Exact and Approximate Task Assignment Algorithms for Pipelined Software Synthesis

Matin Hashemi
Soheil Ghiasi

Laboratory for Embedded and Programmable Systems
http://leps.ece.ucdavis.edu

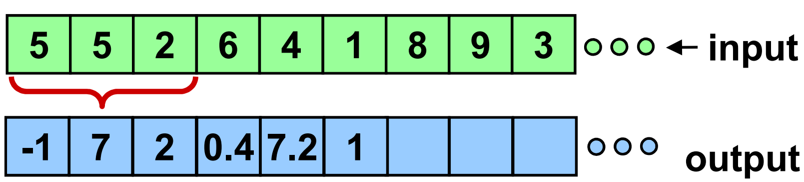
Department of Electrical and Computer Engineering
University of California, Davis
Davis, CA
United States

Streaming Applications

- Widespread



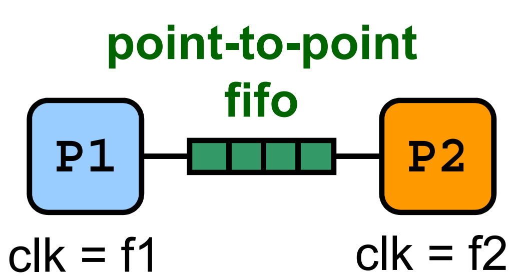

  - Cell phones , mp3 players, video conference, real-time encryption, graphics, HDTV editing, hyperspectral imaging, cellular base stations
- Definition
  - Infinite sequence of data items
  - At any given time, operates on a small window of this sequence
  - Moves forward in data space

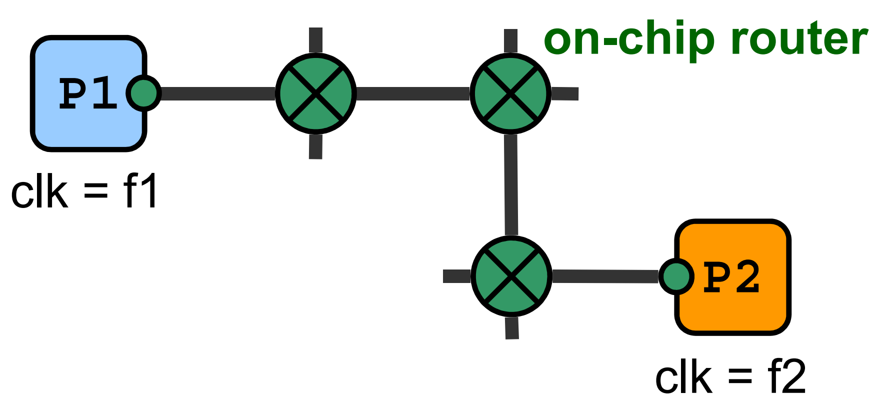



```
//53° around the z axis
const R[3][3]={
      {0.6,-0.8, 0.0},
      {0.8, 0.6, 0.0},
      {0.0, 0.0, 1.0}}
Rotation3D {
 for (i=0; i<3; i++)
  for (j=0; j<3; j++)
   B[i] += R[i][j] * A[j]
 }
```

Programming Model

- Thread-based models
  - Difficult to develop and debug [Sutter and Larus, ACM Queue '05]
  - Fundamentally, unreliable and nondeterministic [Lee, IEEE Computer '06], [Weng, MIT tech report '75]
- To maximize **throughput** of stream applications
  - Pipelined distributed-memory dual-core
  - Connected through on-chip network





Software Synthesis

- Need better CAD tools
  [Rowen, MPSOC ‘3], [Rabaey, Gigascale ’04], [Gordon, ASPLOS ’06], [Martin, DAC 06], [Parkhurst, ICCAD ’06], [Panel, EMSOFT ‘06], [Asanovic, UCB tech report ‘06]

- Need effective **task assignment** methods because of diminishing returns if applications don’t use available processing power
  [Leland, SC ’95], [Karypis, SC ’95], [Parkhurst, ICCAD ’06], [Martin, DAC 06], [Asanovic, UCB tech report ‘06]

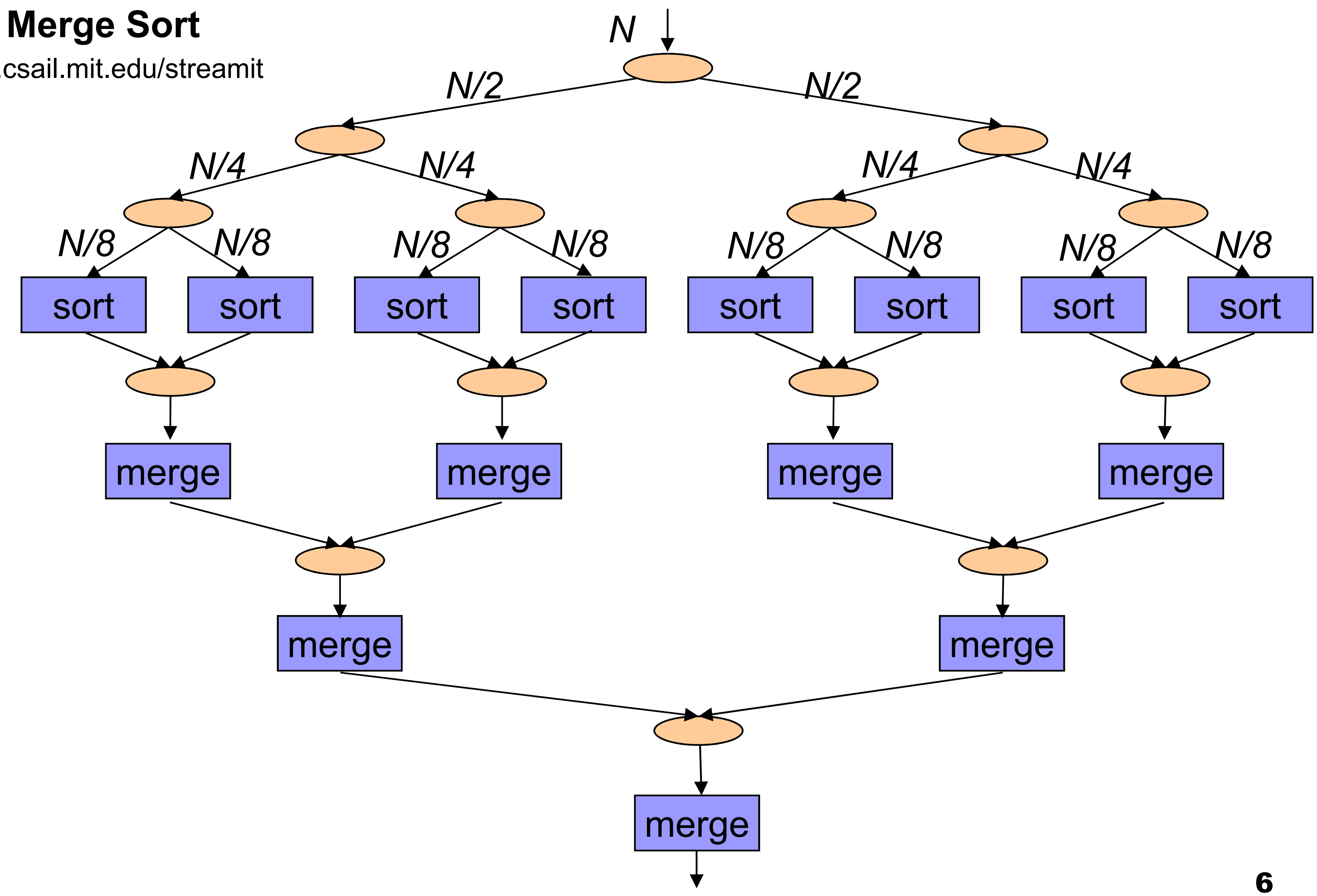
Application Model:
Dataflow Graph

- Vertices or actors
  - functions, computations
- Edges
  - data dependency, communication between actors
- Execution Model
  - any actor can perform its computation whenever all necessary input data are available on incoming edges.
- SDF is one special case
  - statically schedulable [Lee '87]

