Dynafuse: Dynamic Dependence Analysis for FPGA Pipeline Fusion and Locality Optimizations

Jeremy Fowers, Greg Stitt
University of Florida
Department of Electrical and Computer Engineering

Dynafuse Optimization

- High-level synthesis (HLS) useful for FPGA productivity
- Generally unaware of data dependencies
  - Unnecessary PCIe transfers and missed parallelism

Application code:

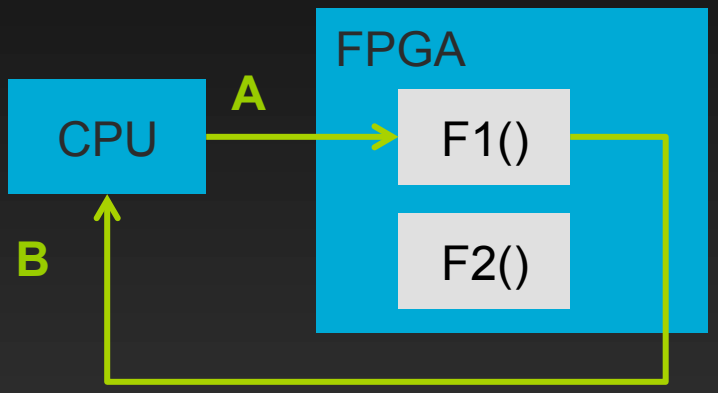
```
void f(int *a, int *b, int *c) {
        …
        *a = f1();
        *b = f2();
        *c = f3();
        f4(a);
}
```

Dynafuse Optimization

- Result: pessimistic data management

Application code:

F1(/*in*/ **A**, /*out*/ **B**)
Other_CPU_Funtions( )
F2(/*in*/ **B**, /*out*/ **C**)

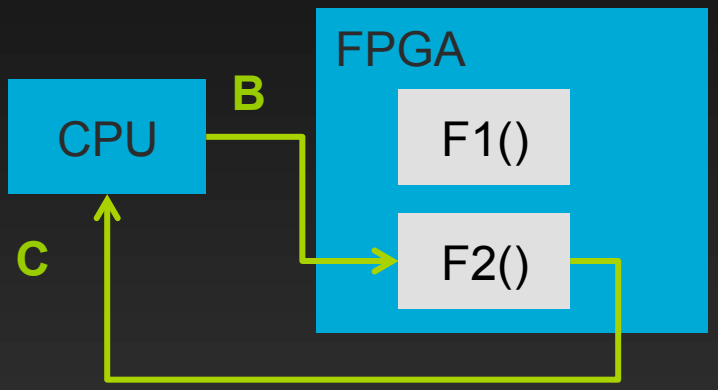


Dynafuse Optimization

- Result: pessimistic data management

Application code:

F1(/*in*/ **A**, /*out*/ **B**)
Other_CPU_Funtions( )
F2(/*in*/ **B**, /*out*/ **C**)

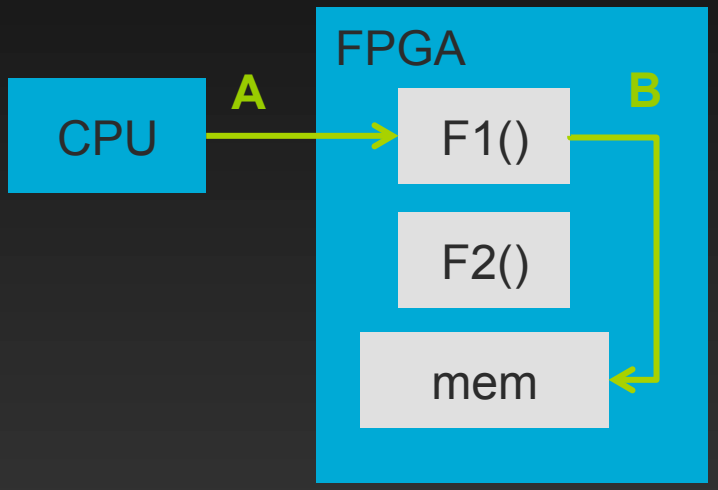


Locality Exploitation

- PCIe bus is inefficient
  - Up to 65% of total execution time
- FPGA boards often have multiple DDR RAMs

Application code:

F1(/*in*/ **A**, /*out*/ **B**)
Other_CPU_Funtions( )
F2(/*in*/ **B**, /*out*/ **C**)

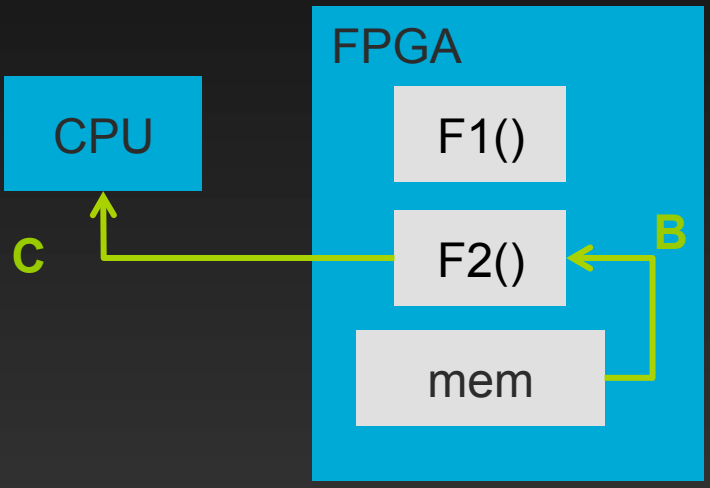


Locality Exploitation

- PCIe bus is inefficient
  - Up to 65% of total execution time
- FPGA boards often have multiple DDR RAMs

