STMicroelectronics
Advanced System Technology

Parallel Hardware for the Computation of Pairings

Luca Breveglieri, Gerardo Pelosi
Politecnico di Milano, Italy
luca.breveglieri,gerardo.pelosi@polimi.it
and
Guido Bertoni, Pasqualina Fragneto - ST Microelectronics, Italy
guido.bertoni,pasqualina.fragneto@st.com
Giampaolo Agosta, Martino Sykora - Politecnico di Milano, Italy
giampaolo.agosta,martino.sykora@polimi.it

Outline

- Recall on Pairings (e.g. Tate)
- Overview of Pairing Options
- Computing Pairings – State of the Art
- Parallel Coprocessors for Pairings
  - Methodology – architecture and design
  - Instant Break – back to elliptic curve crypto
  - Case I – dedicated parallel HW
  - Case II – more on dedicated parallel HW
  - Case III – programmable reconfigurable HW
- Considerations and Conclusion

Recall on Pairings

definition of pairing
and cryptographic relevance

Pairing Recall - I

$$e(p,q): G_1 \times G_1 \rightarrow G_2$$

- groups G_1 and G_2 are additive and multiplicative
- function e ( ) commutes with addition:
  - $e(p_1 + p_2, q) = e(p_1, q) \cdot e(p_2, q)$
  - $e(p, q_1 + q_2) = e(p, q_1) \cdot e(p, q_2)$
- function e ( ) commutes with iterated addition:
  - $e(np, q) = e(p, nq) = e(p, q)^n$
- difficult discrete log. problem both in G_1 and G_2
- for G_1 use groups of points on elliptic curves
- for G_2 use finite field multiplicative groups

Pairing Recall - II

$$e(p,q): G_1 \times G_1 \rightarrow G_2$$

bilinearity: $e(aP,bQ) = e(P,Q)^{ab}$ $\forall P,Q \in G_1$, $a,b \in Z$

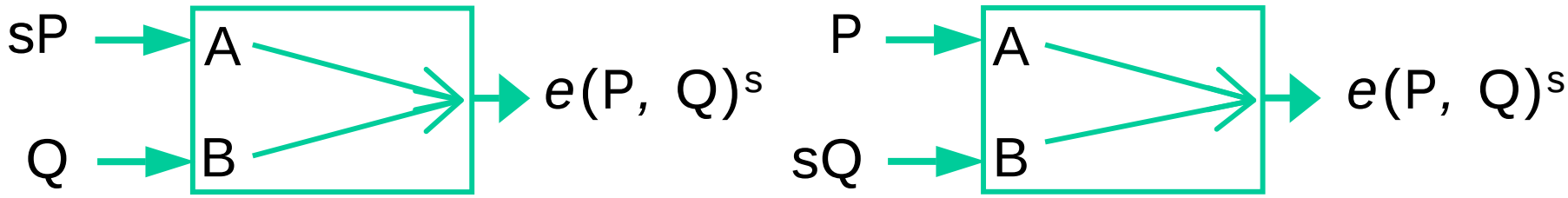
consequence relevant for cryptography

$e(aP, bQ) = e(bP, aQ)$

can swap the two integers *a* and *b*

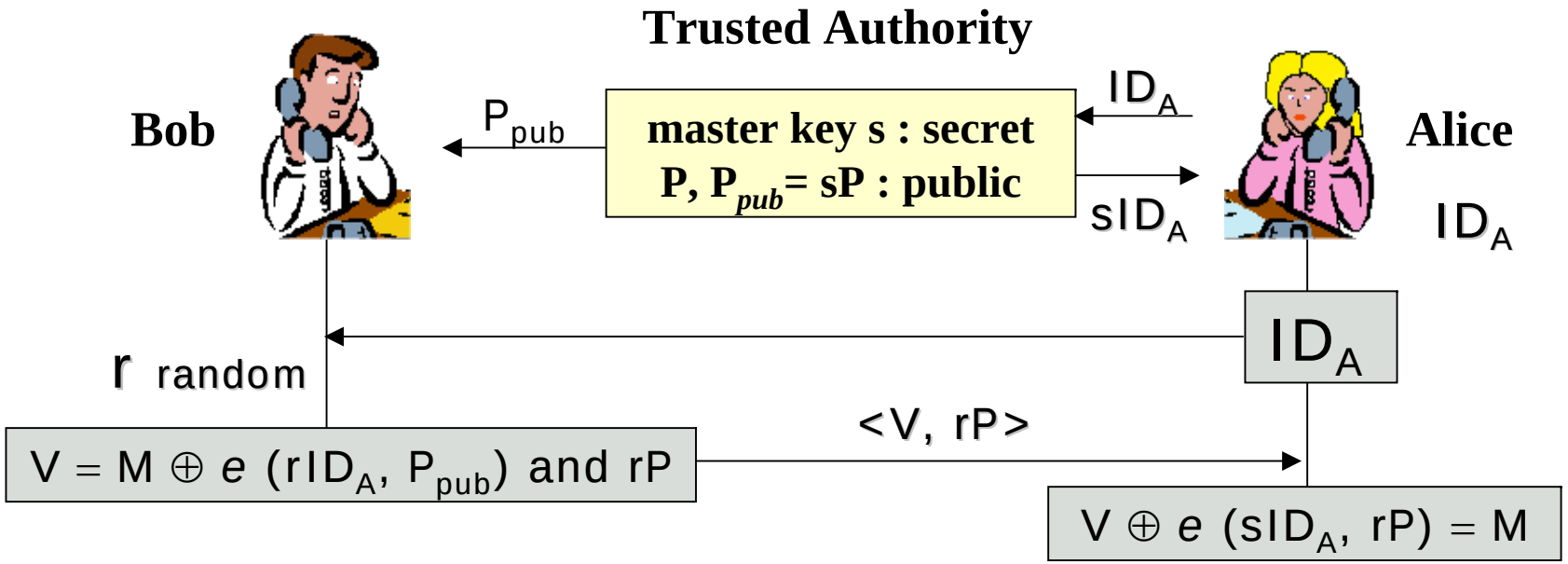
Cryptographic Interpretation of Pairing

- $s \in \mathbf{N}$ is secret P, Q and E (curve) are public
- sP, sQ must be hard discrete log. problems
- e^s must be a hard discrete log. problem



e(P, Q) is the element for establishing the encryption key and is computed by means of a pairing

IBE - Boneh-Franklin Protocol



$$e(rID_A, P_{pub}) = e(rID_A, sP) = e(ID_A, P)^{rs} = e(ID_A, P)^{sr} = e(sID_A, rP)$$

Boneh-Franklin is similar to the ElGamal cryptosystem but $rK_{p,A}$ is replaced by a pairing $e(rID_A, P_{pub})$ (e.g. Tate)

$\oplus$ is a bitwise XOR

Pairing Options

parameters curves and algorithms

Pairing Options - Fields

- F_2^m *binary field*
  - has reduced HW (and partly SW) complexity
  - may not be recommendable due to attacks (Coppersmith attack, and others)
- F_3^m *ternary field*
  - has reduced HW complexity (but worse than F_2^m)
  - is suppsed to be less prone to efficient attacks
- F_p *prime field*
  - is more complex to implement in HW (arithmetic is more similar to that of integers)
  - is a “flat” field, without any specific internal structure

Pairing Options - Types

- **Weil**: is the hystorical pairing definition
  - is less efficient than others invented later
  - becomes more efficient for very high security levels (as it does not have any final exponentiation)
- **Tate**: is the most popular pairing definition in crypto
  - halvens the Weil definition, but adds a final exponentiation
  - is the basis for defining the remaining ones (Eta and Ate)
  - Eta and Ate are more efficient in special cases
- **Eta**: is an optimization of the Tate definition
  - reduces slightly computational complexity
  - but works only for supersingular curves
- **Ate**: is an optimization of the Tate definition
  - reduces slightly computational complexity
  - and is conceived for non-supersingular curves

Pairing Options - Curves

- Pairings *map elliptic curve points* of order *r* to *values in the underlying finite field* or in an extension thereof.
- *Ordinary elliptic curves*
  - are completely generic (but smooth, of course)
  - are definable over the fields F_2^m, F_3^m and F_p
  - the embedding degree is $k > 1$ (and may be $> > > 1$)
    (*k* is the field min. ext. degree to contain all the order *r* points)
  - and hence need have special support algorithms for finding curves with a sufficiently low parameter *k*
- *Supersingular elliptic curves*
  - have a special property related to the # of curve points
  - are definable over the fields with embedding degree:
    - F_2^m $k = 4$
    - F_3^m $k = 6$
    - F_p $k = 2$
  - and allow to obtain simplifications of various kinds

Pairing Options - Algorithms

- binary field F_2^m has algorithm
  - Kwon-ETA, only for supersingular curves
- ternary field F_3^m has algorithms
  - Duursma-Lee (DL) and two variants:
    - Kwon (K)
    - Refined DL (RDL)

    all of them only for supersingular curves
  - ETA, only for supersingular curves
- prime field F_p has algorithms
  - BKLS-GHS, for any type of curve
    (is the basic Miller alg., slightly optimized)
  - ATE, only for non-supersingular curves
- still an open list, there may be more in the future

Example – Basic Miller Alg. in F_p

input: $m = \lceil \log_2 r \rceil$ $r = (r_{m-1}, \ldots, r_0)_2$ $P \in E(F_p)[r]$ $Q \in E(F_{p^k})[r]$

output: $e(P, Q) = e(P, Q) = f_P(D_Q)^{\varepsilon}$ $\varepsilon = (p^k - 1) / r$

final exponent

elliptic curve

rational function associated with P

divisor associated with Q

pick a random point $R \in E$
compute $Q' \leftarrow Q + R$
$V \leftarrow P$
$f \leftarrow 1$
for $i = m - 1$ **downto** 0 **do** -- the loop body is the substantial part of the alg.
  compute the straight lines $g_{V,V}$ and $g_{V,-V}$ for doubling V
  $f \leftarrow f^2 \cdot g_{V,V}(Q') \cdot g_{V,-V}(R) / (g_{V,-V}(Q') \cdot g_{V,V}(R))$
  $V \leftarrow 2V$
  if $r_i = 1$ **then**
    compute the straight lines $g_{V,P}$ and $g_{(V+P),-(V+P)}$ for adding V and P
    $f \leftarrow f \cdot g_{V,P}(Q') \cdot g_{(V+P),-(V+P)}(R) / (g_{(V+P),-(V+P)}(Q') \cdot g_{V,P}(R))$
    $V \leftarrow V + P$
  end if
end for
return f^{ε}

with suited parametrizations this conditional can be solved

Computation - State of the Art

cost and performance for computing pairings

(as of december 2006)

Pairing Computation - SW

field & k	MOV security level	reference & algorithm	time in ms Pentium IV @ 3 GHz
F_2^{239} $k = 4$	956	Galbraith et al. ANTS 02, LNCS BKLS-GHS	10.8
F_2^{239} $k = 4$	956	Barreto et al. IACR org. TR 04 / 375 Kwon-Eta-BGOhS	1.70
F_3^{97} $k = 6$	922	Barreto et al. IACR org. TR 04 / 375 Eta-BGOhS	2.72
F_3^{97} $k = 6$	922	Granger et al. LMS J. Cmp. & Math., 06 Refined Duursma-Lee	4.05
F_2^{379} $k = 4$	1516	Scott et al., CHES 06, LNCS Kwon-Eta-BGOhS	3.88
F_p $p \sim 2^{512}$ $k = 2$	1024	Scott et al., CHES 06, LNCS BKLS (twisted) non supersingular	2.97
F_p $p \sim 2^{256}$ $k = 4$	1024	Scott et al., CHES 06, LNCS ATE non supersingular	3.16

291 ms ARM @ 206 MHz

time is referred to the computation of the entire pairing (with final exp.)

comparisons should be made only between similar field types

(too many arithmetic and programming differences otherwise)

Pairing Computation – HW

reference	field	MOV sec. level	device type	device size (FPGA elm.)	multiplier digit-size	freq. MHz	time μs
Shu et al. ICFPT 06, IEEE-CS	F_2^{239}	956	FPGA	25,287	16	84	34
Shu et al. ICFPT 06, IEEE-CS	F_2^{283}	1132	FPGA	37,803	32	72	49
Kerins et al. CHES 05, LNCS	F_3^{97}	922	FPGA	55,616	4	15	594
Grabher et al. CHES 05, LNCS	F_3^{97}	922	FPGA + emb. processor	4,481	4	150	399
Ronan et al. (**hyperelliptic**) ITNG 06, IEEE –CS	F_2^{103}	1236	FPGA	43,986	16	32	749
ours (Duursma-Lee)	**F_3^{97}**	**922**	**FPGA**	**31,907**	**4**	**61**	**138**

no final exp.

time is referred to the computation of the entire paring (with final exp.)

comparisons should be made only between similar field types

(too many arithmetic and architectural differences otherwise)

Observations on Performance

- After several years of research, SW performance has eventually reached the order of magnitude from 1 to 10 ms, yet running on powerful platforms (P IV @ 3 GHz).
- On embedded systems (which are much less powerful, say ARM @ ≈ 200 MHz or even less), SW performance is in the order of magnitude from 100 to 1000 ms (=1 s).
- HW performance is in the order of magnitude from 100 to 1000 μs, or slightly better in few somewhat special cases, but clock frequencies are rather low.
- The gap between SW and HW performance is not large (say from one to two orders of magnitude only), hence HW implementation is likely to be in a still somewhat primitive state, at least if comparing to SW.
- True, presently HW is only on FPGA, not on ASIC …