Example

N-Element Merge Sort

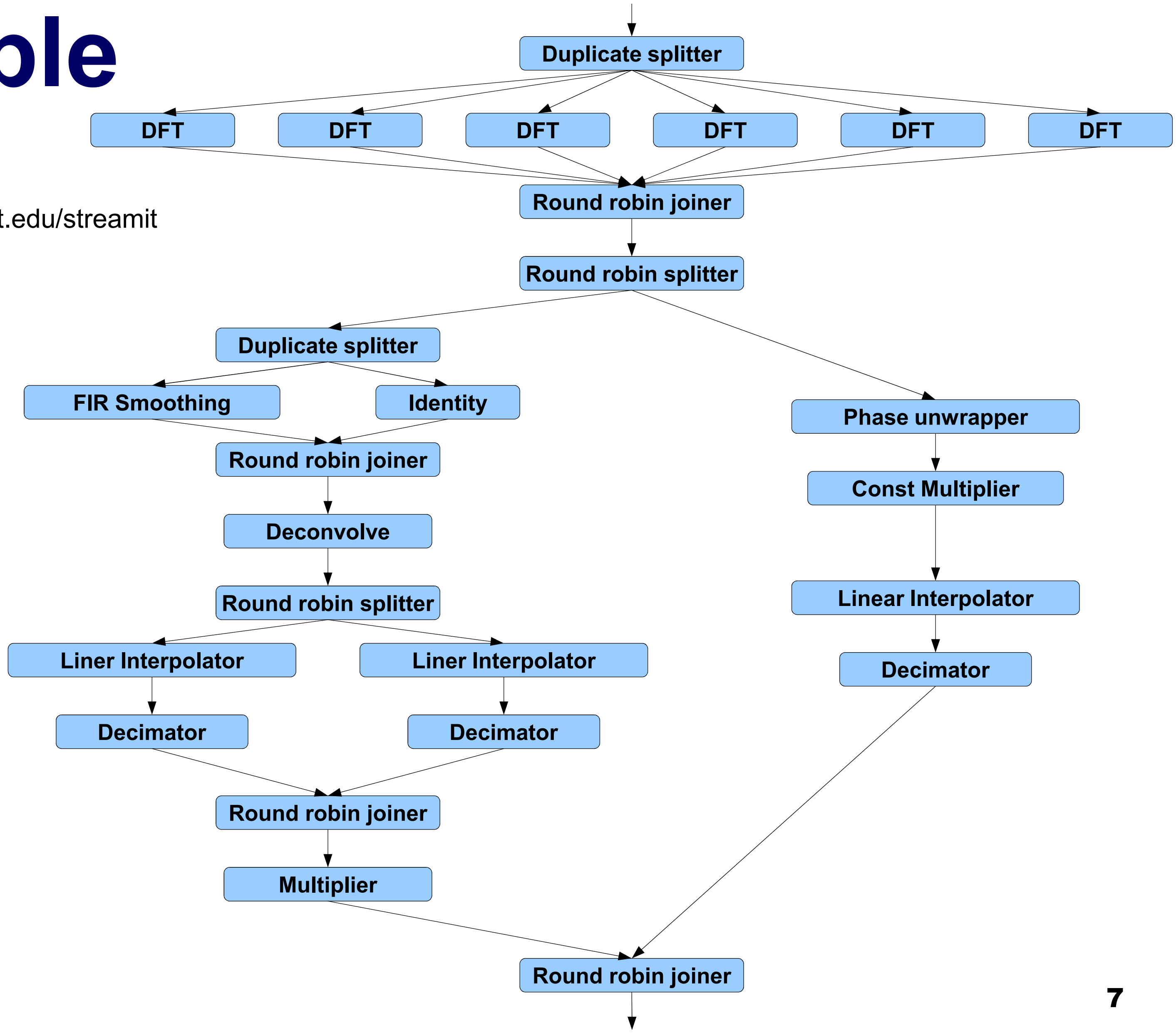
http://www.cag.csail.mit.edu/streamit



Example

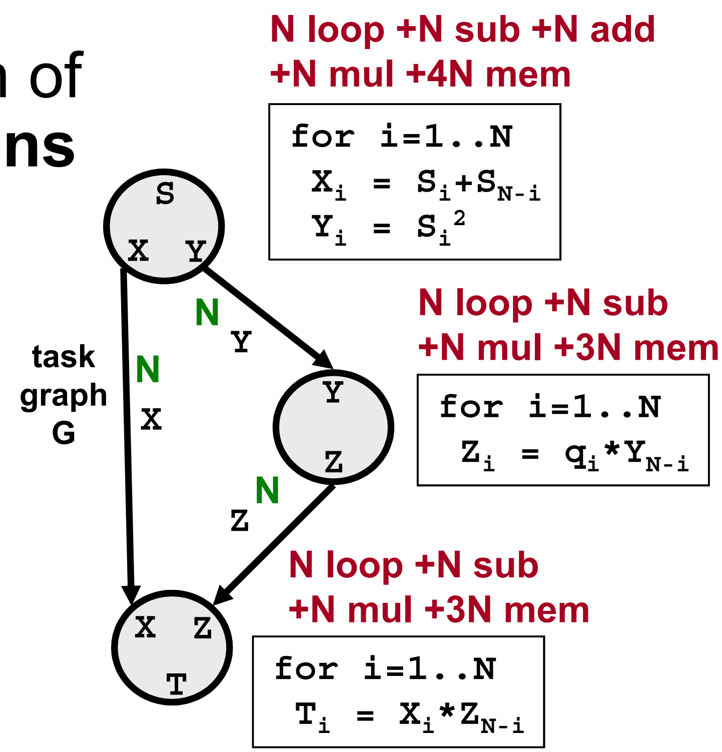
Vocoder

http://www.cag.csail.mit.edu/streamit

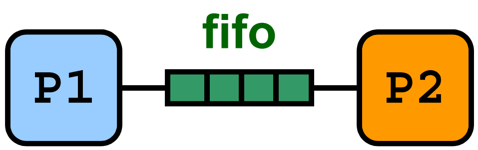


Performance Model:
Implementation Dependant

- Throughput can be any function of **workloads** and **communications**
- W_G
  - computation **workload**, unit time
  - estimated from source code
  - implementation dependant
- Data rates
  - # of data tokens
  - known at compile time [Lee '87]
- C_{CUT}
  - communication **cost**, unit time
  - implementation dependant

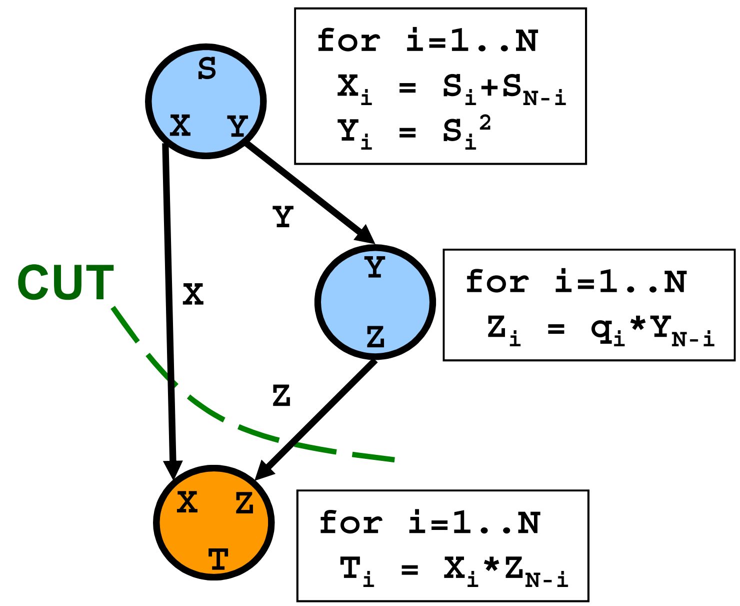


Performance Model:
Example 1




```
while(1) {
 for i=1..n
  X[i]= S[i]+S[n-i]
  Y[i]= S[i]*S[i]
 for i=1..n
  Z[i]= q[i]*Y[n-i]
 for i=1..n
  writef(X[i])
 for i=1..n
  writef(Z[i])
}
```

```
while(1) {
 for i=1..n
  X[i] = readf()
 for i=1..n
  Z[i] = readf()
 for i=1..n
  T[i]= X[i]*Z[n-i]
}
```