Application code:

F1(/*in*/ **A**, /*out*/ **B**)
Other_CPU_Funtions( )
F2(/*in*/ **B**, /*out*/ **C**)



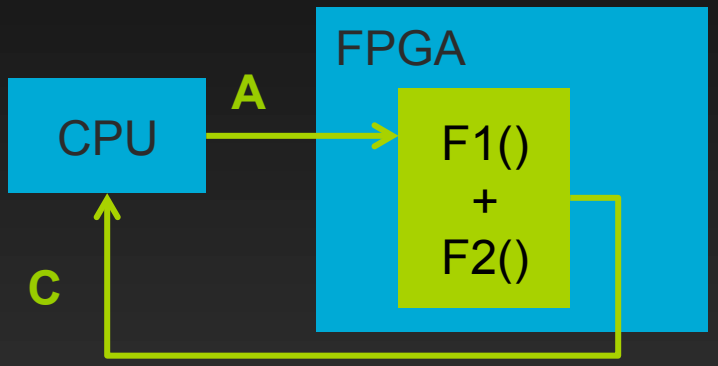
- Saves 2N-2 transfers for N functions, 2 in this case

Pipeline Fusion

- Many FPGA functions are pipelined
- Fuse dependent functions into single pipeline

Application code:

F1(/*in*/ **A**, /*out*/ **B**)
Other_CPU_Funtions( )
F2(/*in*/ **B**, /*out*/ **C**)



- Fusing N pipelines creates Nx speedup

Dynafuse Overview

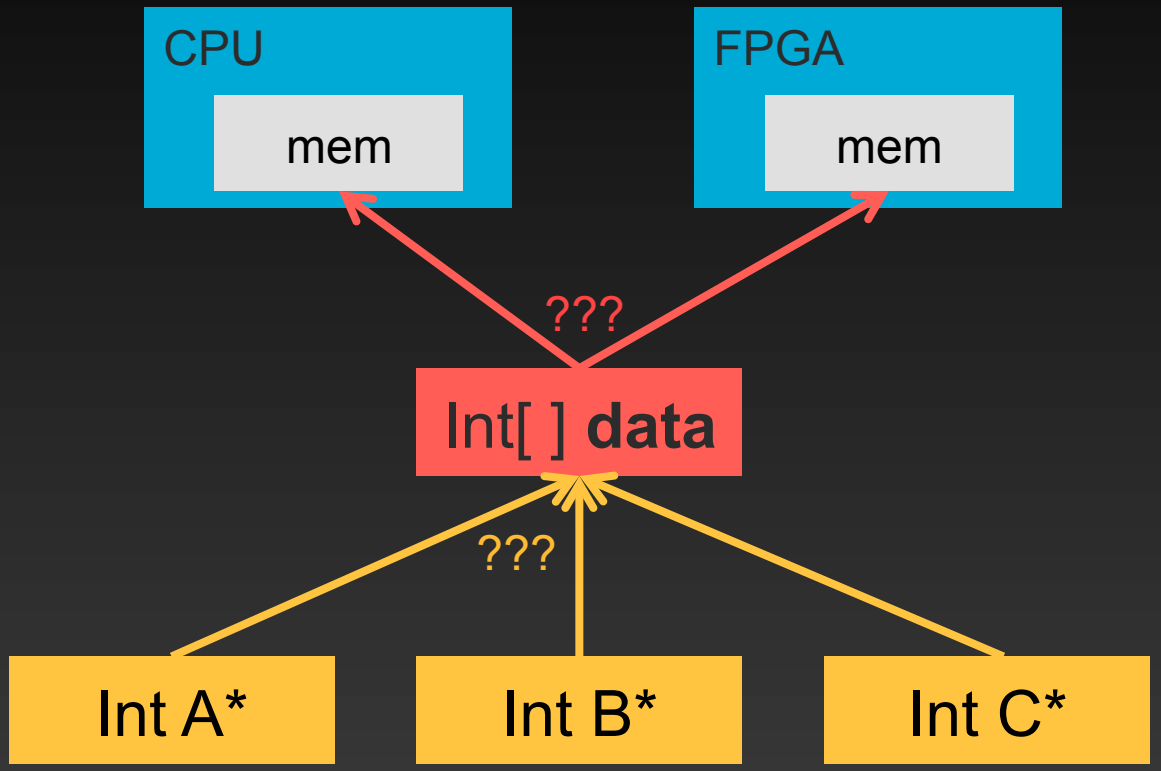
- The Dynafuse optimizations:
  - Locality Exploitation and Pipeline Fusion
  - Automatic & transparent
- SW: Dynamic Dependence Analysis
  - Coherent Arrays
  - Function Queue
- HW: Fusion Network

Dynafuse Challenges

- Information for locality exploitation:
  1. Which device memory has the most recent copy of the data?