Methodology

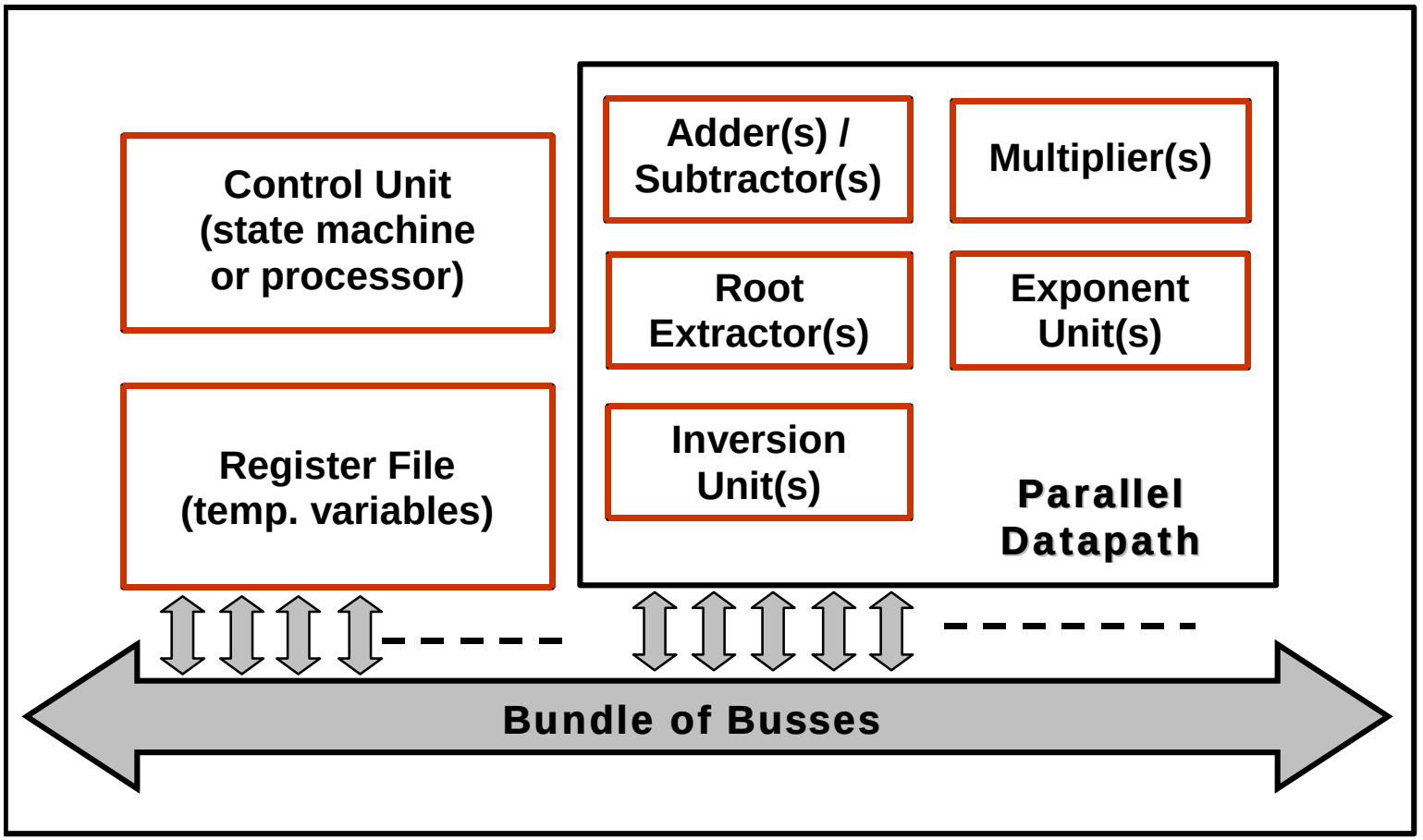
general architectural model and how to design it

Objective

- A methodology for exploring whether pairing algorithms are suited to parallel HW (or partially HW) implementation.
- A general parallel architecture model, with replicated arithmetic function units, connections and registers.
- Some case studies, for evaluation and to identify promising research directions.

Parallel Architecture Model

a dedicated multiple bus datapath architecture

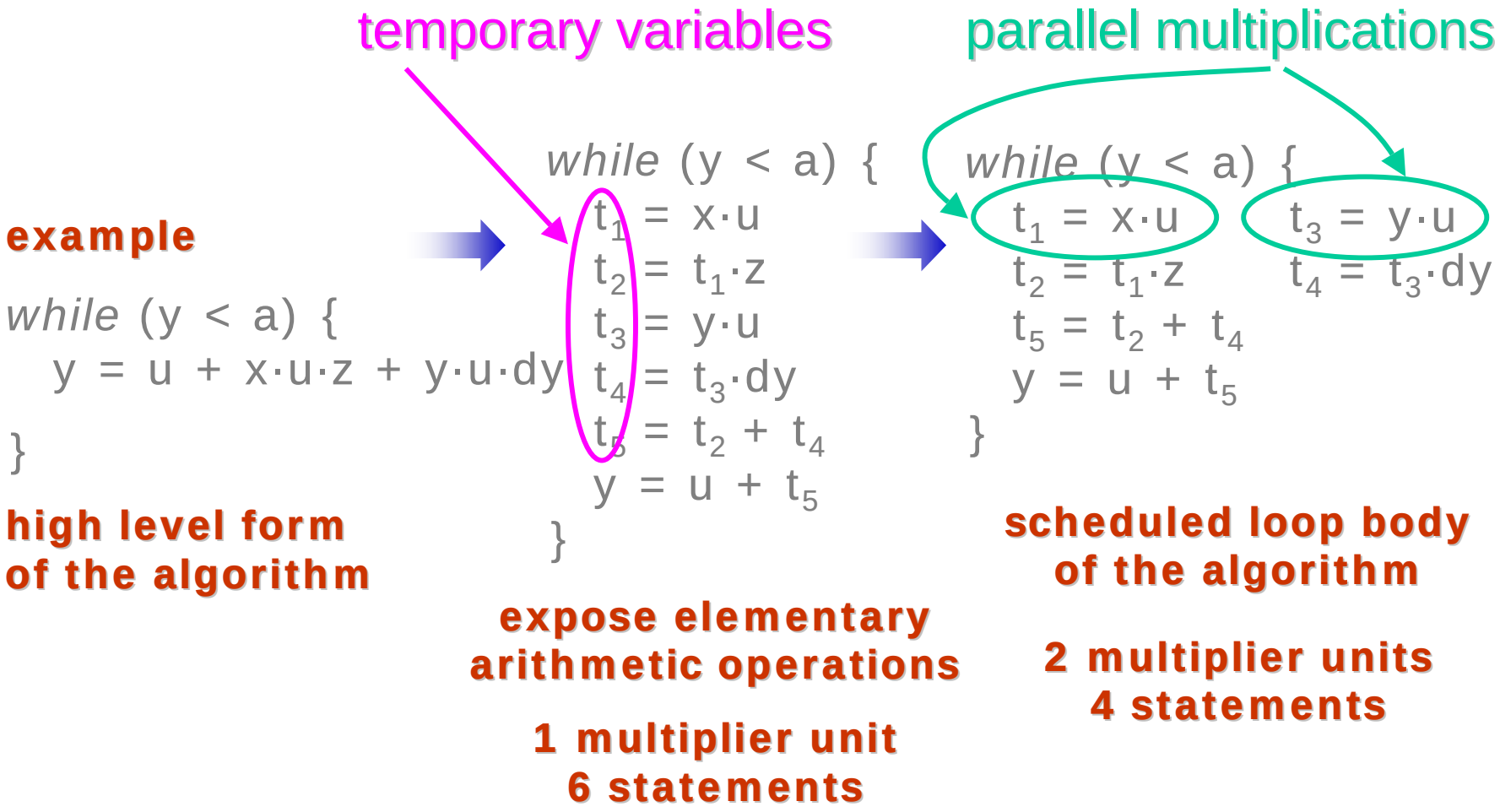


the function units may be replicated

Static Scheduling Concepts - I

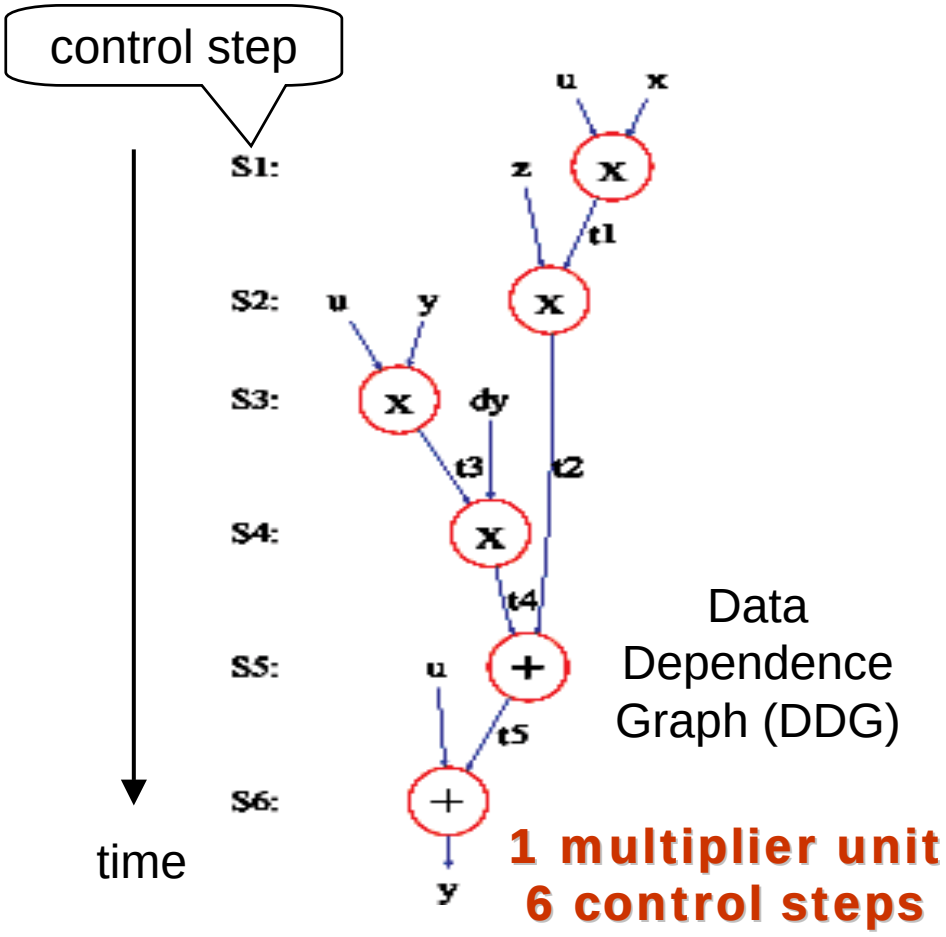
- Model the algorithm as a Data Dependence Graph
  - operation $\Rightarrow$ node (labeled with operation time latency)
  - data dependence $\Rightarrow$ directed arc
- Restructure the DDG and expose as much parallelism as possible among the operations, yet compatibly with additional constraints (cost, etc).
- Design the dedicated parallel architecture that corresponds to the DDG:
  - node $\Rightarrow$ function unit (adder, multiplier, etc)
  - directed arc $\Rightarrow$ internal bus
  - plus possibly registers for temporary variables (register allocation)

Static Scheduling Concepts - II



Static Scheduling Concepts - III

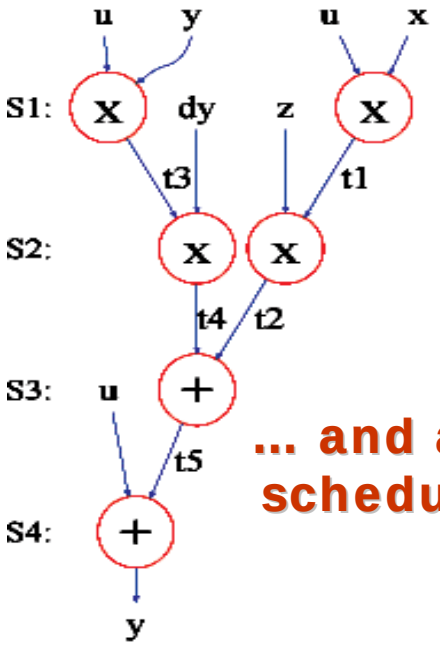
before scheduling ...



Data Dependence Graph (DDG)

1 multiplier unit
6 control steps

As Soon As Possible (ASAP) schedule

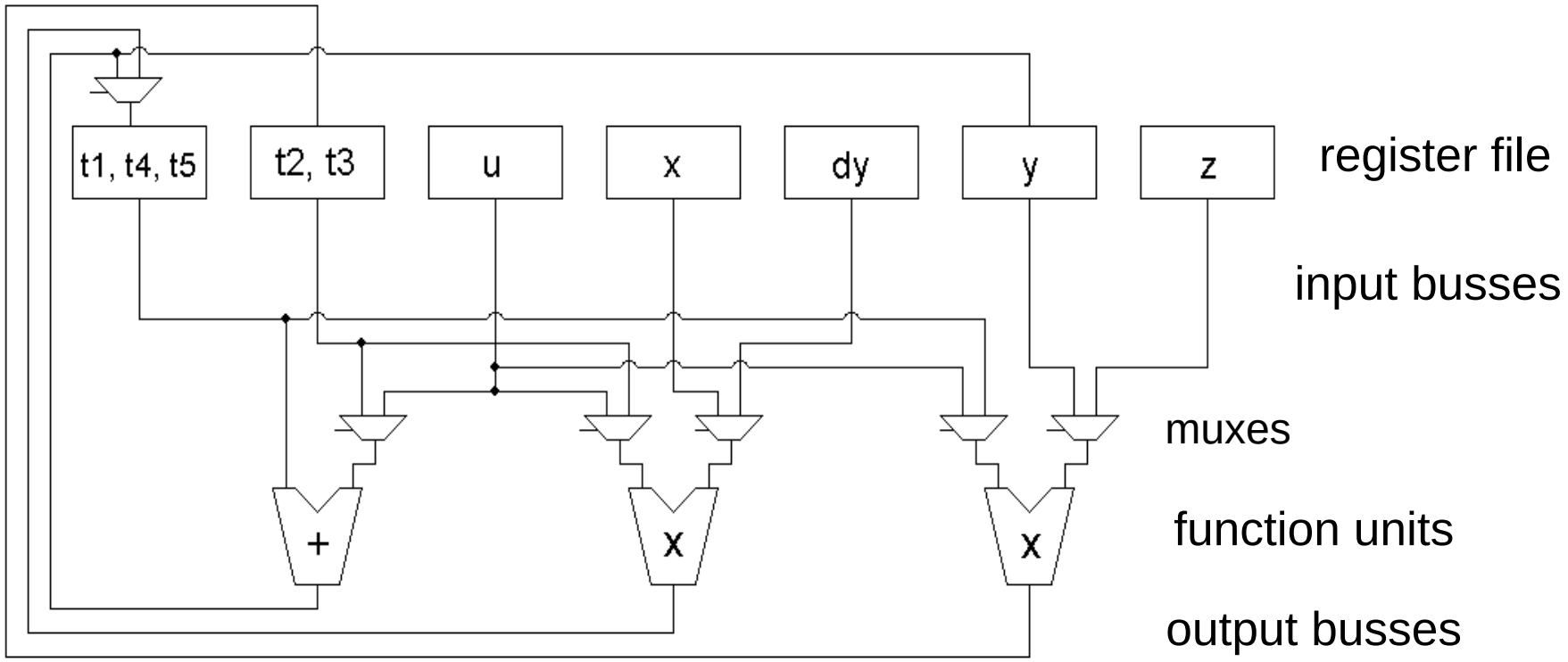


... and after scheduling

2 multiplier units
4 control steps

Static Scheduling Concepts - IV

parallel datapath busses and register file



dedicated architecture corresponding to the previous ASAP schedule – includes as many busses as necessary for connecting all the function units (here mul. and add.)

