CPRE/COMS 583x Adaptive Computing Systems	Fall Semester 1998
November 19, 1998 Lecture 26	
Instructor: Arun K. Somani & Akhilesh Tyagi	Scribe: Jim-Eng NG

Configuration scheduling schemes for striped FPGAs

Field Programmable Gate Arrays (FPGAs) are very commonly used in configurable computing machines (CCMs) as hardware accelerators, in order to perform a better mean of computation. The main purpose of a CCM is to behave as a custom hardware where an application specific circuit can be created to solve a certain problem. Such applications are like encryption-decryption, DSP, image processing, DNA sequence matching, and so on. Second, CCM also increase the flexibility of programmable processor that determine which problem to be solved. Finally CCMs are used for post fabrication application development in order to test the piece of hardware, that will be fabricated, is bug-free.

The basic architecture: of a CCM is shown as in Figure 1. The host processor is a controlling unit that justified which problem to be solved on the hardware and thus configures the FPGA accordingly. The reconfigurable array is a FPGA like XC4000, Altera Flex, etc., and both of these components require some interfacing with memory.

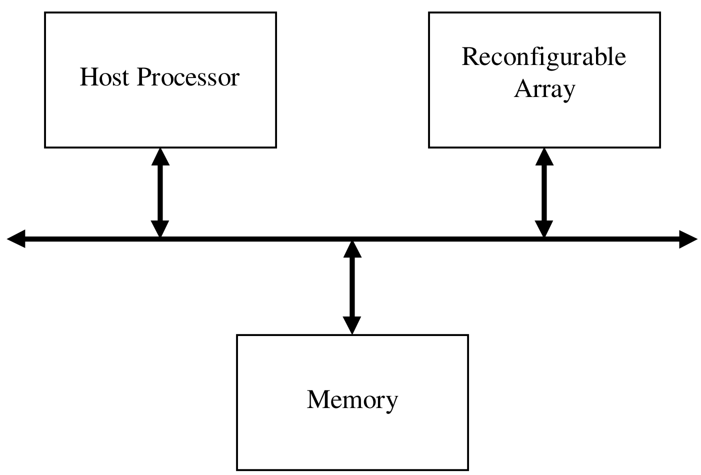

Figure 1: Basic Architecture of CCM

The following table shows the performance comparisons of CCMs to listed general-purpose processor.

Application	CCM/FPGA	Processor Compare	Speedup
DNA Matching	SPLASH-2	SPARC 10	4300
RSA Encryption	PAM	Alpha 150MHz	17.8
FIR Filter	Xilinx FPGA	DSP 50MHz	17.9
Ray Casting	RIPP-10	Pentium 75Mhz	33.8
Stereo Vision	PARTS	RISC 400 MHz	~30
Hidden Markov Model	Xilinx FPGA	SPARC 10	24.4

Table 1: Performance comparison of CCM

These performance gains are partly determined by reconfiguration time and reconfiguration granularity of the FPGA. Traditional FPGAs like XC4000 and Altera Flex do not allow partial reconfiguration, and they have exclusive execution and configuration modes. This results them to yield a long reconfiguration time, and thus not suitable for run-time reconfiguration (RTR). The new generation of FPGAs from Xilinx, Garp, Colt, PipeRench, etc., improved and now, they allow simultaneous reconfiguration, which in turn makes rapid RTR possible.

PipeRench architecture consists of a fabric, which is a set of hardware stripes connected in a pipeline fashion. It allows incremental pipeline reconfiguration, and the basic unit of reconfiguration is a stripe. The stripe configuration time is the same as the execution time, and thus suitable for fast run-time reconfiguration. The features of PipeRench architecture are as follow:

- Concurrent configuration and execution
- Hardware virtualization
- A wide global configuration bus (configuration load takes on execution cycle)
- Large configuration cache
- Suitable for pipelined applicatoins

Let k be the number of stripes in the FPGA fabric. Let *S* be the number of pipeline stages required in an application, and *X* is the number of data elements to be processed by the pipeline. When $S > k$, the application will be scheduled temporally on the FPGA as defined by its corresponding reconfiguration pattern. Three different kind of scheduling schemes are introduced here.