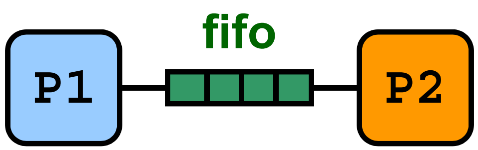


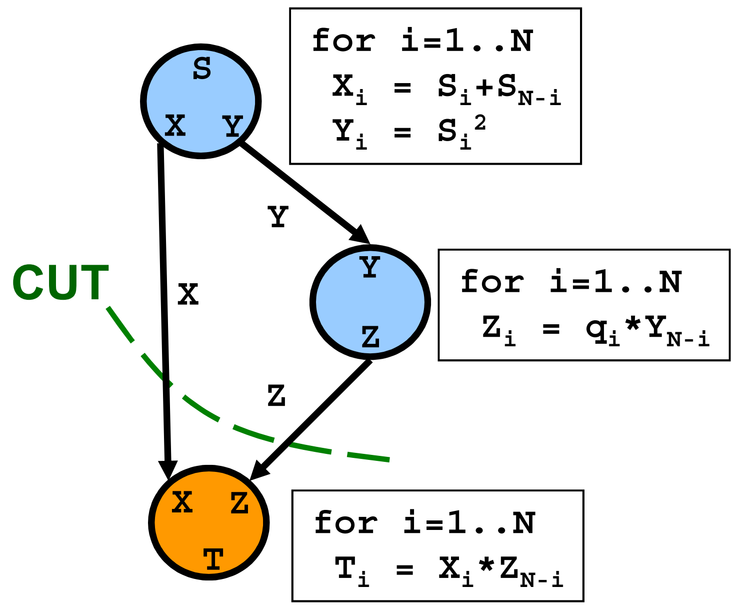
1 / Throughput =

$$\text{Exec. Period} = \text{Max}\{\overbrace{W_1+W_2}^{W_{G1}}+\overbrace{N+N}^{C_{CUT}}\ ,\ \overbrace{N+N}^{C_{CUT}}+\overbrace{W_3}^{W_{G2}}\}$$

correction factors for clock speeds

Performance Model:
Example 2





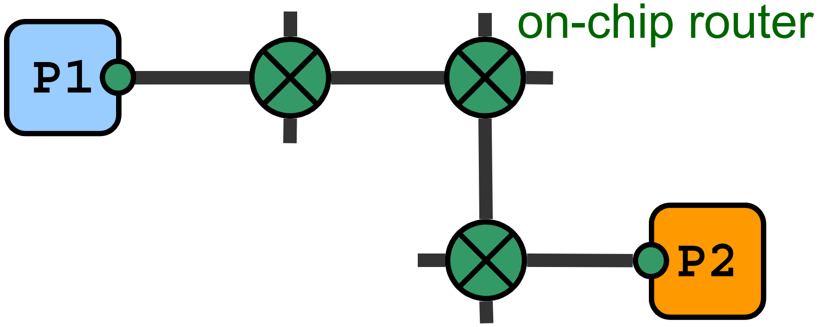

```
while(1) {
 for i=1..n
  X[i]= S[i]+S[n-i]
  writef(X[i])
  Y[i]= S[i]*S[i]
 for i=1..n
  Z[i]= q[i]*Y[n-i]
  writef(Z[i])
}
```

```
while(1) {
 for i=1..n
  X[i] = readf()
 for i=1..n
  Z[i] = readf()
 for i=1..n
  T[i]= X[i]*Z[n-i]
}
```

$$\text{Exec. Period} = \text{Max}\{\ \overbrace{W_1+W_2}^{W_{G1}}-2N\text{xmem}+\overbrace{N+N}^{C_{CUT}}\ ,\ \overbrace{N+N}^{C_{CUT}}+\overbrace{W_3}^{W_{G2}}\ \}$$

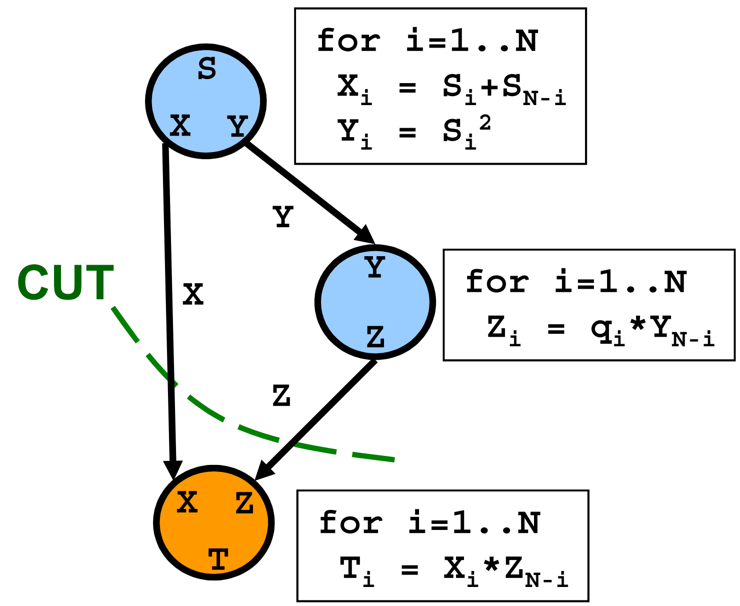
Performance Model:

Example 3




```
while(1) {
 for i=1..n
  X[i]= S[i]+S[n-i]
  Y[i]= S[i]*S[i]
 for i=1..n
  Z[i]= q[i]*Y[n-i]
 for i=1..n
  P[i]=X[i]
  P[i+n]=Z[i]
 writep(P[1..2n])
}
```

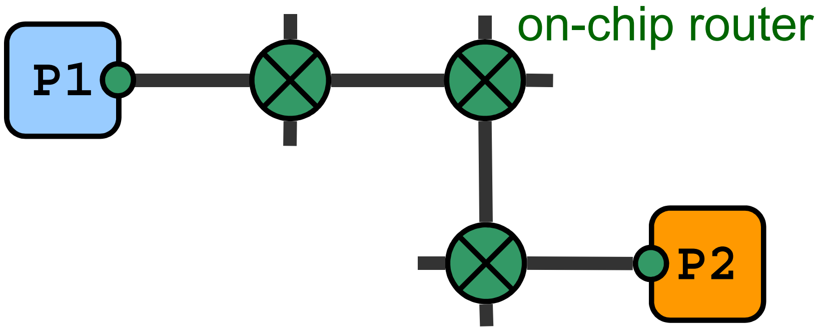
```
while(1) {
 P[1..2n]=readp()
 for i=1..n
  X[i]  = P[i]
  Z[i]  = P[i+n]
 for i=1..n
  T[i]= X[i]*Z[n-i]
}
```



$$\text{Exec. Period} = \text{Max}\{ \underbrace{W_1+W_2+OV}_{W_{G1}}+\underbrace{N+N}_{C_{CUT}} \text{, hop , } \underbrace{N+N}_{C_{CUT}}+\underbrace{OV+W_3}_{W_{G2}} \}$$