Scheduling Disciplines - I

- A scheduling discipline puts together three things:
  - an algorithm to schedule, e.g. a pairing
  - a target function to optimize, e.g. time performance
  - and a set of constraints to fulfill, e.g. the maximum admittable cost (= the num. of func. units) of the circuit
- In general the scheduling problem is NP-complete, and hence hard to solve in an optimal way.
- There are several disciplines, more or less efficient, to compute sub-optimal schedules:
  - As Soon As Possible ASAP
  - As Late As Possible ALAP
  - Operation Scheduling OS
  - List Based Scheduling LBS
  - and others …

Scheduling Disciplines - II

- Scheduling disciplines can be classified depending on:
  - the target function and the set of constraints
  - the algorithm for exploring the solution space
- **Resource-constrained disciplines**
  - minimize ***time*** (circuit ends algorithm as quickly as possible)
  - but keep ***cost*** under control (fix max num. of function units)
- **Time-constrained disciplines**
  - minimize ***cost*** (circuit has as few as possible function units)
  - but keep ***time*** under control (fix max time for the circuit)

List Based Scheduling – LBS - I

- Minimize time, constrain resources (e.g. num. of FUs).
- Idea: if the freedom for placing an operation is high, do not rush to schedule now that operation.
- How to do:
  - schedule progressively the operations, until all are done
  - and give preference to those that have limited freedom
- Is a compromise between ASAP and ALAP.
- Does not have a unique solution, in general.
- Is a *heuristic* scheduling discipline.
- But is proved to be somewhat efficient.
- And explores extensively (though not completely) the solution space (otherwise would be NP-complete).

List Based Scheduling – LBS - II

LBS is a *resource-constrained* discipline.

The LBS discipline works as follows:

- describe the algorithm as a DDG (nodes and arcs)
- label each DDG node with the time latency of the function unit
- set the max number of function units (per unit type, possibly)
- execute the following allocation step:
  - for each not yet allocated node of the DDG, compute its mobility (the time interval between the earliest and latest possible allocation)
  - allocate the node(s) (i.e. the operation(s)) with minimum mobility
  - if there are still unallocated nodes left, repeat the step
- allocate registers for temporary variables, if necessary
- output the scheduled DDG(s) and variable allocation(s)

One of the scheduled DDG(s) (e.g. a time-optimal one) will be used to design the circuit.

LBS – How to Program

- The LBS discipline is too complex to apply manually, therefore design and implement a specific SW toolchain.
- Toolchain inputs (must be prepared manually):
  - a file containing the pairing algorithm, programmed at the level of the base field operations and with temporary variables (use an *ad hoc* defined language to do so, in assembler style)
  - a description file listing the latencies of the FUs and all other constraints (e.g. max # of FUs per type, etc)
- Toolchain steps to execute (are two procedures written in C):
  - run the LB scheduling procedure and collect all possible schedules (may be numerous …) – filter them if necessary
  - run an auxiliary register allocation procedure, at least for the interesting schedules, and collect the allocation table
- Toolchain output: a file listing all possible LB schedules.
- Possibly, automatically design the VHDL model of the schedules of interest, including in particular the controller unit (this part of the toolchain is currently under development).
- Otherwise, do so manually, then synthesize and evaluate.

Instant Break

back to elliptic curve cryptography

Parallelism in ECC

- Parallelism is proved or shown to exist and to be exploitable in ECC.
  - A. Antola, G. Bertoni, L. Breveglieri, P. Maistri, *Parallel Architectures for Elliptic Curve Cryptoprocessors over Binary Extension Fields*, Proc. Midwest Symposium, IEEE, 2003 – parallelism for ECC, static scheduling and evaluation
  - G. Bertoni, L. Breveglieri, C. Paar, T. Wollingher, *Finding Optimum Parallel Coprocessor Design for Genus 2 Hyperelliptic Curve Cryptosystems*, Proc. ITCC, IEEE, 2004 – as above, but for HECC
  - G. Bertoni, L. Breveglieri, F. Sozzani, F. Turcato, *A Parallelized Design for an Elliptic Curve Cryptosystem Coprocessor*, Proc. ITCC, IEEE, 2005 – as above, but in F_p
  - G. Bertoni, L. Breveglieri, C. Paar, T. Wollingher, *Performance of HECC Coprocessors Using Inversion-Free Formulae*, Proc. ICCSA, LNCS 3982, Springer, 2006 – as above, but for HECC
  - L. Batina, B. Preneel, K. Sakiyama, I. Verbauwhede, *Superscalar Coprocessor for High-Speed Curve-Based Cryptography*, Proc. CHES, LNCS 4249, Springer, 2006 – parallelism in (H)ECC, dynamic scheduling, implementation figures, etc
  - more ? … (add if you know)
- ECC precedes pairing, why not extending parallelism to pairing as well ?
  - G. Bertoni, L. Breveglieri, P. Fragneto, G. Pelosi, *Parallel Hardware Architectures for the Cryptographic Tate Pairing*, ITNG, IEEE, 2006 – dedicated parallel hardware
  - G. Agosta, L. Breveglieri, G. Pelosi, M. Sykora, *Programming Highly Parallel Reconfigurable Architectures for Public-Key Cryptographic Applications*, ITNG, IEEE, 2007 (to appear) – reconfigurable parallel programmable hardware
  - more ? ... (add if you know)

Case Study – I

dedicated architectures for Tate pairing in F_3^m

Objective

- Design a parallel HW dedicated coprocessor for the Tate pairing in the base field F_3^m.
- Design and optimize the architecture by means of the LBS scheduling methodology.
- Analyze performance and cost for several automatically scheduled solutions.
- Experimental performance / cost evaluation on FPGA and comparison with litterature.
- Some final considerations …

Basic (Arithmetic) Assumptions

- Elementary arithmetic in $F_3 = \{-1, 0, 1\}$.
- Base field F_3^m ($m = 97$ elements of F_3).
- Representation in standard basis by the trinomial $x^m + x^h - 1$ (for a small h).
- Supersingular elliptic curve over F_3^m.
- Three pairing algorithms to analyze:
  - Duursma-Lee (Miller algorithm in closed form)
  - Kwon-BOGS (an algorithmic variant of DL)
  - Refined DL (unrolled DL plus refinement)

Duursma-Lee Algorithm

formulated at the level of the operations in the base field F_3^m

Input: $P, Q \in E(\mathbb{F}_q)[l] \quad P = (x_P, y_P) \quad Q = (x_Q, y_Q)$
Output: $f = f_P(\phi(Q)) \in \mathbb{F}^*_{q^6}/\mathbb{F}^*_{q^3}$

begin
  $f \leftarrow 1$ // # of operations in the base field $\mathbb{F}_{3^m}$
  for $i = 1$ **to** m **do**
    $x_P \leftarrow x_P^3,\ y_P \leftarrow y_P^3$ // 2 cube powers
    $\mu \leftarrow x_P + x_Q + b$ // 2 additions
    $\lambda \leftarrow -y_P \cdot y_Q\,\sigma - \mu^2$ // 2 multiplications
    $g \leftarrow \lambda - \mu\,\rho - \rho^2$ // no operation
    $f \leftarrow f \cdot g$ // 13 mul.s, 50 add.s or sub.s
    $x_Q \leftarrow \sqrt[3]{x_Q}$ // 1 cube root
    $y_Q \leftarrow \sqrt[3]{y_Q}$ // 1 cube root
    // tot: 52 add.s or sub.s, 15 mul.s, 2 cube pow.s, 2 cube roots
  endfor
  return f
end

total computational load of the DL loop body

Tate Pairing Algorithms

- Operation counting in the loop body of the algorithms for the base field F_3^m.

Algorithm	Cube Power	Add/Sub	Mul	Cube Root
Dursma-Lee (DL)	2	52	15	2
Kwon (K)	10	59	15	not used
Refined-DL (RDL)	4	90	28	4

- These operations will be modeled in HW (i.e. VHDL).
- Evaluation will be done by implementation on FPGA.

Max Degree of Parallelism

ASAP schedules in ideal conditions:

- there are unboundedly many function units and busses
- time latencies are all identical (i.e. all = 1 clock cycle)

Algorithm Type	Min. Exec. Time with ASAP Sched.	F.U. Variability Range			
		Adder	Multiplier	Cube Power	Cube Root
DL	13	18	11	2	2
K	14	19	11	9	not used
RDL	20	26	16	2	2

the “variability range” is max number of FUs beyond which a further time latency reduction does not happen any more

In principle, exploitable parallelism is high !

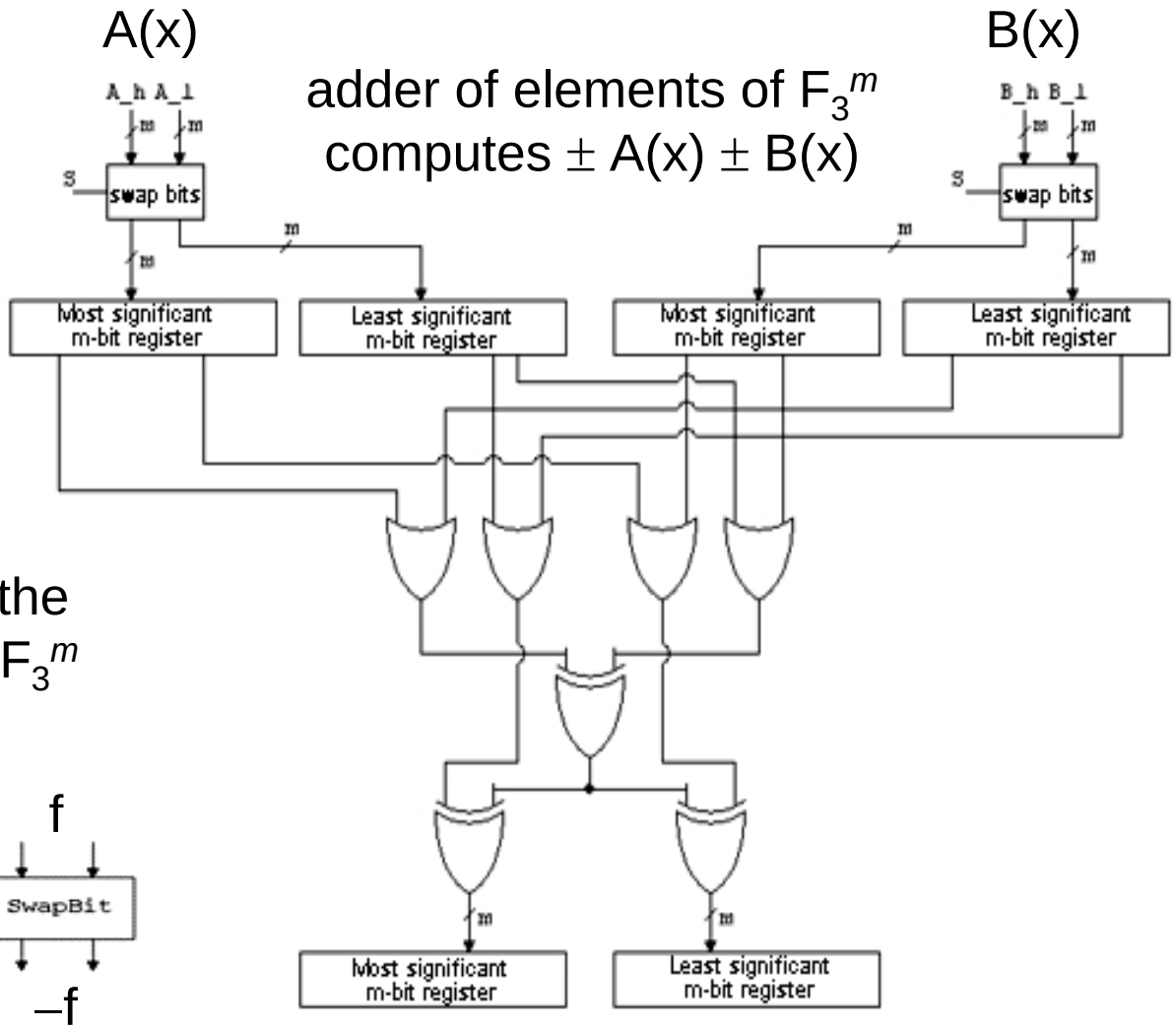
Function Units - HW Description

- Addition and subtraction (same func. unit) are performed in parallel on *m* elements of F_3.
- Multiplication is performed in the digit-serial / parallel to parallel way:
  - first factor completely available in *parallel*
  - second factor: *scanned by a window* of $D \geq 1$ bits at a time (with $D \leq m - h$ to simplify reduction)
  - product completely available in *parallel* at the end
- Cube power: easy, interleave 0's and reduce.
- Cube root: almost easy, so-called "thinning".

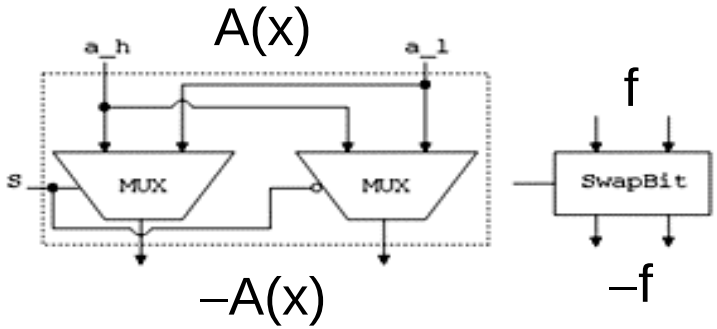
Function Units - Adder

F_3	coded as
–1	1 0
0	0 0
1	0 1

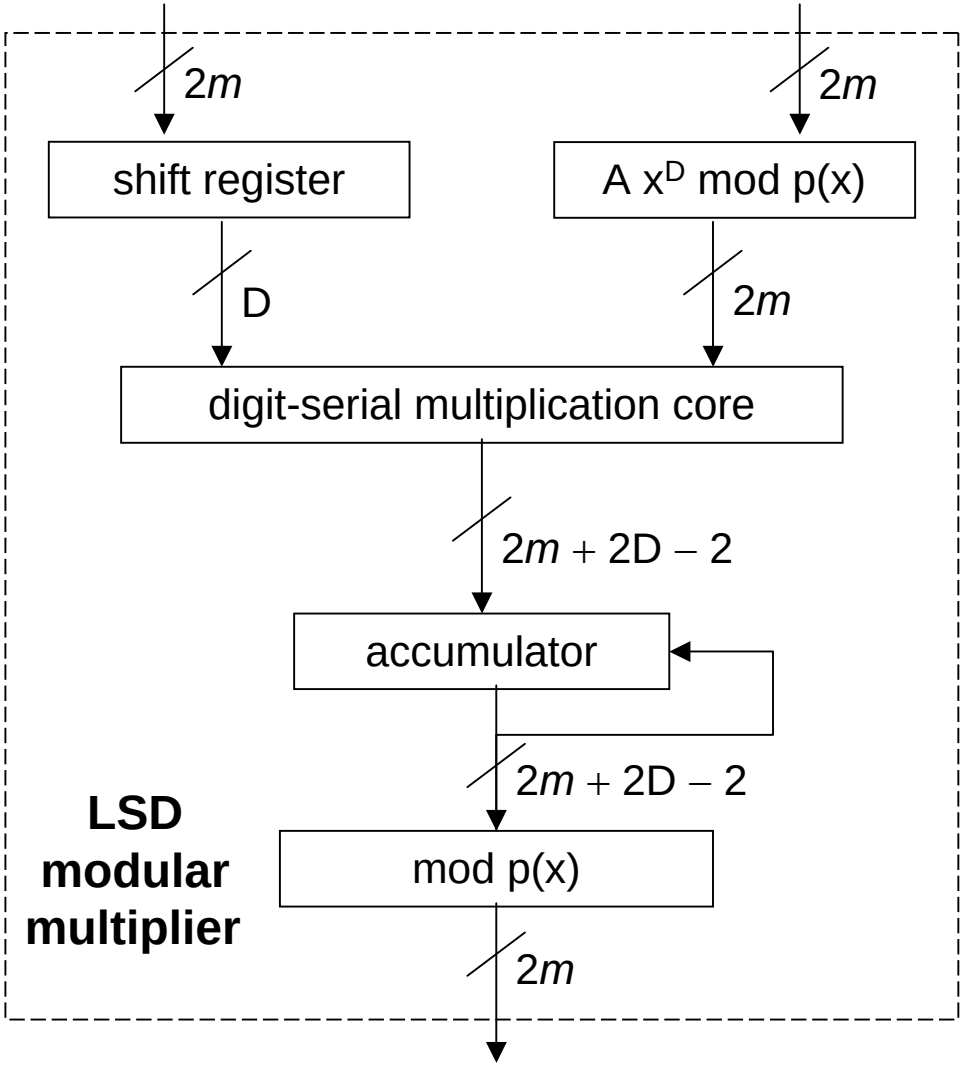
parallel / parallel to parallel addition unit



