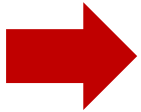
Coarse-Grained Reconfigurable Architectures and Plasticine

Raghu Prabhakar

CS 217 Lecture
Stanford University

Three Important Trends

1. Big data
   Advances in ML, Data analytics



Increasing compute requirements

Data Scaling Trends

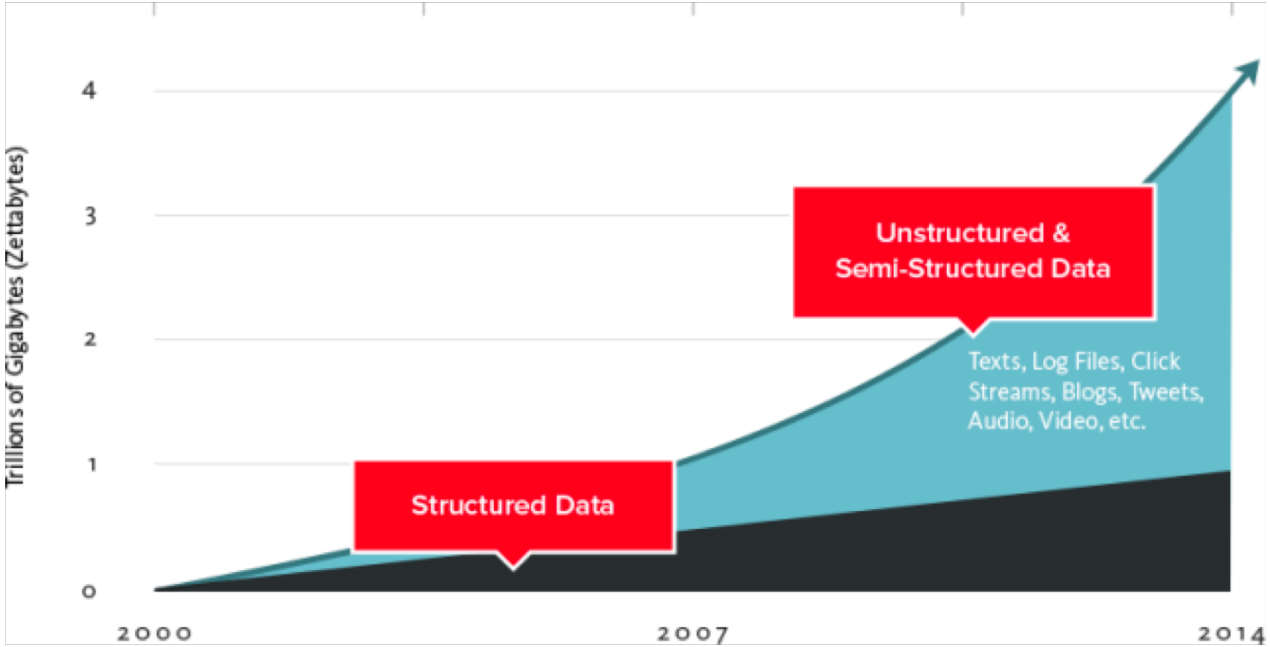


Image credit: Couchbase

- Challenge: Data-driven discovery. Mine, search, analyze data in real time
- Increasing demands for compute, memory

Three Important Trends

1. Big data
   Advances in ML, Data analytics
   → Increasing compute requirements

2. Moore's Law
   Dennard Scaling
   Power Wall, Memory Wall
   → **Specialization**: Use transistors efficiently
   Performance / Watt is key

The Scaling Promise of Moore's Law

2004

3.8 GHz

1 core

90 nm

22 nm

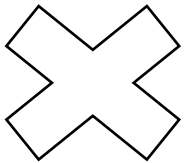
The Scaling Promise of Moore's Law

2004

3.8 GHz

1 core

90 nm



2012, Xistors are:

4x faster

16x more plentiful

22 nm

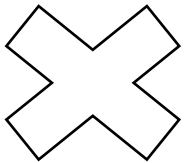
The Scaling Promise of Moore's Law

2004

3.8 GHz

1 core

90 nm

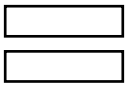


2012, Xistors are:

4x faster

16x more plentiful

22 nm



2012

15.2 GHz

16 Cores

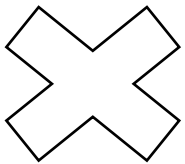
The Scaling Promise of Moore's Law

2004

3.8 GHz

1 core

90 nm

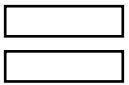


2012, Xistors are:

4x faster

16x more plentiful

22 nm



2012

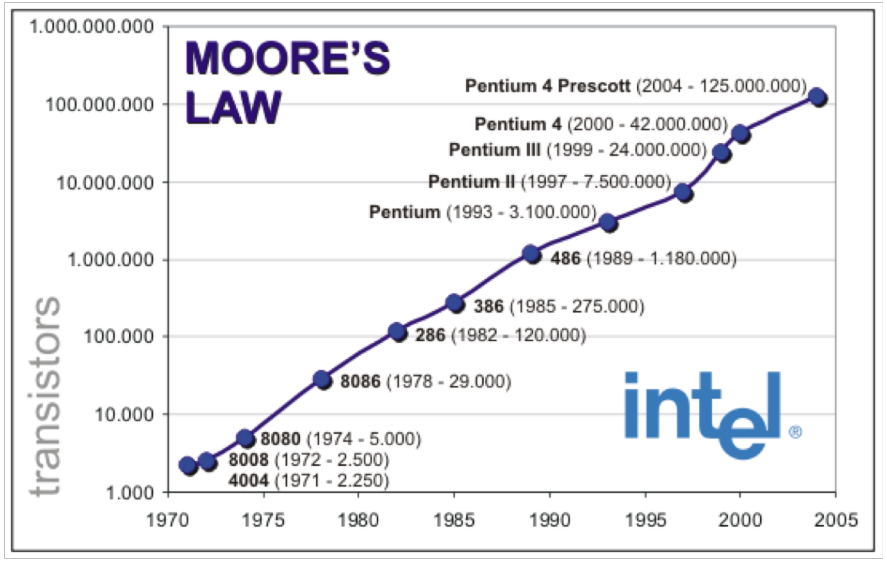
~~15.2~~ GHz

~~16~~ Cores

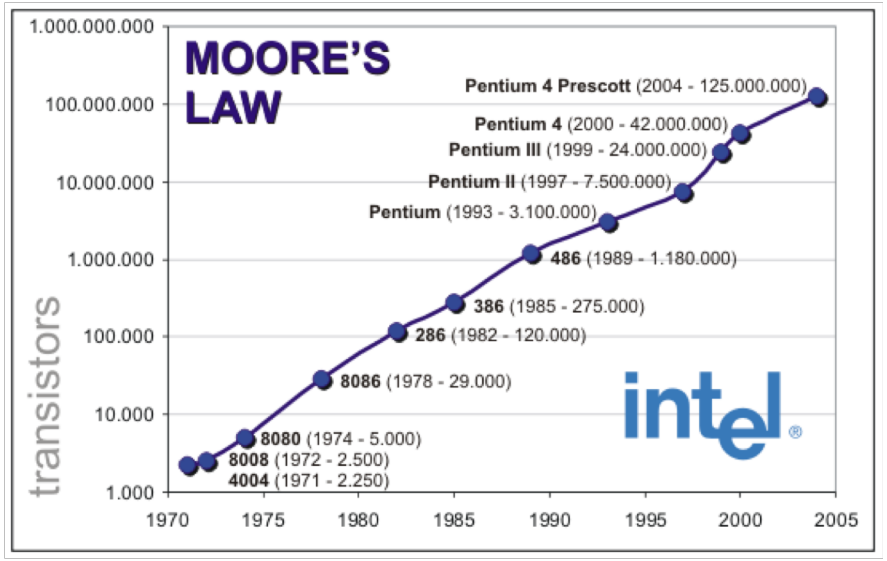
3.6

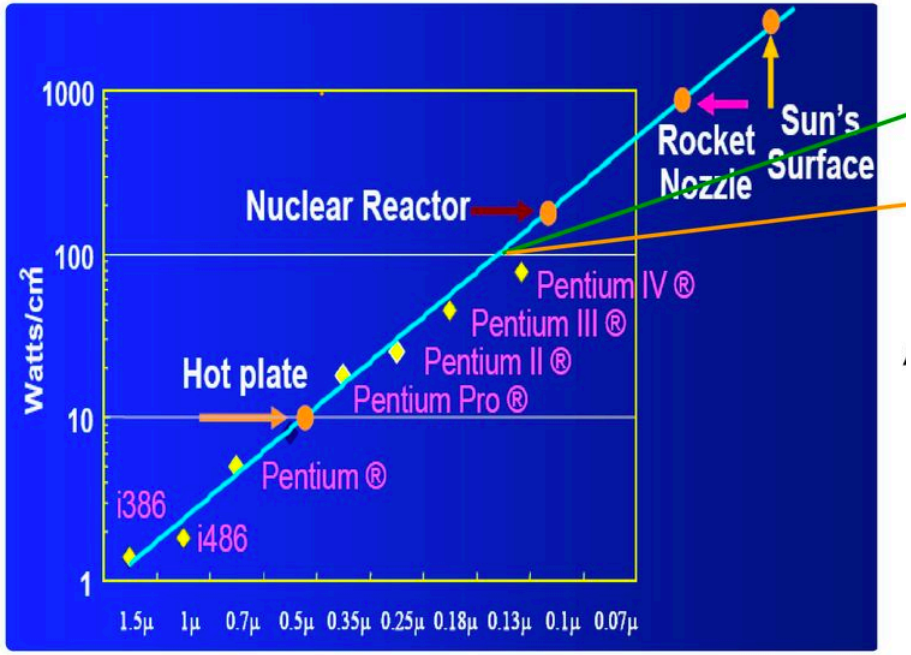
6

The Scaling Promise of Moore's Law



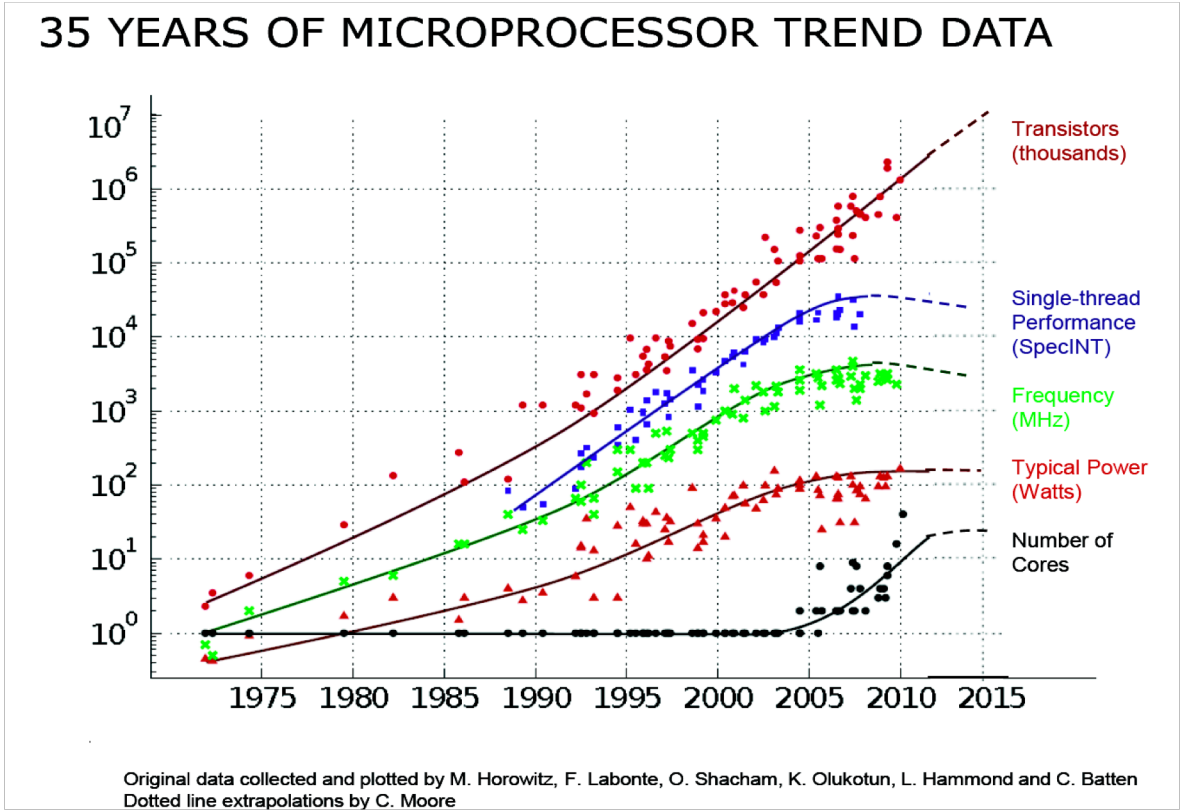
Moore's Law - Power Wall





Source: Shekhar Borkar, Intel

Microprocessor Trends - Multicores

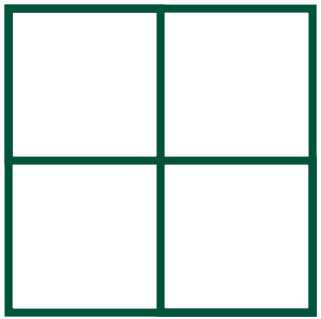


Multicore has hit the Utilization Wall

Spectrum of tradeoffs between # of cores and frequency

Example:
65 nm → 32 nm (S = 2)