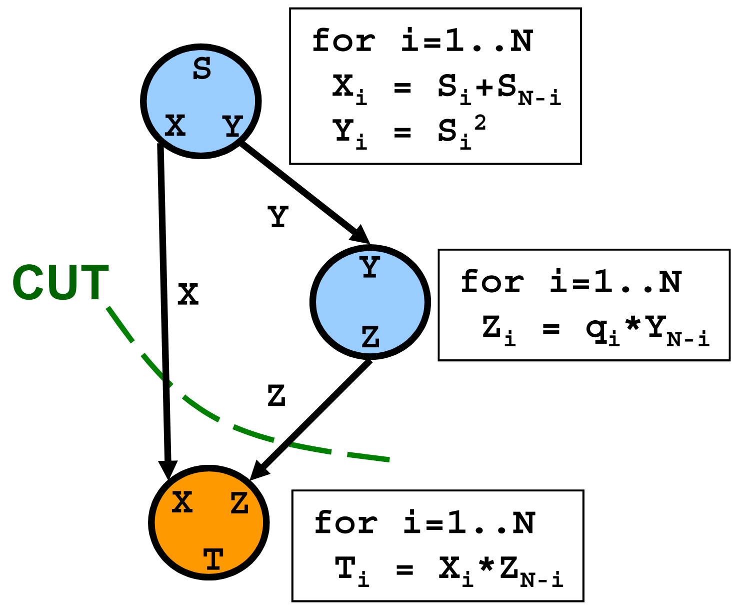
Performance Model:

Example 4




```
while(1) {
 for i=1..n
  X[i]= S[i]+S[n-i]
  writep(X[i])
  Y[i]= S[i]*S[i]
 for i=1..n
  Z[i]= q[i]*Y[n-i]
  writep(Z[i])
}
```

```
while(1) {
 for i=1..n
  X[i] = readp()
 for i=1..n
  Z[i] = readp()
 for i=1..n
  T[i]= X[i]*Z[n-i]
}
```



$$\text{Exec. Period} = \text{Max}\{ \overbrace{W_1+W_2}^{W_{G1}}\text{-2Nxmem+}\overbrace{N+N}^{C_{CUT}} ,\ \overbrace{(N+N)}^{C_{CUT}}\text{hop} ,\ \overbrace{N+N}^{C_{CUT}}+\overbrace{W_3}^{W_{G2}} \}$$

Versatile Cost Function

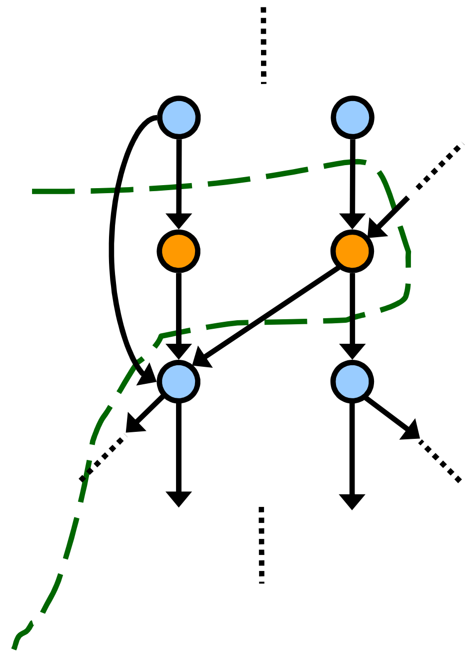
- Throughput = 1 / Execution Period
  - implementation dependant
- Task assignment method has to be versatile: handle any realistic hardware-inspired function of
  - workloads
  - communications
- **$Q_{CUT} = F(W_{G1}, C_{CUT}, W_{G2})$**
- realistic: Q_{CUT} has to be non-decreasing in C_{CUT}

Convex Cut

- To ensure a feasible schedule
  [Cong, FPGA '07]
  - we need all data at the beginning

```
while(1) {
  for i=1..n
    A[i] = readf()
  for i=1..n
    Y[i] = readf()
  //computation
}
```

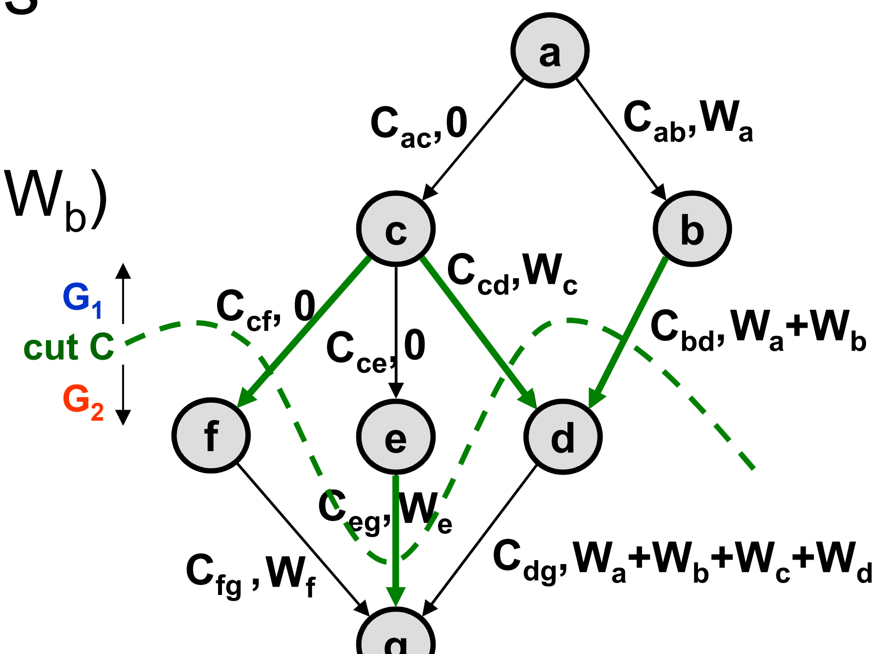
```
while(1) {
  for i=1..n
    X[i] = readf()
  //computation
  for i=1..n
    writef(Y[i])
}
```



- Cycles limit the throughput
  [Rabaey '93], [Wolf '94]

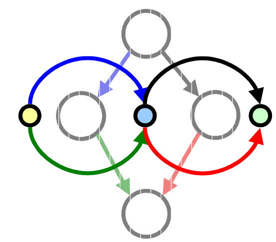
Algorithm Idea

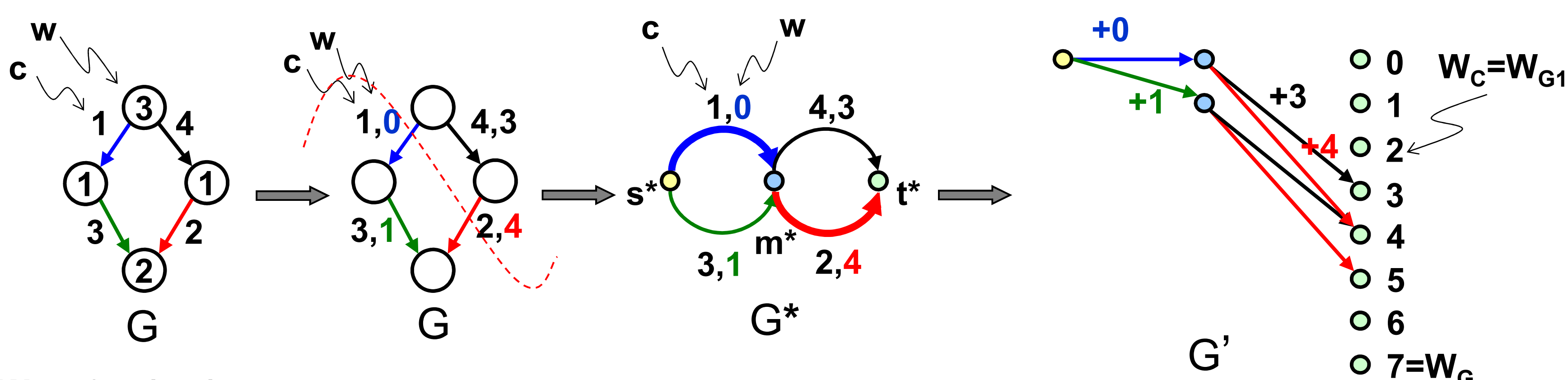
- Calculate cost function Q_C only from cut C, and not other parts of the graph
- Move workloads to edges
- Property:
  - $W_C = (0)+(W_e)+(W_c)+(W_a+W_b) = W_a+W_b+W_c+W_e = W_{G1}$
  - $C_C = C_{cf} + C_{eg} + C_{cd} + C_{bd}$



Algorithm Details

- move node workloads of G to its edges
- for planar graphs, a cut is equal to a path in dual graph
- expand G* to G’

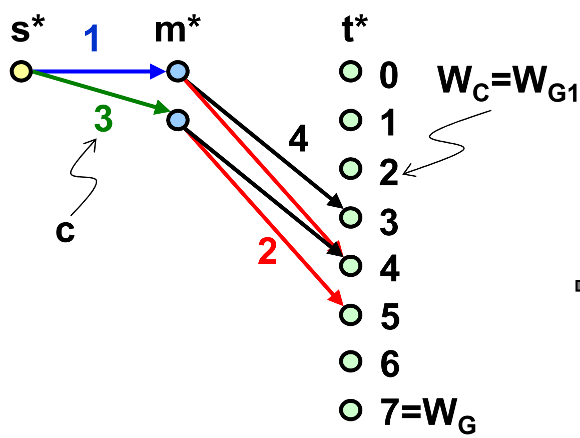




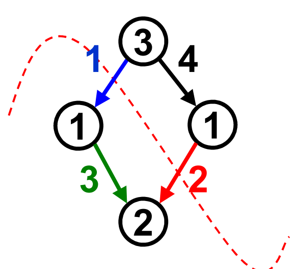
$W_G = 3 + 1 + 1 + 2$

Algorithm Details, cont.

