ECE 598NS: Machine Learning in Silicon Fall 2017

Algorithm-to-Architecture Mapping Techniques

Naresh Shanbhag
Professor
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
• the goal of this lecture is to study algorithm-to-architecture mapping techniques for data-intensive algorithms:
  – data flow-graphs to describe algorithms
  – retiming
  – pipelining and parallelization
  – folding and unfolding
Algorithm Transforms

• There are a few different categories of algorithm transforms:
  – structural, functional, static, or dynamic transforms

• These operate on Data Flow Graphs (DFGs). Part of many automated CAD tool-flows for DSP architecture synthesis, e.g., Hyper-LP [anantha-CAD95]. Useful for manually generated architectures as well.

• In this chapter, we will study retiming, pipelining, parallel/block processing, unfolding, and folding transforms
Data-flow Graphs (DFGs)
Modeling Computation

- learning → repeated computations on different data
- input data $X_i$: $i$ is the sample index
  - For a time series: $i = n$ (time index)
- node: a memoryless computation or mapping
- arc/edge: communication between nodes
- $D$: storage (register) representing 1 sample delay; also called arc weight
Data Flow Graphs (DFG)

- a DFG is composed of a:
  - set of nodes $S_U$ : $S_U = \{A_{0-4}, M_{0-4}\}$
  - set of arcs/edges $S_e$ : $A_0 \rightarrow M_0$, $M_0 \rightarrow A_2 \in S_e$
• $d(U)$: deterministic **worst case delay** of node $U$
• $w(e)$: arc weight
• $w(e) = 0 \rightarrow$ signifies **intra-iteration precedence**
• $w(e) > 0 \rightarrow$ signifies **inter-iteration precedence**
Mapping DFG to Architecture

- each node mapped to a dedicated hardware unit
- each arc mapped to a dedicated interconnect
- referred to as a direct-mapped architecture
Iteration Period (IP) vs. Clock Period

- **IP**: smallest sample period (of the DFG) required to complete execution of one iteration
- **$T_{CLK}$**: smallest clock period (of the direct-mapped architecture) required to execute all computations correctly

- usually $IP = T_{CLK}$

- a DFG may be transformed into another functionally equivalent DFG $\rightarrow$ this can change the $IP$ and the $T_{CLK}$
- **$IPB$**: iteration period bound $\rightarrow$ lower bound on the $IP$
DFG Properties

• **path** \( p: U \rightarrow W \)
  - Sequence of nodes connected by arcs: \( A_0 \rightarrow M_0 \rightarrow A_2 \)
  - \( U \)=source node, \( W \)=destination node, \( w(p) \)=path weight, \( d(p) \)=path delay

\[
w(p) = \sum_{e_i \in p} w(e_i) \quad d(p) = \sum_{U_i \in p} d(U_i)
\]

• **loop**
  - path with identical source and destination nodes
  - E.g., \( A_0 \rightarrow M_0 \rightarrow A_2 \rightarrow A_0 \)
Acyclic DFG (aDFG)

- aDFG: DFG with non-zero weighted arcs removed
Critical Path in a DFG

- path with the maximum delay in the corresponding aDFG
  - $d(M) = 2$ a.u. (arbitrary units), $d(A) = 1$ a.u
  CP: $M_4 \rightarrow A_3 \rightarrow A_2 \rightarrow A_0 \rightarrow A_1$; $d(cp) = T_{CP} = d(M) + 4d(A) = 6$
- $IP = d(cp) = T_{CP}$: iteration period equals the critical path delay
Critical Path in a DFG

- $IP = d(cp) = T_{CP}$: iteration period equals the critical path delay
- $f_{CLK} = \frac{1}{T_{CLK}} = \frac{1}{T_{CP}}$: maximum clock frequency (throughput)

$T_{CP} = 6$
Iteration Period Bound

\[ IPB = \max \frac{\sum_U d(U)}{\sum_e w(e)} \]

- **IPB**: lower-bound on **IP** over all equivalent DFGs
- **IP = IPB** can be achieved in:
  - many-core implementations via *unfolding*
  - dedicated ASIC implementations via *retiming*
• assume : \( d(A) = 1 \); \( d(M) = 2 \) then