4 cores @ 1.8 GHz



65 nm

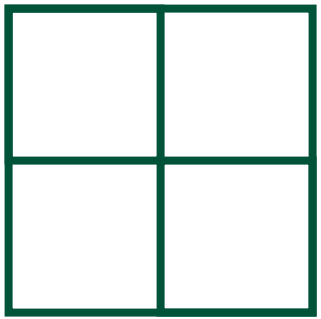
[Taylor, Hotchips 2010]
[Esmaeilzadeh ISCA 2011]

Multicore has hit the Utilization Wall

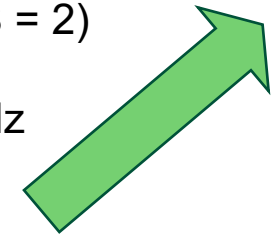
Spectrum of tradeoffs between # of cores and frequency

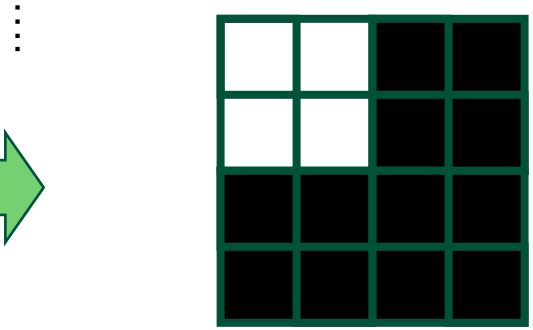
Example:
65 nm → 32 nm (S = 2)

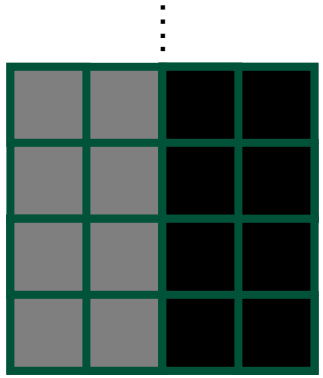
4 cores @ 1.8 GHz



65 nm







4x4 cores @ .9 GHz
(*GPUs of future?)*

2x4 cores @ 1.8 GHz
(8 cores dark, 8 dim)

4 cores @ 2x1.8 GHz
(12 cores dark)

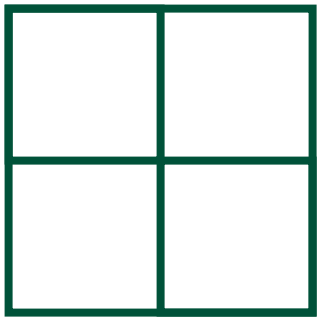
[Taylor, Hotchips 2010]
[Esmaeilzadeh ISCA 2011]

Multicore has hit the Utilization Wall

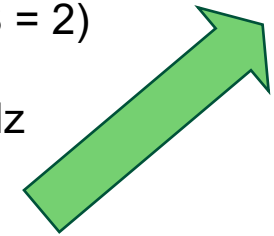
Spectrum of tradeoffs between # of cores and frequency

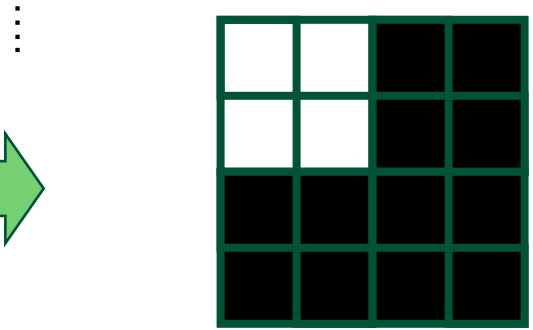
Example:
65 nm → 32 nm (S = 2)

4 cores @ 1.8 GHz

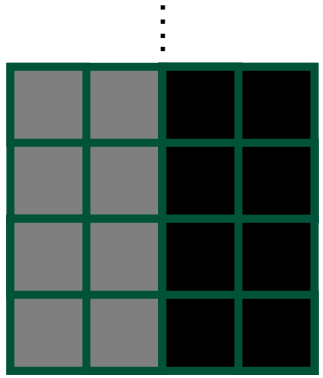


65 nm





4x4 cores @ .9 GHz
(*GPUs of future?)*



2x4 cores @ 1.8 GHz
(8 cores dark, 8 dim)

With each successive process generation, the percentage of a chip that can switch at full frequency drops **exponentially** due to power constraints.

4 cores @ 2x1.8 GHz
(12 cores dark)

[Taylor, Hotchips 2010]
[Esmaeilzadeh ISCA 2011]

ISSCC Chips (22nm – 0.18µm)

Chip	Year	Paper	Description
1	2009	3.8	Dunnington
2	2010	5.7	MSG-Passing
3	2010	5.5	Wire-speed
4	2011	4.4	Godson-3B
5	2013	3.5	Godson-3B1500
6	2011	15.1	Sandy Bridge
7	2012	3.1	Ivy Bridge
8	2011	15.4	Zacate
9	2013	9.4	ARM-v7A

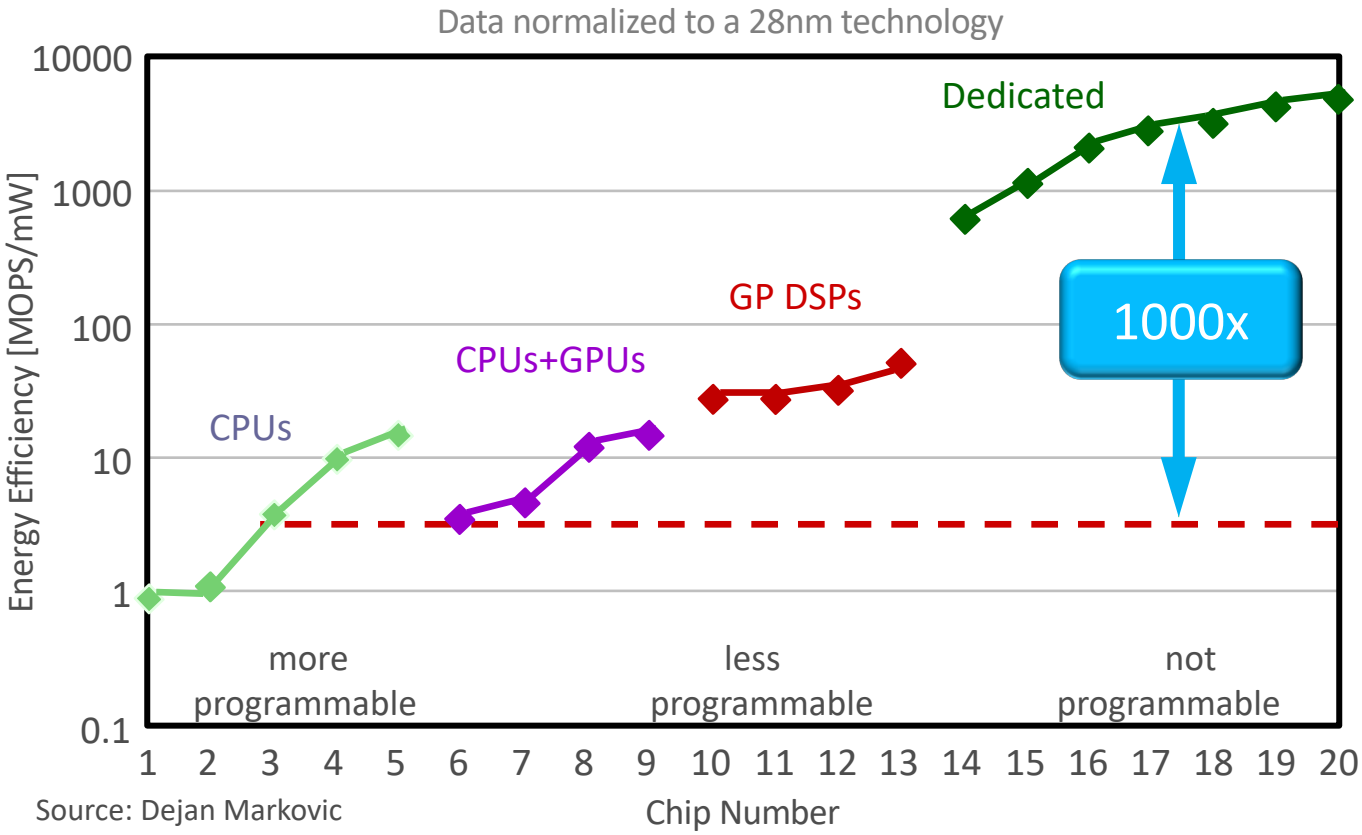
Chip	Year	Paper	Description
10	2012	10.6	3D Proc.
11	2013	9.3	H.264
12	2012	28.8	Razor SIMD
13	2011	7.1	3DTV
14	2011	7.3	Multimedia
15	2011	19.1	ECG/EEG
16	2010	18.4	Obj. Recog.
17	2012	12.4	Obj. Recog.
18	2013	9.8	Obj. Recog.
19	2011	7.4	Neural Net
20	2013	28.2	Visual. Recog.

Chip type: Microprocessor
Microprocessor + GPU
General purpose DSP
Dedicated design

Chips published at ISSCC over a 5-year span

Source: Dejan Markovic

Flexibility vs. Efficiency



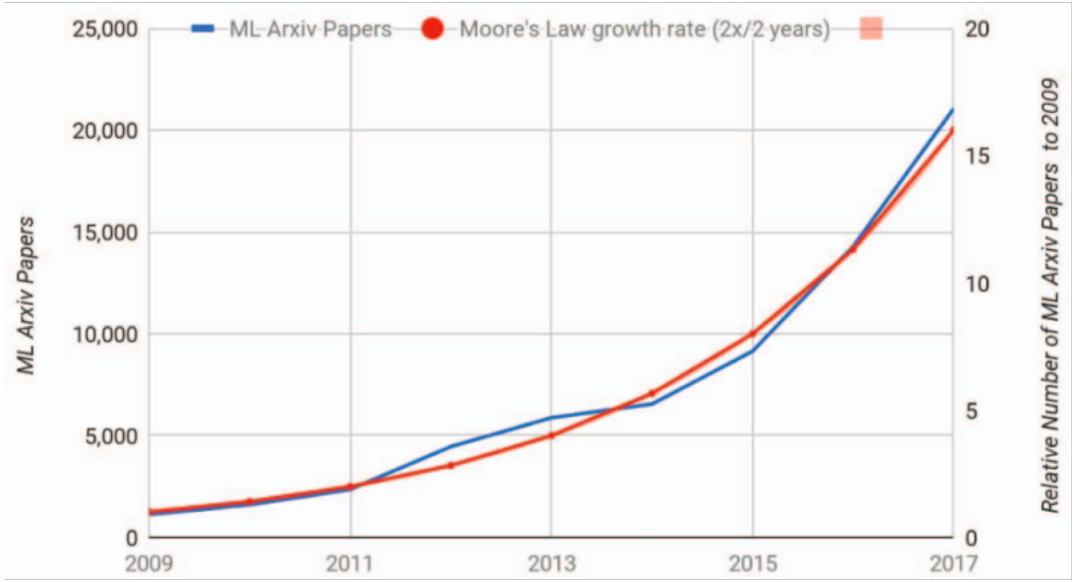
Three Important Trends

1. Big data
   Advances in ML, Data analytics
   → Increasing compute requirements

2. Moore's Law
   Dennard Scaling
   Power Wall, Memory Wall
   → **Specialization**: Use transistors efficiently
   Performance / Watt is key

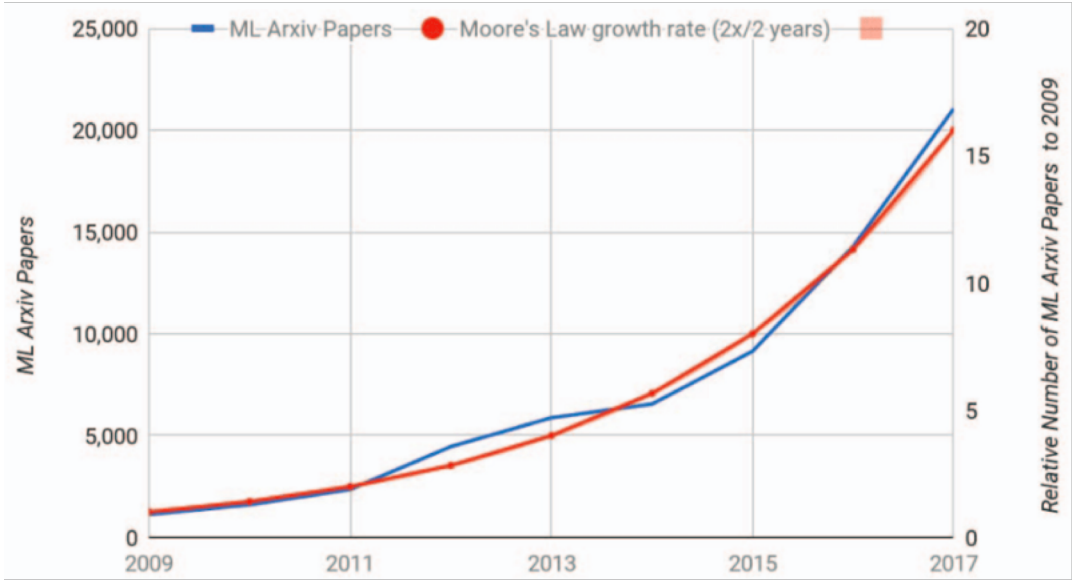
3. Algorithms change rapidly
   High NRE Costs with ASICs
   → **Flexible hardware**

ML Arxiv Papers grows faster than Moore's Law!



