CAMBRIDGE

Cambridge University Press & Assessment
978-0-521-88099-2 — Arithmetic Optimization Techniques for Hardware and Software Design
Ryan Kastner , Anup Hosangadi , Farzan Fallah
Index
More Information

Index

Numbers in *italics* indicate figures.
Numbers in **bold** indicate tables.

abstract syntax tree (AST) high-level program representation, 40
adaptive differential pulse-code modulation (ADPCM) encoder, 42, *42*, 43, *43*, 44, *44*
adders
  adder tree for a FIR filter, 60, *61–2*
  see also multiple-operandadder, two-operand addition
address calculation, 17–18
  address calculation units (ACUs), 18
algebraic optimization, 99–113
  experimental results/comparisons, 113–17: additions/multiplications, 114, **114**; with hardware constraints, 115–17, **115**, **116**; Jouletrack simulator, 114; latency and energy reduction, 114–15, **114**; Synopsis Behavioral Compiler^{TM}, 115: Synopsis Design Compiler^{TM}, 115; voltage scaling, 115
  terminology explanations: co-kernels, 101; cube-free expression, 101; cubes, 101; kernels, 101; literals, 101; sum of products (SOP), 101
  see also divisorextraction methods of optimization, kernels/co-kernels, rectangle covering methods of optimization
algebraic transformations *see* software compilers/compilation, algebraic transformations in optimization
algorithmic optimization, with hardware synthesis, 37, 44–5
analysis stage of compilation, 24
application specific integrated circuits (ASICs), 3
approximation algorithms, 9–10
architectural synthesis for hardware, 35–7
  algorithmic optimization, 37
  lexical and syntactic analysis, 37
  logic synthesis, 38
  physical synthesis, 38
  resource allocation, 37
  resource binding, 37–8
  see also physical synthesis, for architectural synthesis
arithmetic expressions/operations *see* algebraic optimization, integer linear programming (ILP) solutions for reducing operations
arithmetic logic unit (ALU), 50
as late as possible (ALAP) unconstrained scheduling, 51–2, *52*
as soon as possible (ASAP) unconstrained scheduling, 51–2, *52*
asymmetrical encryption, 16–17

Bezier formulation/vector equation, 95–6
binary signed digit representation, 71–3
  Booth's algorithm, 71
  caconical (sparse) form, 71
  Hamming weight, 71
  nonadjacent form, 71
  Parhami's generalized signed digit system, 72–3
  Reitwiesner optimal algorithm, 71–2, *72*
Booth's algorithm, 71
Buchberger algorithm, 98–9

C language, 39
canonical representation, 71
  canonical signed digits (CSDs), 131–2, *132*
  using the ordered matrix, 100–1
canonical/non-canonical number systems, 70–1
carry look-ahead adder (CLA), 79–82
  delay issues, 82
  four-bit CLA, 80, *81*
  three-level eight-bit CLA, 80, *81*
  two-bit CLA, 79, *80*
carry propagate adders (CPAs), 83–5, *84*, *138*, 139
carry save adders (CSAs), 86–8, *86*, *87*, *88*
  with multiple operand addition, 158–60, *159*
  trees, 158–9, *159*
Chatterjee *et al*. technique, 141
co-kernels, 101

© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE

Cambridge University Press & Assessment
978-0-521-88099-2 — Arithmetic Optimization Techniques for Hardware and Software Design
Ryan Kastner , Anup Hosangadi , Farzan Fallah
Index
More Information

code generation stage for compilation, 25
combinational functional units, 46
common subexpression elimination (CSE)
  with computer graphics, 11
  with delay-aware optimization, 164–74
  for polynomial optimization, 97–8
  result comparisons, 114–16, **114**, **115**
  in software compilers, 25–7, *26*
  with two-operand adder synthesis, 145–7
compilers/compilation *see* software compilers/compilation, structure
compressors, 89–90, *89*, *90*, *91*
computer graphics
  Horner form, 11
  multiply accumulate (MAC) operations, 11
  polynomials for, 10–12
  quartic spline polynomial optimization, 11–12
  spline interpolation, 10–11
  two-term CSE algorithm with, 11
concurrent arithmetic extraction (CAX), 112–13, *113*
  with CSE algorithm, *146*
  with FIR filter optimization, 153–4
  with multiple-operand addition, 162
condense algorithm, 99, 111
configurable logic blocks (CLBs), 155
conjunctive normal form (CNF) to ILP clauses, 118–19
control data flow graph (CDFG), 26–7
control flow graph (CFG) representation
  for hardware synthesis, 40, *41*
  for software compilation, 23–4, *24*
counters and compressors, 89–90
cryptography, 16–17
  asymmetrical encryption, 16–17
  exponentiation, 18, *19*
  method of squaring, 18
  public key cryptography, 16–17
cube intersection matrix (CIM), 109–11, *110*, *111*
cube polynomial expressions, 101
  cube-free expressions, 101

