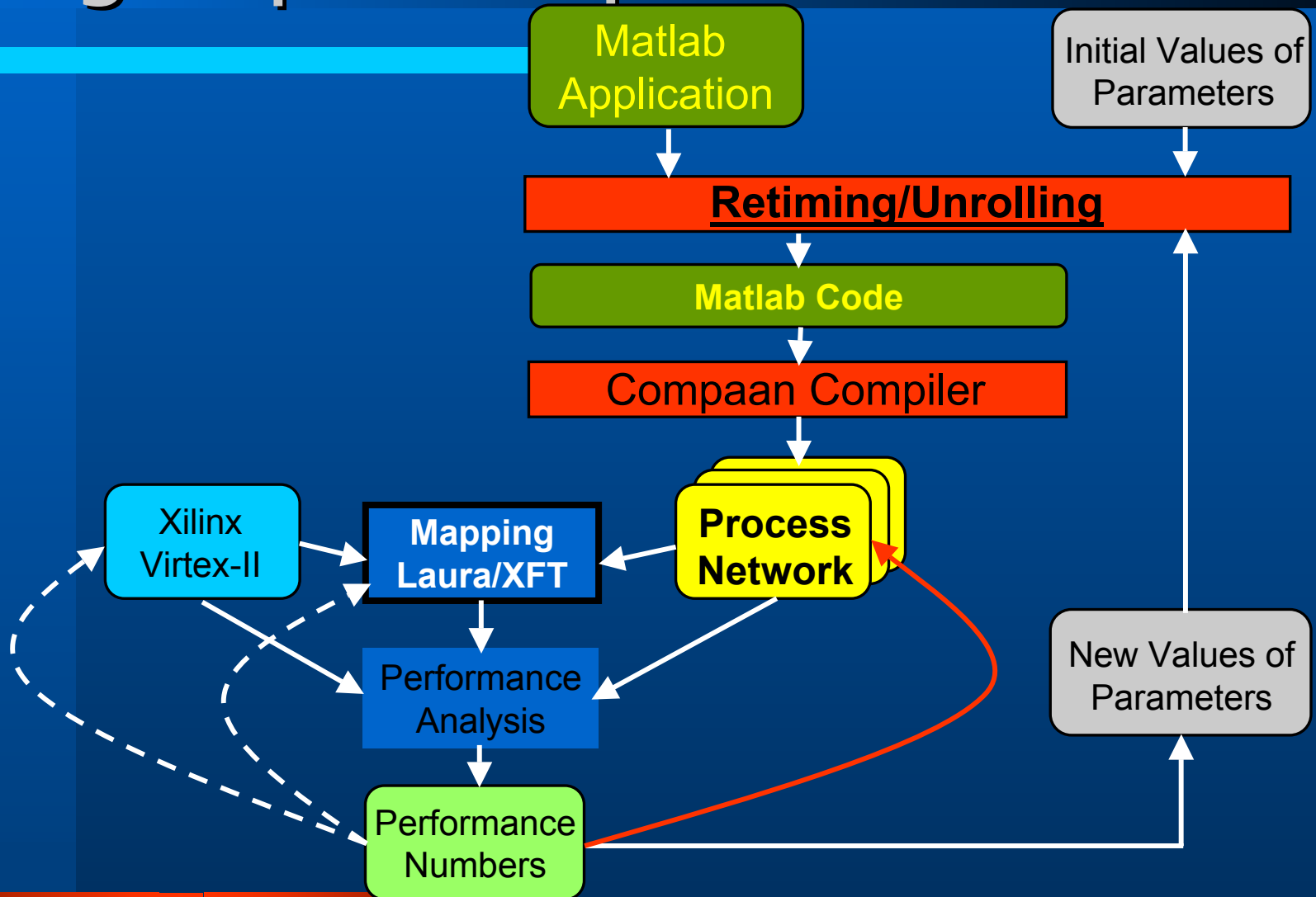


```
for j = 2:1:N+K,
  for i = max(2, j-N):1:min(j-1, K),
    for i = max(1, j-1):1:min(j-1, K),
      [y(i), x(j-i)] = F(y(i), x(j-i));
    end
  end
end
if mod(j, 2) = 0,
  for i = max(1, j-N):1:min(j-1, K),
    [y(i), x(j-i)] = F(y(i), x(j-i));
  end
end
end
```

↓ **Compaan**



Design Space Exploration



Conclusions

- To satisfy tomorrow's applications, we will see hierarchical multiprocessor systems with a number of CPUs, Memories, IPcores, and RPU.
- Programming these system will be difficult unless the MoC is changed to take Concurrency into account. The key items will be
 - Distributed Memory
 - Distributed Control
- Kahn Process Networks seem to be a very promising programming formalism for tomorrow's HW/SW codesign platforms

Conclusions

- We showed proof-of-concept with a case in which we Convert M-JPEG into a KPN of which the processes are mapped either on hardware or software.
- In the M-JPEG case, the hardware and software were running concurrently, exploiting task-level parallelism
- Having good tools, we can start from (legacy) code in Matlab, C, or other imperative languages.
- The results are just the beginning. There is more to achieve when more mature / commercial products are used (RTOS, Compiler, Target Platform, Virtex Pro)

Publications

- Bart Kienhuis, Edwin Rijpkema, and Ed F. Deprettere ``Compaan: Deriving Process Networks from Matlab for Embedded Signal Processing Architectures.", 8th International Workshop on Hardware/Software Codesign (CODES'2000), May 3 -- 5 2000, San Diego, CA, USA.
- Alexandru Turjan, Bart Kienhuis, and Ed Deprettere ``A compile time based approach for solving out-of-order communication in Kahn Process Networks", in proceeding of IEEE 13th International Conference on Application-specific Systems, Architectures and Processors (ASAP'2002), San Jose, CA, USA, July 17-19, 2002
- Tim Harriss, Richard Walke, Bart Kienhuis, and Ed Deprettere ``Compilation from Matlab to Process Networks Realized in FPGA", In journal on Design Automation of Embedded Systems, Kluwer, Vol 7, Issue 4, 2002
- Todor Stefanov, Bart Kienhuis, and Ed Deprettere ``Algorithmic Transformation Techniques for Efficient Exploration of Alternative Application Instances", in proceeding of Tenth International Symposium on Hardware/Software Codesign CODES'2002, Stanley Hotel, Estes Park, Colorado, USA, May 6 -- 8, 2002
- Edwin Rijpkema, ``Modeling Task Level Parallelism in Piece-wise Regular Programs", PhD thesis, Leiden University, Leiden Institute of Advanced Computer Science (LIACS), The Netherlands, Sept 2002.
- Alexandru Turjan, Bart Kienhuis, and Ed Deprettere, ``Solving out-of-order communication in Kahn Process Networks ", submitted for publication in Journal on VLSI Signal Processing-Systems for Signal, Image, and Video Technology. Kluwer Academic Publishers., 2003
- Claudiu Zissulescu , Todor Stefanov, Bart Kienhuis and Ed Deprettere, "*Laura: Leiden Architecture Research and Exploration Tool*", submitted to The International Conference on Field Programmable Logic and Applications, September 1-3, 2003 Lisbon, Portugal
- Todor Stefanov, Claudiu Zissulescu, Alexandru Turjan, Bart Kienhuis and Ed Deprettere, "*System Design using Kahn Process Networks: The Compaan/Laura Approach*", submitted for review to ICCAD, November 9 --13, 2003, San Jose, CA, USA.