Source: Jeff Dean, David Patterson, "A New Golden Age In Computer Architecture", IEEE Micro, 2018

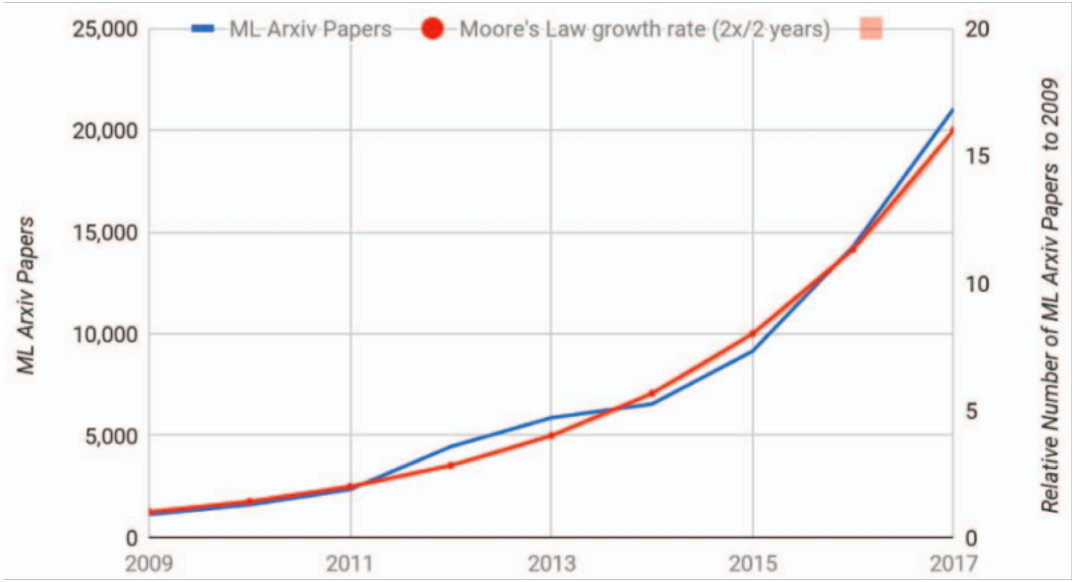
ML Arxiv Papers grows faster than Moore's Law!



Source: Jeff Dean, David Patterson, "A New Golden Age In Computer Architecture", IEEE Micro, 2018

- **Hardware Design Cycle = ~2 years**
  - Inflexible hardware will be obsolete by the time it is built!
- **We need flexible hardware that is efficient**

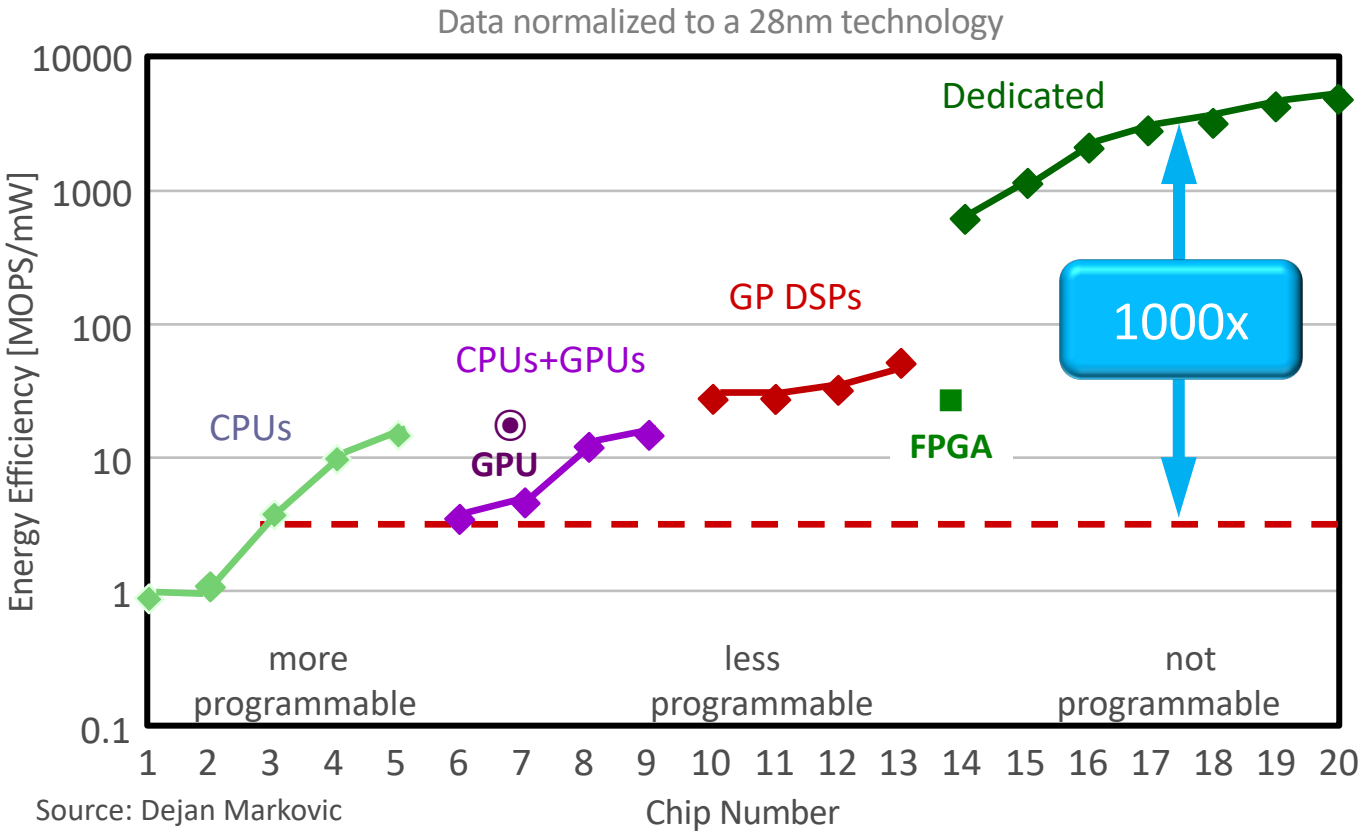
ML Arxiv Papers grows faster than Moore's Law!



- **Hardware Design Cycle = ~2 years**
  - Inflexible hardware will be obsolete by the time it is built!
- **We need flexible hardware that is efficient**
- **What are our choices?**

Source: Jeff Dean, David Patterson, "A New Golden Age In Computer Architecture", IEEE Micro, 2018

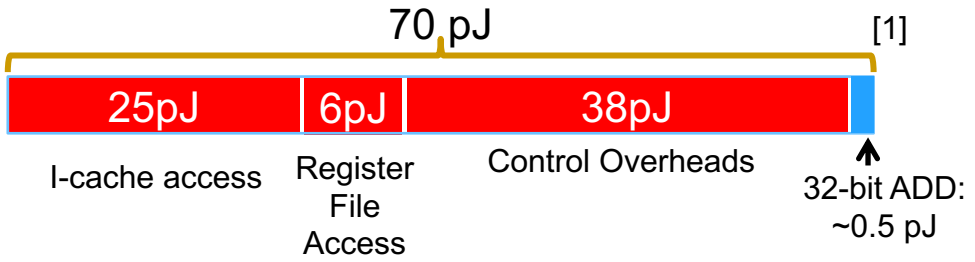
Flexibility vs. Efficiency: GPUs, FPGAs



Flexibility: Instructions

- **CPUs, GPGPUs**

- **Instructions add overheads:**
  - Instruction fetch, decode, register file
  - 40% of datapath energy on CPU[2]
  - 30% of dynamic power on GPU [3]



- **Can we avoid instruction overheads?**

[1] Mark Horowitz, Computing's Energy Problem (and what we can do about it), ISSCC 2014
[2] Hameed et al, Understanding Sources of Inefficiency in General-purpose Chips, ISCA 2010
[3] Leng et al, GPUWattch: Enabling Energy Optimizations in GPGPUs, ISCA 2013

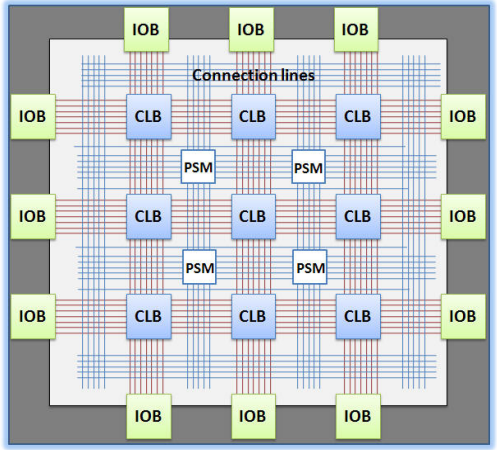
Flexibility: Reconfigurable Hardware

- **FPGAs, CGRAs**
- **Sea of reconfigurable elements**
- **Programmable interconnect**
- **Statically programmed**
- **No instruction overheads**
  - But, granularity of configurability matters