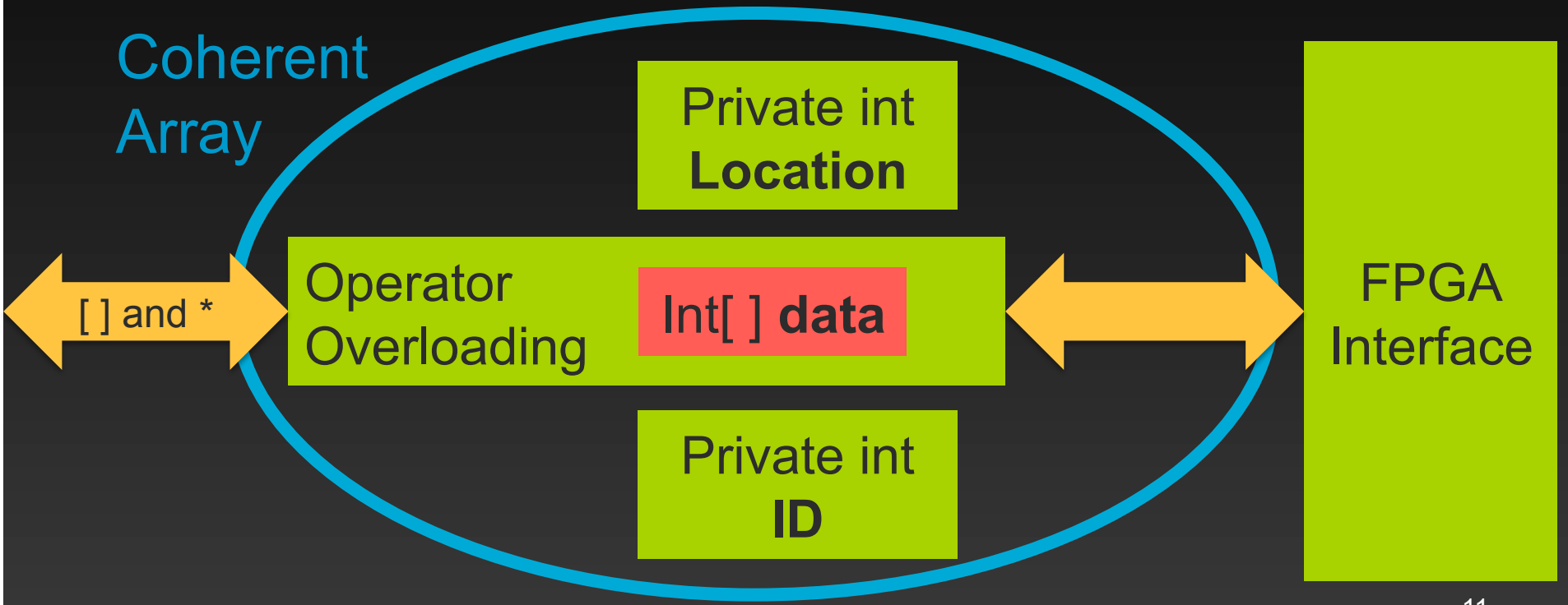
Coherent Arrays

- Std arrays have locality, alias issues
- Resolved by Coherent Arrays



Coherent Arrays

- Std arrays have locality, alias issues
- Resolved by Coherent Arrays



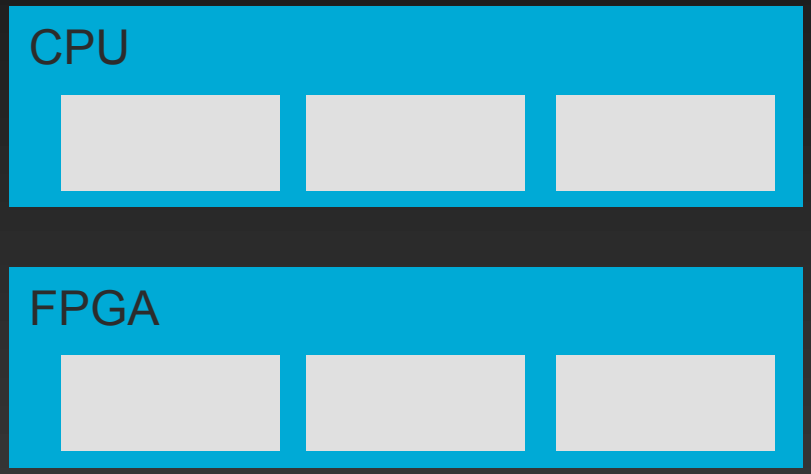
Data management

1. All data starts on the CPU
2. Coherent array does *not* immediately return output
3. Reads/writes ensure locality before returning

Application code:

```
1  CoherentA
2  CoherentArray B
3  CoherentArray C
4  FPGA_F1(A, B)
5  Other_Functions()
6  FPGA_F2(B, C)
7  File.write(C)
```

A



Data management

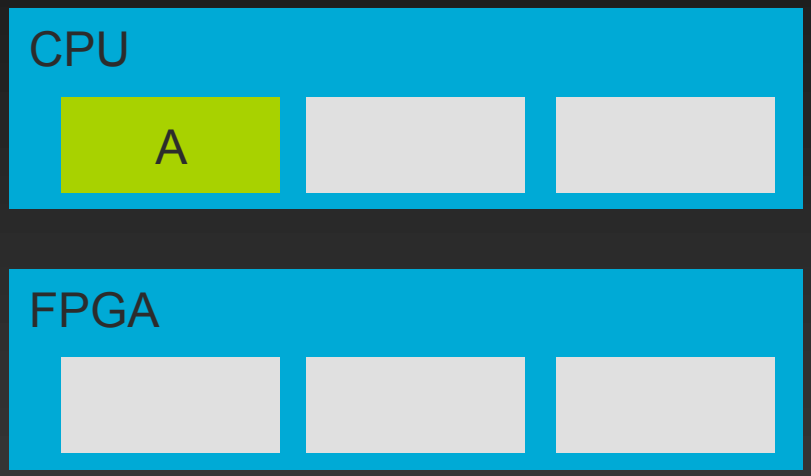
1. All data starts on the CPU
2. Coherent array does *not* immediately return output
3. Reads/writes ensure locality before returning