1. Configuration Caching

Figure 2 shows the execution cycle of configuration caching with 6 pipeline stages, 6 data elements, on a 3-stripes FPGA.

CONF f1	f1(x1)	f1(x2)	CONF f4	f4(x1)	f4(x2)	CONF f1	f1(x3)	f1(x4)	CONF f4
	CONF f2	f2(x1)	f2(x2)	CONF f5	f5(x1)	f5(x2)	CONF f2	f2(x3)	f2(x4)
		CONF f3	f3(x1)	f3(x2)	CONF f6	f6(x1)	f6(x2)	CONF f3	f3(x3)

f4(x3)	f4(x4)	CONF f1	f1(x5)	f1(x6)	CONF f4	f4(x5)	f4(x6)		
CONF f5	f5(x3)	f5(x4)	CONF f2	f2(x5)	f2(x6)	CONF f5	f5(x5)	f5(x6)	
f3(x4)	CONF f6	f6(x3)	f6(x4)	CONF f3	f3(x5)	f3(x6)	CONF f6	f6(x5)	f6(x6)

Figure 2: Execution cycle using Configuration Caching approach

The execution of this caching approach has the following features:

- All configurations are cached
- One stripe is configured every cycle
- Life of one stripe configuration = $k - 1$, for $k < S$
- Global I/O bus to feed data to the first stage
- Global bus connecting first and last stripe
- No additional storage for intermediate results
- Latency independent of total data elements

2. Data Caching

Figure 3 shows the execution cycle of data caching with the same parameters used in configuration caching.

Pipeline stages, $S=6$ Number of stripes, $k=3$
Number of data elements, $X=6$

CONF f1	f1(x1)	f1(x2)	f1(x3)	f1(x4)	f1(x5)	f1(x6)	CONF f4	f4(x1)	f4(x2)
	CONF f2	f2(x1)	f2(x2)	f2(x3)	f2(x4)	f2(x5)	f2(x6)	CONF f5	f5(x1)
		CONF f3	f3(x1)	f3(x2)	f3(x3)	f3(x4)	f3(x5)	f3(x6)	CONF f6

f4(x3)	f4(x4)	f4(x5)	f4(x6)		
f5(x2)	f5(x3)	f5(x4)	f5(x5)	f5(x6)	
f6(x1)	f6(x2)	f6(x3)	f6(x4)	f6(x5)	f6(x6)

Figure 3: Execution cycle using Data Caching approach

The execution of this caching approach has the following features:

- Cache data and intermediate results
- One bubble every *(X + 1)* cycles
- Life of a stripe = *X* cycles
- Better resource utilization
- Latency proportional to *X*
- Execution time possibly smaller

3. Hybrid Caching

Hybrid caching is a combined approach to schedule data and configurations through the use of FPGA fabric. Figure 4 shows the execution cycle of hybrid caching with the same parameters used in previous two cases. The major difference to the previous schemes is that hybrid caching is block-oriented. At anytime, a block with B data elements will get executed before any reconfiguration takes place in the pipeline.

Pipeline stages, $S=6$ Number of elements in a block, $B=3$
Number of data elements, $X=6$ Number of stripes, $k=3$

CONF f1	f1(x1)	f1(x2)	f1(x3)	CONF f4	f4(x1)	f4(x2)	f4(x3)	CONF f1	f1(x4)
	CONF f2	f2(x1)	f2(x2)	f2(x3)	CONF f5	f5(x1)	f5(x2)	f5(x3)	CONF f2
		CONF f3	f3(x1)	f3(x2)	f3(x3)	CONF f6	f6(x1)	f6(x2)	f6(x3)

f1(x5)	f1(x6)	CONF f4	f4(x4)	f4(x5)	f4(x6)		
f2(x4)	f2(x5)	f2(x6)	CONF f5	f5(x4)	f5(x5)	f5(x6)	
CONF f3	f3(x4)	f3(x5)	f3(x6)	CONF f6	f6(x4)	f6(x5)	f6(x6)

Figure 4: Execution cycle using Hybrid Caching approach

The execution of this hybrid caching approach has the following characteristics:

- Combination of previous two schemes
- I/O traffic reduction for data and configurations
- No need to wait for all data elements
- Latency independent of total data elements
- Reduction in number of data stalls
- Cache *C* configurations and *2B* data elements

$$M = CW_c + 2BW_d$$

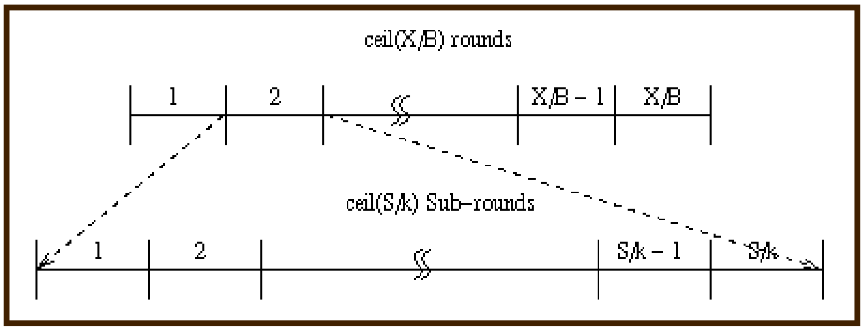

Figure 5: Rounds and Sub-rounds in Hybrid Caching

Round: data caching for B data elements

- Number of $R_h = \left\lceil \frac{X}{B} \right\rceil$ rounds,

Sub-round: Similar to the round of data caching

- Number of sub-rounds in a $R_{sub} = \left\lceil \frac{S}{k} \right\rceil$ round,

Parameters

In modeling the execution time for each of the schemes above, certain assumptions are made. Parameters used throughout the paper are shown in Table 1. The list of assumptions made is listed next.

k	Number of stripes in the fabric
W_c	Configuration word size in bytes
n_c	Number of IO cycles to fetch one configuration
W_d	Data element size in bytes
n_d	Number of IO cycles to fetch one data element
M	Total on-chip memory in bytes
$C_{max}=M/W_c$	Maximum number of configurations that can be cached
$X_{max}=M/W_d$	Maximum number of data elements that can be cached
S	Number of pipeline stages in the application
N	Number of distinct configurations representing all S pipeline stages
X	Number of data elements processed by the application
$f(n)$	Coverage function

Table 2: Architectural and Application Parameters

Assumptions

1. $S > k$
2. No stalls due to data output
3. Prefetch buffers for k configurations and $k - 1$ data elements
4. Prefetch buffer and cache are dual ported
5. Data read from cache + processing in 1 ex. Cycle
6. Configuration loads from cache in one clock cycle
7. Linear pipeline applications
8. Pipeline stages are distinct
9. Cached configurations required uniformly in the set of k
10. Intermediate and output data have the same size as the input, e.g. IDEA, DES encryption

The coverage functions $f(n)$ denotes the number of pipeline stages represented by the most frequently used n distinct configurations. It is a cumulative function that has the following properties.

- $n = 1$, the most frequently used configuration and
- $n =$ N, all distinct configurations

Consider $f(N) = S$; when $N = S, f(n) = n$; when $N < S, f(n) > n$. Therefore the function appears to be convex for $N < S$ and linear for $N = S$. Worst case scenario will be a linear function.

By caching C configurations, the configuration hit ratio, h, is determine as follows:

$$N = S, h = \frac{C}{S}$$

$$N < S, h = \frac{f(C)}{S}$$

Therefore, for configuration caching,

$$h = \frac{\min\{S, f(C_{\max})\}}{S}$$

For data caching,

$$h = 0$$

For hybrid caching,

$$h = \frac{f(C)}{S}$$

Since assumption is made that all data elements and configurations are available without any stalls, I/O overhead is not considered in deriving expressions for the execution cycles.