sub-unit for inverting the sign of an element of F_3^m



Function Units - Multiplier



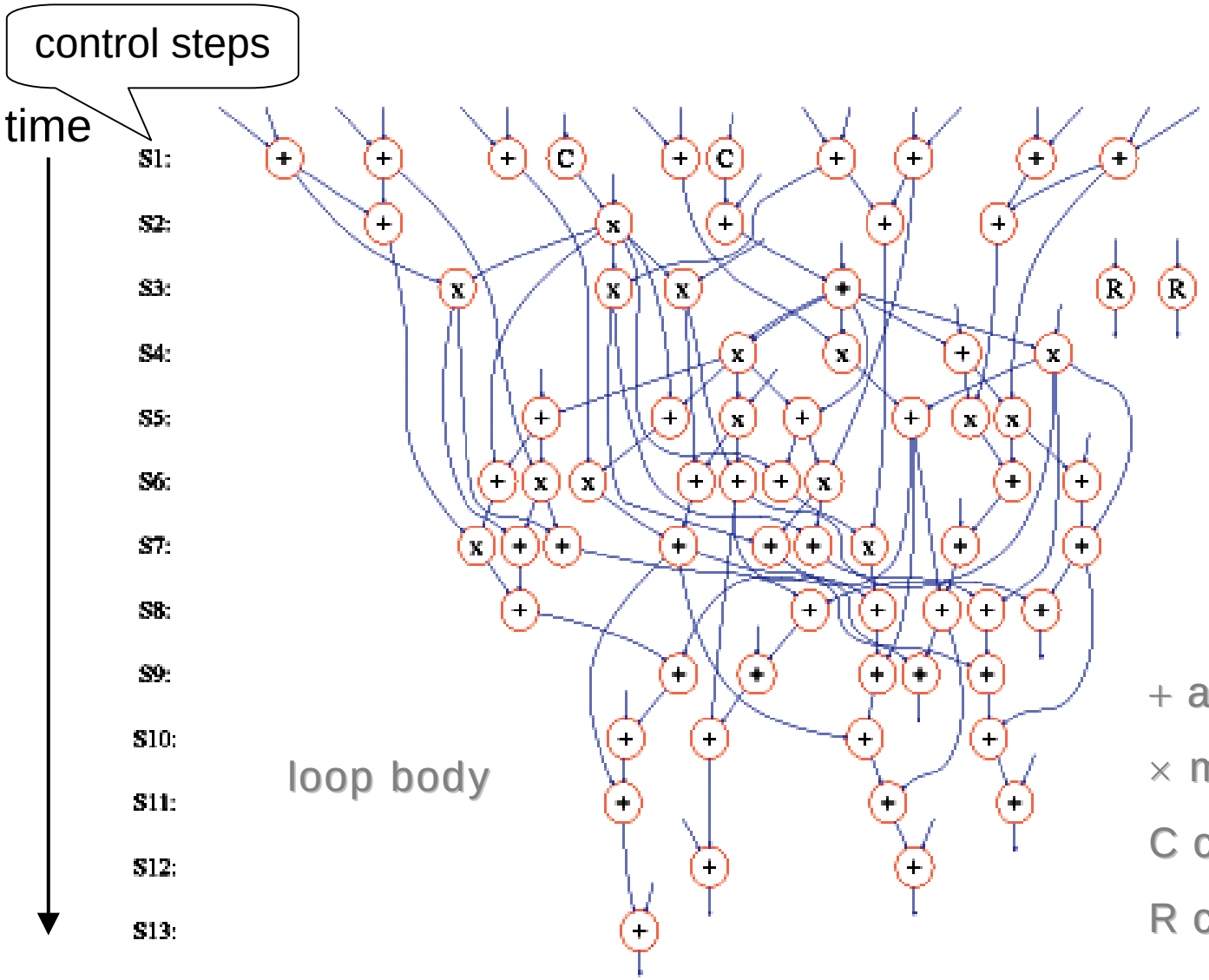
parallel / digit-serial to parallel multiplication unit

multiplier of elements of $F_3{}^m$
computes $A(x) \times B(x)$

input: $A, B \in F_3{}^m$
output: $C \equiv A \times B \bmod p(x)$

$C \leftarrow 0$
for $i = 0$ **to** $\lceil m / D \rceil - 1$ **do**
  $C \leftarrow B_i A + C$
  $A \leftarrow A\, x^D \bmod p(x)$
return C

ASAP Schedule - Duursma-Lee



unitary latencies & unconstrained resources

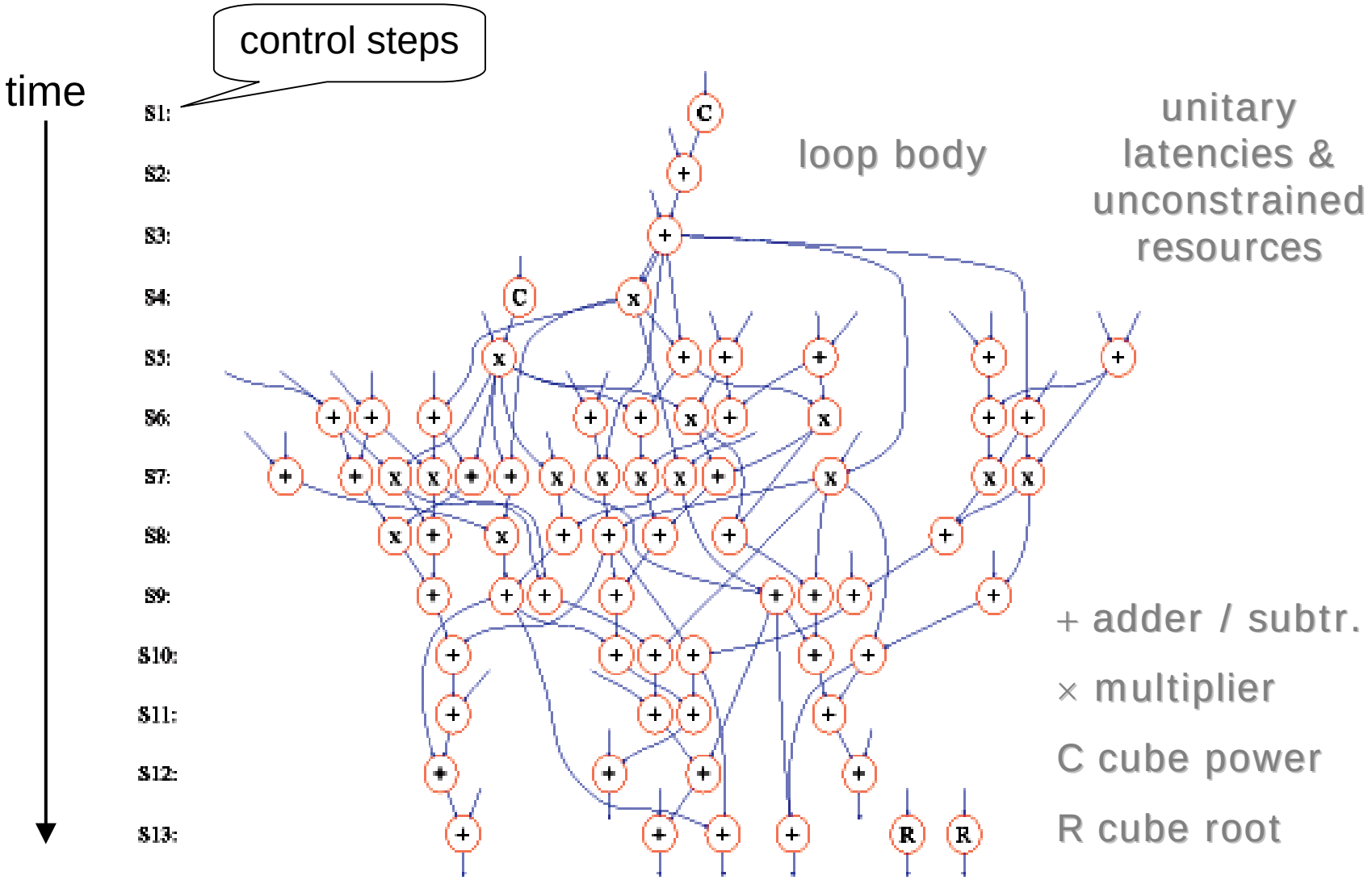
+ adder / subtr.

× multiplier

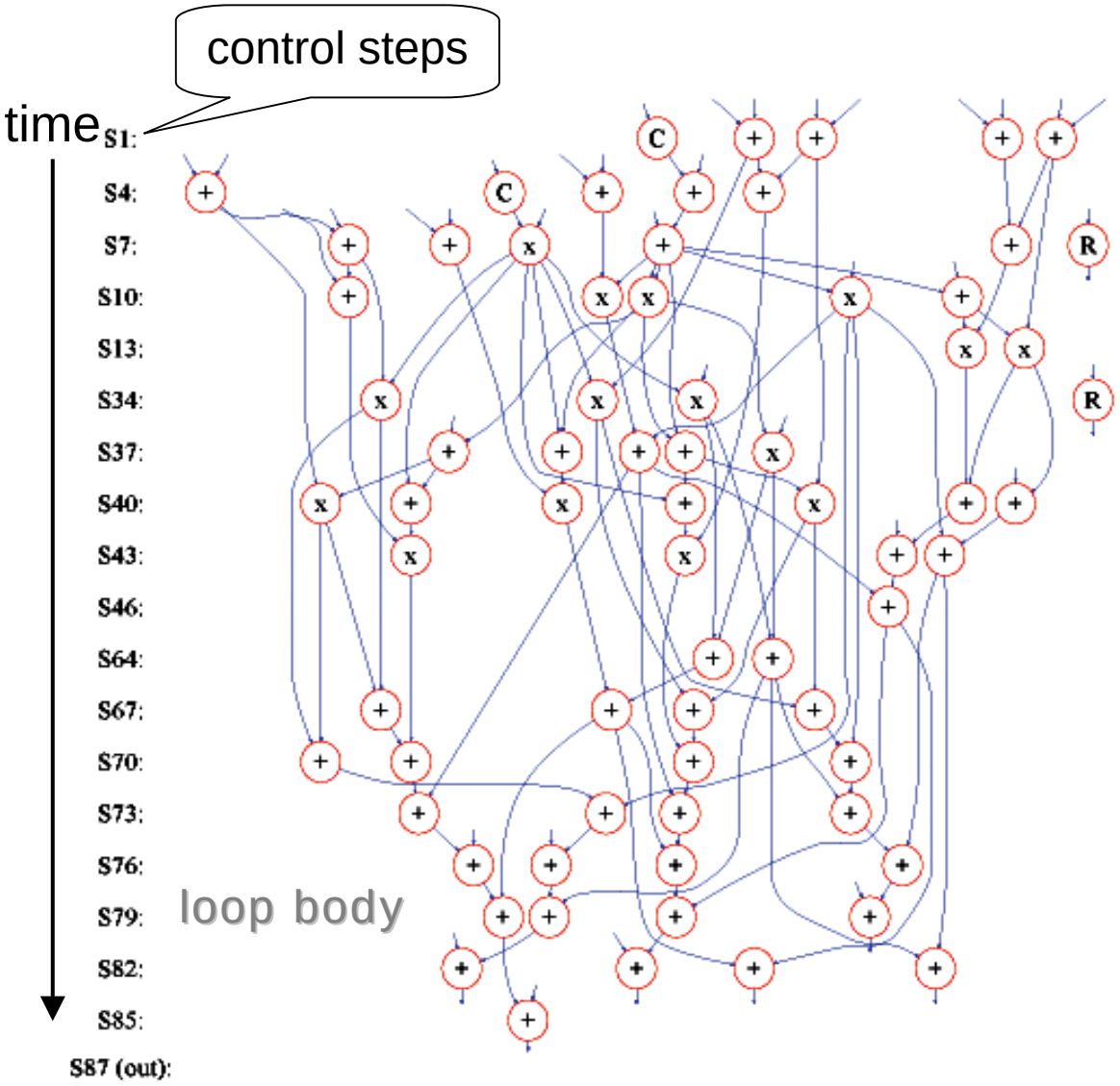
C cube power

R cube root

ALAP Schedule - Duursma-Lee



LBS Schedule - Duursma-Lee



($m = 97$ D = 4)