Application code:

```
1  CoherentArray A
2  CoherentA
3  CoherentA
4  FPGA_F1(A, B)
5  Other_Functions()
6  FPGA_F2(B, C)
7  File.write(C)
```

B

C

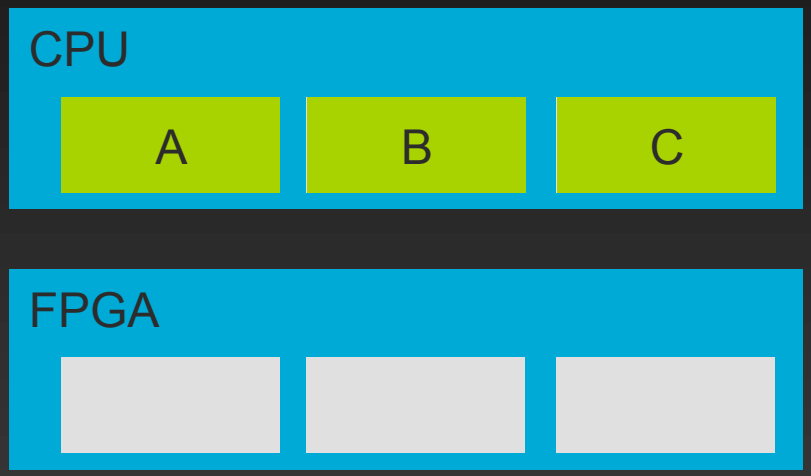


Data management

1. All data starts on the CPU
2. Coherent array does *not* immediately return output
3. Reads/writes ensure locality before returning

Application code:

```
1  CoherentArray A
2  CoherentArray B
3  CoherentArray C
4  FPGA_F1(A, B)
5  Other_Functions()
6  FPGA_F2(B, C)
7  File.write(C)
```

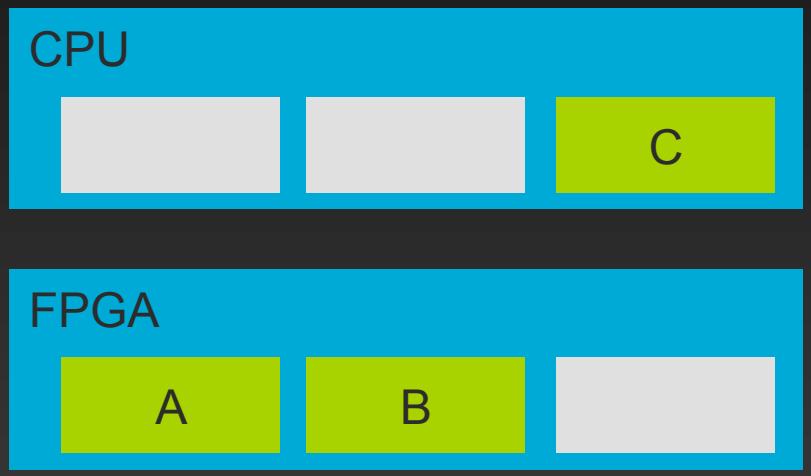


Data management

1. All data starts on the CPU
2. Coherent array does *not* immediately return output
3. Reads/writes ensure locality before returning

Application code:

```
1  CoherentArray A
2  CoherentArray B
3  CoherentArray C
4  FPGA_F1(A, B)
5  Other_Functions()
6  FPGA_F2(B, C)
7  File.write(C)
```

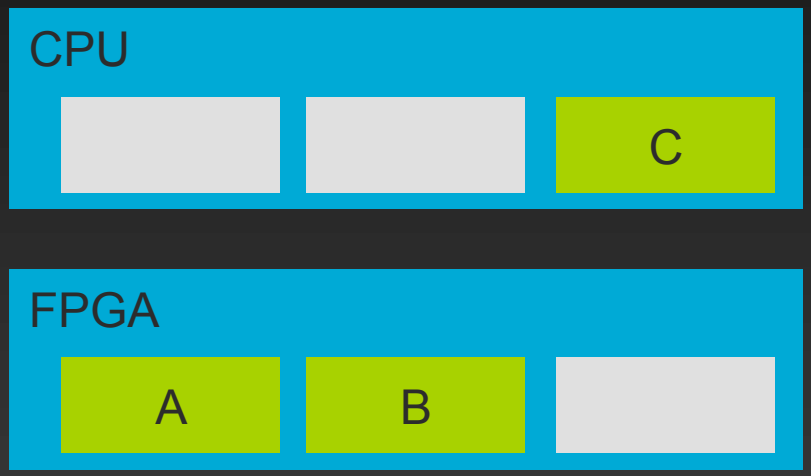


Data management

1. All data starts on the CPU
2. Coherent array does *not* immediately return output
3. Reads/writes ensure locality before returning

Application code:

```
1  CoherentArray A
2  CoherentArray B
3  CoherentArray C
4  FPGA_F1(A, B)
5  Other_Functions()
6  FPGA_F2(B, C)
7  File.write(C)
```