- single-source shortest-path on G'
- pick the best cut



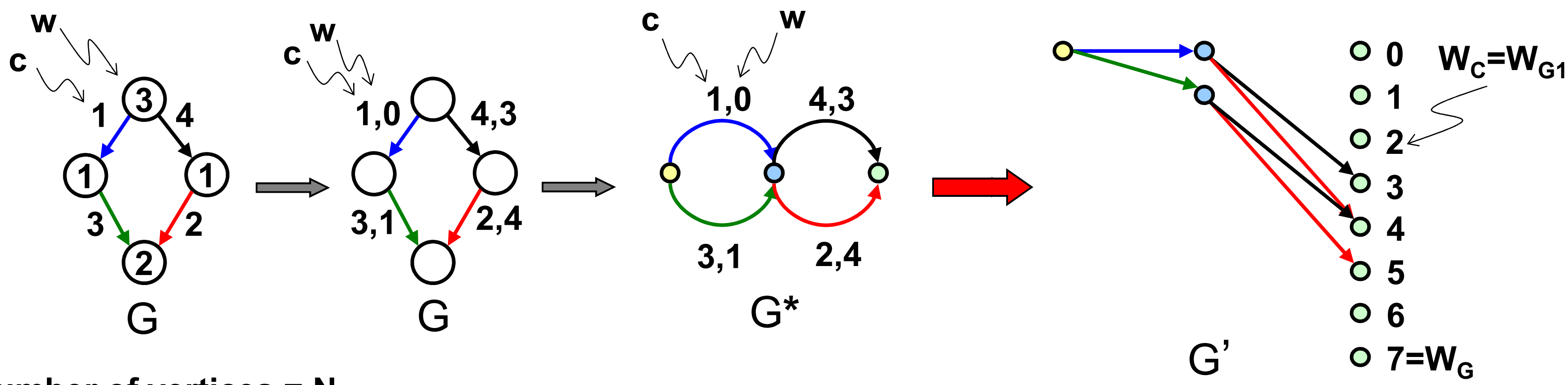
W_{G1}	C_C	shortest path for this W_{G1}?	$W_{G2} = W_G - W_{G1}$	cost function Q_C
3	1+4 =5	✓	7-3=4	9
4	1+2 =3	✓	7-4=3	7
4	3+4 =7		7-4=3	
5	3+2 =5	✓	7-5=2	10



provably **optimal** in minimizing any realistic cost function

$\text{Max}\ \{W_{G1}+C_C\ ,\ W_{G2}+C_C\}$

Complexity



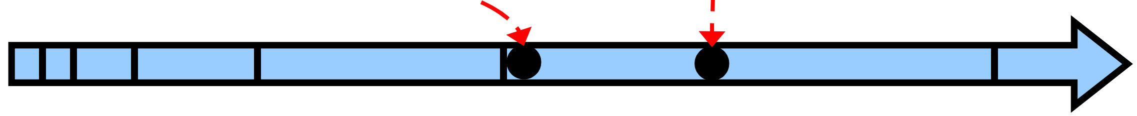
number of vertices = N

- Constructing G' is the most complex part of the algorithm
- Both **runtime** and **memory consumption** depend on the number of vertices in G'
- **O(N x W_G)**
- NP Complete: reduction from set partitioning

Approximate Algorithm

- Approximate **workload** values in graph G'. A **range** of workload values **w** is represented by one single **y** value, where **y=f(w)** is the approximation function, and **δ** is a constant parameter:

$$f(w) = (1+\delta)^{\lfloor \log_{1+\delta} w \rfloor}$$



- Example (1+δ=2)

w	y=f(w)
0	0
1	1
2-3	2
4-7	4
8-15	8

Approximate Algorithm, cont.

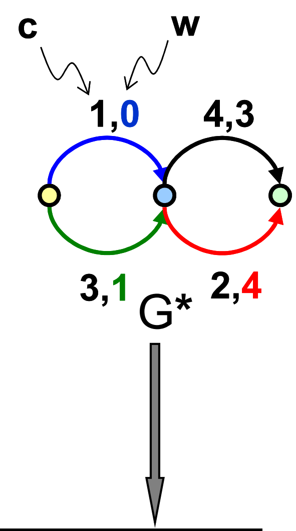
- Theorem:

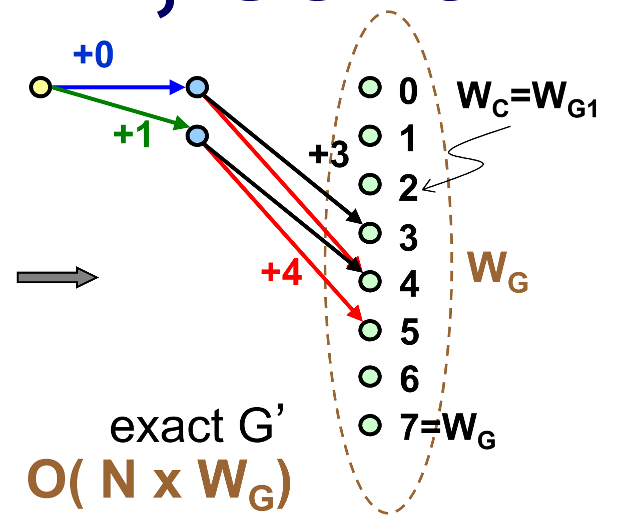
$$\frac{w}{1+\delta} < y \le w$$

$$\frac{W_C}{(1+\delta)^k} < \Psi_C \le W_C$$

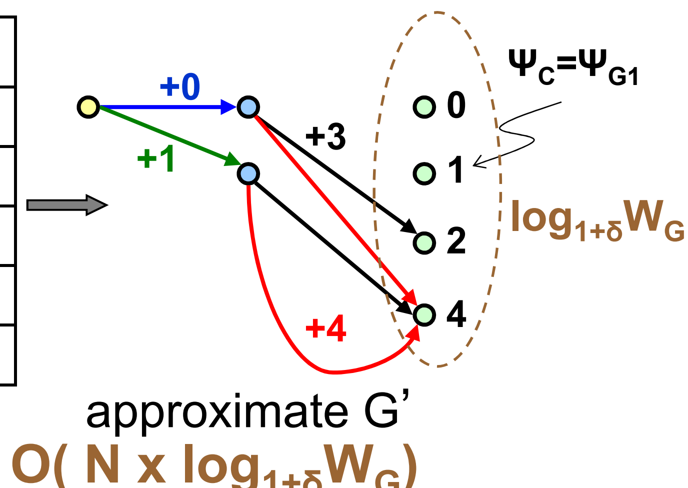
- Cost function
  - $Q_C = F(W_{G1}, C_C, W_{G2})$
  - $\Omega_C = F(\Psi_{G1}, C_C, \Psi_{G2})$
- $Q_{C,min} < \Omega_{C,min} < (1+\varepsilon)\, Q_{C,min}$
- $\varepsilon = \delta k$

- Error in calculating cost function is bounded within an adjustable factor.