FPGAs

- **Bit-level reconfigurable logic elements**
- **Static interconnect**

- **Flexibility**
- **Performance / Watt**
- **Mature toolchain**

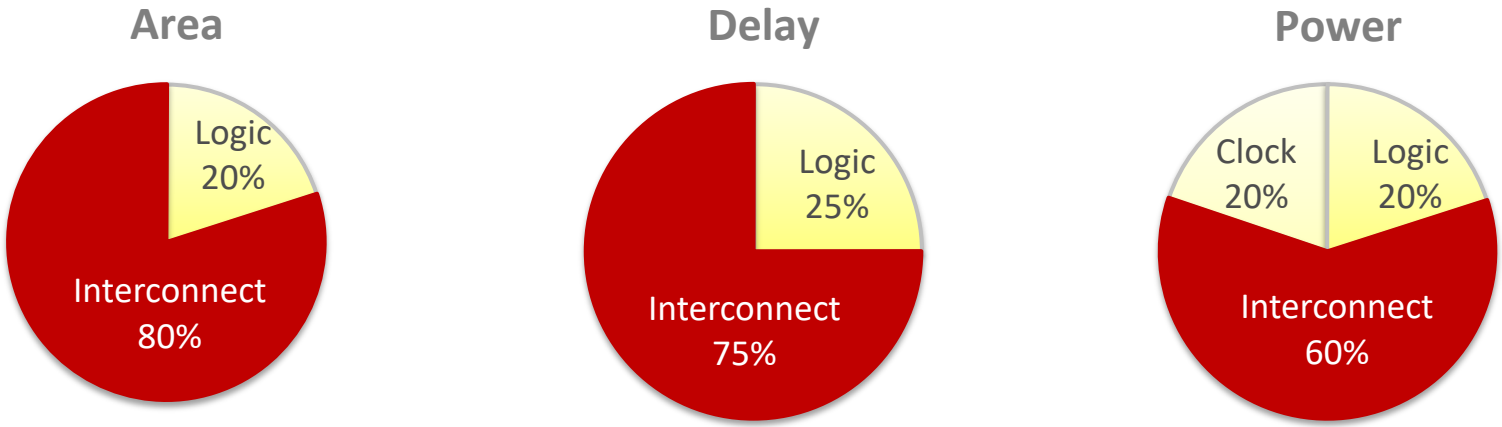


Baidu 百度 Microsoft amazon web services intel FPGA

Great ! But …

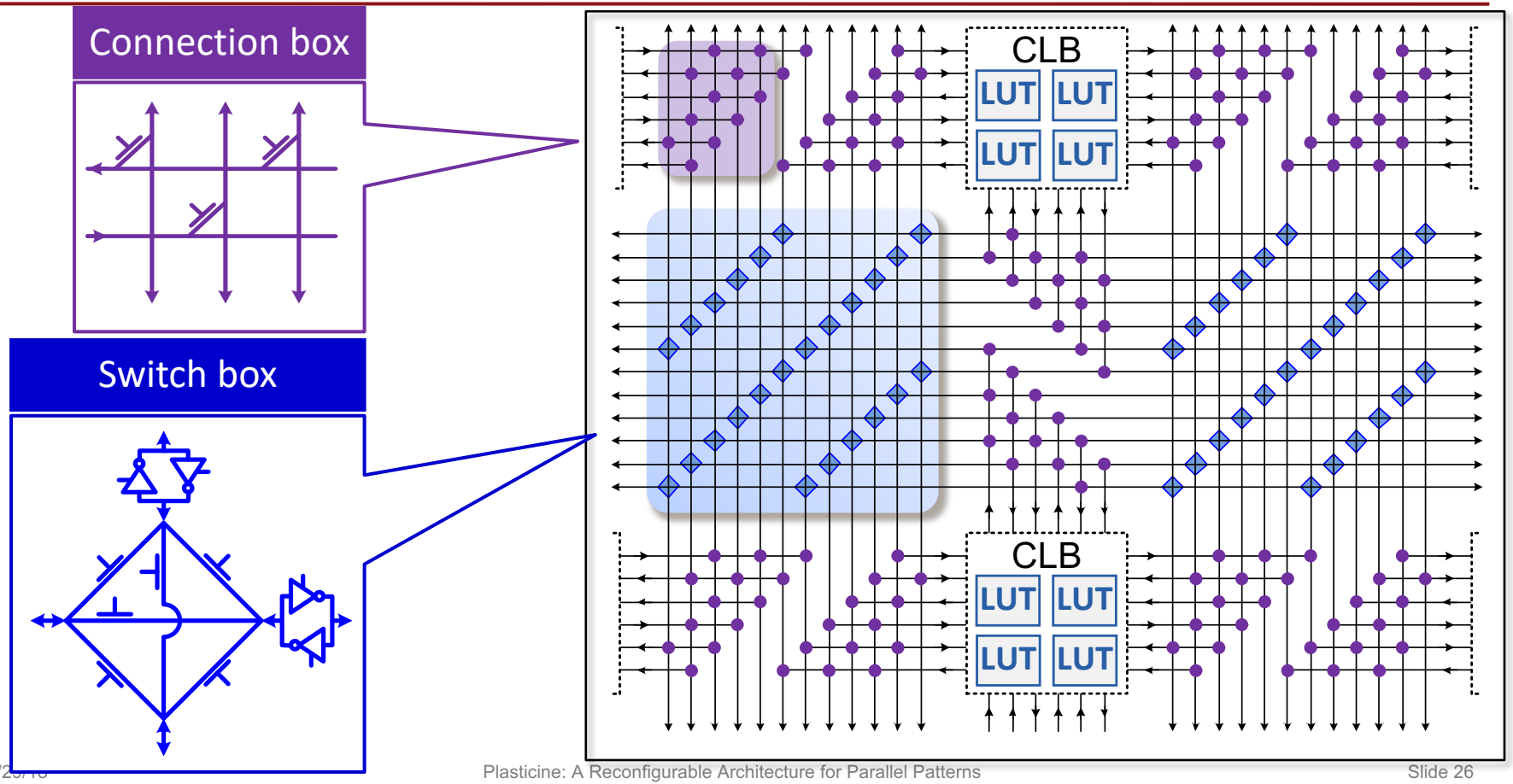
FPGAs are Flexible but Inefficient

- **Compared to dedicated chips, FPGAs have**
  - Area (17x – 54x)
  - Speed (3x – 7x)
  - Power (6x – 62x)
- **It's the interconnect !**



I. Kuon, et al., Found. & Trends in Elec. Design Automation 2007
I. Bolsens, MPSOC 2006; B. Calhoun, et al., Proc. IEEE 2010

FPGA 2D Mesh Interconnect



CGRA: Overcoming FPGA Limitations

- **Fine-grained reconfigurability introduces overheads**
  - Much higher area, delay and power vs. standard cell ASIC
  - Introduce coarse-grained building blocks, much less interconnect
- **Programmability**
  - Fine grained architecture leads to long place and route times (> 2 hours)
  - Not a good computing substrate target for a compiler
- **CGRA architecture**
  - New reconfigurable architectures and new compiler technology
  - How do we go about designing a CGRA?

Several CGRAs have been proposed

Coarse grain reconfigurable architecture (embedded tutorial), Reiner Hartenstein, ASP-DAC 2001

Project	first publ.	Source	Architecture	Granularity	Fabrics	Mapping	intended target application
PADDI	1990	[24]	crossbar	16 bit	central crossbar	routing	DSP
PADDI-2	1993	[26]	crossbar	16 bit	multiple crossbar	routing	DSP and others
DP-FPGA	1994	[3]	2-D array	1 & 4 bit, multi-granular	inhomogenous routing channels	switchbox routing	regular datapaths
KressArray	1995	[4] [9]	2-D mesh	family: select pathwidth	multiple NN & bus segments	(co-)compilation	(adaptable)
Colt	1996	[10]	2-D array	1 & 16 bit inhomogenous	(sophisticated)	run time reconfiguration	highly dynamic reconfig.
RaPID	1996	[21]	1-D array	16 bit	segmented buses	channel routing	pipelining
Matrix	1996	[12]	2-D mesh	8 bit, multi-granular	8NN, length 4 & global lines	multi-length	general purpose
RAW	1997	[14]	2-D mesh	8 bit, multi-granular	8NN switched connections	switchbox rout	experimental
Garp	1997	[13]	2-D mesh	2 bit	global & semi-global lines	heuristic routing	loop acceleration
Pleiades	1997	[27]	mesh / crossbar	multi-granular	multiple segmented crossbar	switchbox routing	multimedia
PipeRench	1998	[23]	1-D array	128 bit	(sophisticated)	scheduling	pipelining
REMARC	1998	[15]	2-D mesh	16 bit	NN & full length buses	(information not available)	multimedia
MorphoSys	1999	[16]	2-D mesh	16 bit	NN, length 2 & 3 global lines	manual P&R	(not disclosed)
CHESS	1999	[17]	hexagon mesh	4 bit, multi-granular	8NN and buses	JHDL compilation	multimedia
DReAM	2000	[18]	2-D array	8 &16 bit	NN, segmented buses	co-compilation	next generation wireless
Chameleon	2000	[19]	2-D array	32 bit	(not disclosed)	co-compilation	tele- & datacommunication
MorphICs	2000	[20]	2-D array	(not disclosed)	(not disclosed)	(not disclosed)	next generation wireless

2D Mesh Architectures

Coarse grain reconfigurable architecture (embedded tutorial), Reiner Hartenstein, ASP-DAC 2001

Project	first publ.	Source	Architecture	Granularity	Fabrics	Mapping	intended target application
PADDI	1990	[24]	crossbar	16 bit	central crossbar	routing	DSP
PADDI-2	1993	[26]	crossbar	16 bit	multiple crossbar	routing	DSP and others
DP-FPGA	1994	[3]	2-D array	1 & 4 bit, multi-granular	inhomogenous routing channels	switchbox routing	regular datapaths
KressArray	1995	[4] [9]	2-D mesh	family: select pathwidth	multiple NN & bus segments	(co-)compilation	(adaptable)
Colt	1996	[10]	2-D array	1 & 16 bit inhomogenous	(sophisticated)	run time reconfiguration	highly dynamic reconfig.
RaPID	1996	[21]	1-D array	16 bit	segmented buses	channel routing	pipelining
Matrix	1996	[12]	2-D mesh	8 bit, multi-granular	8NN, length 4 & global lines	multi-length	general purpose
RAW	1997	[14]	2-D mesh	8 bit, multi-granular	8NN switched connections	switchbox rout	experimental
Garp	1997	[13]	2-D mesh	2 bit	global & semi-global lines	heuristic routing	loop acceleration
Pleiades	1997	[27]	mesh / crossbar	multi-granular	multiple segmented crossbar	switchbox routing	multimedia
PipeRench	1998	[23]	1-D array	128 bit	(sophisticated)	scheduling	pipelining
REMARC	1998	[15]	2-D mesh	16 bit	NN & full length buses	(information not available)	multimedia
MorphoSys	1999	[16]	2-D mesh	16 bit	NN, length 2 & 3 global lines	manual P&R	(not disclosed)
CHESS	1999	[17]	hexagon mesh	4 bit, multi-granular	8NN and buses	JHDL compilation	multimedia
DReAM	2000	[18]	2-D array	8 &16 bit	NN, segmented buses	co-compilation	next generation wireless
Chameleon	2000	[19]	2-D array	32 bit	(not disclosed)	co-compilation	tele- & datacommunication
MorphICs	2000	[20]	2-D array	(not disclosed)	(not disclosed)	(not disclosed)	next generation wireless

Linear Array Architectures

Coarse grain reconfigurable architecture (embedded tutorial), Reiner Hartenstein, ASP-DAC 2001

Project	first publ.	Source	Architecture	Granularity	Fabrics	Mapping	intended target application
PADDI	1990	[24]	crossbar	16 bit	central crossbar	routing	DSP
PADDI-2	1993	[26]	crossbar	16 bit	multiple crossbar	routing	DSP and others
DP-FPGA	1994	[3]	2-D array	1 & 4 bit, multi-granular	inhomogenous routing channels	switchbox routing	regular datapaths
KressArray	1995	[4] [9]	2-D mesh	family: select pathwidth	multiple NN & bus segments	(co-)compilation	(adaptable)
Colt	1996	[10]	2-D array	1 & 16 bit inhomogenous	(sophisticated)	run time reconfiguration	highly dynamic reconfig.
RaPID	1996	[21]	1-D array	16 bit	segmented buses	channel routing	pipelining
Matrix	1996	[12]	2-D mesh	8 bit, multi-granular	8NN, length 4 & global lines	multi-length	general purpose
RAW	1997	[14]	2-D mesh	8 bit, multi-granular	8NN switched connections	switchbox rout	experimental
Garp	1997	[13]	2-D mesh	2 bit	global & semi-global lines	heuristic routing	loop acceleration
Pleiades	1997	[27]	mesh / crossbar	multi-granular	multiple segmented crossbar	switchbox routing	multimedia
PipeRench	1998	[23]	1-D array	128 bit	(sophisticated)	scheduling	pipelining
REMARC	1998	[15]	2-D mesh	16 bit	NN & full length buses	(information not available)	multimedia
MorphoSys	1999	[16]	2-D mesh	16 bit	NN, length 2 & 3 global lines	manual P&R	(not disclosed)
CHESS	1999	[17]	hexagon mesh	4 bit, multi-granular	8NN and buses	JHDL compilation	multimedia
DReAM	2000	[18]	2-D array	8 &16 bit	NN, segmented buses	co-compilation	next generation wireless
Chameleon	2000	[19]	2-D array	32 bit	(not disclosed)	co-compilation	tele- & datacommunication
MorphICs	2000	[20]	2-D array	(not disclosed)	(not disclosed)	(not disclosed)	next generation wireless

Hexagon-D Mesh Architectures

Coarse grain reconfigurable architecture (embedded tutorial), Reiner Hartenstein, ASP-DAC 2001

Project	first publ.	Source	Architecture	Granularity	Fabrics	Mapping	intended target application
PADDI	1990	[24]	crossbar	16 bit	central crossbar	routing	DSP
PADDI-2	1993	[26]	crossbar	16 bit	multiple crossbar	routing	DSP and others
DP-FPGA	1994	[3]	2-D array	1 & 4 bit, multi-granular	inhomogenous routing channels	switchbox routing	regular datapaths
KressArray	1995	[4] [9]	2-D mesh	family: select pathwidth	multiple NN & bus segments	(co-)compilation	(adaptable)
Colt	1996	[10]	2-D array	1 & 16 bit inhomogenous	(sophisticated)	run time reconfiguration	highly dynamic reconfig.
RaPID	1996	[21]	1-D array	16 bit	segmented buses	channel routing	pipelining
Matrix	1996	[12]	2-D mesh	8 bit, multi-granular	8NN, length 4 & global lines	multi-length	general purpose
RAW	1997	[14]	2-D mesh	8 bit, multi-granular	8NN switched connections	switchbox rout	experimental
Garp	1997	[13]	2-D mesh	2 bit	global & semi-global lines	heuristic routing	loop acceleration
Pleiades	1997	[27]	mesh / crossbar	multi-granular	multiple segmented crossbar	switchbox routing	multimedia
PipeRench	1998	[23]	1-D array	128 bit	(sophisticated)	scheduling	pipelining
REMARC	1998	[15]	2-D mesh	16 bit	NN & full length buses	(information not available)	multimedia
MorphoSys	1999	[16]	2-D mesh	16 bit	NN, length 2 & 3 global lines	manual P&R	(not disclosed)
CHESS	1999	[17]	hexagon mesh	4 bit, multi-granular	8NN and buses	JHDL compilation	multimedia
DReAM	2000	[18]	2-D array	8 &16 bit	NN, segmented buses	co-compilation	next generation wireless
Chameleon	2000	[19]	2-D array	32 bit	(not disclosed)	co-compilation	tele- & datacommunication
MorphICs	2000	[20]	2-D array	(not disclosed)	(not disclosed)	(not disclosed)	next generation wireless