• critical path : \( M_4 \rightarrow A_4 \rightarrow A_2 \rightarrow A_0 \rightarrow A_1 \rightarrow \)

• \( IP = 2 + 1 + 1 + 1 + 1 + 1 = 6 \)

• \( IPB = \max \left[ \frac{4}{1}, \frac{5}{2}, \frac{5}{3} \right] = 4 \)

• no equivalent DFG can achieve an \( IP < 4 \)
Example: 3-Tap LMS Adaptive Filter

- Critical path delay:
  \[ T_{cp} = 2T_m + 4T_A \]
- Throughput = \( \frac{1}{T_{cp}} \)
- \( IPB = \max \left[ \frac{2T_m + 4T_A}{1}, \frac{T_A}{1} \right] = T_{cp} \)
- \( T_{cp} \) can be reduced via retiming, pipelining and parallelization

D: one sample delay (register)
Retiming
Retiming
[leiserson]

• relocate delays in DFG without changing input-output characteristics

• useful for:
  – reduces $IP$ without changing $IPB$
  – reducing power
  – reducing the number of registers and resources
  – improving scheduling on multi-core architectures
  – designing systolic (regular) architectures

• 2 types
  – cutset retiming
  – systolic retiming
Cutset Retiming

• 3 steps
  – identify a cutset (set of arcs that result in 2 disjoint DFGs when removed)
  – delay-scaling
  – delay transfer
Delay-Transfer

- transfer of $K$ delays from in-bound (out-bound) to out-bound (in-bound) arcs of a cutset
Cutset Retiming Example

- one delay transferred from lower arc of cutset to upper arc
- assume $T_M = 2\text{ns}$ and $T_A = 1\text{ns}$;
  - $T_{cp} = 4T_M + 4T_A = 12\text{ns}$ (original)
  - $T_{cp} = 2T_M + 3T_A = 7\text{ns}$ (retimed)
Delay Scaling

- replace all $D$ by $\alpha D$ ($\alpha > 1$, delay scaling factor)
- interleave input stream by $\alpha - 1$ zero/null or independent input streams
- use multichannel processing to avoid underutilizing H/W
Delay Scaling Example

• $IP$ is unaltered
Retiming Lemma

- assign integers $\text{lag}(U)$ to each node $U$ in the original DFG
- assign new weights to each arc $e(U \rightarrow V)$: (new DFG is generated)
  $$w_r(e) = w(e) + \text{lag}(V) - \text{lag}(U)$$
- Lemma guarantees: new (retimed) DFG is equivalent to the original one

Original DFG

Retimed DFG

$\text{lag}(A), \text{lag}(C), \text{lag}(V_h) = 0$;
$\text{lag}(B) = -1$
• a DFG with all arc weights $w(e) > 0$ is called systolic

• retiming can be used to systolize a DFG
• how to find the systolizing $lag()$ function?
Systolic Conversion Theorem

• given DFG $G$, construct a constraint graph $G_{-1}$ by reducing all arc weights in $G$ by 1. If $G_{-1}$ does not have any negative weight cycles, then a DFG $G$ can be systolized via retiming

• systolizing $\text{lag}(V) = \text{the smallest weight of any path from } V \text{ to } V_h \text{ in } G_{-1} \text{ where } V_h \text{ is a host node with } \text{lag}(V_h) = 0$, and all arcs entering/exiting $V_h$ have weight zero

$\text{lag}(A), \text{lag}(C), \text{lag}(V_h) = 0; \\
\text{lag}(B) = -1$
Pipelining and Parallelization
Latency vs. Throughput

Latency: time for the input to propagate to the output
Throughput: rate at which outputs are generated
In general: Throughput \neq 1/Latency

Latency = T_{A1} + T_{A2}
Throughput = \frac{1}{T_{A1} + T_{A2}}

Latency = T_{A1} + T_{A2} + \Delta
Throughput = \frac{1}{\max(T_{A1}, T_{A2})}

\Delta: timing overhead or pipelining/parallelization

Latency = T_{A1} + T_{A2} + \Delta
Throughput = \frac{2}{T_{A1} + T_{A2}}
Pipelining Non-recursive DFGs