Function Units

9 multipliers

4 adders/subtr.s

1 cube root unit

1 cube power unit

FU Latency Time

real timings

Execution Time

87 clock cycles

Temp. Registers

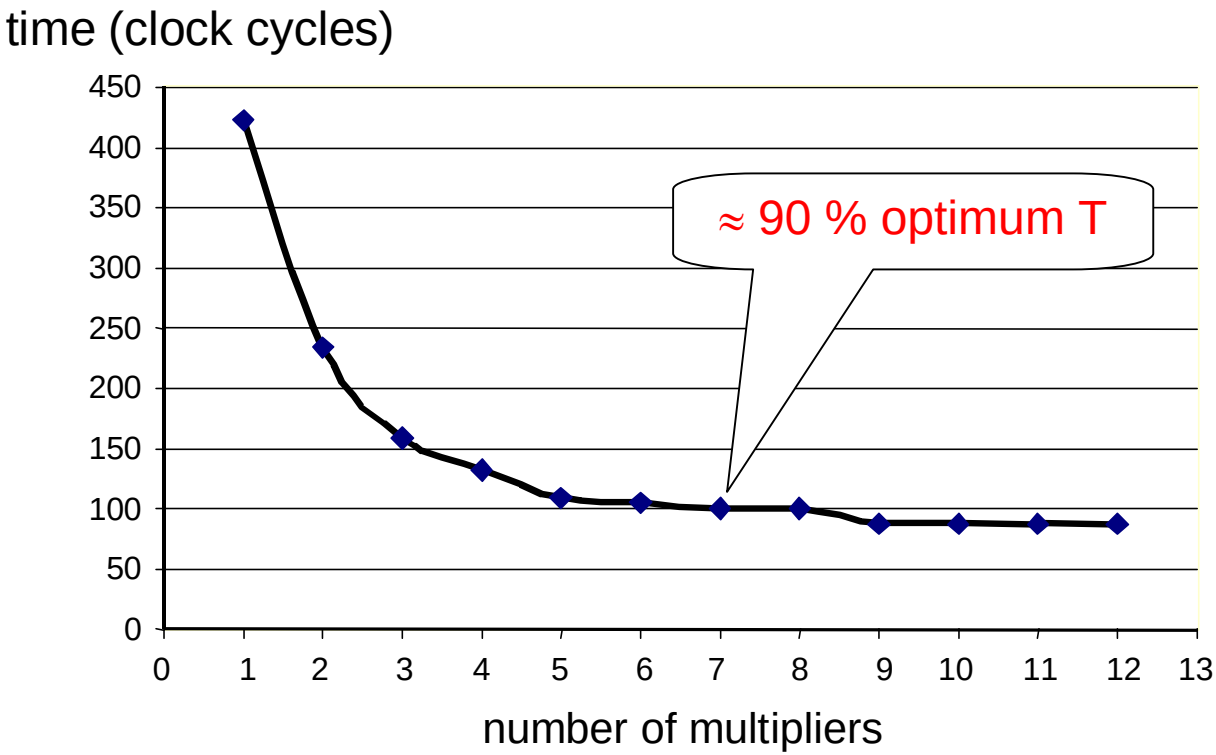
35 each of $2m$ bits

Architecture Evaluation – I

optimal time

×	^	+	√	T
1	2	1	1	423
2	2	4	1	234
3	2	4	1	159
4	2	4	1	132
5	1	4	1	108
6	1	3	1	105
7,8	**2**	**3**	**1**	**99**
≥ 9	1	4	1	87

optimal T for the loop body of the DL alg.



identification of the optimal time schedule, depending on the number of multipliers (the cost-dominant function unit)

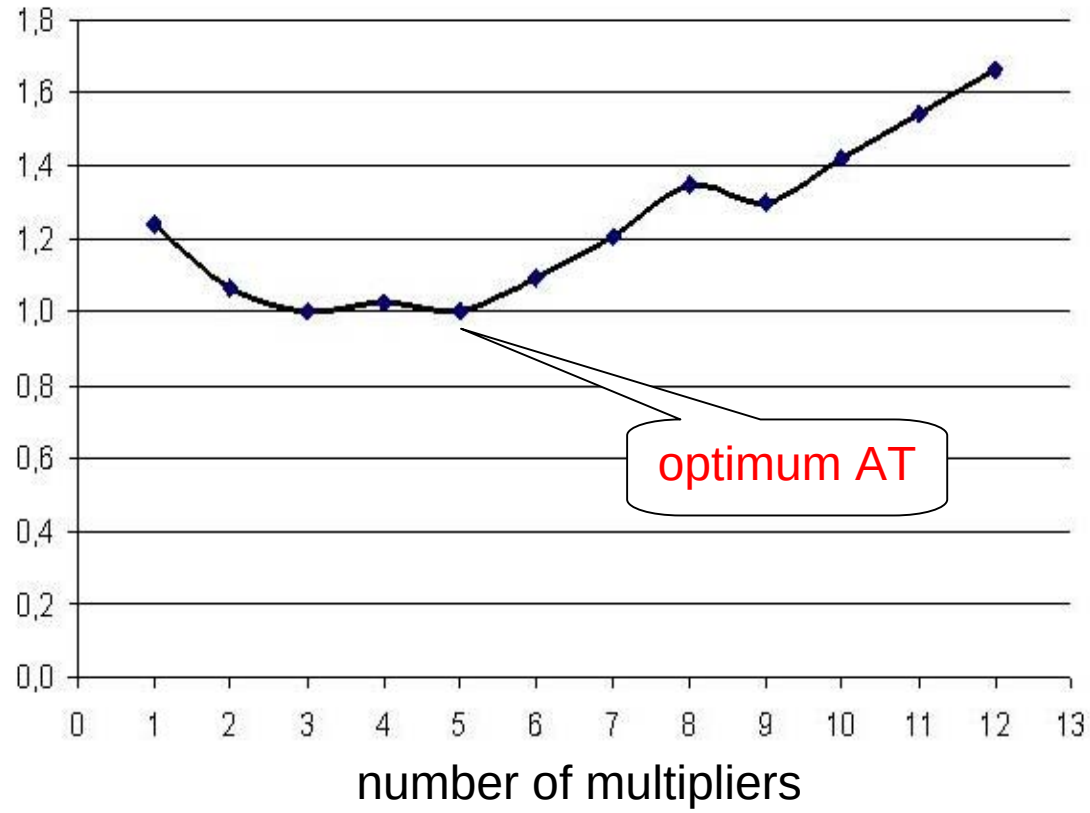
Architecture Evaluation - II

optimal area-time

non optimal

×	^	+	√	AT %	T
1	1	1	1	23.8	426
2	1	1	1	6.3	246
3	1	2	1	0.2	168
4	1	3	1	2.7	135
5	**1**	**4**	**1**	**min**	**108**
6	1	3	1	9.6	105
7	2	3	1	20.6	99
8	2	3	1	34.6	99
≥ 9	1	4	1	29.7	87

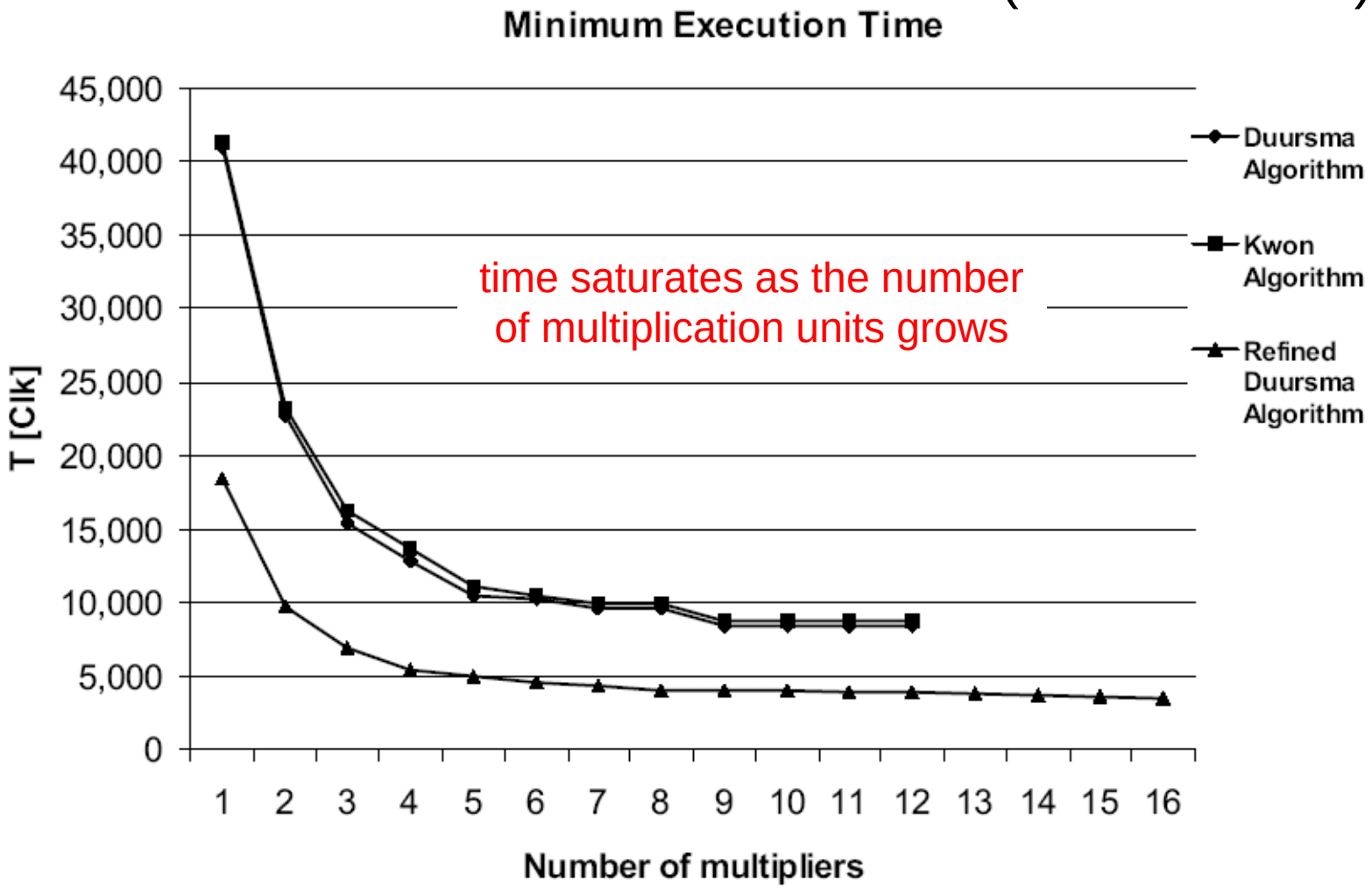
optimal AT for the loop body of the DL alg.



identification of the optimal area-time product schedule, depending on the number of multipliers (the cost-dominant function unit)

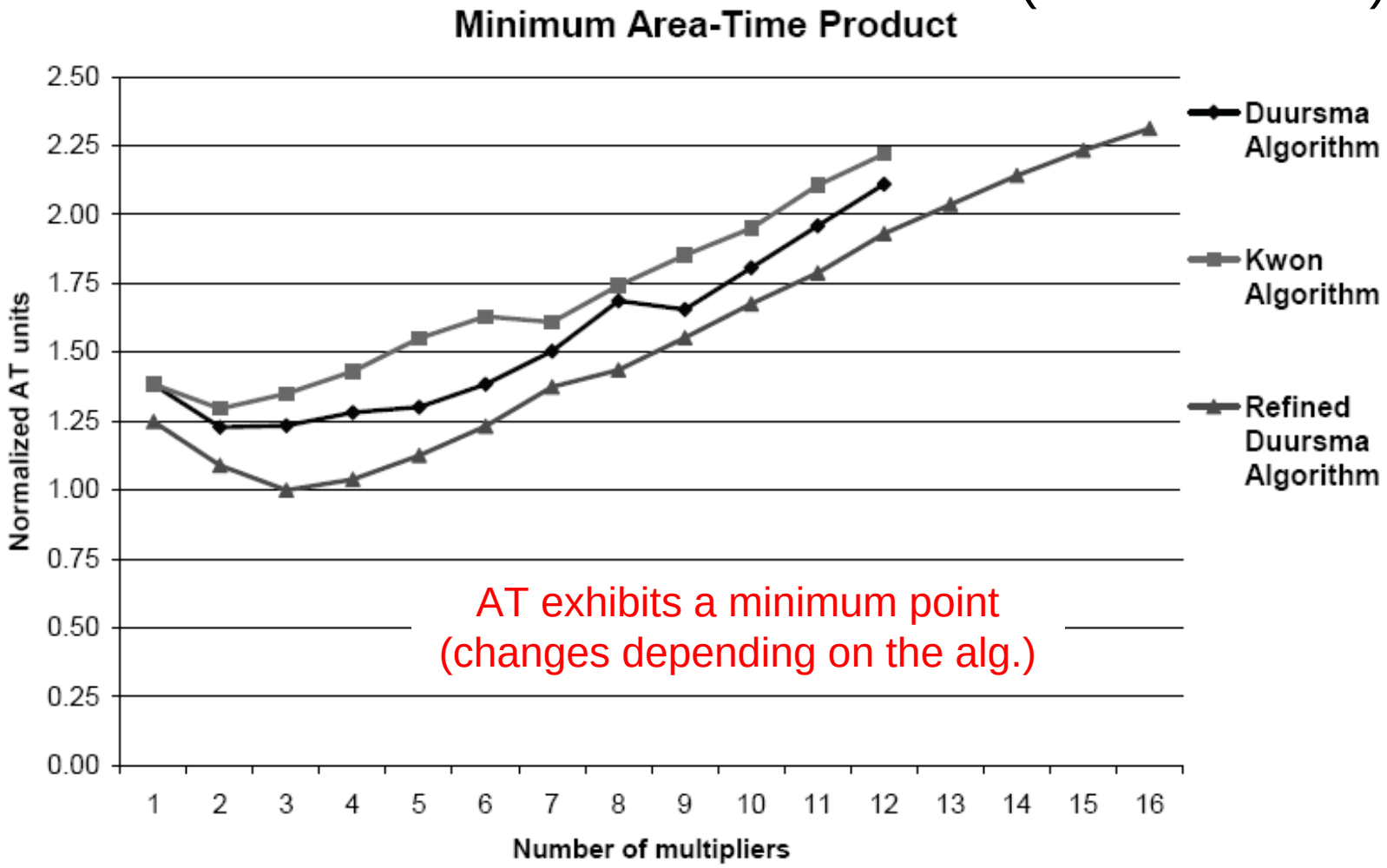
Architecture Evaluation - III

complete pairing algorithms $(m = 97 \quad D = 4)$



Architecture Evaluation - IV

complete pairing algorithms ($m = 97$ D $= 4$)



Architecture Evaluation - V

time-optimal algorithms when varying the number of multipliers, the digit size D and the base field size *m* (= 97, 193, 239 F_3 elements)

Number of Multipliers	Multiplier Digit Size D			
	1	2	4	8
[1, 8]	RDL	RDL	RDL	RDL
[9, 13]	DL	DL	DL	RDL
[14, 16]	RDL	RDL	RDL	RDL

minimum execution time (in clock cycles) for the loop body of the DL \ RDL algorithms, with 5 multipliers

Field Size m	Multiplier Digit Size D			
	1	2	4	8
97	324 \ 317	180 \ 174	108 \ 104	72 \ 64
193	612 \ 500	324 \ 318	180 \ 175	108 \ 103
239	750 \ 615	393 \ 387	213 \ 170	123 \ 95

Function Units - Summary

area cost and time latency of the various function units, depending on *m* and D

Function Unit	Area	Latency
Adder	m F3A + $4m$ MX	1
Multiplier	$(m+1)\,D$ F3A + $((m+2)D-2)$ F3M + $8m$ MX + $(6m+2D-2)$ FF	$\lceil m/D \rceil$
Cube Power	$2m$ F3A + $2m$ MX	1
Cube Root	$(m-1)$ F3A + $2m$ MX	1