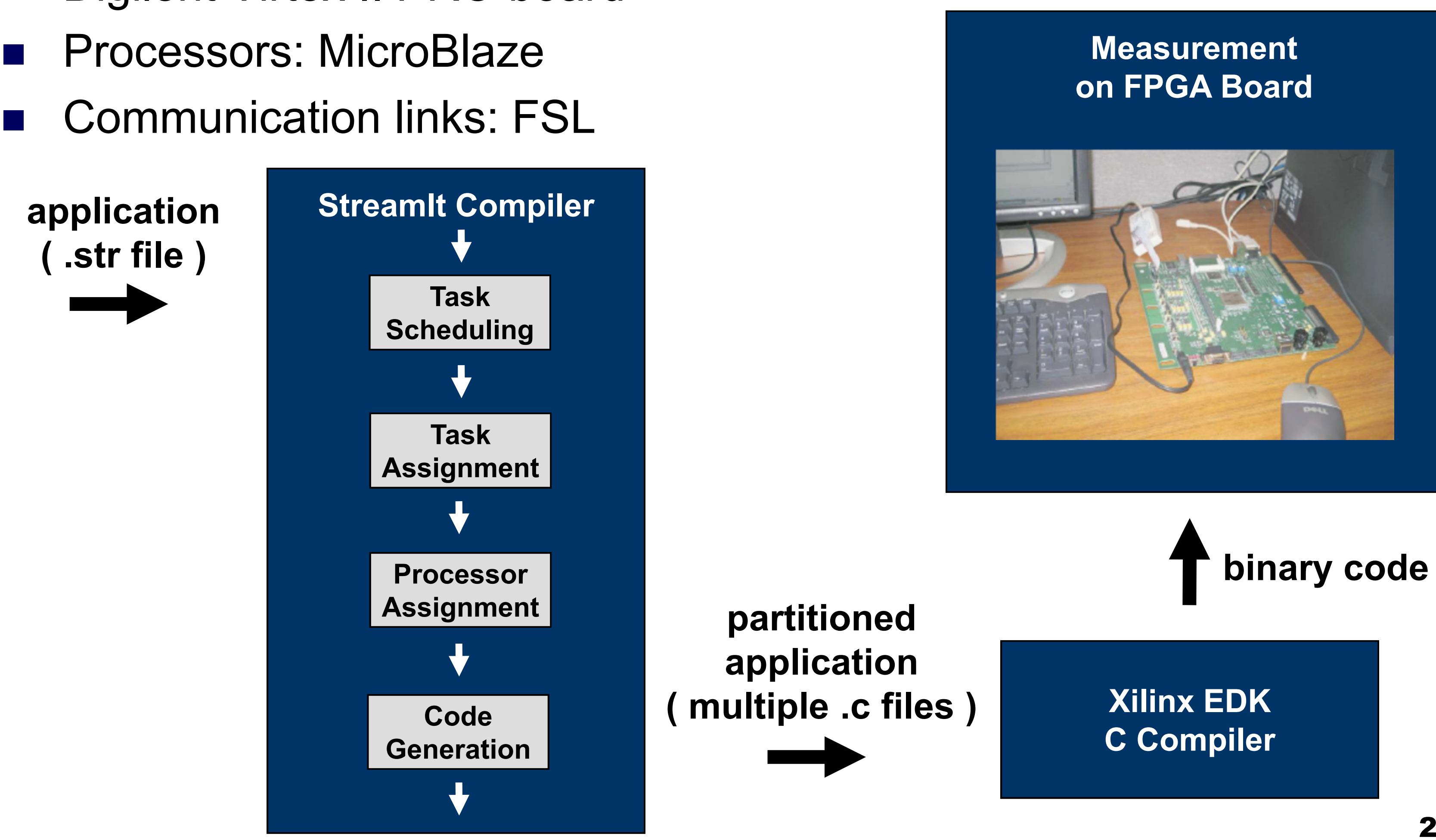


w	y=f(w)
0	0
1	1
2-3	2
4-7	4
8-15	8



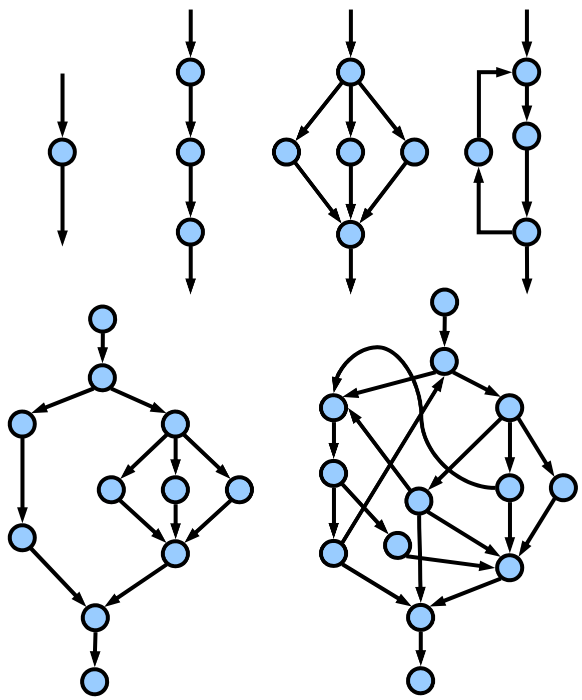
Experiment Platform

- Digilent Virtex II PRO board
- Processors: MicroBlaze
- Communication links: FSL

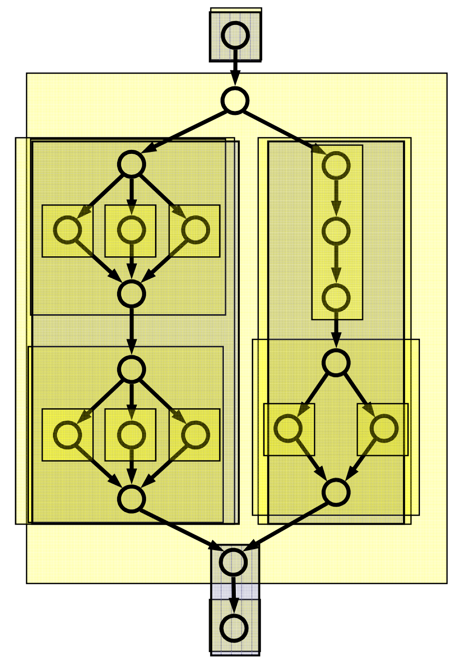


StreamIt

- basic element: `Filter`
- constructs:
  `Pipeline, SplitJoin, Feedback`
- planar graph



- Partitioning Algorithm:
  [Thies, MIT tech report '03]
  - limited to structured graphs
  - dynamic programming



$B = 9 + 3 + 2 + 3$

$\sum_{b \text{ in } B} X_b Y_b = O(N^2 B)$

Benchmarks

	Description	Task Graph Structure			Single-processor Throughput (K sample / sec.)
		$\|V_G\|$	$\|E_G\|$	$\|F_G\|$	
BSORT	Bitonic Sort	756	1012	259	187
MATMUL	Blocked Matrix Multiply	23	23	3	135
FFT	Fast Fourier Transform	152	207	58	264
TDE	Frequency Domain Convolution	46	52	9	580
FILTER	Discrete Filter	53	59	9	18.0

Throughput

	StreamIt		TAP		Throughput vs single-processor		Additional Throughput (TAP-StreamIt)
	Workload Imbalance %	Throughput	Workload Imbalance %	Throughput	StreamIt	TAP	
BSORT	7.7	296.3	3.7	319.2	1.58	1.70	.12
MATMUL	6.5	186.3	9.7	208.0	1.38	1.55	.17
FFT	11	417.7	4.9	470.4	1.58	1.77	.19
TDE	4.1	933.8	4.1	933.8	1.61	1.61	.00
FILTER	0.7	34.6	0.7	34.6	1.88	1.88	.00
Avg.					1.61	1.70	.09

Throughput of dual-processor hardware for both StreamIt and TAP algorithms.