- Assume ideal registers → zero set-up, hold, clk-to-Q times
- Assume timing constraints met: $T_{fast} \geq 0$ and $T_{cp} \geq T_{CLK}$
- Maximum serial architecture clock frequency:

$$f_{CLK, serial} = \frac{1}{T_{cp, serial}}$$
• maximum serial architecture clock frequency:

\[ f_{CLK,serial} = \frac{1}{T_{cp,serial}} \]

• pipelined critical path delay:

\[ T_{cp,pipe} = \max\{T_{cp,A1}, T_{cp,A2}\} \leq T_{cp,serial} \]

• for uniform pipelining \( T_{cp,A1} = T_{cp,A2} = 0.5T_{cp} \)
  
  – 2x speedup
• place $M - 1$ registers at a feed-forward (FF) cutset
• a FF cutset has all arcs pointing in the same direction
• speed-up compared to serial architecture=$M$
  – $M \times$ faster
• watch out for fast path constraint violation
Example - Pipelining an FIR Filter

- Assume \( d(M) = 3\text{ns}, d(A) = 1\text{ns} \)
- \( T_{cp,\text{serial}} = d(M) + 2d(A) = 5\text{ns} \)
- \( T_{cp,\text{pipe}} = \max(d(M), 2d(A)) = 3\text{ns} \)
- \( \left(\frac{5}{3}\right) \times \text{speed-up} \)
Uniform Pipelining

- pipeline with equal delay stages
- split multiplier: \( d(M_1) = d(M_2) \) + 2\( d(A) \) = 2.5 ns
- speed-up=2×
- practical speed-up < 2× due to non-zero register delay
Pipelining DFGs with Loops

- \( IPB \) dominated by slowest loop
- no FF cutset exists
- need to introduce delays in loops to reduce \( IPB \) without altering functionality

\[
Y_i = A(X_i, Y_{i-1})
\]

\[
Y_{i-2} = \begin{cases} \text{D} \\ 2D \end{cases}
\]

\[
Y_i = A(X_i, Y_{i-2})
\]
Look-ahead Pipelining

- $y[n] = b_0 x[n] + a_1 y[n - 1] + a_2 y[n - 2]$
- back substitute for $y[n - 1]$ in terms of $y[n - 2], y[n - 3]$
  
  $$y[n] = b_0 x[n] + a_1 [b_0 x[n - 1] + a_1 y[n - 2] + a_2 y[n - 3]] + a_2 y[n - 2]$$
  
  $$= b_0 x[n] + a_1 b_0 x[n - 1] + (a_1^2 + a_2) y[n - 2] + a_1 a_2 y[n - 3]$$

- FF section (overhead) can be cutset pipelined
Pipelining the LMS Algorithm

- also has loops
- difficult to apply look-ahead directly (try it!) → need to relax the requirements of functional invariance → relaxed look-ahead

\[ e[n] = d[n] - \mathbf{W}^T[n] \mathbf{X}[n] \]

\[ \mathbf{W}[n + 1] = \mathbf{W}[n] + \mu e[n] \mathbf{X}[n] \]
\[ e[n] = d[n] - W^T [n - M_1]X[n] \]

\[ w[n + M_2] = w[n] + \mu \sum_{i=0}^{M_3-1} e^*[n - M_2 + 1 + i]x[n - M_2 + 1 + i] \]

- \( M_1 (M_2) \) delays to pipeline outer (inner) loop
- convergence behavior is altered slightly (hence relaxed look-ahead)
Block/Parallel Processing

- improves throughput
  - unlike pipelining, does so without relying on high frequency clocks
  - useful for high-sample rate applications, e.g., optical (10+ Gb/s), chip-to-chip signaling
- reduces \( IPB \)
Block Architecture