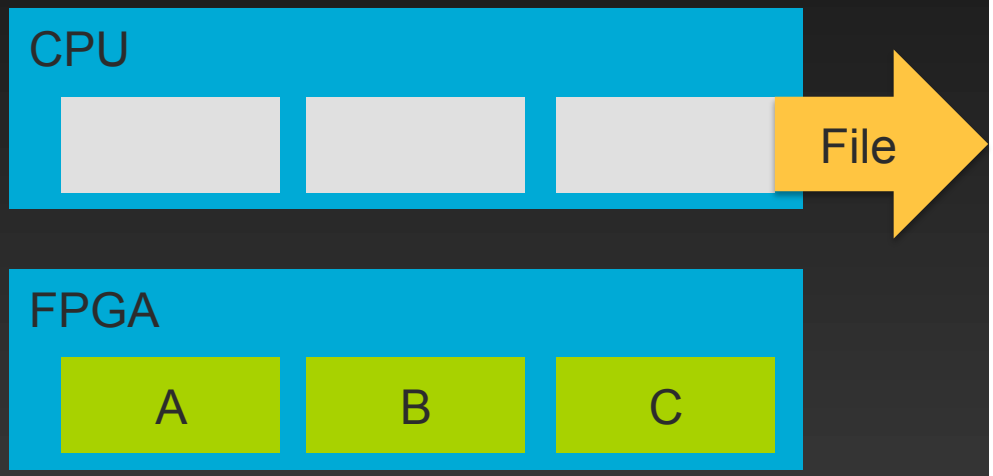


Data management

1. All data starts on the CPU
2. Coherent array does *not* immediately return output
3. Reads/writes ensure locality before returning

Application code:

```
1  CoherentArray A
2  CoherentArray B
3  CoherentArray C
4  FPGA_F1(A, B)
5  Other_Functions()
6  FPGA_F2(B, C)
7  File.write(C)
```

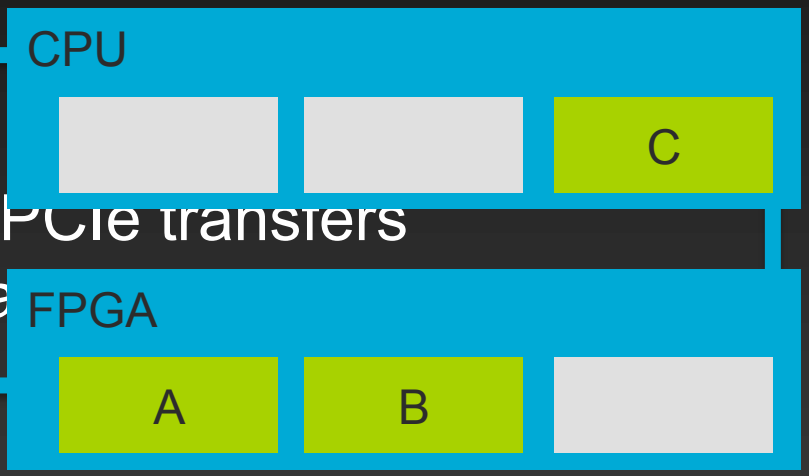


Data management

1. All data starts on the CPU
2. Coherent array does *not* immediately return output
3. Reads/writes ensure locality before returning

Application code:

```
1  CoherentArray A
2  CoherentArray B
3  CoherentArray C
4  FPGA_F1(A, B)
5  Other_Functions()
6  FPGA_F2(B, C)
7  File.write(C)
```



Dynafuse Challenges

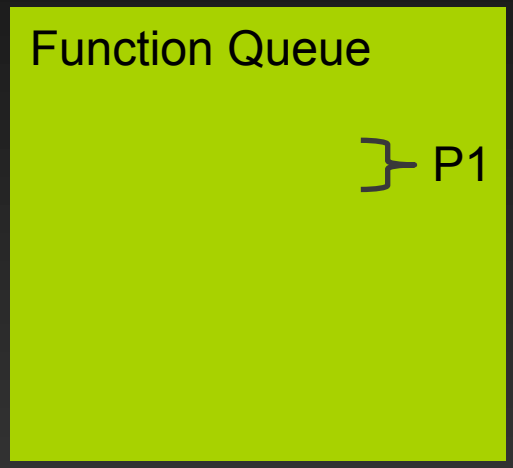
- Information for locality exploitation:
  1. Which device memory has the most recent copy of the data?
- For pipeline fusion:
  2. Which future functions need the output of the current function?

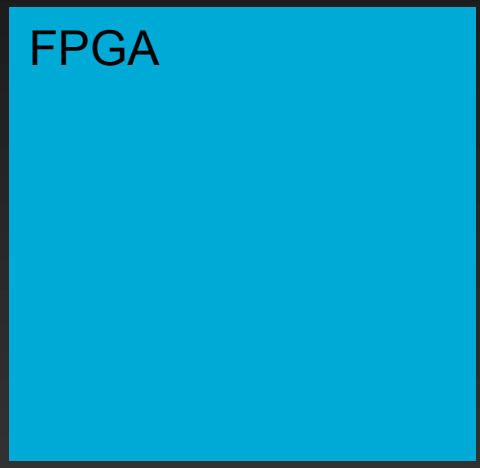
Dynamic Dependence Analysis

1. FPGA functions are placed on the queue when called—*not executed*
2. Coherent arrays used to establish dependence chains
3. CPU functions trigger the entire chain that produces the requested value