FF one flip-flop

MX one 2-to-1 mux

F3A one adder in F_3

F3M one multiplier in F_3

eventually measure all the costs as FPGA elements

Architecture Comparisons

FPGA implementation - VirteX2P100 – performance vs. cost

(time is referred to the entire computation of the DL algorithm, but no final exp.)

type	number of adders / multipliers	clock frequency (MHz)	area (FPGA slices)	time (clock cycles)
Kerins at al. (CHES 2005)	$\geq 100 / \approx 18$	≈ 15	55,616	8,924
scheduled DL (2006)	4 / 9	60.9	31,986	8,439
Grabher et al. (CHES 2005)	not a parallel design (serial, one function unit per type)			59,946

ESTIMATE not fully implemented

$(D = 4)$

FPGA percentage occupation

% FPGA	9 multipliers	4 adders / subtr.s	cube power & root	busses registers & controller
72 %	27 %	2.8 %	0.9 %	41.3 %

suggests to bound connections

Some Considerations

- The Tate pairing algorithms in F_3^m over supersingular curves are well suited for parallelism at the level of the operations in the base field.
- The complexity of the analysis and synthesis of the parallel HW solutions is beyond the reach of intuitive or hand-made techniques.
- Formal methodologies and disciplines (scheduling) are advisable to carry out an extensive exploration of the architectural solution space.
- More extensive exploration is however necessary:
  - other types of fields curves and algorithms wanted
  - more implementations wanted (and ASIC, not only FPGA)
  - include connections in the constrained resources - next
  - is scheduling limited to be useful only in pure HW ? - next

Case Study – II

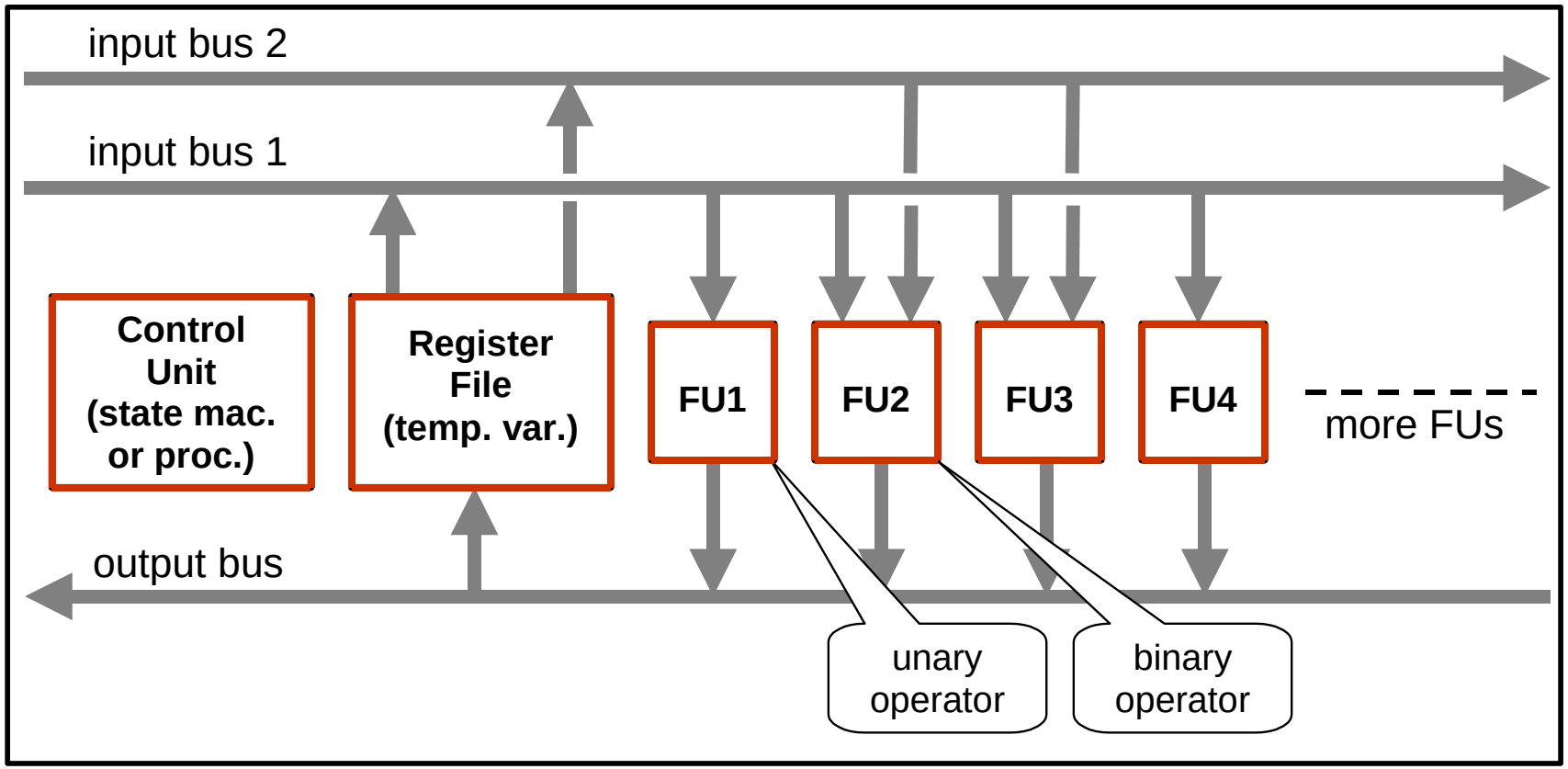
more on dedicated architectures for Tate pairing in F_3^m and F_p

Objective

- Design a more “realistic” parallel architecture:
  - adopt a somewhat standard datapath model
  - possibly more suited to multiple algorithms
- Constrain the following computing resources:
  - the number of function units
  - the number of interconnections
- Schedule a few algorithms and compare the costs and performances of the solutions.
- Some final considerations …

Parallel Architecture Model

a standard 3-bus datapath architecture



the FUs may be replicated

Duursma-Lee Algorithm in $F_3{}^m$

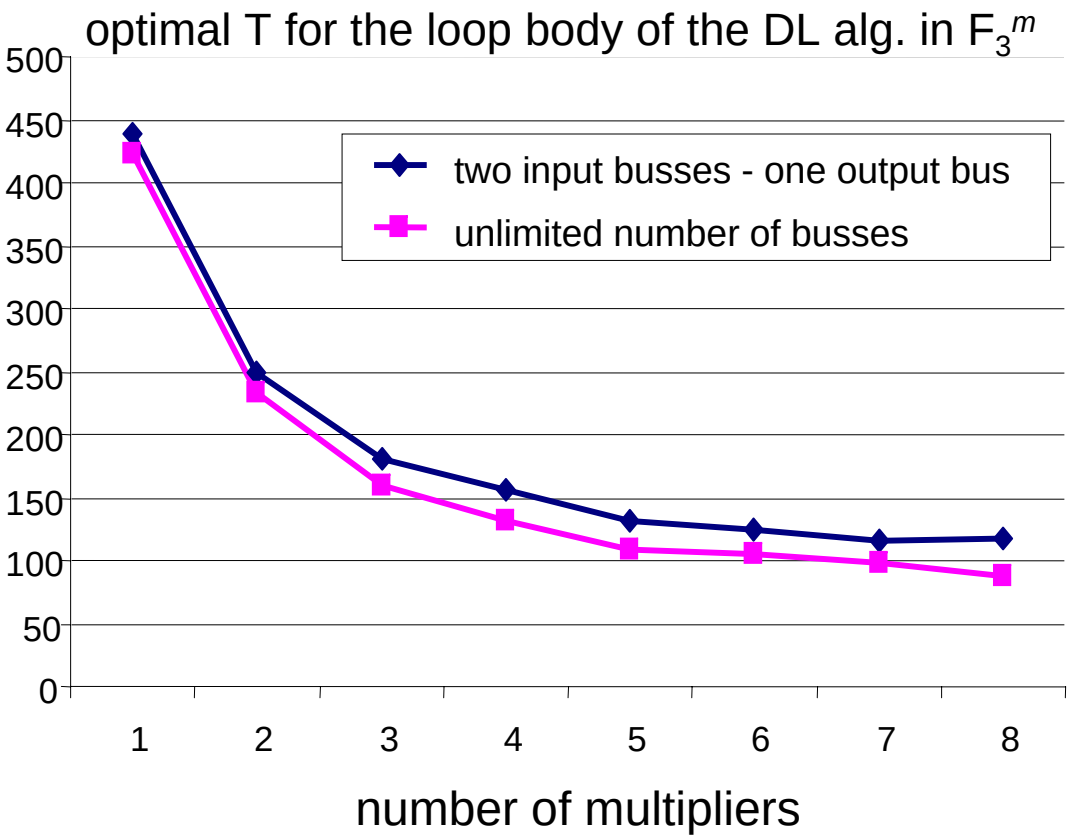
($m = 97$ D = 4)

with 3 busses

unlimited busses

×	^	+	√	T	T
1	2	1	1	423	438
2	2	4	1	234	250
3	2	4	1	159	180
4	2	4	1	132	157
5	1	4	1	108	131
6	1	3	1	105	124
7, 8	**2**	**3**	**1**	**99**	**116**

under 20% slow down



BKLS Algorithm in F_p - I

optimized BKLS alg. for supersingular curve $p \approx 2^{512}$ & $k = 2$

loop body only

time is in clock cycles

AT is almost constant

schedule (limited # of busses)					
×	**input**	**output**	+	<<	**time**
1	1	1	1	1	9420
2	1	1	1	1	4780
3	3	1	1	1	3380
4	1	1	1	1	2810
5	**2**	**1**	**1**	**1**	**2380**

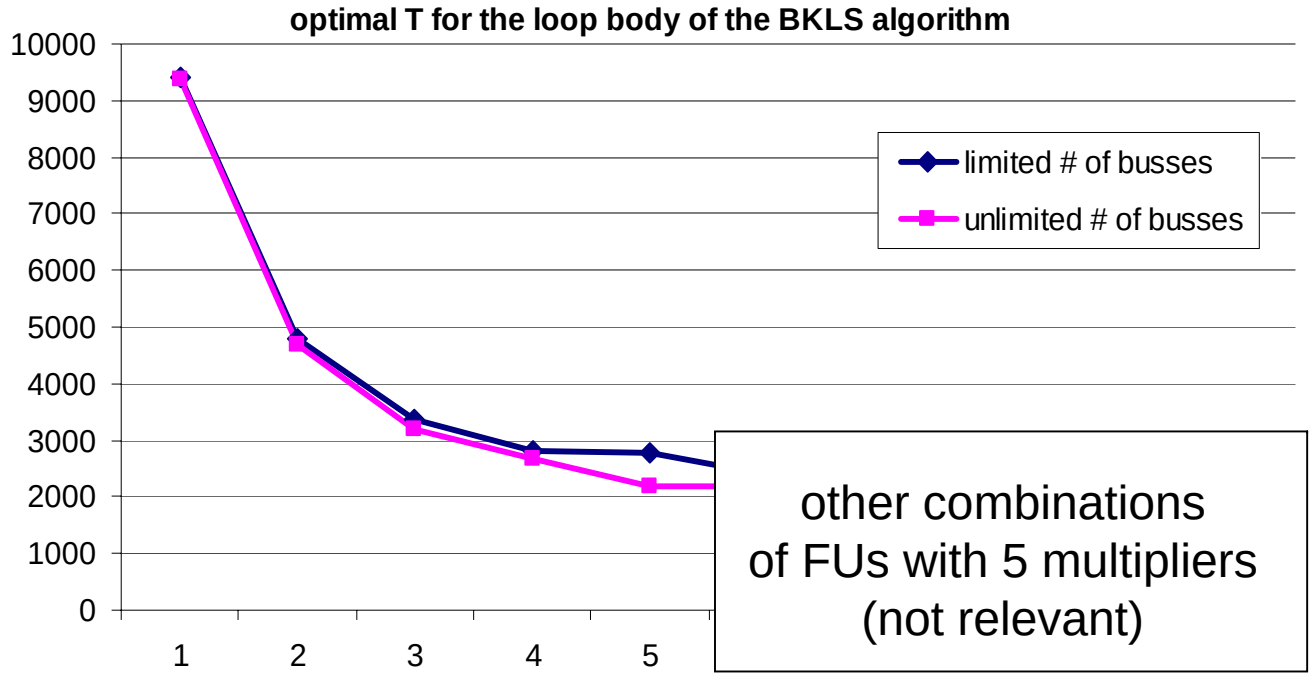
under 10% slow down

schedule (unlimited # of busses)			
×	+	<<	**time**
1	1	1	9380
2	1	3	4700
3	1	2	3180
4	1	2	2660
5	**1**	**2**	**2180**

L. Marnane, preprint

FU time latencies in clock cycles				
×	**input**	**output**	+	<<
520	10	10	20	20

BKLS Algorithm in F_p - II



other combinations
of FUs with 5 multipliers
(not relevant)

very small changes

effect of some combinations
of input & output busses

schedule (limited # of busses)					
×	**input**	**output**	+	<<	**time**
5	1	1	1	1	2400
5	**2**	**1**	**1**	**1**	**2380**
5	2	2	2	1	2360

Some Considerations

- When bounding the number of connections, the incidence on time performance is small (roughly from 10 % to 20 % at worst).
- There is a relevant exploitable parallelism (e.g. up to 5 multipliers).
- The architecture is somewhat standard and:
  - could be easily rescheduled for other algorithms
  - provided these alg.s use the same function units
- For instance, one might schedule also:
  - the final exponentiation (for Tate pairing)
  - elliptic curve auxiliary stuff (e.g. scalar mult.)

Case Study – III

reconfigurable programmable architectures for Tate pairing in F_p

Motivation

- Reconfigurable parallel programmable architectures have gained popularity, as it is possible to integrate many (say 10-100) simple small processors, each with a local program memory, on a single chip.
- A reconf. par. programmable architecture is similar to a dedicated one, but the FUs are programmable.
- This kind of architectural model:
  - is object of intense research nowadays, and various research projects exist as well as some commercial solutions of differing types
  - is proposed for several application types, (e.g. multimedia), where cryptography is likely to exist as well
- Therefore, such a model is an interesting case study for the parallel computation of pairings.

