Scheduling of Iterative Algorithms with Matrix Operations for Efficient FPGA Design

(Implementation of Finite Interval Constant Modulus Algorithm)

P.Šůcha, Z.Hanzálek

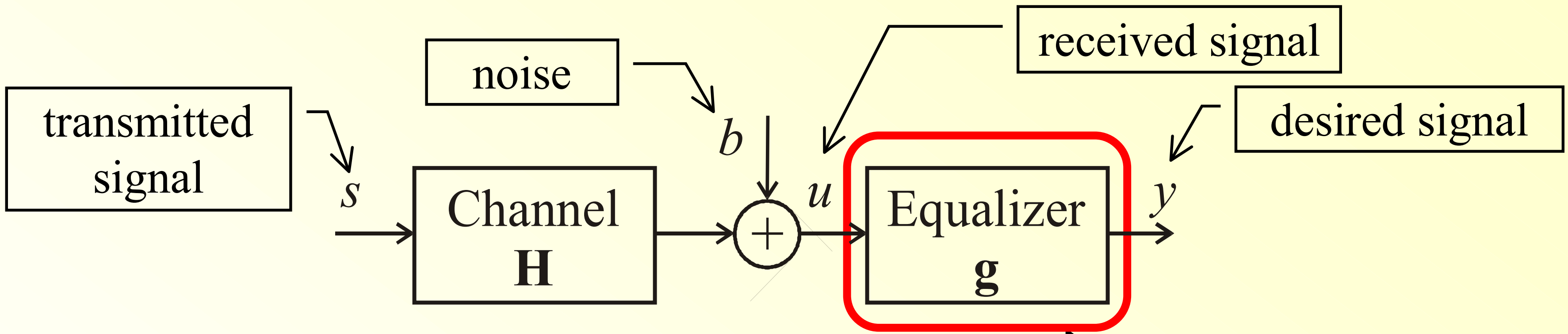
Centre for Applied Cybernetics
Department of Control Engineering
Czech Technical University in Prague
Karlovo nam. 13, 121 35 Prague 2
email: {suchap,hanzalek}@fel.cvut.cz

A.Heřmánek, J.Schier

Institute of Information Theory and Automation
Academy of Sciences of the Czech Republic
Pod vodarenskou vezi 4, 182 08 Praha 8
email: {hermanek,schier}@utia.cas.cz

Contents

1. Motivation



GSM:

- data transmission rate of approximately 270 kbps
- training sequence approx. 25%

Constant Modulus Algorithm [Godard80]:

- algorithms with no training sequence
- computations in floating-point

for $k = 1$ **to** K **do**

$\mathbf{y}(k) = \mathbf{Q} \cdot \mathbf{w}(k\text{-}1)$

$\mathbf{v}(k) = \mathbf{w}(k\text{-}1) \cdot \mathbf{Q}^T \cdot \mathbf{y}(k)^3 \,/\, F(k)$

$\mathbf{w}(k) = \mathbf{v}(k) \,/\, \|\mathbf{v}(k)\|$

end

2. Scheduling of Iterative Algorithms

Operations in a computation loop can be considered as a set of n tasks *T* performed *N* times in iterations.

Cyclic scheduling:
- *N*, the number of iterations, is large enough
- results in the **periodic schedule** (an iteration is repeated each **period** *w*)
- can lead to the **overlapped schedule** (operations belonging to different iterations can be execute simultaneously)

Objective: to find a periodic schedule with the **minimum period** (is NP-hard)

Related work:

[C. Hanen and A. Munier 1995] - **Basic Cyclic Scheduling** – infinite number of processors – $O(n^3 \log n)$

[D. Fimmel and J. Müller 2001] - solution by ILP for limited number of processors