data flow graph (DFG) representation, 23–4, *24*
  for hardware synthesis, 37, 40–1, *42*, 58–9
dataflow optimization in modern software compilers, 25–6
  CDFG, 26–7
  DU chains, 25
  flowgraphs, 26, *26*
  local and global CSE, 25, *26*
define use (DU) chains, 25
delay-aware optimization with three-term CSE, 172–4
  delay model, 172–4
  delay-aware optimization algorithm, 174
  delay-aware three-term CAX, 173, *173*
  delay-ignorant three-term CAX, 172–3, *173*
  experimental results, 174, *174*
delay-aware optimization with two-term CSE, 164–72
  about the technique, 165
  arrival times of the variables, 167, *168*
  delay calculation, 168–9, *169*
  delay-aware CSE algorithm, 167–70
  divisor's true value algorithm, 170, *170*
  experimental results: for multi-variable DSP transforms, 171–2, **171**; with NRCSE, delay ignorant and delay aware algorithms, 170, **171**
  impact of optimization on delay, 165–6, *166*
  problem formulation, 165–7
  procedure for calculating delay, 167–8, *168*
  recursive and nonrecursive CSE, 166–7, *167*
digit-based recording algorithms, 135
digital arithmetic *see* multiple-operand adder; number representation; two-operand addition
digital signal processing (DSP), 12–16
  FIR filters, 12–13
  linear transforms, 13–16
directed acyclic graph (DAG), 41
discrete cosine transform (DCT), 128–9
  four-point DCT, *14*, *15*
  for JPEG and MPEG compression, 13–16
discrete Fourier transform (DFT), 127
discrete Hartley transform (DHT), 128
discrete sine transform (DST), 129
distill algorithm, 99, 111
distributed arithmetic, 90–3, *92*
divisor extraction methods of optimization, 111–13, *112*
  concurrent arithmetic extraction (CAX), 112–13, *113*
  divisor subexpressions, 111–13

embedded computing/systems, 3
  design flow, 3–4, *4*
exponentiation, 18, *19*

field programmable gate arrays (FPGAs), 155–8
finite impulse response (FIR) filter, 12–13, 148–51
  architecture with adders and latches, 151–3, *152*
  L-tap FIR filters, 12, 148–9; direct form, 148–9, *148*; distributed arithmetic implementation, 149–50, *150*, *151*
  with multiple-constant multiplication (MCM), 133–4, *134*
  reducing the number of registered adders, 152–3, *153*
  tapped delay line representation, *13*

© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE

Cambridge University Press & Assessment
978-0-521-88099-2 — Arithmetic Optimization Techniques for Hardware and Software Design
Ryan Kastner , Anup Hosangadi , Farzan Fallah
Index
More Information

finite impulse response (FIR) filter case study for hardware synthesis, 58–64
adder tree, 60, *61–2*
area/latency relationship, 62–3, *64*
C code for a 64-tap FIR filter, 58, *58*
DFG of the unrolled 64-tap filter, 58–9, *59*
minimizing area of computational resources, 60–1, *62*
most area efficient design, 61–2, *63*
resource sharing, 61–2
technological library example, **60**
finite impulse response (FIR) filter optimization, 147–58
experimental results using FPGAs, 155–8; additions before and after optimization, 156, **156**; comparisons of slices, LUTs, flip flops and performance, 156–7, **156**; MAC-based FIR filter comparison, 157, **158**; power consumption comparisons, 157, *158*; reduction in slices, LUTs and FFs, 157
FIR filter fundamentals, 148–51
optimization algorithm, 153–5: algorithm for reducing area, 154, *155*; divisor value approach, 154; fast evaluation with extra registers, 153, *154*; use of CAX algorithm, 153–4
fixed point number representations, 73–4
accuracy, 73
range, 73
resolution, 73
two's complement representation, 74
flip flops (FFs), 82
floating point number representations, 3–4, 74–5
advantages/disadvantages, 75
exponent, 74
IEEE 754 standard, 74–5
significand, 74
flow graphs
control data flow graph (CDFG), 26–7
see also controlflow graph (CFG) representation, data flow graph (DFG) representation
force directed scheduling (FDS), 54–5, *55*
functional units in hardware synthesis, 46–7