Execution Cycles Analysis

1. Stalls for Configuration Caching

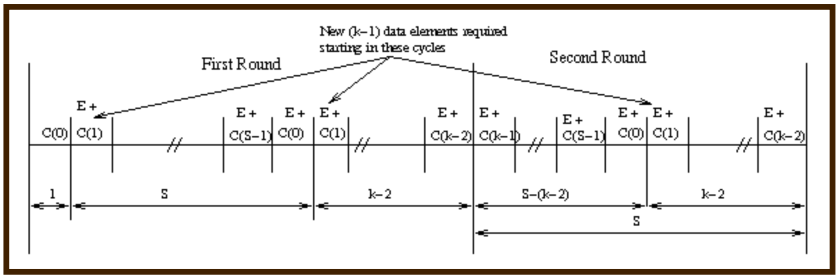

Figure 6: Configuration Caching Execution Cycles without I/O stalls

A round is considered as one sweep of an application

Number of Rounds, $R_c = \left\lceil \frac{X}{k-1} \right\rceil$

Round 1, no of cycles: $S + k - 1$

Round 2 to $(R_c - 1)$: $S-(k-2)+(k-2)=S$

Last Rd.: $S-(k-2)+X-(k-1)\left\lfloor \frac{X}{k-1} \right\rfloor - 1 = S + X - (k-1)\left\lceil \frac{X}{k-1} \right\rceil$

$$EX_c = k-1+X+(S-k+1)\left\lceil \frac{X}{k-1} \right\rceil$$

2. Stalls for Data Caching

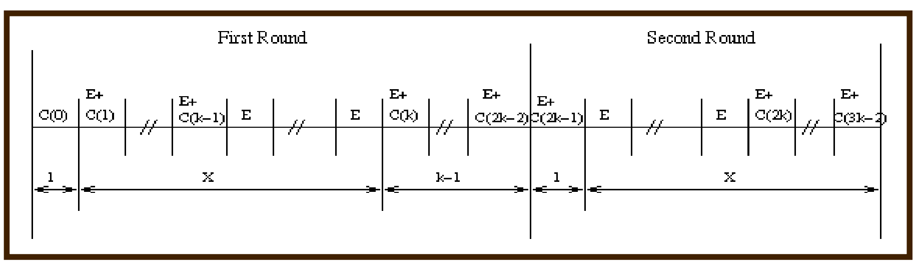

Figure 7: Data Caching Execution Cycles without I/O stalls

A round is considered as a sweep of all data elements through k stages

Number of rounds, $R_d = \left\lceil \frac{S}{k} \right\rceil$

Round 1: $k + X$

Round 2 to $(R_d - 1)$: $X + 1$

Last Round: $X+1-\left[k-\left(S-k\left\lfloor \frac{S}{k} \right\rfloor\right)\right] = X+S+1-k\left\lceil \frac{S}{k} \right\rceil$

$$EX_d = k - 1 + S + \left\lceil \frac{S}{k} \right\rceil (X - k + 1)$$

With some approximations, $EX_d < EX_c$ when $X > k - 1$

3. Stalls for Hybrid Caching

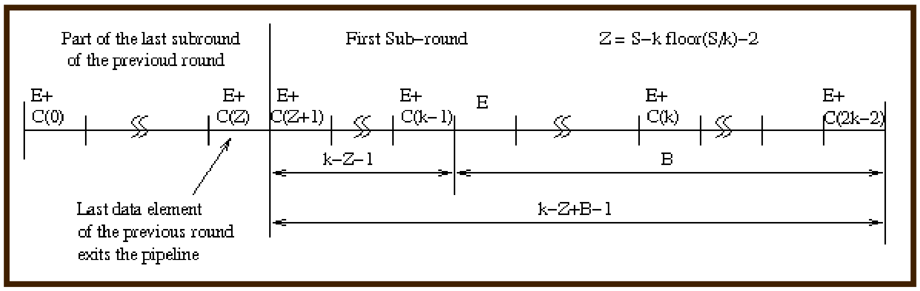

Figure 8: Hybrid Caching Execution Cycles without I/O stalls

Round 1: Similar to data caching round 1

- No. of cycles $k - 1 + S + (B - k + 1)\left\lceil \frac{S}{k} \right\rceil$:

Round 2 to $(R_h\text{-}1)$:

- Sub-round 1: $k - \left(S - k\left\lfloor \frac{S}{k} \right\rfloor - 2 \right) + B - 1 = k\left\lceil \frac{S}{k} \right\rceil - S + B + 1$