Cyclic Scheduling

for $k = 1$ **to** K **do**

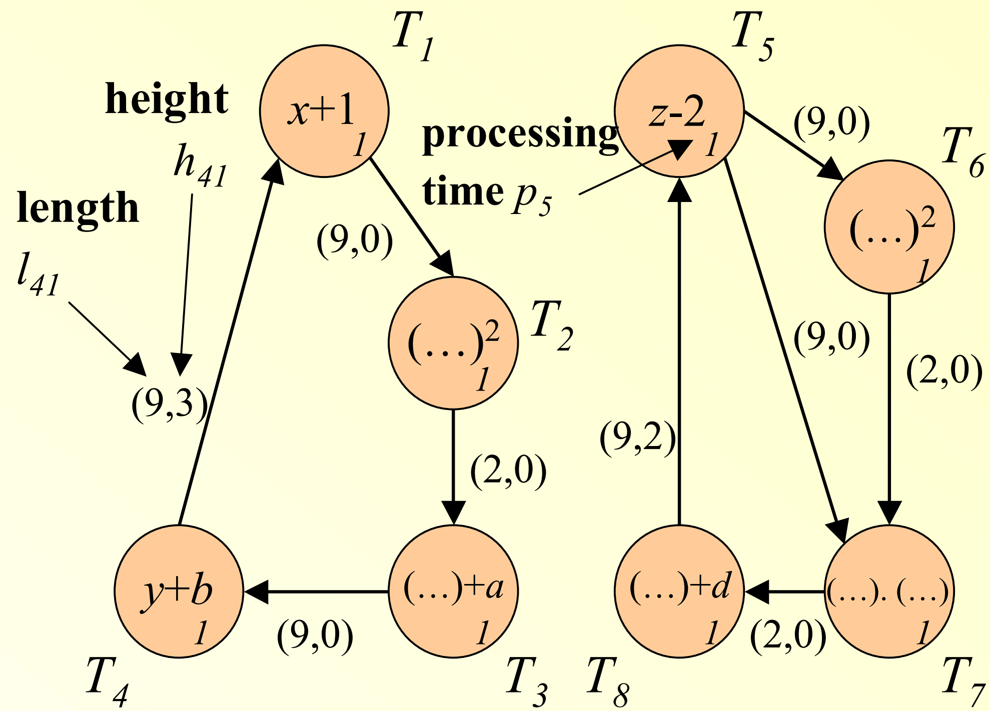
$$y(k) = \left(x(k\text{-}3) + 1\right)^2 + a$$

$$x(k) = y(k) + b$$

$$z(k) = \left(z(k-2) - 2\right)^3 + d$$

end

Operation on HSLA	+ (-)	*, /, ^{2}, $\sqrt{\ }$
Processing time p [clk]	1	1
In-Out Latency [clk]	9	2



<u>Algorithm representation by oriented graph *G*:</u>

- vertex ~ instruction ~ task
  - **processing time** p_i (time to "feed" the processor)
- arc ~ precedence relation
  - arc **height** h_{ij} (shift of the iteration index)
  - arc **length** l_{ij} (input-output latency of the unit)

$$s_j - s_i \geq l_{ij} - w \cdot h_{ij}$$

ILP program for fixed w

$$\min \sum_{i=1}^{n} \hat{q}_i$$

subject to:

$$\hat{s}_j + \hat{q}_j \cdot w - \hat{s}_i - \hat{q}_i \cdot w \geq l_{ij} - h_{ij} \cdot w$$

$$p_j \leq \hat{s}_i - \hat{s}_j + w \cdot \hat{x}_{ij} \leq w - p_i$$

where:

$$\hat{s}_i \in \langle 0, w-1 \rangle, \hat{q}_i \geq 0, \hat{x}_{ij} \in \{0,1\}$$

$\hat{q}_i, \hat{x}_{ij}$ are integers.

objective function - minimizes the iteration overlap

precedence constraint - restriction corresponding to algorithm of filter

processor constraints - one task at maximum is executed at a given time

w*- the shortest period resulting in feasible schedule is found iteratively by formulating one **ILP program for each integer *w*** ∈ [lowerbound, upperbound] … **interval bisection** method

3. Cyclic Scheduling with Nested Loops

Complex data computations (e.g. matrix operations) are implemented as nested loops.

perfectly nested loops – all elementary operations are contained in the innermost loop

imperfectly nested loops – some elementary operations are not contained in the innermost loop

Objective: to find a periodic schedule with the **minimum period** and efficient FPGA implementation of nested loops.