H.264 video codec, 14–16
Hamming weight, 71
hardware description languages (HDLs), 4
hardware synthesis, 35–64
algorithmic optimization, 44–5
hardware description languages, 35
resource binding, 56, *57*
system specification, 38–9
see also finiteimpulse response (FIR) filter case study for hardware synthesis
hardware synthesis, design flow, 35–8
DFG program representation, 37
program representation, 37
stages for, *36*
see also architecturalsynthesis for hardware, physical synthesis, for architectural synthesis
hardware synthesis, operation scheduling, 49–56
constrained scheduling algorithms, 52–6
force directed scheduling (FDS), 54–5, *55*
heterogeneous scheduling, 50
homogeneous scheduling, 50
list scheduling, 53–4, *53*
problem definition, 50–1
resource constrained scheduling (RCS), 49, 50–1
timing constrained scheduling (TCS), 49, 51, 55
unconstrained scheduling algorithms, 51–2
hardware synthesis, program representation, 39–44
ADCPM encoder, 42, *42*, 43, *43*, 44, *44*
AST high-level program, 40
CFG representation, 40, *41*
DFG representation, 40–1, *42*
PDG representation, 42–4, *43*
RTL description, 44, *44*
SSA intermediate representation, 42, *42*
hardware synthesis, resource allocation, 45–9
about resource allocation, 45–6
clock periods, 46–7
combinatorial functional units, 46
interconnect logic, 48
nonpipelined sequential function units, 46–7
pipelined sequential function units, 46
potential implementations for multipliers, *47*
problem definition, 49
storage elements, 48: on-chip memory blocks, 48; registers, 48
timing attributes, 47
hardware/software choices, 4
heterogeneous scheduling, 50
homogeneous scheduling, 50
Horner form/method
with computer graphics, 11
for evaluating polynomial approximations, 31–3, 96–8: advantages, 31–2: disadvantages, 32–3; limitations, 98; optimization of the quartic spline polynomial, *98*
result comparisons, 114–15, **114**, **115**

IEEE 754 standard, 74–5
integer linear programming (ILP) solutions for reducing operations, 117–23
CNF to ILP clauses, 118–19
generalizing the ILP model for redundancy elimination, 119–23
GenCkt algorithm to generate AND-OR circuits, 119–23, *120*, *122*

© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE

Cambridge University Press & Assessment
978-0-521-88099-2 — Arithmetic Optimization Techniques for Hardware and Software Design
Ryan Kastner , Anup Hosangadi , Farzan Fallah
Index
More Information

modeling CSE as an ILP problem, 117–19
intellectual property (IP) cores, 39
interconnect logic, hardware synthesis, 48
intermediate code generation for compilation, 23–4
DFG and CFG representations, 23–4, *24*
inverse discrete cosine transform (IDCT), 128–9
inverse modified discrete cosine transform (IMDCT), 129

Jouletrack simulator, 114
JPEG compression, 13–14

kernels/co-kernels, 101–9
co-kernels, 101
examples, *106*
expressions after extraction, *109*
finding intersections, 106–9, *108*
generating kernels of polynomial expressions, 102–3
kernel generation algorithm, 103–5, *103*; divide function, 104; merge function, 104
kernel-cube matrix (KCM), 106–9: prime rectangles in, *108*, *109*; rectangles in, 107
kernelling, 100
kernels algorithm, 99, *105*
performing factorizations, 106–9
kill, generate, propagate unit (KGP), 77, *78*