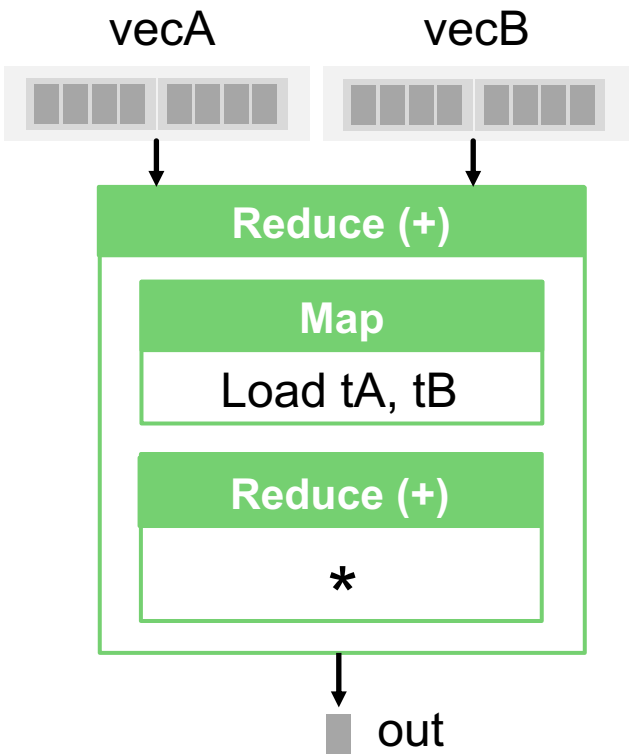
Anatomy of a basic CGRA

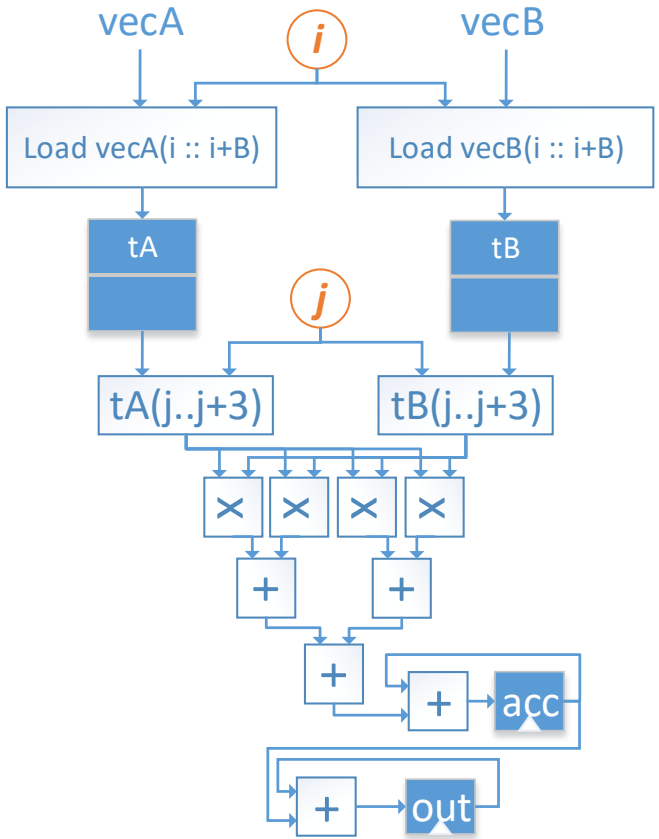
- **Hardware Building blocks: Compute, Memory, Interconnect**
  - Compute: ALUs of varying capability
  - Memory: Programmer-managed scratchpads, caches
  - Interconnect: Statically programmed paths vs. dynamically routed data
- **Hardware Organization: Topology**
  - Data path hierarchy: ALUs vs. clusters of ALUs
  - Communication granularity: bit-level vs. word-level
  - Interconnect topology: Mesh, Torus, …
- **Software: Programming Model**
  - Software abstraction: Threads, VLIW, spatially configurable ALUs, …
  - Compiler technology to map high-level applications to CGRAs

Top-Down Design of a CGRA

- **Observation: We can abstract key software constructs that are amenable to hardware acceleration**
  - Nested data and pipeline parallelism
  - Data locality
- **Parallel Patterns: Software abstractions that capture parallelism and locality**
  - Loops with special properties
  - Map, reduce, zip, filter, groupBy, etc
  - Expressive over wide range of domains (ML, SQL, Graph analytics, etc)
  - Enables building optimizing compilers with aggressive compiler optimizations
- **Design a CGRA to accelerate parallel patterns**

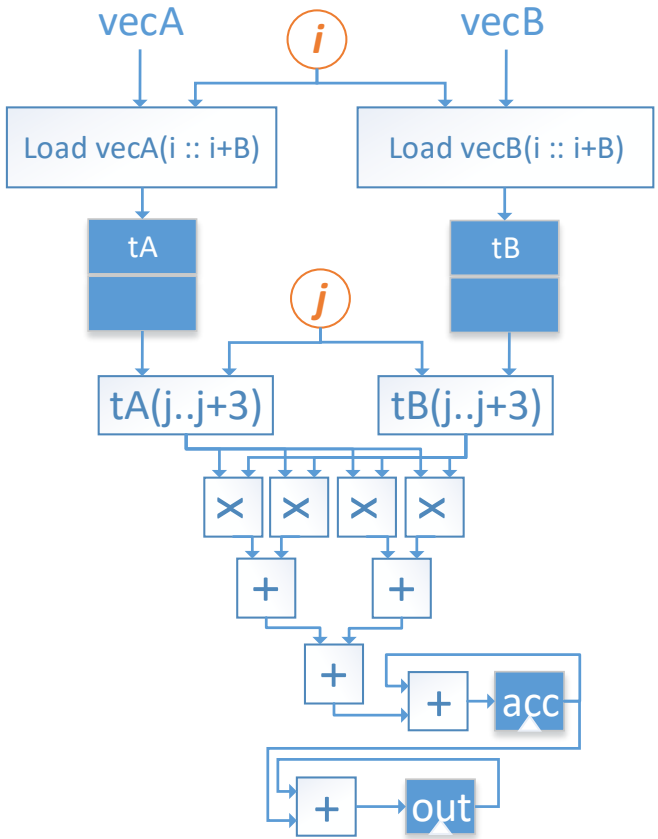
Example: Dot Product





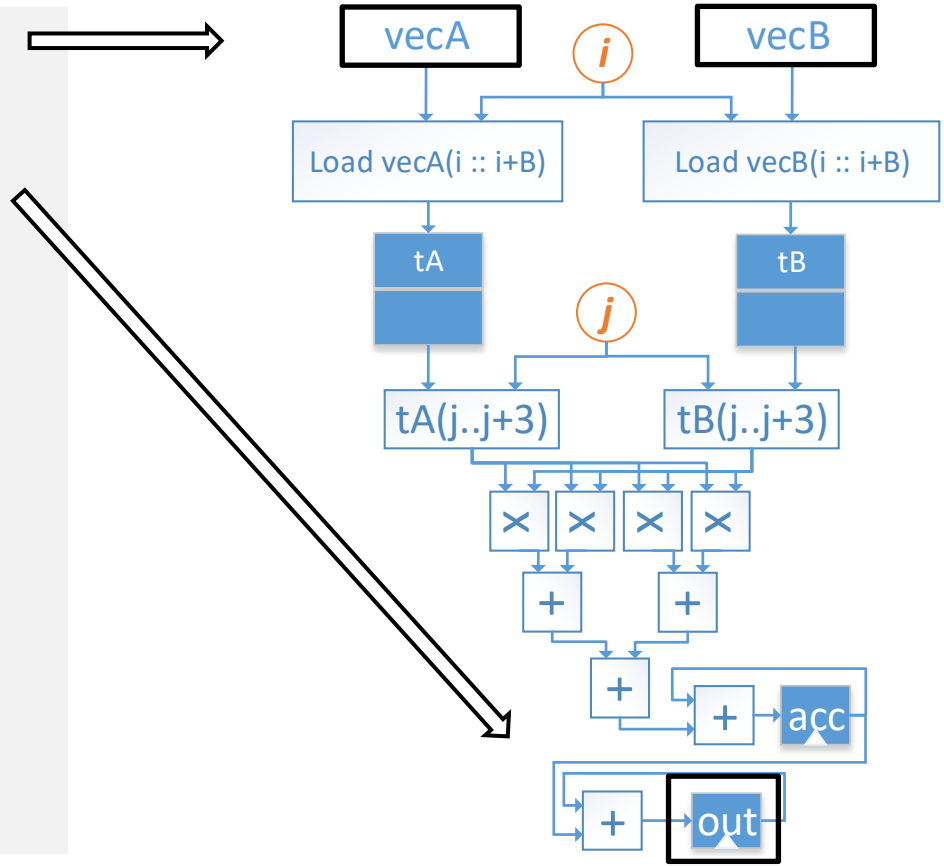
Example: Dot Product

```
val vecA    = DRAM[Float](N)
val vecB    = DRAM[Float](N)
val out     = Reg[Float]

Reduce(N by B)(out) { i =>
   val tA    = SRAM[Float](B)
   val tB    = SRAM[Float](B)
   val acc   = Reg[Float]

   tA load vecA(i :: i+B)
   tB load vecB(i :: i+B)

   Reduce(B by 1)(acc){ j =>
      tA(j) * tB(j)
   }{_+_}
}{_+_}
```



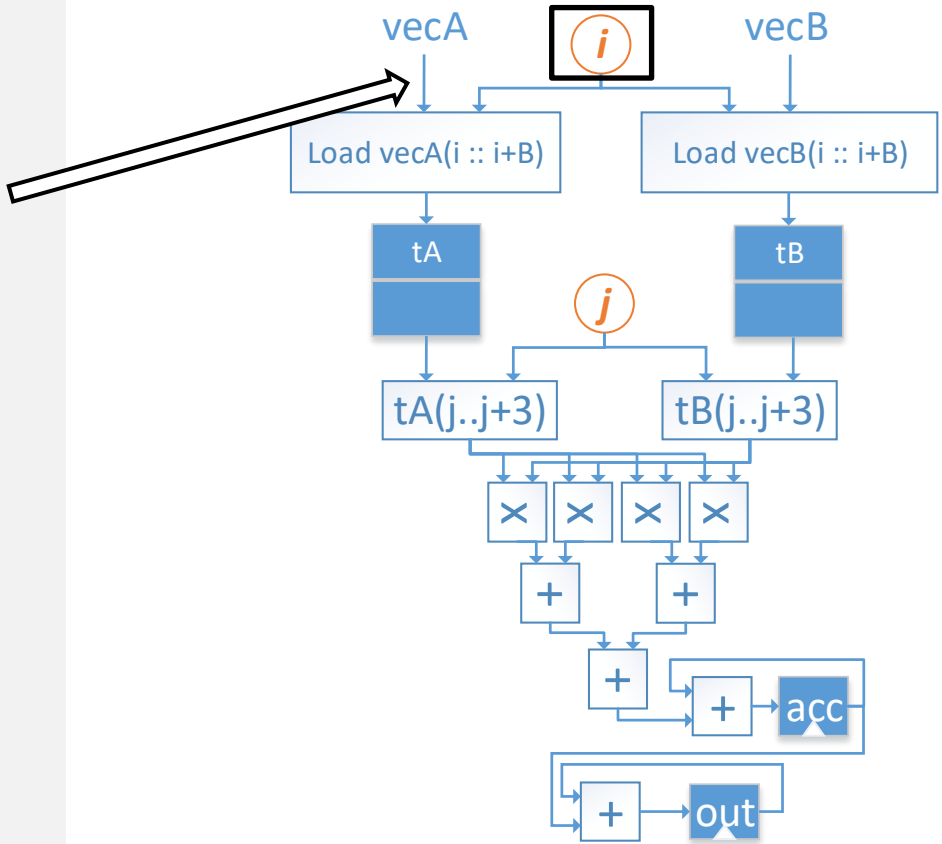
Example: Dot Product

```
val vecA    = DRAM[Float](N)
val vecB    = DRAM[Float](N)
val out     = Reg[Float]

Reduce(N by B)(out) { i =>
   val tA    = SRAM[Float](B)
   val tB    = SRAM[Float](B)
   val acc   = Reg[Float]

   tA load vecA(i :: i+B)
   tB load vecB(i :: i+B)

   Reduce(B by 1)(acc){ j =>
      tA(j) * tB(j)
   }{_+_}
}{_+_}
```



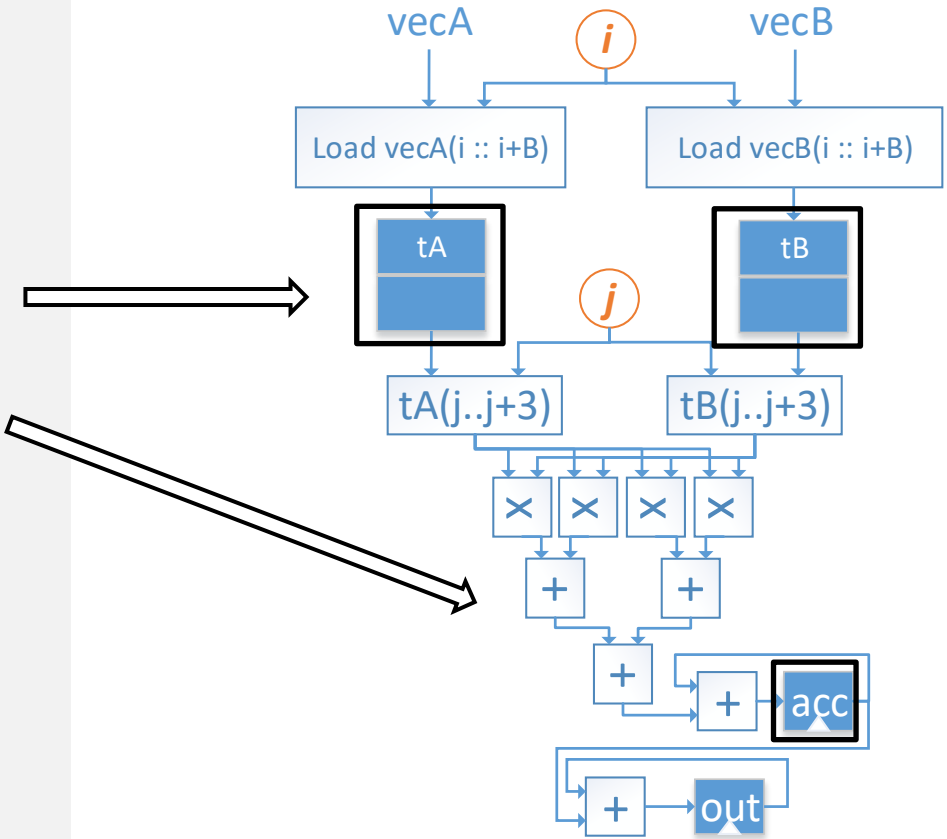
Example: Dot Product

```
val vecA    = DRAM[Float](N)
val vecB    = DRAM[Float](N)
val out     = Reg[Float]

Reduce(N by B)(out) { i =>
   val tA    = SRAM[Float](B)
   val tB    = SRAM[Float](B)
   val acc   = Reg[Float]

   tA load vecA(i :: i+B)
   tB load vecB(i :: i+B)

   Reduce(B by 1)(acc){ j =>
      tA(j) * tB(j)
   }{_+_}
}{_+_}
```