Related work:

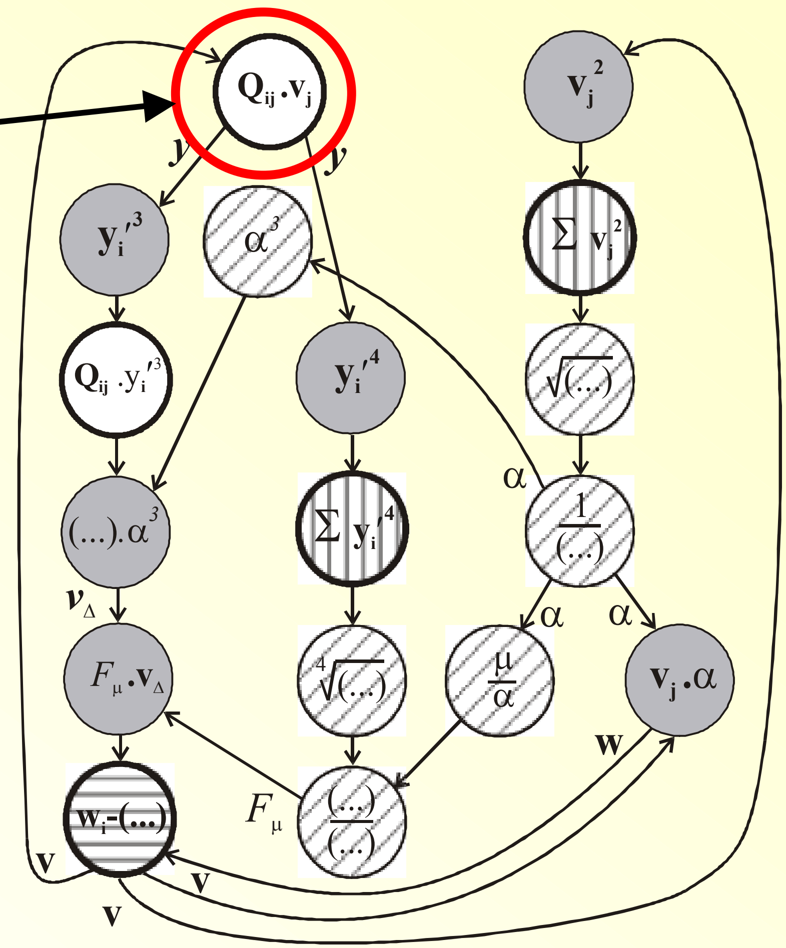
[N. Ahmed, N. Mateev and K.Pingali 2000] – "Tiling imperfectly nested loops"
- heuristics transformation of imperfectly nested loops

[Q.Zhuge, Z.Shao, and E.Sha 2005] – "Optimization of Nest-Loop Software Pipelining"
- timing and code size requirements optimization

Equalizer algorithm

$$
\begin{array}{l}
\textbf{for } k = 1 \textbf{ to } K \textbf{ do} \\
\quad \mathbf{y}'(k) = \mathbf{Q} \cdot \mathbf{v}(k\text{-}1) \\
\quad \alpha(k-1) = \dfrac{1}{\sqrt{\sum \mathrm{v}(k-1)^2}} \\
\quad \mathbf{w}(k-1) = \mathbf{v}(k) \cdot \alpha(k-1) \\
\quad F_\mu(k) = \dfrac{\mu}{\sqrt[4]{\mathbf{y}'(k)^4} \cdot \alpha(k-1)} \\
\quad \mathbf{v}_\Delta(k) = \mathbf{Q}^T \cdot \mathbf{y}'(k)^3 \cdot \alpha(k-1)^3 \\
\quad \mathbf{v}(k) = \mathbf{w}(k-1) - F_\mu(k) \cdot \mathbf{v}_\Delta(k) \\
\textbf{end}
\end{array}
$$