L-tap FIR filters *see under* finite impulse response (FIR) filter
lexical analysis (lexing/scanning)
for compilation, 22
for hardware synthesis, 37
linear systems
about linear systems, 2, 126, 178
basics, 126–9: additivity, 126–7; familiar linear transforms, 127–9; homogeneity, 126–7; transformation matrices, 127
problem formulation, 129–30: for a four-point DCT, 130
for signal processing, 2
see also multiple-operandaddition, synthesis for, multiplication, polynomial expressions/functions
linear systems, optimization of, 140–58
about linear system optimization, 140–2
Chatterjee *et al.* technique, 141: example, *141*
and the MCM problem, 140–1: example, *140*
for synthesis using two-operand adders, 143–7: Boolean expression techniques, 143; common subexpression elimination (CSE), 145–7; generation of two-term divisors, 143–5, *144*; H.264 example, *147*; two-term CAX, algorithm for, *146*
with transformation into a polynomial expression, 142
see also delay-aware optimization. . ., finite impulse response (FIR) filter optimization, software optimization
linear transforms, 13–16, 127–9
DCT for JPEG and MPEG compression example, 13–16
discrete cosine transform (DCT), 128–9
discrete Fourier transform (DFT), 127
discrete Hartley transform (DHT), 128
discrete sine transform (DST), 129
with H.264 video codec, 14–16
inverse discrete cosine transform (IDCR), 128–9
inverse modified discrete cosine transform (IMDCT), 129
modified discrete cosine transform (MDCT), 129
multipliers with, 16
real discrete Fourier transform (RDFT), 128
Walsh–Hadamard transform (WHT), 127
list scheduling, 53–4, *53*
literals, 101
logic synthesis, for architectural synthesis, 38
look up table (LUT), 150
loop invariant code motion, 28–9

Manchester adder, 77
MATLAB system design language, 39
method of squaring, 17
modified discrete cosine transform (MDCT), 129
MPEG compression, 13–14
multiple-constant multiplication (MCM) *see* multiplication
multiple-operand adder, 82–93
about multiple-operand adders, 82–3
carry propagate adders (CPAs), 83–5, *84*
carry save adders (CSAs), 86–8, *86*, *87*, *88*
counters and compressors, 89–90
definitions, 83
distributed arithmetic, 90–3, *92*
parallel carry propagation summation, 85, *85*
redundant digit summation, 86
sequential carry propagation summation, 83–5
signed/unsigned operands, 83
summation of *m* operands, 83, *84*
multiple-operand addition, synthesis for, 158–64
algorithm complexity, 163
with carry save adder (CSA) trees, 158–60, *159*
experimental results, 163–4: areas before/after optimization, **164**
iterative CSE algorithm, 162–3, *163*; three-term CAX algorithm, 162, *162*
linear transform in linear/matrix/polynomial form, 160, *160*
three-term divisor extraction algorithm, 160–2, *161*
multiple-variable multiplication, 137–9

© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE

Cambridge University Press & Assessment
978-0-521-88099-2 — Arithmetic Optimization Techniques for Hardware and Software Design
Ryan Kastner , Anup Hosangadi , Farzan Fallah
Index
More Information

multiplication
  multiple-constant multiplication (MCM), 133–9: with carry propagate adders (CPA), *138*, 139; CSE algorithms, 135–7; digit-based recording algorithms, 135; example with two constants, 134, *135*; with FIR filters, 133–4, *134*; graph-based algorithms, 137; multiple-variable problems, 137–9, *138*; with Synopsis Design Compiler Ultra^{TM}, 139, **139**
  single-constant multiplication, 130–3: canonical signed digits (CSD) with, 131–2; decomposition, 130; four-point DCT transformation, 132–3; not-polynomial (NP) complete issue, 130–1; optimal case, 132, *133*; shifting issues, 131
multipliers, with linear transforms, 16
multiply accumulate (MAC) operations
  with computer graphics, 11
  with the Horner form, 32

nonadjacent form, 71
not-polynomial (NP) complete, 130–1
number representation, 68–75
  binary numbers, 69
  binary signed digit representation, 71–3
  canonical/non-canonical systems, 70–1
  digital vectors, 69
  fixed point representations, 73–4
  floating point representations, 74–5
  negative numbers, 69
  number system properties, 68–71
  radix systems, 70
  redundant/nonredundant systems, 70
  weighted systems, 70

on-chip memory blocks, for hardware synthesis, 48
operation scheduling *see* hardware synthesis, operation scheduling
operator strength reduction, 30–1
optimization stage for compilation, 25
ordered matrix for canonical representation, 100–1

parallel carry propagation summation, 85, *85*
Parhami's generalized signed digit system, 72–3
partial-redundancy elimination (PRE), 29–30, *30*
physical synthesis, for architectural synthesis, 38
  floorplanning, 38
  placement, 38
  routing, 38
pipelined adder, 82, *82*
  flip flops (FFs) in the design, 82
pipelined/nonpipelined sequential functional units, 46–7
polynomial expressions/functions, 95–123
  about polynomials, 1, 2, 10, 18–19, 95–6
  address calculation, 17–18
  approximation algorithms, 9–10
  basic purpose/function, 95
  Bezier formulation, 95–6
  Buchberger algorithm, 98–9
  for computer graphics, 10–12, 95
  condense algorithm, 99
  cryptography, 16–17
  distill algorithm, 99
  for DSP, 12–16
  for function evaluation, 1–2
  kernelling algorithm, 99
  optimizing, 5–6
  problem formulation, 96
  simplification modulo set of polynomials, 98–9
  using the CSE algorithm, 97–8
  see also algebraicoptimization, Horner form/method, integer linear programming (ILP) solutions for reducing operations, kernels/co-kernels, rectangle covering methods of optimization
processing time issues, 2
program dependence graph (PDG) representation, 42–4, *43*
public key cryptography, 16–17