Objective

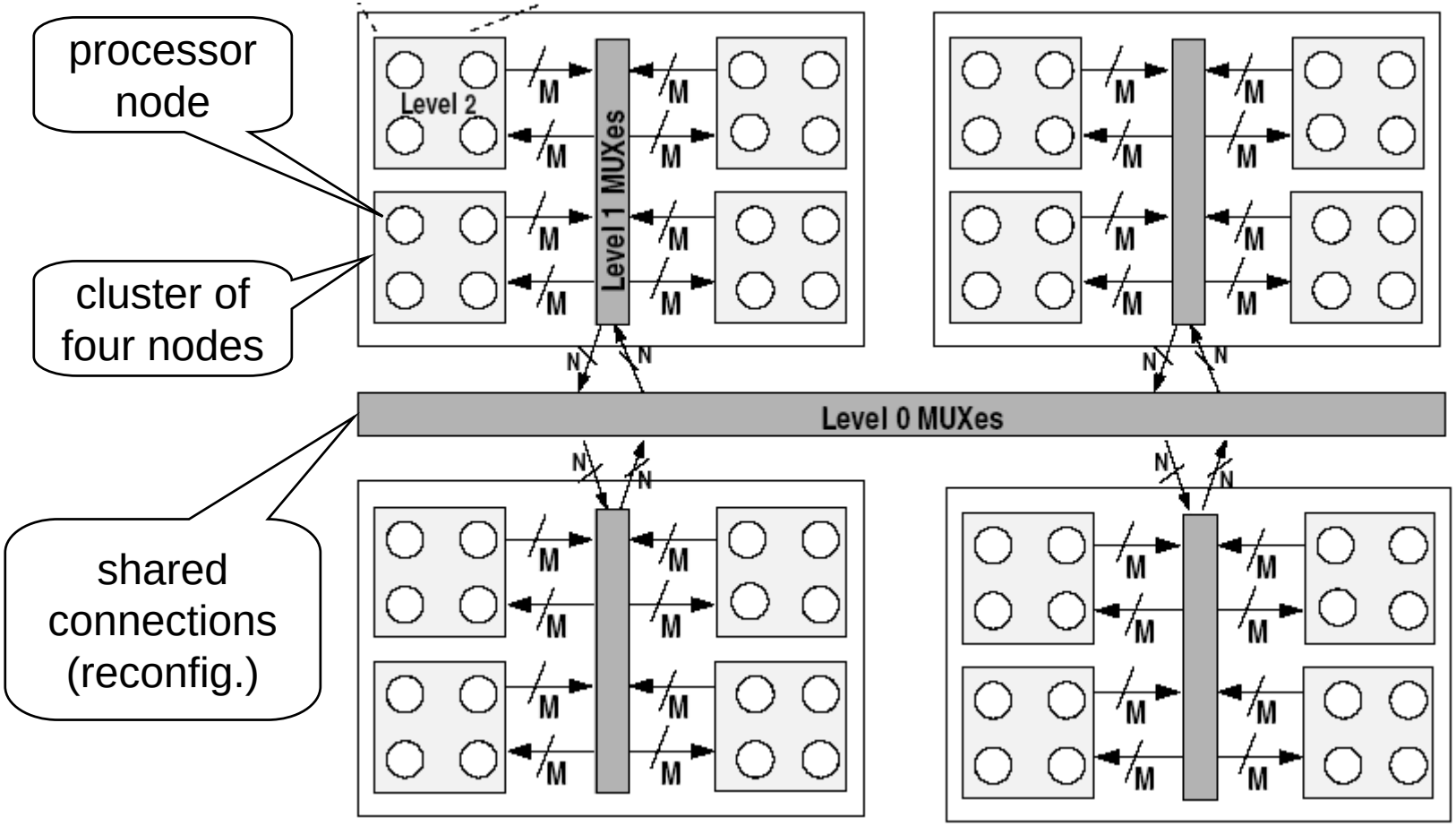
- Explore the feasibility of *reconfigurable parallel programamble architectures* for pairing in F_p.
- Program a simple such architecture model, namely the future multi-processor DSP-Fabric chip by ST Microelectronics (also called Tiled Architecture).
- Use an appropriate parallel programming technique, that includes scheduling, to achieve a satisfctory parallelism level for the entire pairing algorithm.
- Analyze time performance for several schedules, with respect to the number of processing nodes.
- Experimental evaluation of time performance and comparisons with possible similar solutions.
- Some final considerations …

Generalities

- According to AsTrO taxonomy, reconfigurable architectures of the Scalar Operand Network type (those of interest here) can be classified as below, depending on the method for:
  - assigning instructions blocks to nodes
  - moving data through memory and nodes
  - scheduling the instruction flow in each node
- The three tasks above can be carried out in a *static* way (decided at compile-time) or in a *dynamic* way (decided at execution-time).
- DSP-Fabric is of the *static/static/static* type.

Tiled Architecture Model - I

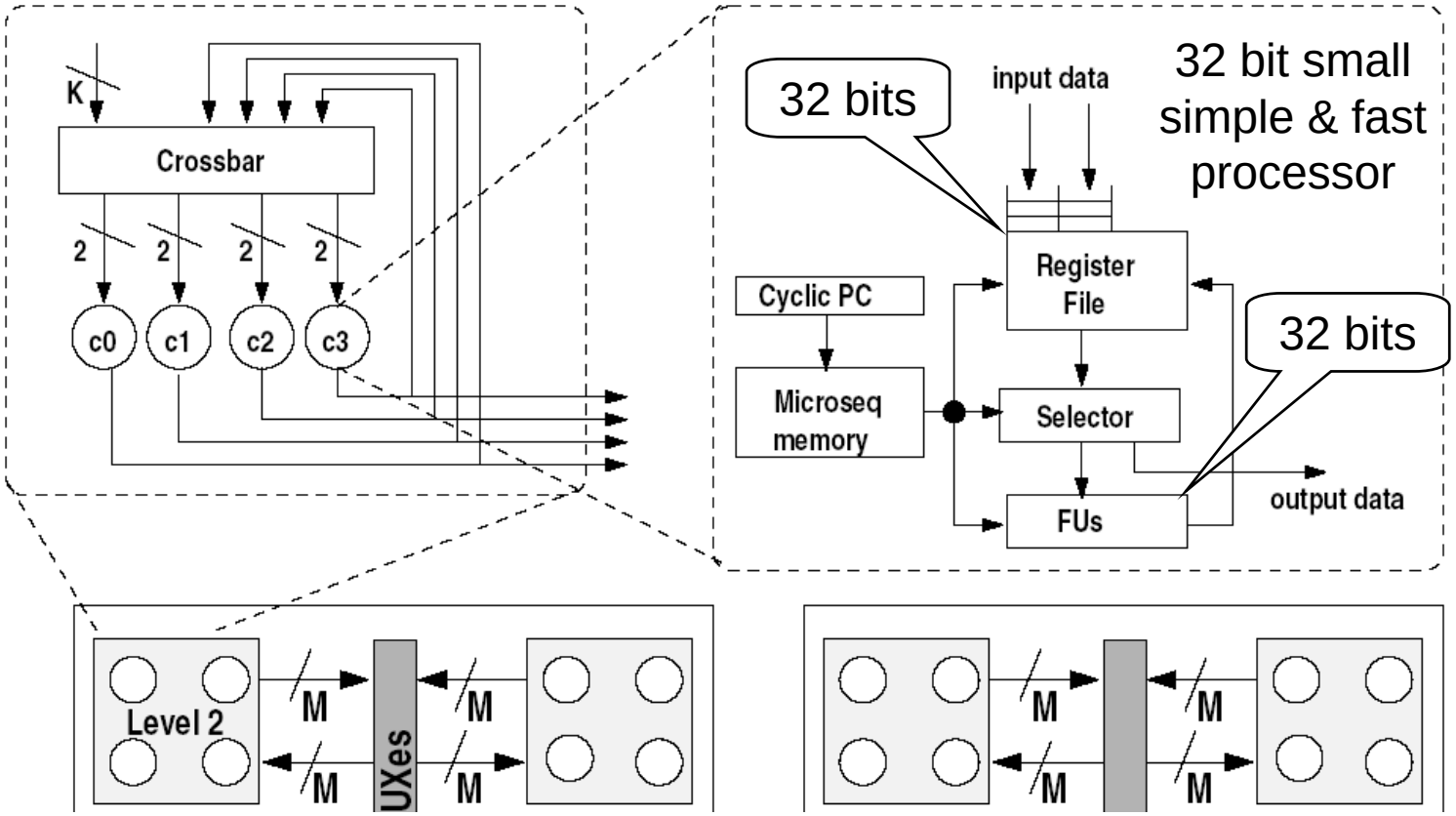
DSP-Fabric (by ST Microelectronics) with 64 nodes



Tiled Architecture Model - II

cluster of four nodes
(fully interconnected)

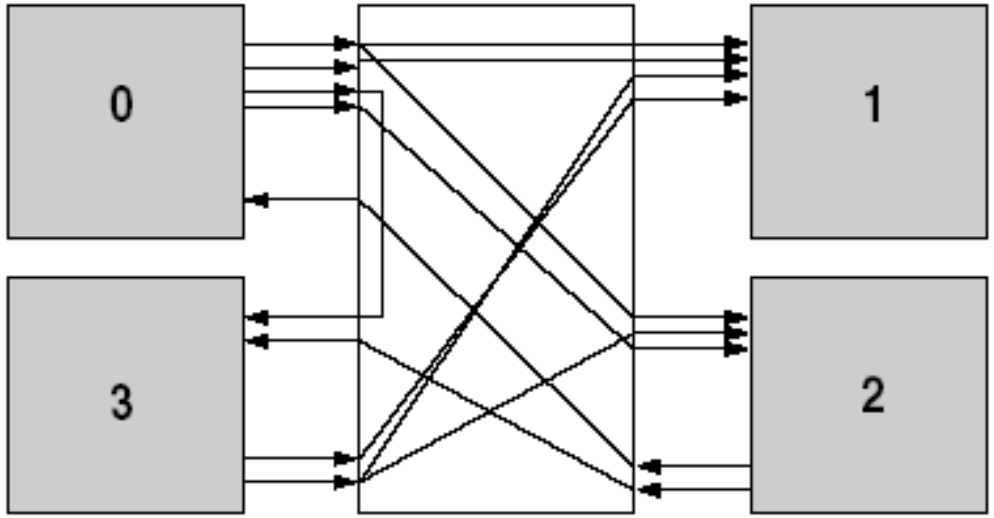
architecture of the node
(datapath & control)



Tiled Architecture Model - III

hierarchical cluster of 16 nodes (partially interconnected)

shared connections



a feasible static interconnection of clusters

if both the fan-in and fan-out of one 4-node cluster are of 4 wires, clusters 0 and 1 saturate, respectively, while 2 and 3 do not

Pairing on Tiled Architecture

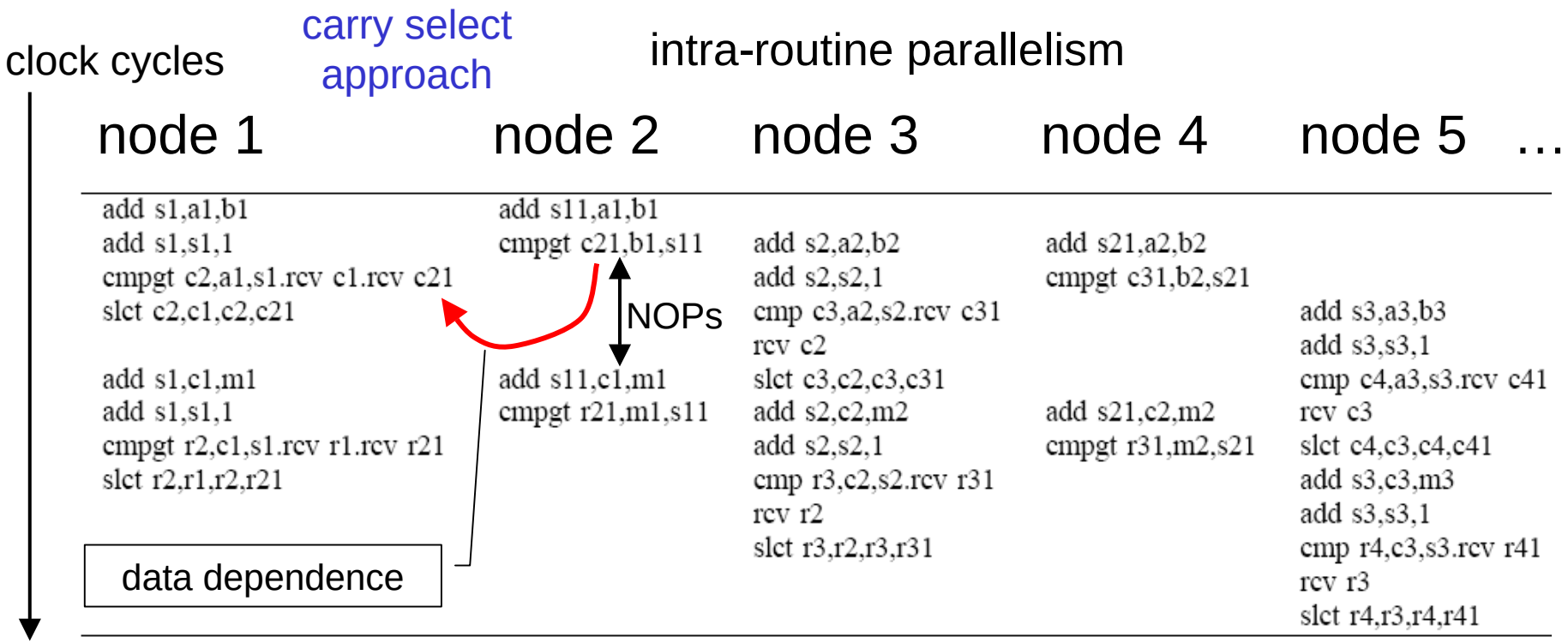
- Idea: imagine of replacing the arithmetic HW function units (add, mul) with equivalent *SW function units*:
  - each SW function unit is a program routine
  - and runs split in parallel on a group of nodes
- Optimize each SW FU independently and use as few nodes as possible (intra-routine parallelism):
  - so far compilers can not do so efficiently
  - intra-routine parallelism need be done manually
- Program in parallel the pairing algorithm and use the SW FUs as black boxes (inter-routine parallelism):
  - schedule the pairing algorithm for parallel execution
  - inter-routine parallelism can be determined in an automated way similarly to the case of dedicated HW architectures
- Execute the pairing algorithm with a cycle-accurate simulation tool and evaluate performance vs. cost.

SW Function Unit – Adder - I

- Modular addition (integer addition plus reduction) is computable on a subset of nodes and is organized in a parallel way.
- The addition algorithm is divided into smaller blocks (e.g. $32 + 32 \rightarrow 32$ bits) as well as reduction, and is programmed as a SW function unit.
- Here a carry-select approach is followed, to speed up the carry propagation chain.
- Some intra-routine optimization is necessary to parallelize well (hand-made).
- The resulting SW routines must then be scheduled to expose parallelism compatibly with the selected constraints (e.g. the max number of available nodes).