Equalizer algorithm

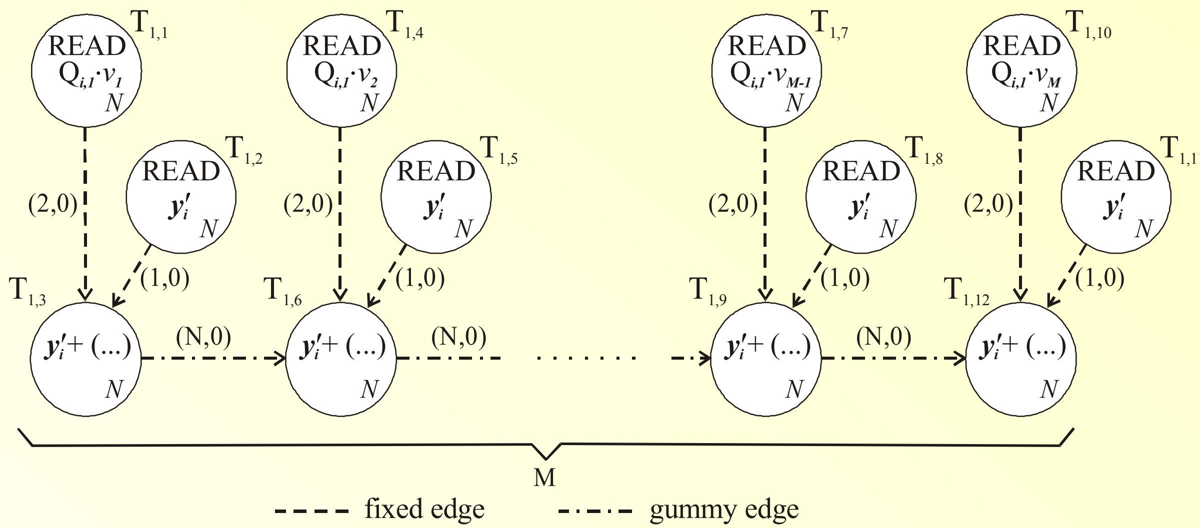


Data dependencies represented by a **condensed graph.**

- scalar operation
- element-wise operation
- sum of vector elements
- vector subtraction
- matrix-vector multiplication

Expansion of Imperfectly Nested Loops

```
for k = 1 to K do
:
    for i = 1 to M do
        for j = 1 to N do
            y'_i(k) = y'_i(k) + Q_ij · v_j(k-1)
        end
    end
:
end
```



- **Processing Time Fusion** – elementary operations are fused into single task.

- **United Edges** – keeps regularity of the loop.

$$s_{i,2} - s_{i,1} - z_i = l_i \;\; ; \;\; s_{i,3} - s_{i,2} - z_i = l_i \;\; ; \;\; \ldots$$

- **Fixed Edges** – direct data flow (e.g. memory → arithm. unit).

$$s_j - s_i = l_{ij} - w \cdot h_{ij}$$

Model Optimization

Optimization of graph model

• approximated expansion

- Iterations of the nested loop are divided into a **prologue** $\mathcal{P}$, **body** $\mathcal{B}$ and an **epilogue** $\mathcal{E}$.
- The body is represented using one task exploiting dedicated processors

Optimization on ILP model

• elimination of redundant processor constraints

- Method is based on Linear Programming.

• estimation of variable bounds

- Calculation of the longest paths in the graph.

4. Experimental Results

Architecture with HSLA (19-bit precision)
(one twin-adder, four multipliers)

One iteration of equalizer algorithm: 11ms on XC2V1000-5.
⇒ fast enough to perform 8 iterations in GSM.

	XCV2000E-6		XC2V1000-5	
Block RAM	16	10%	16	40%
SLICEs	4349	22%	4222	82%
MULT 18×18	-	-	9	22%
TBUFs	192	1%	192	7%
Clock Rate	35 MHz		50MHz	
Performance	210 MFlops		300 MFlops	

5. Conclusions

- ILP gives rather good results even for realistic examples in reasonable time (3,4 seconds).

- model is dependent on number of tasks but it is independent of period *w*.

- Equalizer performance increased by 46%.

- Automatic scheduling

  (*algorithm* → *graph* → *schedule* → *code*)

- Rapid prototyping (allows to compare different HW architectures prior to time consuming implementation).