quartic spline polynomial optimization, 11–12

radix number systems, 70
real discrete Fourier transform (RDFT), 128
rectangle covering methods of optimization, 100–11
  for finding single-term subexpressions, 109–11, *110*: condense algorithm, 111; cube intersection matrix (CIM), 109–11, *110*, *111*; Distill algorithm, 111
  ordered matrix for canonical representation, 100–1
  rectangle covering algorithm, 101
  see also kernels/co-kernels
redundant digit summation, 86
registers for hardware synthesis, 48
  register transfer level (RTL), 4, 36, 44, *44*
Reitwiesner optimal algorithm, 71–2, *72*
resource allocation, for architectural synthesis, 37
resource binding
  for architectural synthesis, 37–8
  for hardware synthesis, 56, *57*
resource constrained scheduling (RCS), 49, 50–1
resource sharing, FIR filter case study, 61–2
ripple carry adder (RCA), 77, *78*
  speed/delay issues, 77

scanning (lexical analysis), 22
scheduling *see* hardware synthesis, operation scheduling
semantic checking for compilation, 23

© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE

Cambridge University Press & Assessment
978-0-521-88099-2 — Arithmetic Optimization Techniques for Hardware and Software Design
Ryan Kastner , Anup Hosangadi , Farzan Fallah
Index
More Information

sequential carry propagation summation, 83–5
signal processing, 2
single-constant multiplication *see* multiplication
software compilers/compilation, algebraic transformations in optimization, 25–33
  CSE procedure, 26–7, *26*
  dataflow optimization, 25–6
  drawbacks of conventional techniques, 33
  Horner form for evaluating polynomial approximations, 31–3
  loop invariant code motion, 28–9
  operator strength reduction, 30–1
  PRE operation, 29–30, *30*
  value numbering, 27–8
software compilers/compilation, structure, 21–5
  about compilers, 21, 33–4
  analysis stage, 24
  basic compilation steps, *22*
  benefits of compilers, 21
  code generation stage, 25
  intermediate code generation, 23–4
  lexical analysis (lexing/scanning), 22
  optimization stage, 25
  semantic checking, 23
  syntactic analysis, 22–3
software optimization, 174–8
  after constant expansion, 176–8: H.264's Luma interpolation equations, 177, *178*; VC-1's integer transform equations, 177, *177*
  without constant expansion, 175–6: 4×4 VC-1 video codec, *175*; H.264 video compression for Luma interpolation, 175–6, *176*; use of two-term CAX algorithm, 175
spline interpolation, 10–11
static single assignment (SSA) intermediate representation, 42, *42*
storage elements, hardware synthesis, 48
sum of products (SOP), 101
Synopsis Behavioral Compiler^{TM}, 115
Synopsis Design Compiler^{TM}, 115
Synopsis Design Compiler Ultra^{TM}, 139, **139**
syntactic analysis
  for compilation, 22–3
  for hardware synthesis, 37

Taylor expansion, 9–10
technological library example, **60**
three-term divisor extraction algorithm, 160–2
timing constrained scheduling (TCS), 49, 51, 55
timing/clock period issues for hardware synthesis, 46–7
transformation matrices, 127
trees
  abstract syntax tree (AST), 40
  adder trees, 61–2
  with carry save adder (CSA) trees, 158–60, *159*
  tree height reduction (THR), 165
two-operand addition, 75–82
  addition of two one-bit numbers, 76, *76*
  carry look-ahead adder (CLA), 79–82
  half and full adders, 76–7
  kill, generate, propagate unit (KGP), 77, *78*
  Manchester adder, 77
  pipelined adder, 82, *82*
  ripple carry adder (RCA), 77, *78*
  speed/delay issues, 77, 82
two's complement number representation, 74

unisolvence theorem, 9

value numbering, for redundancy elimination, 27–8
voltage scaling, 115

Walsh–Hadamard transform (WHT), 127
weighted number systems, 70

© in this web service Cambridge University Press & Assessment www.cambridge.org