Application code:

```
1  FPGA_F1(A, B)
2  FPGA_F2(B, C)
3  FPGA_F3(X, Y)
4  File.write(C)
5  File.write(Y)
```





Dynamic Dependence Analysis

1. FPGA functions are placed on the queue when called—*not executed*
2. Coherent arrays used to establish dependence chains
3. CPU functions trigger the entire chain that produces the requested value

Application code:

```
1  FPGA_F1(A, B)
2  FPGA_F2(B, C)
3  FPGA_F3(X, Y)
4  File.write(C)
5  File.write(Y)
```

Function Queue

FPGA_F1(A, B) } P1

FPGA

Dynamic Dependence Analysis

1. FPGA functions are placed on the queue when called—*not executed*
2. Coherent arrays used to establish dependence chains
3. CPU functions trigger the entire chain that produces the requested value

Application code:

```
1  FPGA_F1(A, B)
2  FPGA_F2(B, C)
3  FPGA_F3(X, Y)
4  File.write(C)
5  File.write(Y)
```

Function Queue

FPGA_F1(A, B)
FPGA_F2(B, C)
} P1

FPGA

Dynamic Dependence Analysis

1. FPGA functions are placed on the queue when called—*not executed*
2. Coherent arrays used to establish dependence chains
3. CPU functions trigger the entire chain that produces the requested value

Application code:

```
1  FPGA_F1(A, B)
2  FPGA_F2(B, C)
3  FPGA_F3(X, Y)
4  File.write(C)
5  File.write(Y)
```

Function Queue

FPGA_F1(A, B)
FPGA_F2(B, C)
} P1

FPGA

Dynamic Dependence Analysis

1. FPGA functions are placed on the queue when called—*not executed*
2. Coherent arrays used to establish dependence chains
3. CPU functions trigger the entire chain that produces the requested value

Application code:

```
1  FPGA_F1(A, B)
2  FPGA_F2(B, C)
3  FPGA_F3(X, Y)
4  File.write(C)
5  File.write(Y)
```

Function Queue

FPGA_F1(A, B)
FPGA_F2(B, C)
} P1

FPGA_F3(X, Y)
} P2

FPGA

Dynamic Dependence Analysis

1. FPGA functions are placed on the queue when called—*not executed*
2. Coherent arrays used to establish dependence chains
3. CPU functions trigger the entire chain that produces the requested value

Application code:

```
1  FPGA_F1(A, B)
2  FPGA_F2(B, C)
3  FPGA_F3(X, Y)
4  File.write(C)
5  File.write(Y)
```

Function Queue

FPGA_F1(A, B)
FPGA_F2(B, C)
} P1

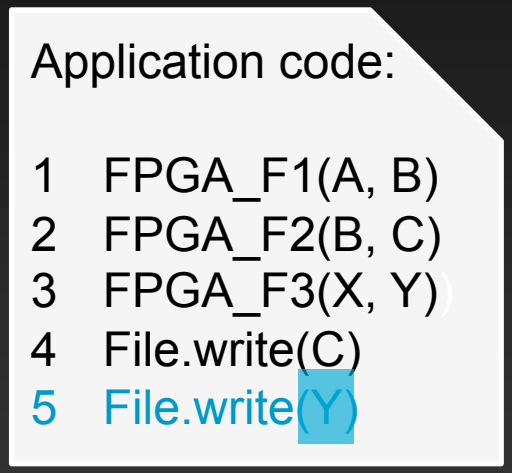
FPGA_F3(X, Y)
} P2

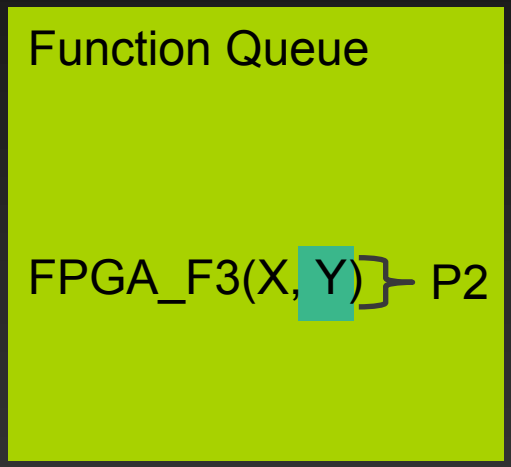
FPGA

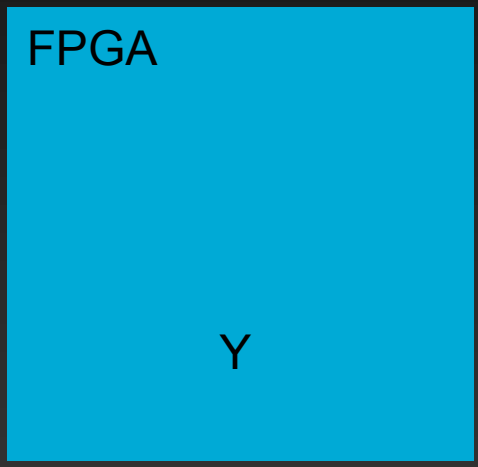
C

Dynamic Dependence Analysis

1. FPGA functions are placed on the queue when called—*not executed*
2. Coherent arrays used to establish dependence chains
3. CPU functions trigger the entire chain that produces the requested value