\( k \): block index; \( L \): block length

- input block \( \mathbf{x}[k] = [x[Lk], x[Lk + 1], \ldots, x[Lk + L - 1]]^T \)
- output block \( \mathbf{y}[k] = [y[Lk], y[Lk + 1], \ldots, y[Lk + L - 1]]^T \)
- \( L \)-slow processor: block frequency \( f_b = \frac{f_s}{L} \), i.e., \( L \)-times slower than sample frequency \( f_s \)
- \( L \)-times throughput increase without increasing clock frequency
in a block architecture, all block delay elements are $L$-slow
Example: L=2, N=4, Block FIR Filter

• 4-tap FIR serial filter:
  \[ y[n] = h_0 x[n] + h_1 x[n-1] + h_2 x[n-2] + h_3 x[n-3] \]

• \( L = 2 \), substitute \( n \) with \( 2k \) and \( 2k + 1 \):
  \[
  y[2k] = h_0 x[2k] + h_1 x[2k-1] + h_2 x[2k-2] + h_3 x[2k-3] \\
  y[2k+1] = h_0 x[2k+1] + h_1 x[2k] + h_2 x[2k-1] + h_3 x[2k-2]
  \]

• \( LN \) MACs needed without computation sharing
Block Processing for DFGs with Loops

- A block IIR filter requires $y[k]$ depend only on $y[k - 1]$ and its delayed versions

$$L = 3; \quad N = 1$$

$$y[3k + 2] = f(x[3k + 2], y[3k - 1])$$
$$y[3k + 1] = f(x[3k + 1], y[3k - 2])$$
$$y[3k] = f(x[3k], y[3k - 3])$$

- How to get the function $f()$?
• use \((L = 3\text{-step})\) look-ahead to obtain \(f()\)

\[
\]

• Substitute \(n = 3k + 2, 3k + 1,\) and \(3k \rightarrow 3\)-parallel IIR filter:

\[
\begin{align*}
\end{align*}
\]
• $T_{cp}$ is identical (after pipelining FF section) to that of a serial architecture, i.e., $T_{cp} = T_M + T_A$

• However, $L$ samples processed per $T_{cp}$ seconds → $L$ -fold speed-up is achieved
• processes block of data $x[k]$ of length $L$ to generate a block of output $\hat{d}[k]$

• weights adjusted once per block → batch-mode processing

• equivalent to serial LMS when $L = 1$
\[e[k] = d[k] - \chi[k]W[k]\]

\[W[k + 1] = W[k] + \frac{\mu}{L} \sum_{i=0:L-1} e[kL - i] x_i[k]\]

- \(W[k] = [w_0[k], ..., w_{N-1}[k]]^T\): same length as serial LMS; updated once per block of \(L\) samples
- update term is \(L\)-times larger hence step-size \(\mu\) (same as in serial LMS) is reduced by a factor of \(L\)
\[ e[k] = d[k] - \chi[k]W[k] \]
\[ W[k + 1] = W[k] + \mu \frac{\mathcal{L}}{L} \sum_{i=0:L-1} e[kL - i]x_i[k] \]

\[ \chi_1 = \begin{bmatrix} x_1 & 0 & 0 \\ x_2 & x_1 & 0 \\ x_3 & x_2 & x_1 \\ \vdots & \vdots & \vdots \end{bmatrix}, \quad \chi_2 = \begin{bmatrix} x_4 & x_3 & x_2 \\ x_5 & x_4 & x_3 \\ \vdots & \vdots & \vdots \end{bmatrix} \]

\[ Y_1 = \hat{d}[k] \]

- \( W[k] = [w_0[k], ..., w_{N-1}[k]]^T \)
- \( \chi[k] = [x_0[k], x_1[k], ..., x_{L-1}[k]]^T \)
- \( x_i[k] = [x[kL - i], x[kL - i - 1], ..., x[kL - i - N + 1]]^T \)
Convergence Properties