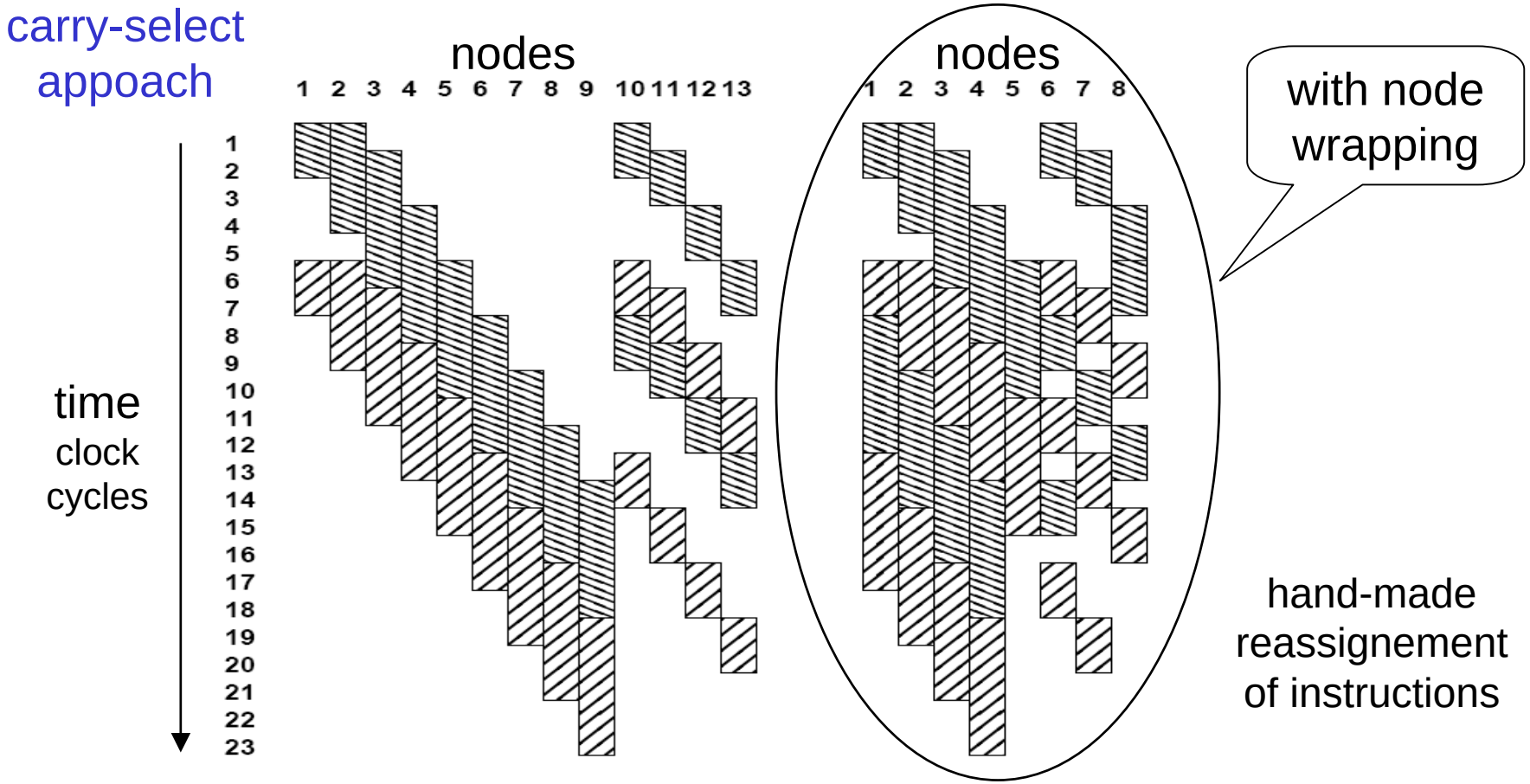
SW Function Unit – Adder - II

part of the SW implementation of the modular adder as a SW function unit (for 2 summands of 3 words each)



SW Function Unit – Adder - III

sample time / space scheduling of a 128-bit modular addition
dark and light shaded areas represent integer addition and reduction, resp.

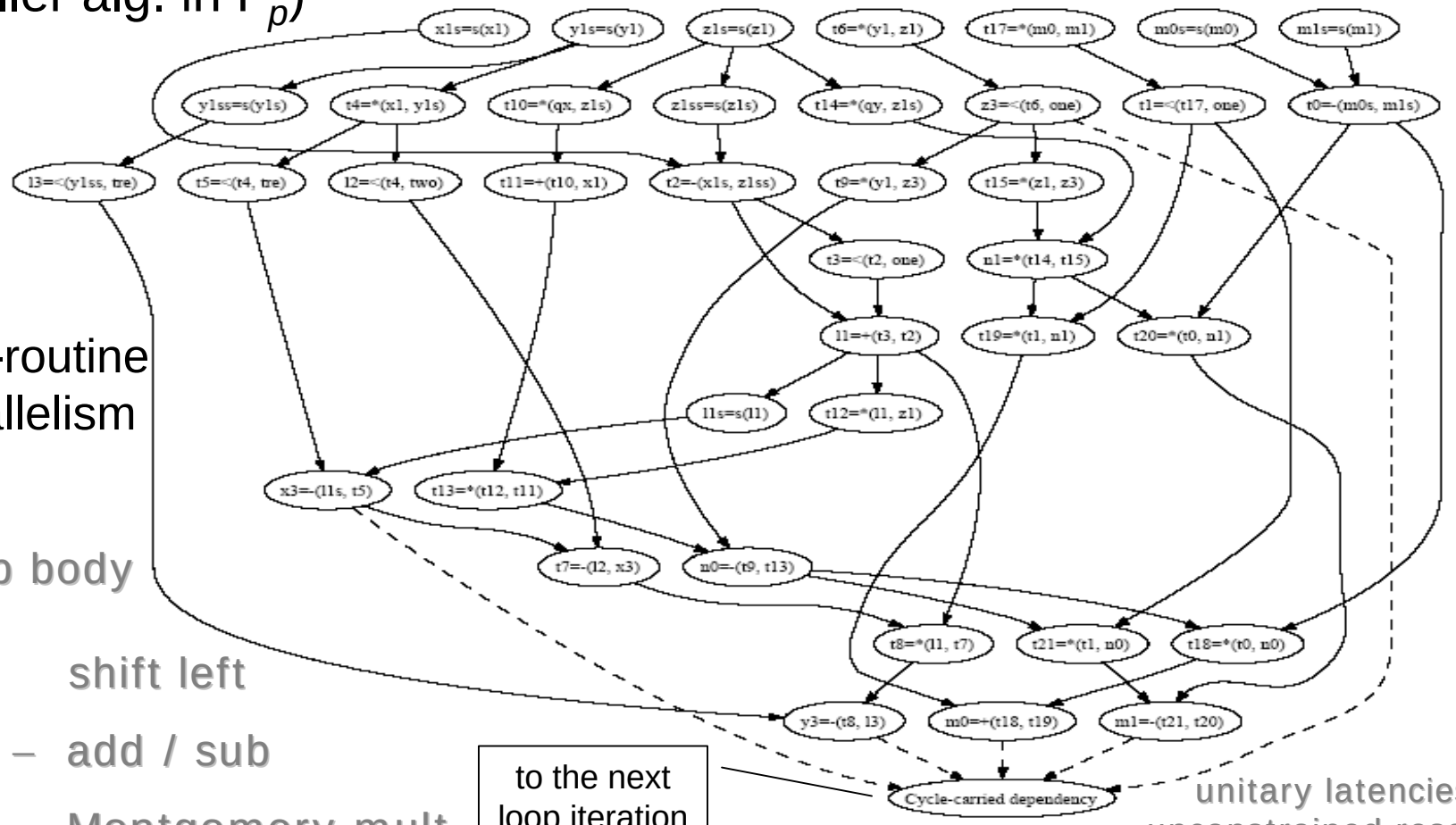


SW Function Unit – Multiplier

- Modular multiplication is computable on a subset of nodes, similarly to what done for addition.
- It is computed by the well-known Montgomery alg. of the parallel / digit-serial to parallel type.
- The implementation is similar to that of addition (details omitted) and intra-routine optimization is again necessary to parallelize well (hand-made).
- The resulting SW routines must then be scheduled (along with addition) to expose parallelism compatibly with the constraints (e.g. the max number of nodes).

Algorithm Scheduling

high-level automatic LB scheduling of the DDG for the doubling step (Miller alg. in F_p)



inter-routine parallelism

loop body

< shift left

+ / − add / sub

∗ Montgomery mult.

to the next loop iteration

unitary latencies & unconstrained resources

SW Function Unit – Summary

Finite Field Operations	Clock Cycles	# of CPUs
$x \pm y \bmod m$	$2n+6$	8
$x \cdot y \bmod m$	$(n+1)(2n+19)$	$2n$
$x << z \bmod m$	$2n+2$	8

time and area/time product for the software implementation of the Montgomery multiplier as a function of input words and number of nodes

Input size n	Number of CPUs	Time [clk]	Time × Area [clk×#CPU]
8	16	315	5040
16	16	1088	17408
16	32	867	27744

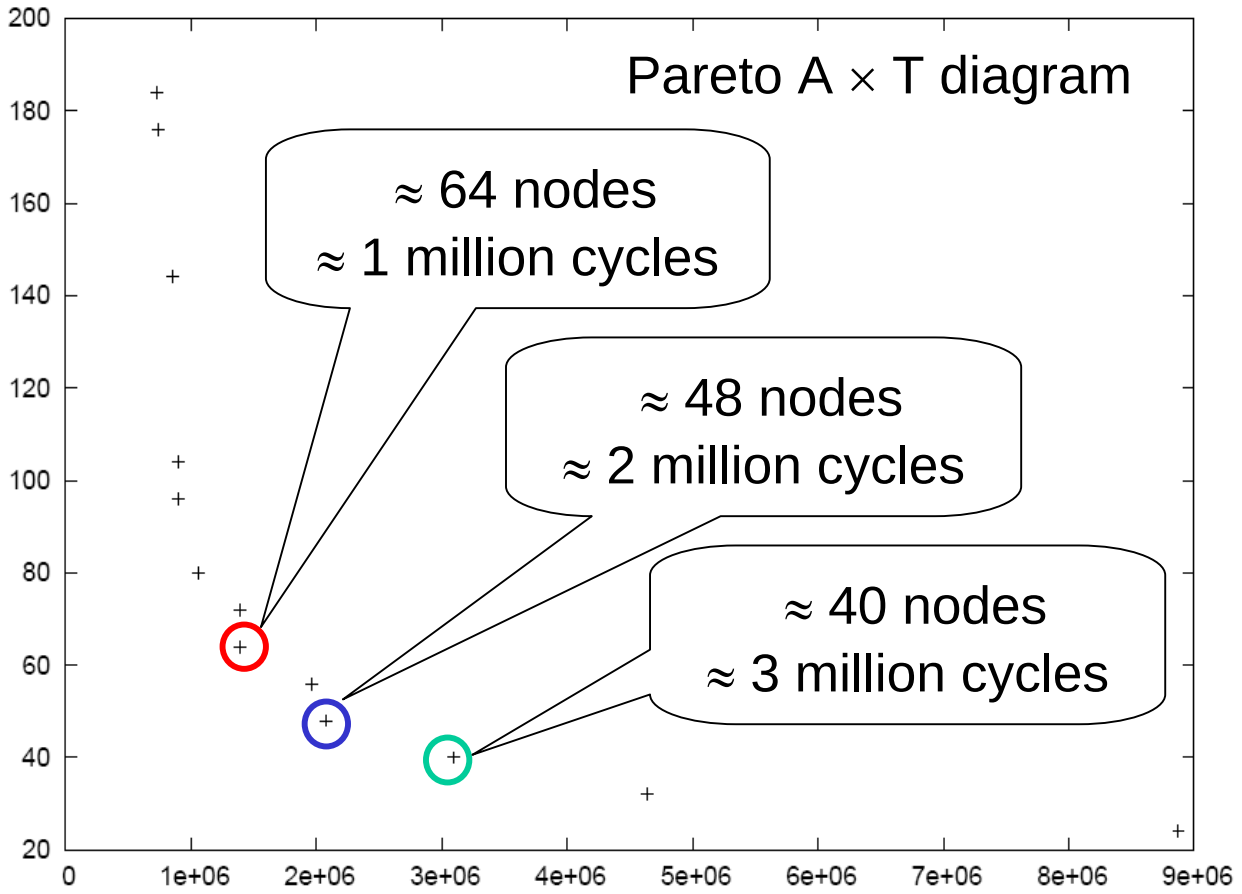
hand-made reassignment of instructions

Parameter *n* is the number of words (here 32 bit words).

Area is evaluated as the number of nodes (processors).

Architecture Evaluation

area (# of nodes)



time (clock cycles)

Architecture Comparisons

- Difficult to compare DSP-Fabric with FPGA, however case study 1 reports 8.439 @ 60.9 MHz, time 138 μs (but for F_3^m with $m = 97$, not for F_p)
- DSP-Fabric (F_p, $p \approx 2^{512}$): ≈ 1 million cycles, @ up to 400 MHz, 64 nodes, time 2.5 ms (chip ≈ 7 mm^2 in ASIC technology, planned for 2009).
- Strong ARM (F_p, $p \approx 2^{512}$): ≈ 60 million cycles @ 206 MHz, 1 node, time 291 ms (Scott et al. 2006).
- See also Scott et al. 3 ms (on a P IV @ 3 GHz).

Parallelism does very well in DSP-Fabric !

time performance span

FPGA = 1 DSP-Fabric ≈ 10-10^2 ARM ≈ 10^3-10^4

Some Considerations

- Reconfigurable programmable parallel architectures seem to be a promising technology for paring (and also for ECC, as most operations are common).
- Present compilers for reconfig. architectures can not exploit the parallel features of the current pairing algorithms (and of ECC algorithms either).

  $\Rightarrow$ need for the automation of parallelization

  $\Rightarrow$ other case studies (algorithms, fields, FUs)
- Moreover, such architectures have interesting power consumption features, which fact perhaps could be exploited for cryptography as well (against attacks).

The End (by now)

overall conclusions
and future research

Overall Conclusions

- Pairing algorithms exhibit a notable degree of parallelism, in several different conditions:
  - parallelism is intrinsic to the algorithm and stable
  - is due not only to “side” effects (parameters or simplifying hypotheses or technology)
- Parallelism is better exposed at the level of the operations in the base field (not at high level).
- Formalization and automation are necessary to expose and exploit parallelism (e.g. scheduling).
- Various technological options are available to compute pairings (not only, e.g. ECC) in parallel.
- This seems to be a research worthy of pursuing.

Future Research

- Continue exploration with other current and possibly newcoming pairing algorithms.
- Complete and improve design toolchain:
  - in particular add a VHDL modeling procedure
  - possibly add a graphical output facility
- And extend methodology to the scheduling of multiple algorithms:
  - pairings are unlikely to be used in isolation
  - need be mixed with other algorithms (e.g. ECC)
  - identify reasonable criteria for assigning relative weights to multiple algorithms
  - criteria might come from the mix of algorithms needed by high-level cryptographic protocols (e.g. IBE)
- Add more if you want …