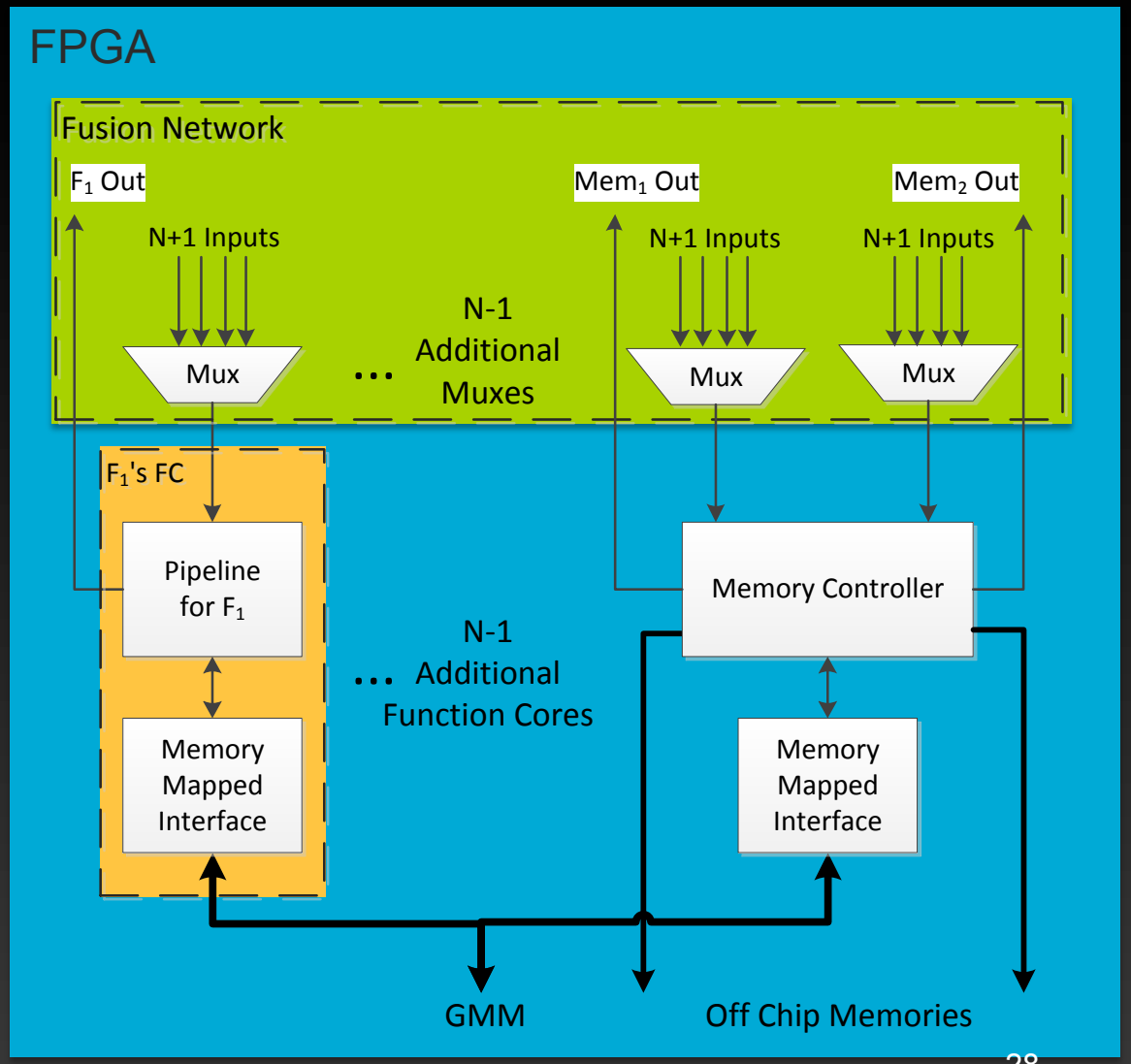


Dynafuse Challenges

- Information for locality exploitation:
  1. Which device memory has the most recent copy of the data?
- For pipeline fusion:
  2. Which future functions need the output of the current function?
  3. Can the desired pipeline be created at runtime?

Fusion Network

- All-to-all network
- Muxes configured by function queue



Experiments

- Created 8 pipelined functions in VHDL
- Wrote 2 SW applications in C++

Image Segmentation 1080p:

1. RGB to HSV Conversion
2. HSV Threshold Filter
3. Morphological Erosion
4. Morphological Dilation
5. Sobel Edge Detection

FFT Averaging Filter:

1. Fast Fourier Transform (FFT)
2. Frequency-domain Average
3. Inverse FFT

- Altera Stratix III E260 on Gidel ProcStar III with PCIe x8

Dynafuse Results

Image segmentation:

Activity	Execution Time (s)	Speedup
No optimizations	0.2	-
w/ Locality Exploitation	0.12	1.6x
w/ Pipeline Fusion	0.04	5x

FFT Averaging Filter:

Activity	Execution Time (s)	Speedup
No optimizations	0.0042	-
w/ Locality Exploitation	0.0024	1.75x
w/ Pipeline Fusion	0.0014	3x

- Fusion network overhead < 1% total area

Conclusions

- Optimizations would benefit HLS
  - Require dependence information
- Transparent, dynamic approach
- Pipeline fusion: Nx speedup for N functions
- Locality exploitation: save 2N-2 transactions
- Future: dynamic fission study

Thanks!