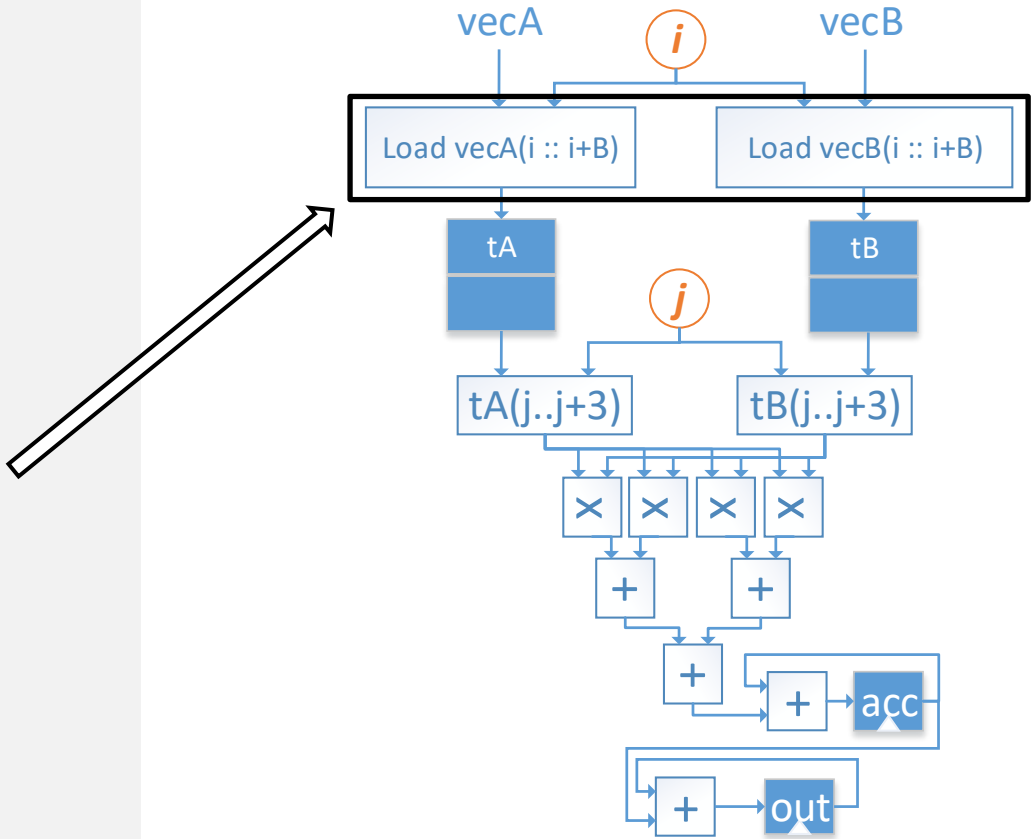
Example: Dot Product

```
val vecA    = DRAM[Float](N)
val vecB    = DRAM[Float](N)
val out     = Reg[Float]

Reduce(N by B)(out) { i =>
   val tA     = SRAM[Float](B)
   val tB     = SRAM[Float](B)
   val acc    = Reg[Float]

   tA load vecA(i :: i+B)
   tB load vecB(i :: i+B)

   Reduce(B by 1)(acc){ j =>
      tA(j) * tB(j)
   }{_+_}
}{_+_}
```



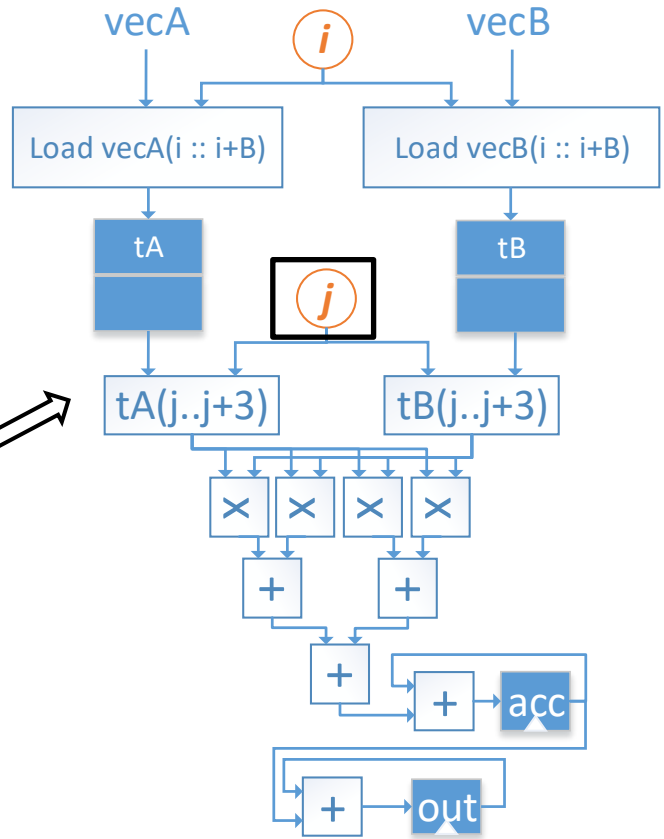
Example: Dot Product

```
val vecA    = DRAM[Float](N)
val vecB    = DRAM[Float](N)
val out     = Reg[Float]

Reduce(N by B)(out) { i =>
   val tA    = SRAM[Float](B)
   val tB    = SRAM[Float](B)
   val acc   = Reg[Float]

   tA load vecA(i :: i+B)
   tB load vecB(i :: i+B)

   Reduce(B by 1)(acc){ j =>
      tA(j) * tB(j)
   }{_+_}
}{_+_}
```



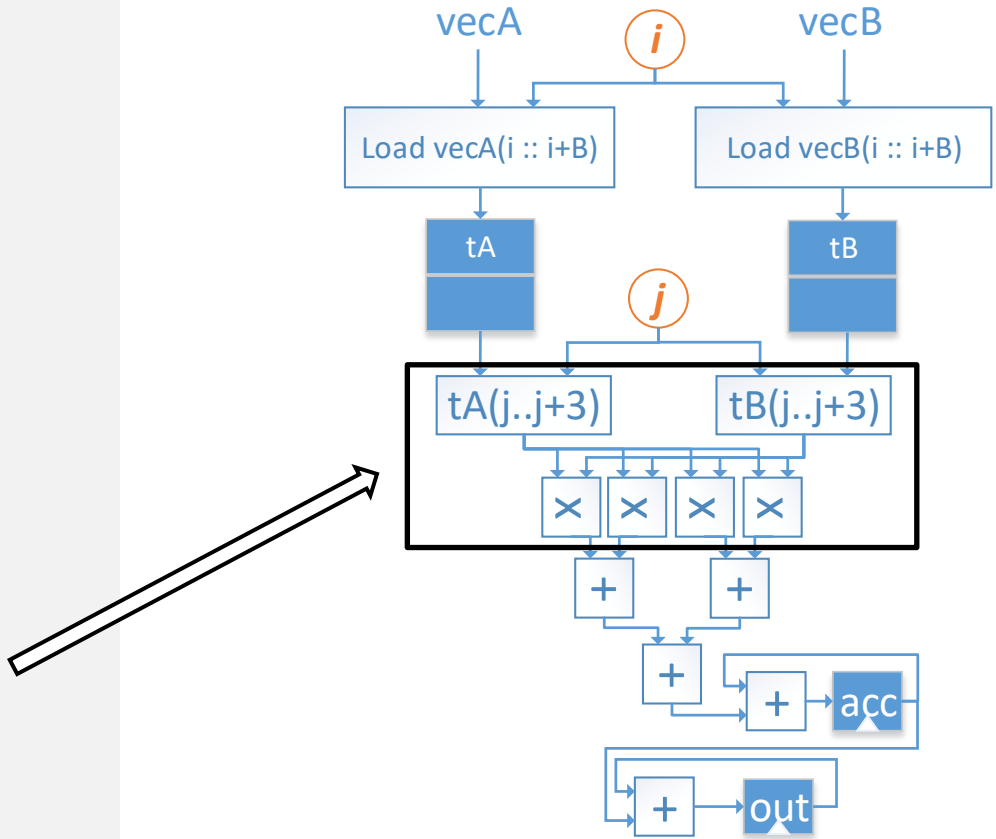
Example: Dot Product

```
val vecA    = DRAM[Float](N)
val vecB    = DRAM[Float](N)
val out     = Reg[Float]

Reduce(N by B)(out) { i =>
   val tA    = SRAM[Float](B)
   val tB    = SRAM[Float](B)
   val acc   = Reg[Float]

   tA load vecA(i :: i+B)
   tB load vecB(i :: i+B)

   Reduce(B by 1)(acc){ j =>
      tA(j) * tB(j)
   }{_+_}
}{_+_}
```



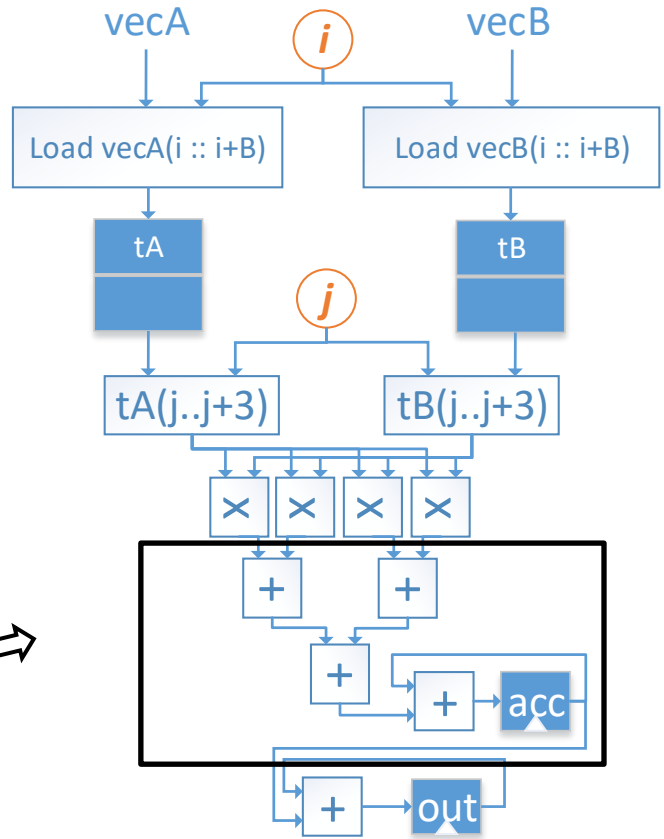
Example: Dot Product

```
val vecA    = DRAM[Float](N)
val vecB    = DRAM[Float](N)
val out     = Reg[Float]

Reduce(N by B)(out) { i =>
   val tA    = SRAM[Float](B)
   val tB    = SRAM[Float](B)
   val acc   = Reg[Float]

   tA load vecA(i :: i+B)
   tB load vecB(i :: i+B)

   Reduce(B by 1)(acc){ j =>
      tA(j) * tB(j)
   }{_+_}
}{_+_}
```