- Sub-rounds 2 to $(R_{sub}$ - $1)$: $B + 1$

- Last sub-round: $B + 1 - \left(k - S + k\left\lfloor \frac{S}{k} \right\rfloor \right) = B + S + 1 - k\left\lceil \frac{S}{k} \right\rceil$

$$EX_{round} = \left\lceil \frac{S}{k} \right\rceil (B + 1)$$

Last Round: $\left\lceil \frac{S}{k} \right\rceil \left(X - B\left\lfloor \frac{X}{B} \right\rfloor + 1 \right)$

Total, $EX_h = k - 1 + S + \left\lceil \frac{S}{k} \right\rceil \left(X + \left\lceil \frac{X}{B} \right\rceil - k \right)$

$EX_h > EX_d$ when $\left\lceil \frac{X}{B} \right\rceil > 1$

$\left\lceil \frac{X}{B} \right\rceil$ determines the performance

Stalls Computation

Total number of execution cycles is obtained by adding stalls to the number of execution cycles without I/O overhead. The stalls relationship is very much dependent on $(C_{max} + k)$, which is what the prefetch buffer can store.

1. Configuration Caching

Case 1: $S \le (C_{max} + k)$

Round 1: S configurations and $2(k-1)$ data elements

$$W_{cr1} = Sn_c + (k-1)n_d - (S + k - 2)$$

Round 2 to $(R_c - 1)$: $(k-1)$ data elements

$$W_{cr2} = \max\{0, (k-1)n_d - S\}$$

Total stalls,

$$W_{ct1} = W_{cr1} + W_{cr2}\left(\left\lceil \frac{X}{k-1} \right\rceil - 2\right)$$

Total execution cycles,

$$T_{c1} = EX_c + W_{ct1}$$

Case 2: $S > (C_{max} + k)$

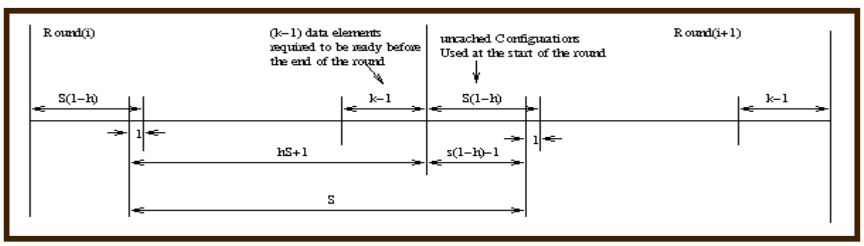

Figure 9: Configuration Caching Stalls analysis for $S > (C_{max} + k)$

Uncached configurations: $k - 1$ to $k - 2 + S(1 - h)$ stages

Round 1: W_{cr1}

Round 2: $W'_{cr2} = W'_{cr2c} + W'_{cr2d}$

- Configurations stalls: $W'_{cr2c} = S(1-h)(n_c - 1) + 1$
- Data stalls: $W'_{cr2d} = \max\{0, (k-1)n_d - hS - 1\}$

Round 3 to $(R_c - 1)$:

- Data stalls $= W'_{cr2d}$
- Configurations prefetch possible if $W'_{cr2d} = 0$
- No. of configuration that can be prefetched,

$$\beta = \min\left\{k, \frac{hS + 1 - (k-1)n_d}{n_c}\right\}$$

- Configuration stalls, $W'_{cr3c} = \max\{0, [S(1-h) - \beta]n_c - S(1-h) + 1\}$

Last round: only configuration stalls

Total Stalls, $W_{ct2} = W_{cr1} + W'_{cr2} + (W'_{cr2c} + W'_{cr2d})\left(\left\lceil \frac{X}{k-1} \right\rceil - 3\right) + W'_{cr2c}$

2. Data Caching

Case 1: $X < (X_{max} + k)$

Round 1: *2k* configurations and *X* data elements

$$W_{dr1} = 2kn_c + Xn_d - (X + k - 1)$$

Round 2 to $(R_d - 1)$: *k* configurations

$$W_{dr2} = \max\{0, kn_c - (X + 1)\}$$

Total stalls,

$$W_{dt1} = W_{dr1} + W_{dr2}\left(\left\lceil \frac{S}{k} \right\rceil - 2\right)$$