- optimum Wiener-Hopf equation identical to serial LMS: $w_{opt} = R^{-1}P$
- stability bounds identical to serial LMS: $0 < \mu < \frac{2}{N\sigma_X^2}$
- convergence speed $\rightarrow L$-times slower due to infrequent updates
- $L$ times better accuracy (misadjustment) than serial LMS

$$\eta = \frac{J(\infty) - J_{\text{min}}}{J_{\text{min}}} = \frac{\mu}{2L} \text{tr}(R)$$
Pipelining & Block Processing for Low-Power

Low-Power CMOS Digital Design
Anantha P. Chandrakasan, Samuel Sheng, and Robert W. Brodersen, Fellow, IEEE

- basic idea: trade-off throughput increase from pipelining with power via supply voltage $V_{dd}$ reduction
- what is the relationship between $V_{dd}$, throughput, power, energy? (next lecture)
Folding and Unfolding
Unfolding

• also known as loop unrolling
• a one-to-one transform
• exposes inter-iteration precedence in a DFG
  – Unfolding by a factor $J$ exposes $J$ iterations
• benefits
  – generation of rate-optimal multi-core/processor schedules
  – systematic design of digit-serial architectures from bit-serial architecture
  – circuit/logic level power and speed optimization
• does not reduce $IPB$
Unfolding Example

- second order IIR filter: \( y[n] = b_0 x[n] + a_1 y[n-1] + a_2 y[n-2] \)
- substitute \( n = 2k \) and \( n = 2k + 1 \) (\( J = 2 \))

\[
\begin{align*}
y[2k] &= b_0 x[2k] + a_1 y[2k-1] + a_2 y[2k-2] \\
y[2k+1] &= b_0 x[2k+1] + a_1 y[2k] + a_2 y[2k-1]
\end{align*}
\]
Unfolding Example (J=2)

\[ y[2k] = b_0 x[2k] + a_1 y[2k - 1] + a_2 y[2k - 2] \]
\[ y[2k + 1] = b_0 x[2k + 1] + a_1 y[2k] + a_2 y[2k - 1] \]
Unfolding and Block Processing

Unfolding and block processing are identical for non-recursive DFGs. For recursive DFGs, unfolding computes current state from immediate past state while block processing uses look-ahead.

- Unfolding can achieve $IP = IPB$ without fine-grain pipelining.
Unfolding Property 1

- unfolding preserves the number of delays, and leads to a $J$-fold increase in the number of nodes
Unfolding Property 2

- $J$-fold unfolding of a loop with $N_D$ delays and $N_n$ nodes results in $N_l = \text{GCD}(N_D, J)$ loops each with $N_D/N_l$ delays and $JN_n/N_l$ nodes.
- $J$-fold unfolding increases $T_{cp}$ by a factor of $J$.
Unfolding Property 3

- an arc/path $P$ with weight $w(P) \geq J$ in the original DFG, will result in $J$ paths with one or more delays in the $J$-fold unfolded DFG
Unfolding Property 4

- If $J$ is a common factor of all arc weights in a DFG, then the $J$-fold unfolded DFG will have $J$ decoupled sub-DFGs that are topologically identical to the original DFG with all arc weights scaled down by $J$.

suitable for many-core implementation
Folding

- Maps algorithmic operations to hardware
  - Scheduling algorithmic operations onto hardware cycles, and binding them to hardware units
  - Algorithmic DFG is different from hardware DFG (hDFG)
- Folding reduces area. It is a one-to-many mapping (unlike folding). Also referred to as multiplexing or shared-resource architecture.
- Folding by a factor $J$ followed by unfolding by a factor $J$ leads to a retimed version of original DFG
Folding a 4-tap FIR Filter

- If $T_m = 7\, ns$, $T_a = 3\, ns$, $T_s = 40\, ns$, and $T_{cp} = 16\, ns$
- $J = 4$ folded architecture
  - 1 MAC unit, data, and coefficient registers
Folding a 4-tap FIR Filter: Timing Diagram

- $T_{cp} = 10\text{ns}$
- folding factor of 4 matches the hardware speed of the application assuming $T_s = 40\text{ns}$ and $T_{CLK} = 10\text{ns}$
References

https://courses.engr.illinois.edu/ece598ns/fa2017

http://shanbhag.ece.uiuc.edu