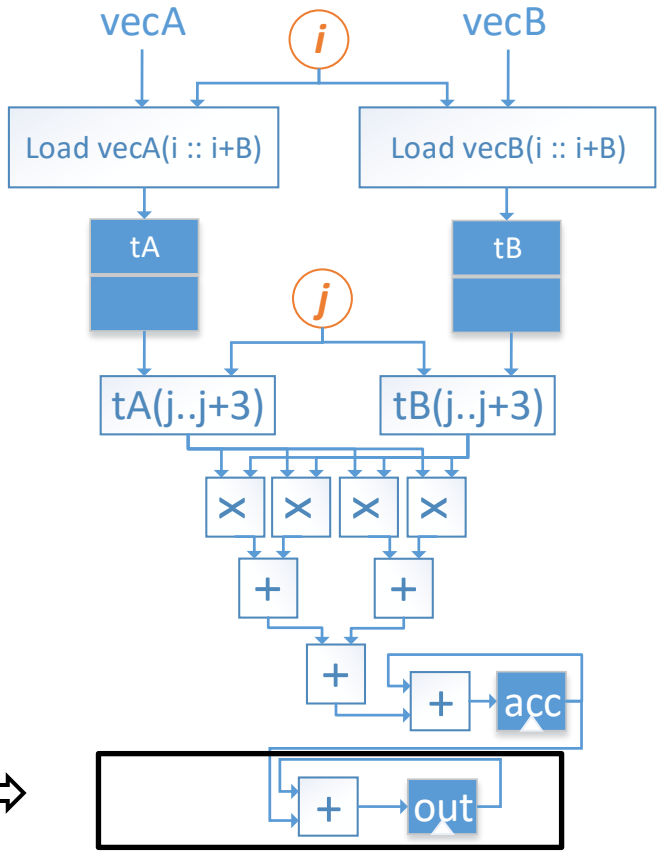
Example: Dot Product

```
val vecA    = DRAM[Float](N)
val vecB    = DRAM[Float](N)
val out     = Reg[Float]

Reduce(N by B)(out) { i =>
   val tA    = SRAM[Float](B)
   val tB    = SRAM[Float](B)
   val acc   = Reg[Float]

   tA load vecA(i :: i+B)
   tB load vecB(i :: i+B)

   Reduce(B by 1)(acc){ j =>
      tA(j) * tB(j)
   }{_+_}
}{_+_}
```

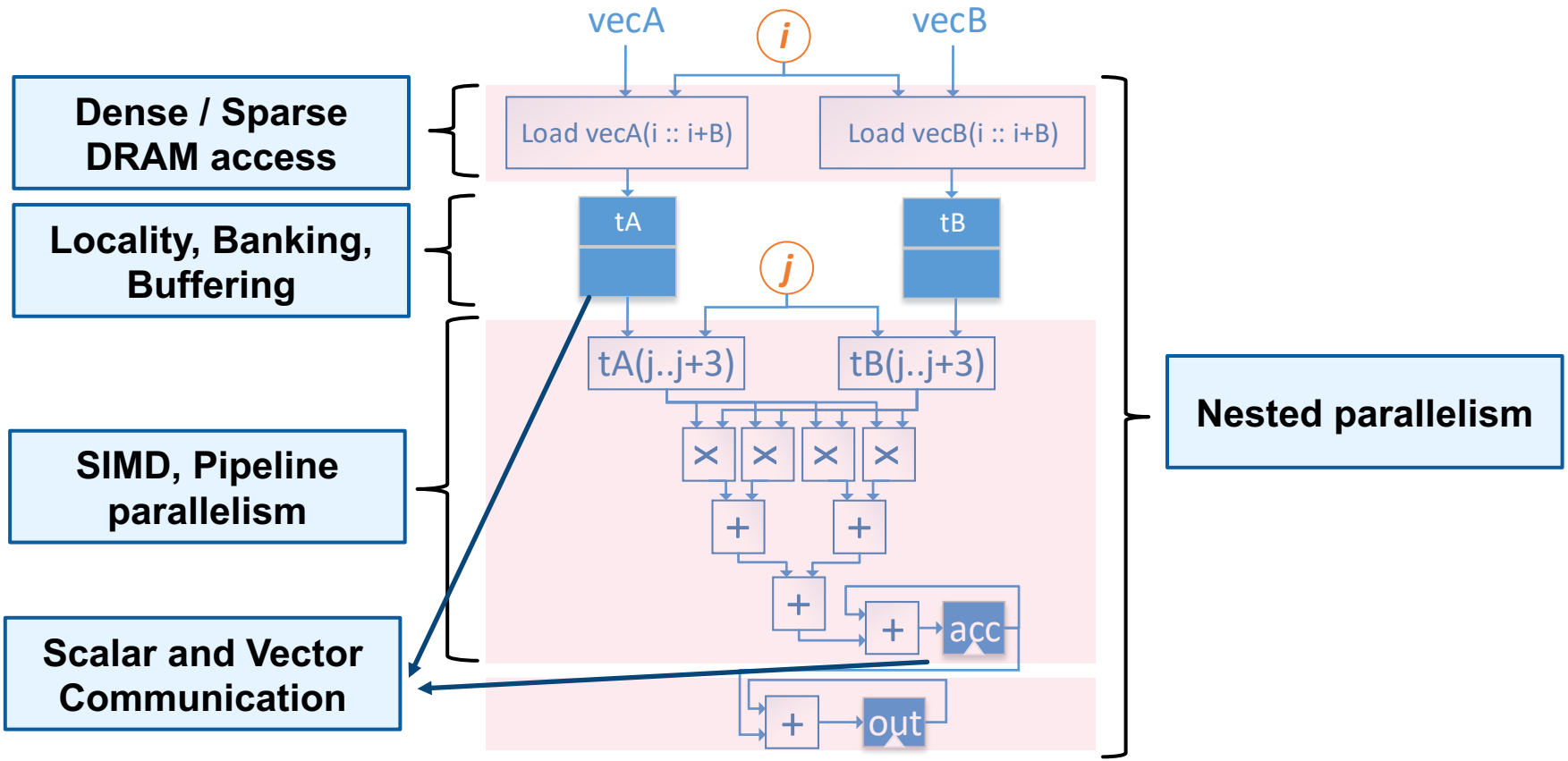


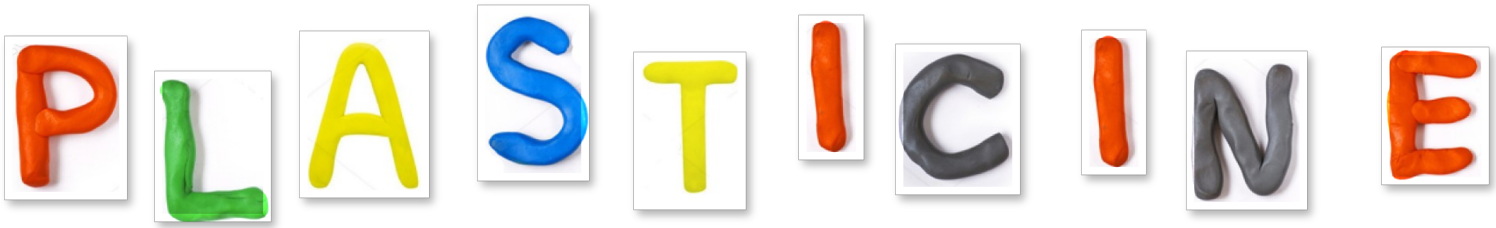
Example: Dot Product

```
val vecA    = DRAM[Float](N)
val vecB    = DRAM[Float](N)
val out     = Reg[Float]

Reduce(N by B)(out) { i =>
   val tA    = SRAM[Float](B)
   val tB    = SRAM[Float](B)
   val acc   = Reg[Float]

   tA load vecA(i :: i+B)
   tB load vecB(i :: i+B)

   Reduce(B by 1)(acc){ j =>
      tA(j) * tB(j)
   }{_+_}
}{_+_}
```



Key Characteristics




PLASTICINE

Plasticine Architecture

- New Reconfigurable Architecture to Accelerate Parallel Patterns
  - Published in ISCA (Int'l Symposium on Computer Architecture) 2017
  - Up to **95x** perf, up to **75x** perf/W vs. Altera Stratix V FPGA

- IEEE Micro Top Pick 2017
  - Appeared in IEEE Micro's May/June special issue of "most significant papers in computer architecture based on novelty and long-term impact" for 2017

- Covered by EETimes: http://www.eetimes.com/document.asp?doc_id=1331781&page_number=2

Plasticine: Key Features

Nested parallelism	**Hierarchical Datapath** **Flexible Control Mechanism**
Locality, Banking, Buffering	**On-chip Scratchpads + Configurable banking** **Address partitioning for double buffering**
Dense / Sparse DRAM access	**Dense access: Dedicated DRAM address generators** **Sparse access: Scatter-gather Units**
Scalar and Vector Communication	**Interconnect with multiple levels of granularity**

Hasn't this been done before?

Previous Work	Nested Parallelism	Configurable Scratchpads	Hierarchical Interconnect	Programmability
ADRES				
DySER				
Garp				
PipeRench				
Tartan				
RaPiD				
HRL				
Triggered Instructions				
Mosaic				
FPGA				

Hasn't this been done before?

Previous Work	Nested Parallelism	Configurable Scratchpads	Hierarchical Interconnect	Programmability
ADRES				
DySER				
Garp				
PipeRench				
Tartan				
RaPiD				
HRL				
Triggered Instructions				
Mosaic				
FPGA				

Hasn't this been done before?

Previous Work	Nested Parallelism	Configurable Scratchpads	Hierarchical Interconnect	Programmability
ADRES				
DySER				
Garp				
PipeRench				
Tartan				
RaPiD				
HRL				
Triggered Instructions				
Mosaic				
FPGA				

Hasn't this been done before?

Previous Work	Nested Parallelism	Configurable Scratchpads	Hierarchical Interconnect	Programmability
ADRES				
DySER				
Garp				
PipeRench				
Tartan				
RaPiD				
HRL				
Triggered Instructions				
Mosaic				
FPGA				

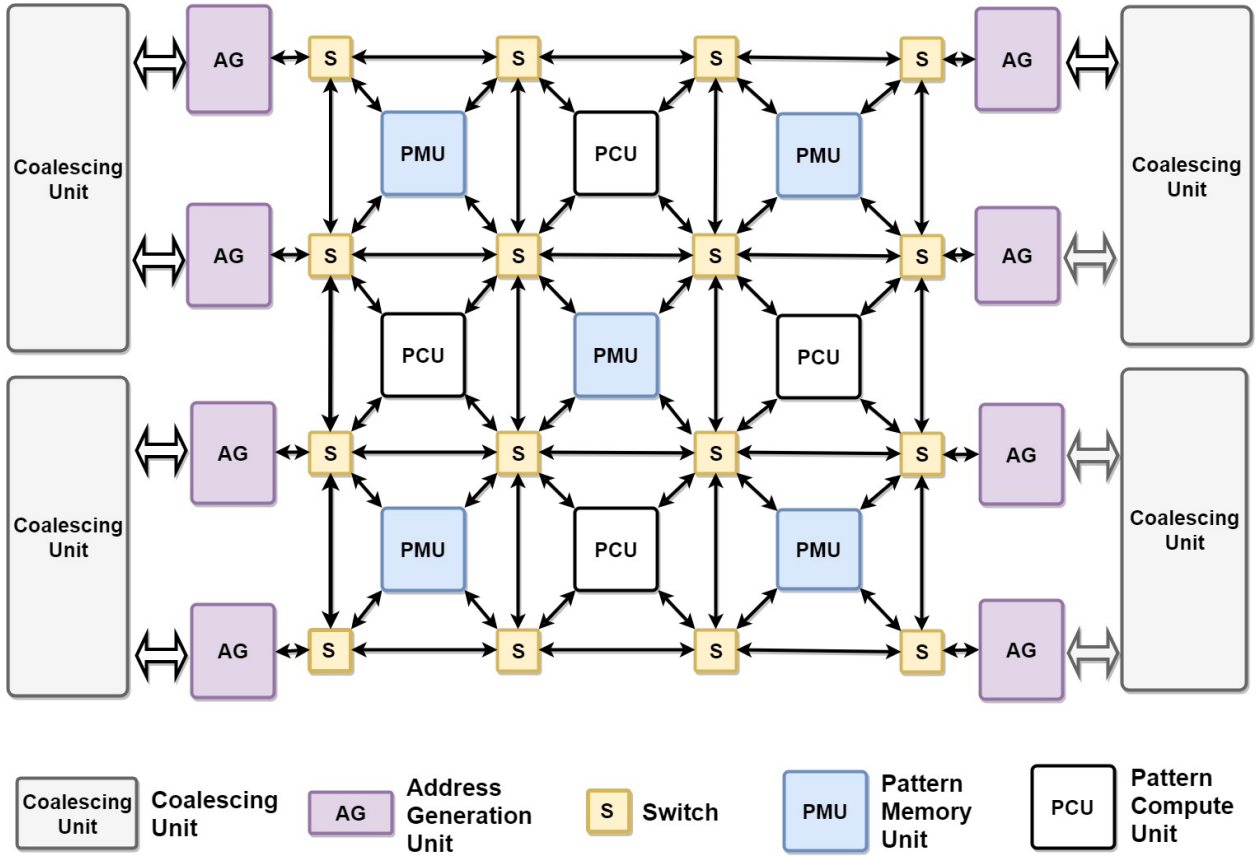
Hasn't this been done before?