Total execution cycles,

$$T_{d1} = EX_d + W_{dt1}$$

Case 2: $X \geq (X_{max} + k)$

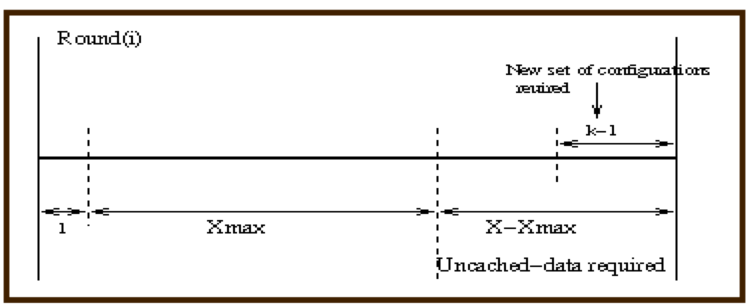

Figure 10: Data Caching Stalls analysis for $X \geq (X_{max} + k)$

Uncached data: $X_{max} - (X - 1)$
Round 1: W_{dr1}
Round 2 to $(R_d - 1)$:

$$\beta_d = \min\left\{k - 1, \frac{X_{\max} + 1}{n_d}\right\}$$

$$\beta_c = \min\left\{k, \frac{\max\{0, X_{\max} + 1 - \beta_d n_d\}}{n_c}\right\}$$

Round 2 to $(R_d - 1)$ stalls: $W'_{dr2} = \max\{0, (k-\beta_c)n_c + (X - X_{\max} - \beta_d)n_d - (X - X_{\max})\}$

Last Round: $W'_{dl} = \max\{0, (X - X_{\max} - \beta_d)n_d - (X - X_{\max})\}$

Total Stalls, $W_{dt2} = W_{dr1} + W'_{dr2}\left(\left\lceil \frac{S}{k} \right\rceil - 2\right) + W'_{dl}$

Total execution cycles,

$$T_{d2} = EX_d + W_{dt2}$$

3. Hybrid Caching

Compute Stalls for 4 rounds:

- Round 1
- Round 2, when $R_h > 2$
- Round 3 and onwards
- Last Round

Round 1: Similar to Data Caching stalls,

$$W_{hr1} = W_{r1s1} + W_{r1s2}\left(\left\lceil \frac{S}{k} \right\rceil - 2\right)$$

$$W_{r1s1} = 2kn_c + Bn_d + (B + k - 1)$$

$$W_{r1s2} = \max\{0, kn_c - (B+1)\}$$

Round 2:

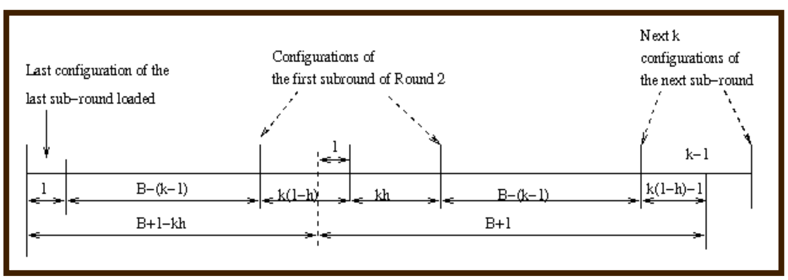

Figure 11: Hybrid Caching Stalls analysis for Round 2

Uncached configurations of the first sub-round $W_{r2s1c} = \max\{0, k(1-h)n_c - (B + 1 - kh)\}$

Prefetched data elements,

$$\beta_{r2d} = \min\left\{B, \frac{\max\left\{0, B + 1 - kh - k(1-h)n_c + \left(\left\lceil \frac{S}{k} \right\rceil - 2\right)\max\{0, B + 1 - kn_c\}\right\}}{n_d}\right\}$$

1st sub-round, stalls $W_{r2s1} = W_{r2s1c} + \max\{0, (B - \beta_{r2d})n_d + k(1-h)n_c - (B+1)\}$

Remaining sub-rounds stalls, $W_{r2s2} = \max\{0, k(1-h)n_c - (B+1)\}$

Round 2 stalls, $W_{hr2} = W_{r2s1} + W_{r2s2}\left(\left\lceil \frac{S}{k} \right\rceil - 2\right)$