FFT

StreamIt / TAP

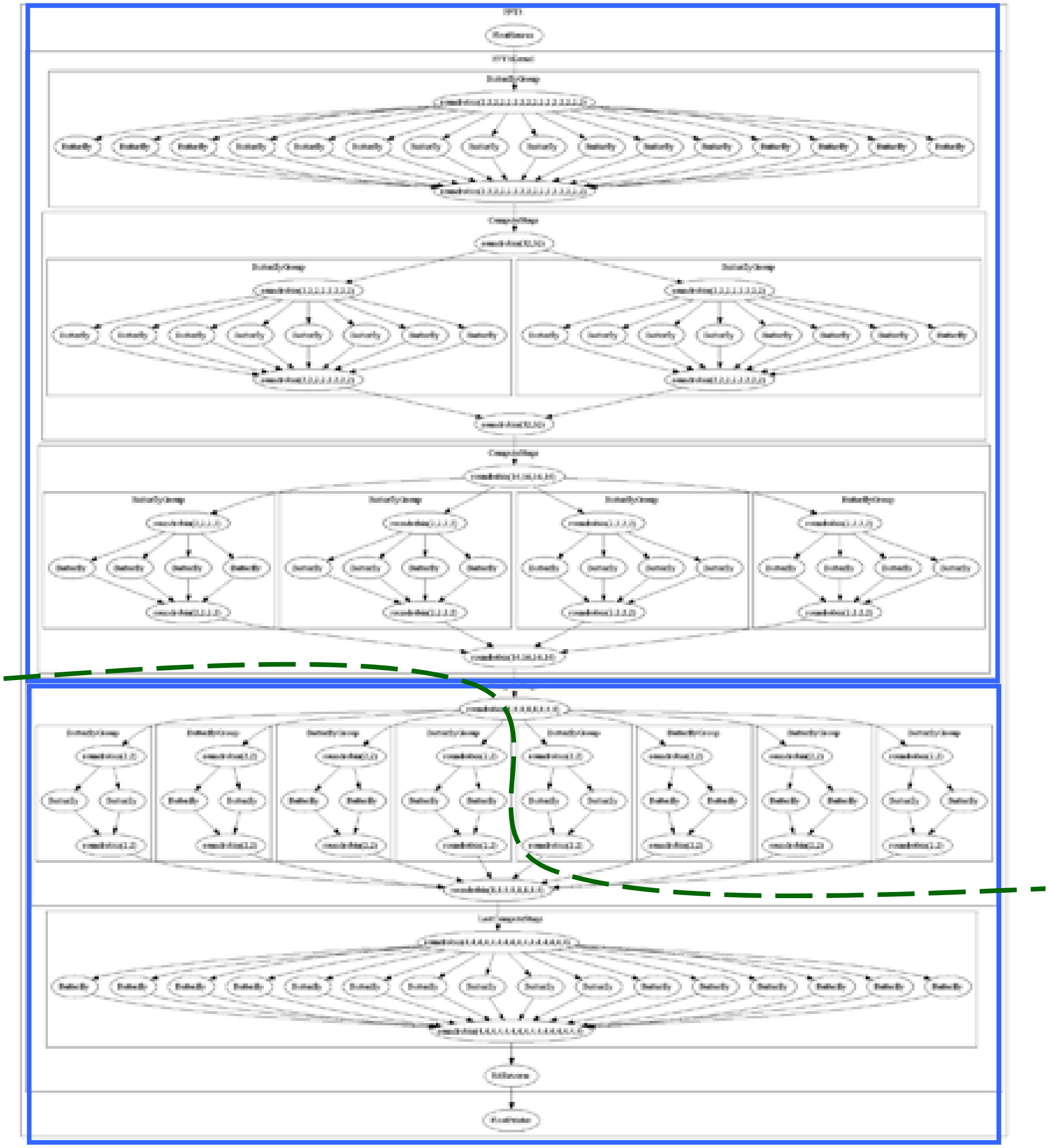
-
-

39%

45%

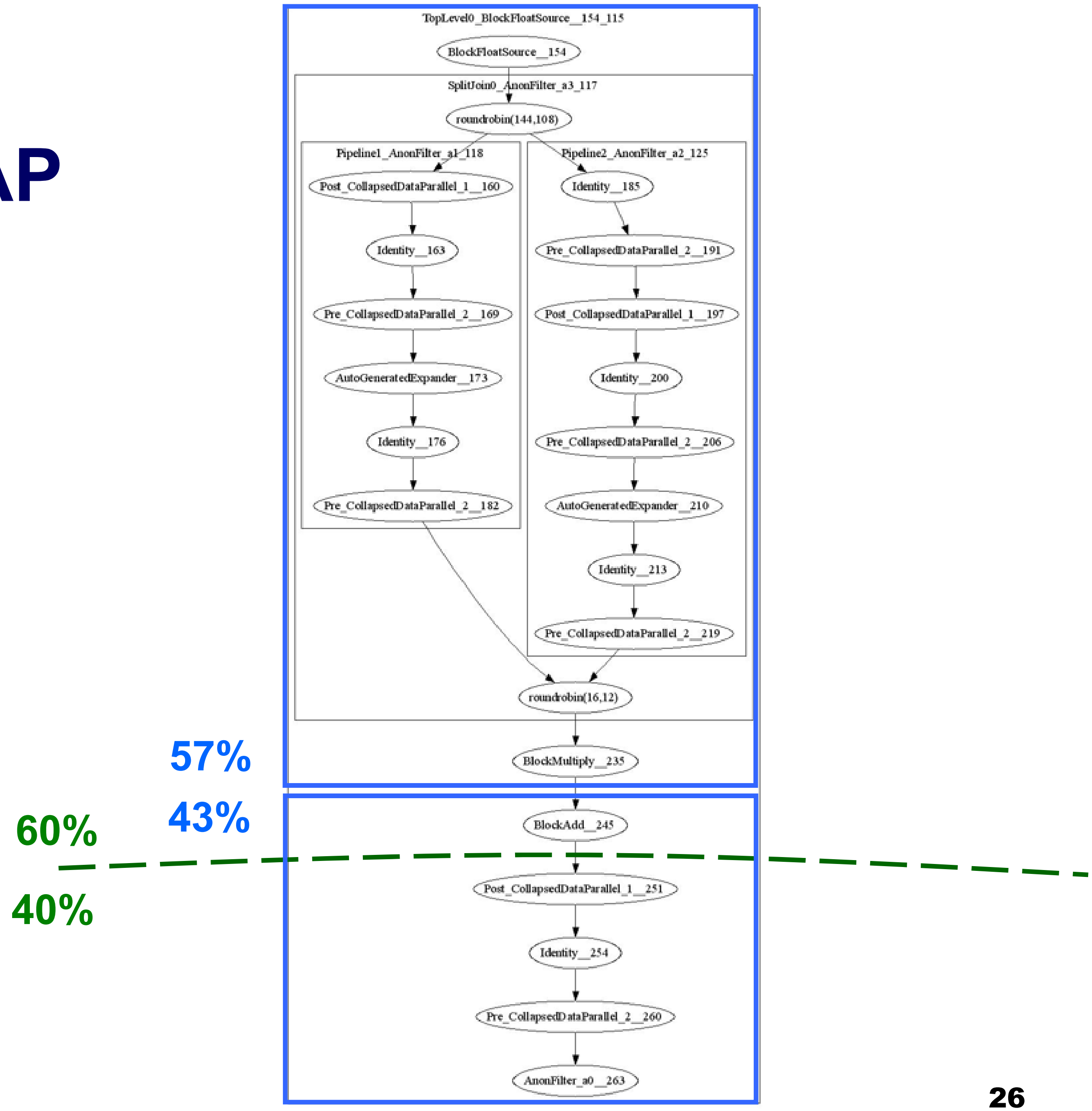
55%

61%



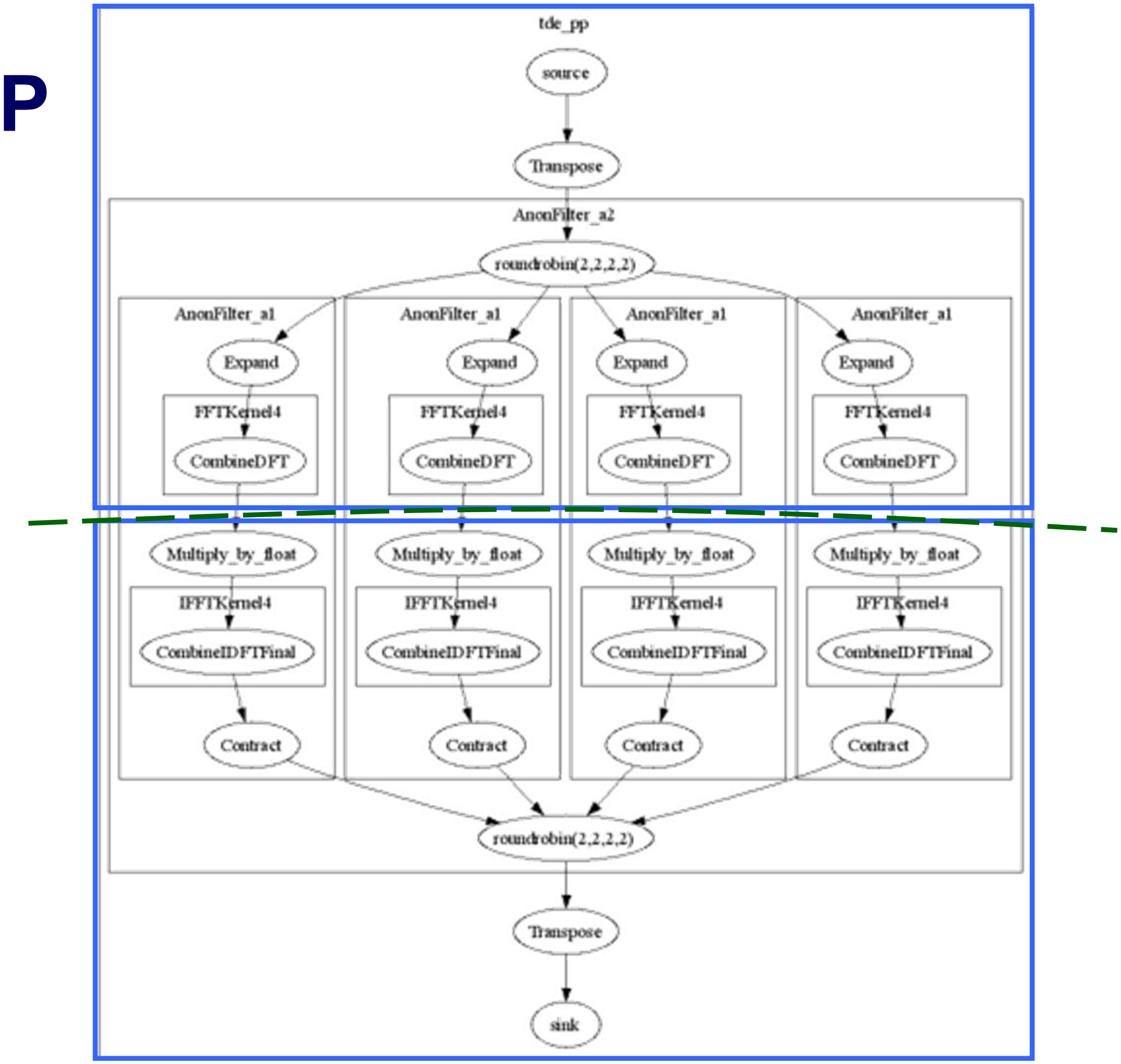
MATMUL

StreamIt / TAP



TDE

StreamIt / TAP

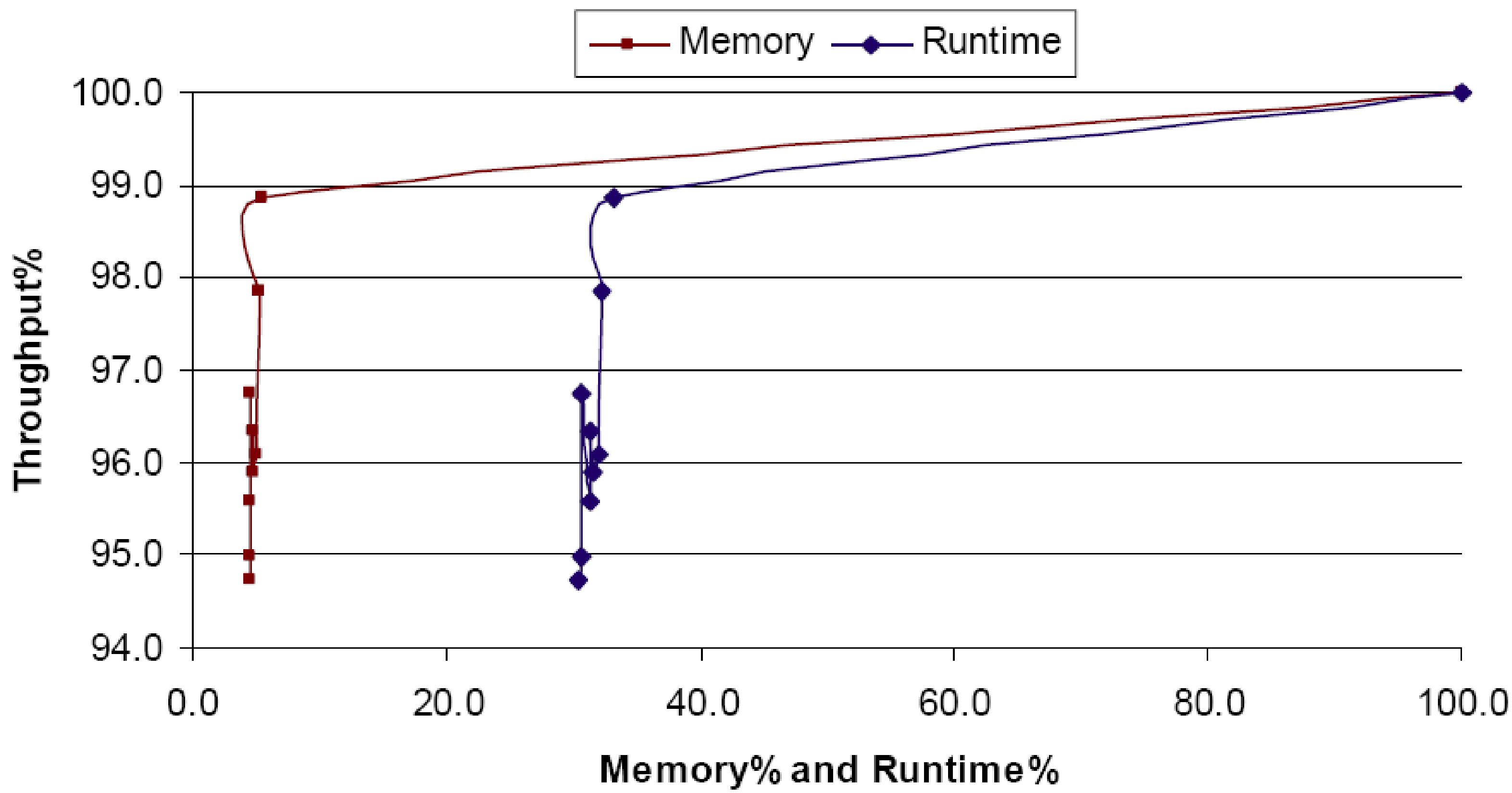


Exact Algorithm

	Runtime (second)	Memory Consumption (MB)	Dual-processor Throughput (K sample / sec.)
BSORT	31.8	2543	319
MATMUL	57.5	321	208
FFT(64)	46.7	2553	470
TDE	76.6	844	933
FILTER	121.5	1366	34.6

Throughput of dual-processor hardware when using the exact partitioning algorithm. It requires the mentioned time and memory to perform.

Approximate Algorithm



Throughput degradation versus reduction in runtime and memory consumptions when using the approximate partitioning algorithm. All values are normalized against the exact algorithm.

questions?