Any Questions?

Sample Application

Goal: Apply Locality Exploitation and Pipeline Fusion Transparently

Regular code:

```
1  int* A = (int*)malloc(500);
2  int* B = (int*)malloc(500);
3  int* C = (int*)malloc(500);
4  for(i = 0; i < 500; i++) A[i] = i;
5  FPGA_F1(A, B);
6  Other_Functions();
7  FPGA_F2(B, C);
8  File.write(C);
```

Sample Application

Goal: Apply Locality Exploitation and Pipeline Fusion Transparently

Regular code:

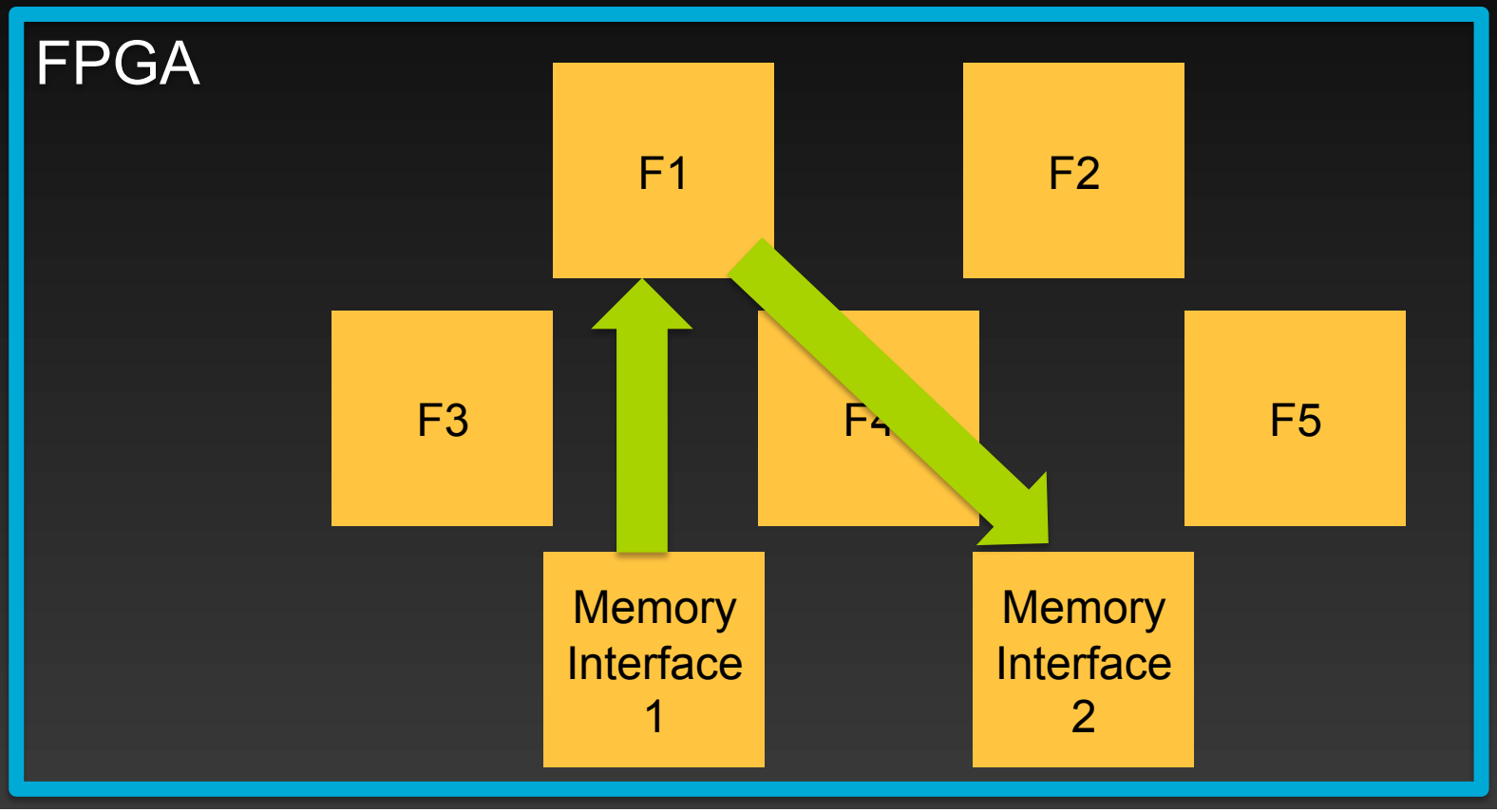
```
1  int* A = (int*)malloc(500);
2  int* B = (int*)malloc(500);
3  int* C = (int*)malloc(500);
4  for(i = 0; i < 500; i++) A[i] = i;
5  FPGA_F1(A, B);
6  Other_Functions();
7  FPGA_F2(B, C);
8  File.write(C);
```

Dynafuse code:

```
1  Dynafuse context;
2  CoherentArray<int> A (500, context);
3  CoherentArray<int> B (500, context);
4  CoherentArray<int> C (500, context);
5  for(i = 0; i < 500; i++) A[i] = i;
6  FPGA_F1(A, B, context);
7  Other_Functions();
8  FPGA_F2(B, C, context);
9  File.write(C);
```

Dynamic Fusion

- Execute an arbitrary number of arbitrarily configured functions



Dynamic Fusion

- Execute an arbitrary number of arbitrarily configured functions