Round 3 and onwards

- $R_h > 3$
- Prefetched data, $\beta_{r3d} = \min\left\{B, \frac{\max\{0, (B+1-k(1-h)n_c)\left(\left\lceil \frac{S}{k} \right\rceil - 1\right)}{n_d}\right\}$

- Stalls in first sub-round,

$$W_{r3s1} = \max\{0, (B - \beta_{r3d})n_d + k(1-h)n_c - (B+1)\}$$

- Total of a round,

$$W_{hr3} = W_{r3s1} + W_{r2s2}\left(\left\lceil \frac{S}{k} \right\rceil - 2\right)$$

Last Round:

Case I: $R_h = 2$

- Data prefetched,

$$\beta_{l1d} = \min\left\{X - B\left\lfloor \frac{X}{B} \right\rfloor, \frac{\max\{0, B+1-kh-k(1-h)n_c\} + \left(\left\lceil \frac{S}{k} \right\rceil - 2\right)\max\{0, B+1-kn_c\}}{n_d}\right\}$$

- First sub-round stalls,

$$W_{l1s1} = W_{r2s1c} + \max\left\{0, \left(X - B\left\lfloor \frac{X}{B} \right\rfloor - \beta_{l1d}\right)n_d + k(1-h)n_c - \left(X - B\left\lfloor \frac{X}{B} \right\rfloor + 1\right)\right\}$$

- Other sub-rounds,

$$W_{l1s1} = \max\left\{0, k(1-h)n_c - \left(X - B\left\lfloor \frac{X}{B} \right\rfloor + 1\right)\right\}$$

- Last round, $W_{l1} = W_{l1s1} + W_{l1s2}\left(\left\lceil \frac{S}{k} \right\rceil - 2\right)$
- Total, $W_{ht1} = W_{hr1} + W_{l1}$

Case II: $R_h > 2$

- Data prefetched

$$\beta_{l2d} = \min\left\{X - B\left\lfloor \frac{X}{B} \right\rfloor, \frac{\max\left(0, (B+1-k(1-h)n_c)\left(\left\lceil \frac{S}{k} \right\rceil - 1\right)\right)}{n_d}\right\}$$

- First sub-round stalls,

$$W_{l2s1} = \max\left\{0, \left(X - B\left\lfloor \frac{X}{B} \right\rfloor - \beta_{l2d}\right)n_d + k(1-h)n_c - \left(X - B\left\lfloor \frac{X}{B} \right\rfloor + 1\right)\right\}$$

- Other sub-rounds, W_{l1s2}

- Last round, $W_{l2} = W_{l2s1} + W_{l1s2}\left(\left\lceil \frac{S}{k} \right\rceil - 2\right)$

- Total, $W_{ht2} = W_{hr1} + W_{hr2} + W_{hr3}\left(\left\lceil \frac{X}{B} \right\rceil - 3\right) + W_{l2}$

Summary

Three configuration scheduling schemes, configuration caching, data caching, and hybrid caching, are evaluated in the paper, based on the performance by computing their total execution cycles taking into account the effect of I/O with the external memory.

In general, when the number of pipeline stages of an application is less than the number of stripes in the FPGA fabric, all the three performance are the same. When the number of pipeline stages go more than the number of stripes, the performance issue is determined by the scheduling scheme used. Configuration caching configures the pipeline stage every S cycles for every set of $(k - 1)$ data elements. Data caching configures only once for all data elements. Hybrid caching lies in between, where it configures the same pipeline stage for every block of B $(k - 1 < B < X)$ data elements.

Latency issue in configuration caching is minimal, as long as all the configurations corresponding to the application are present in the cache. Data caching latency depends on the number of data elements processed by the application. Hybrid caching can adapt to the required latency by selecting an appropriate value for the size of the block. Performance improvement of hybrid caching is 1.1 – 1.3 times over data caching, and 1.1 – 1.5 times, sometimes even 6 times that of configuration caching.

The selection of which scheduling scheme depends on the requirement on smaller latency or smaller total execution time. Configuration caching will provide a fare result in latency issue, as long as all configurations can fit in the cache. For the latter requirement, hybrid caching will provide the good result.