Previous Work	Nested Parallelism	Configurable Scratchpads	Hierarchical Interconnect	Programmability
ADRES				
DySER				
Garp				
PipeRench				
Tartan				
RaPiD				
HRL				
Triggered Instructions				
Mosaic				
FPGA				

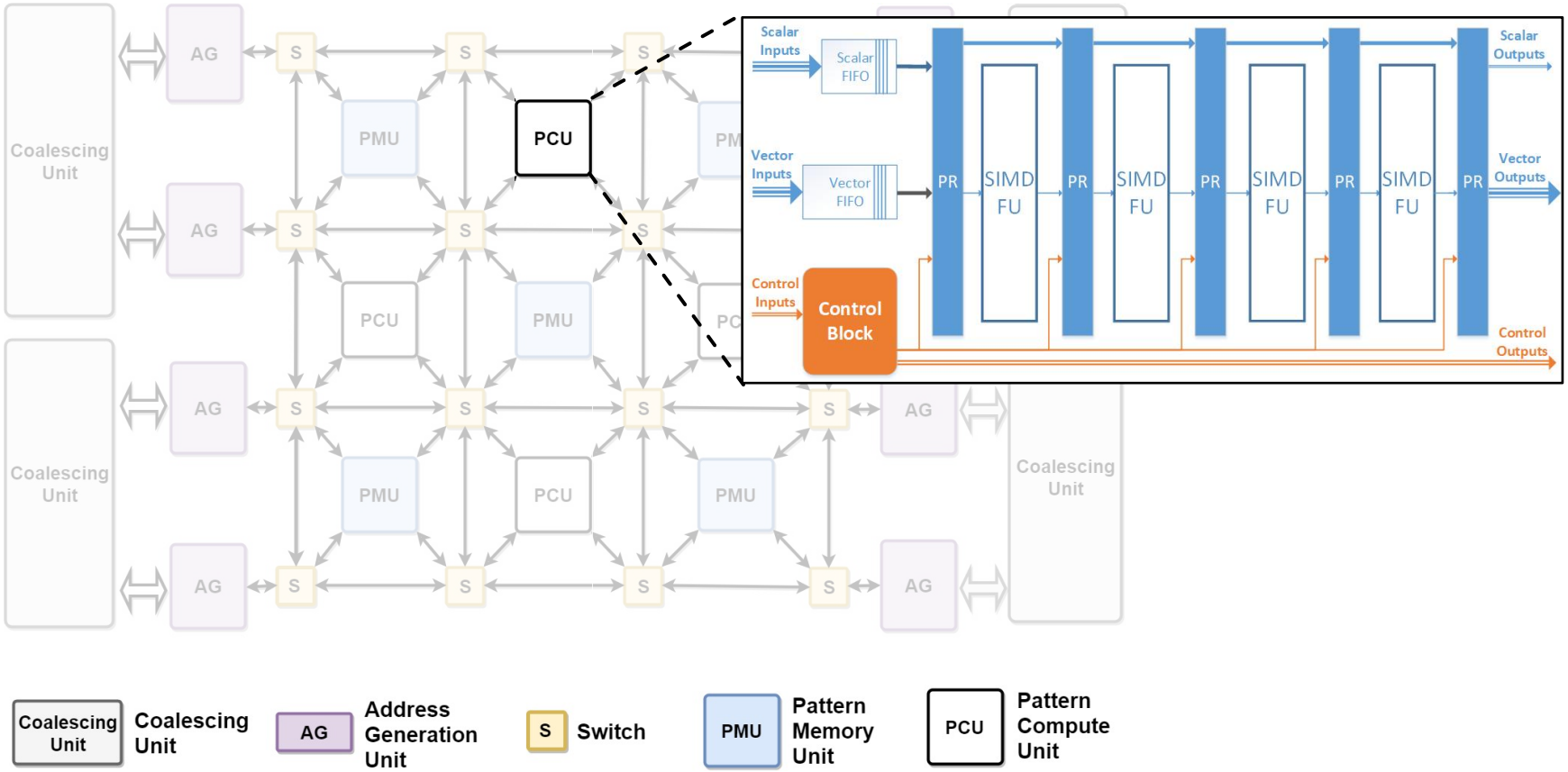
Hasn't this been done before?

Previous Work	Nested Parallelism	Configurable Scratchpads	Hierarchical Interconnect	Programmability
ADRES				
DySER				
Garp				
PipeRench				
Tartan				
RaPiD				
HRL				
Triggered Instructions				
Mosaic				
FPGA				
Plasticine				

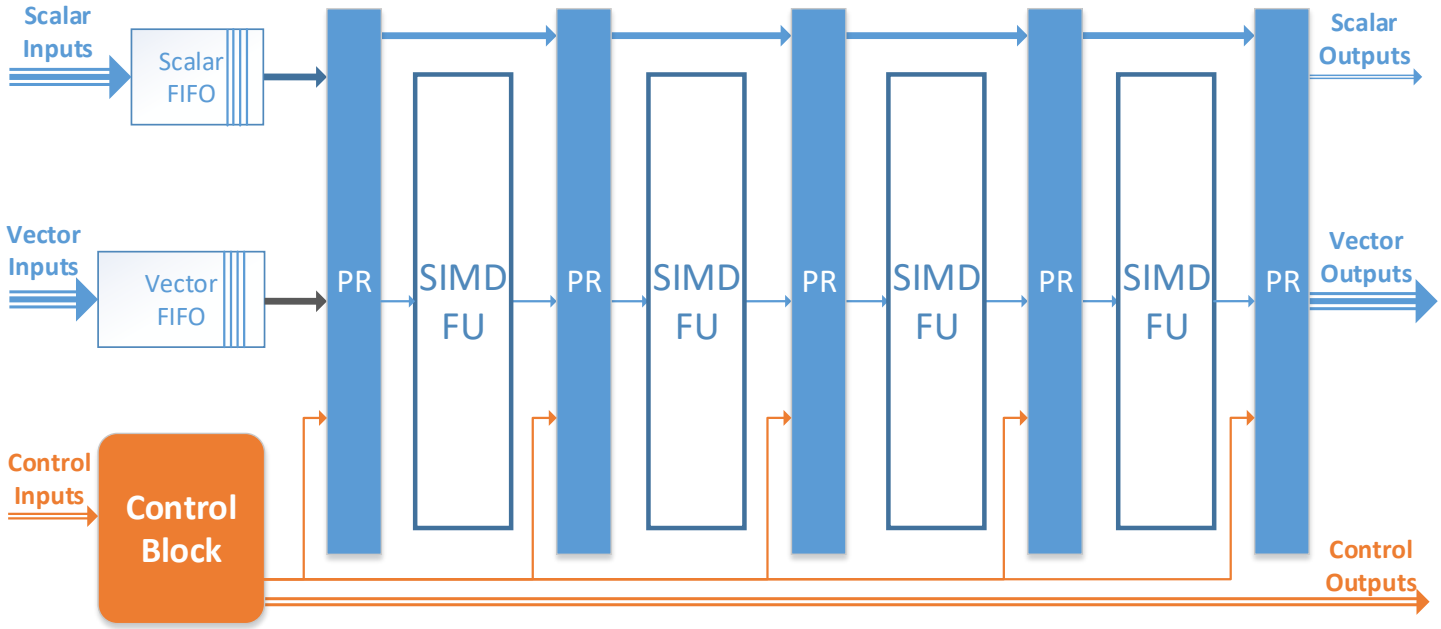
Plasticine: Top-Level



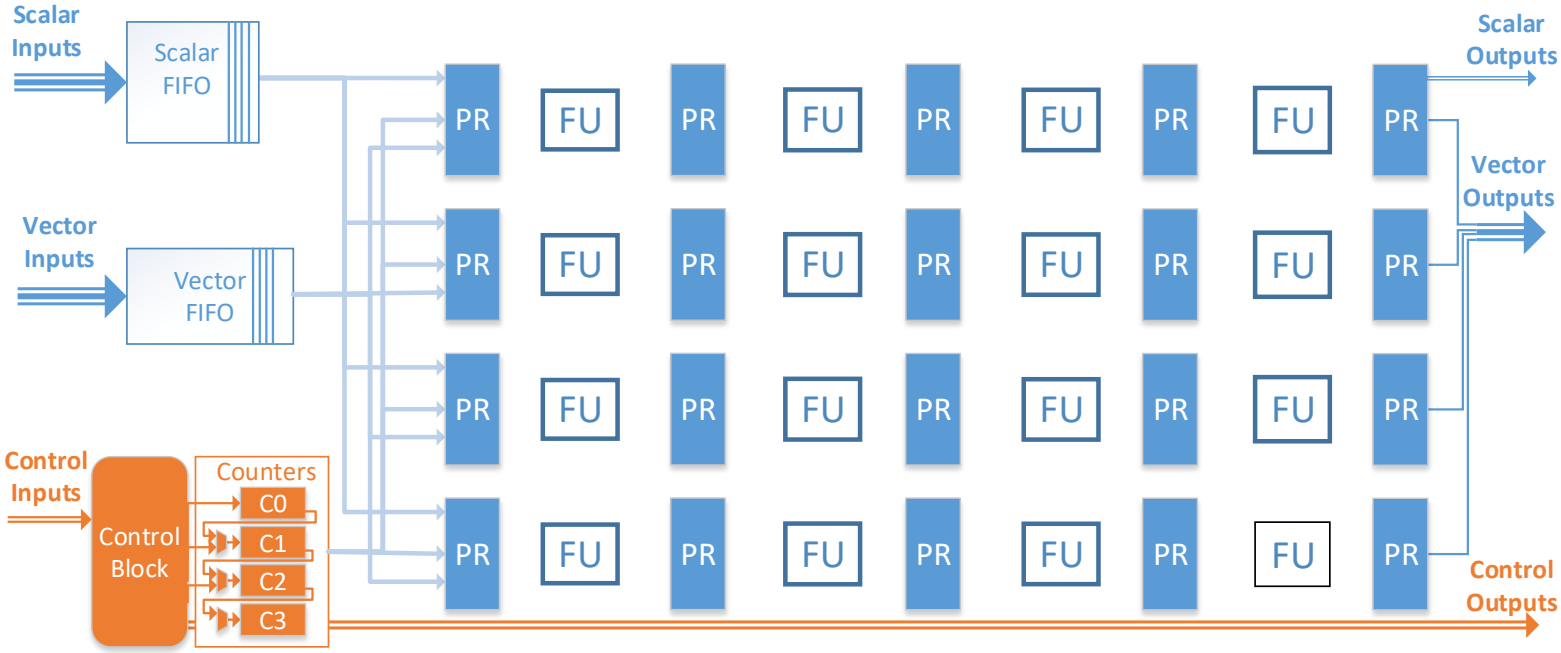
Plasticine: PCU



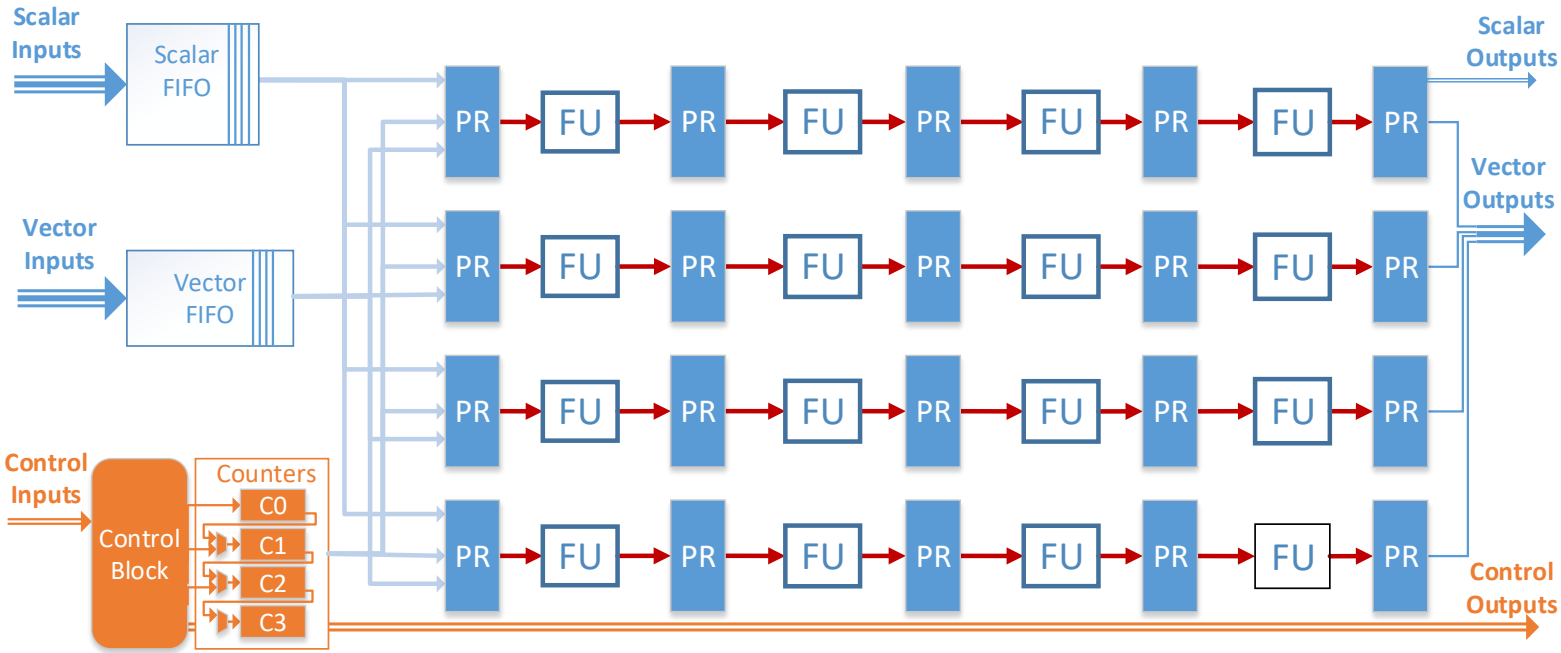
Pattern Compute Unit (PCU)



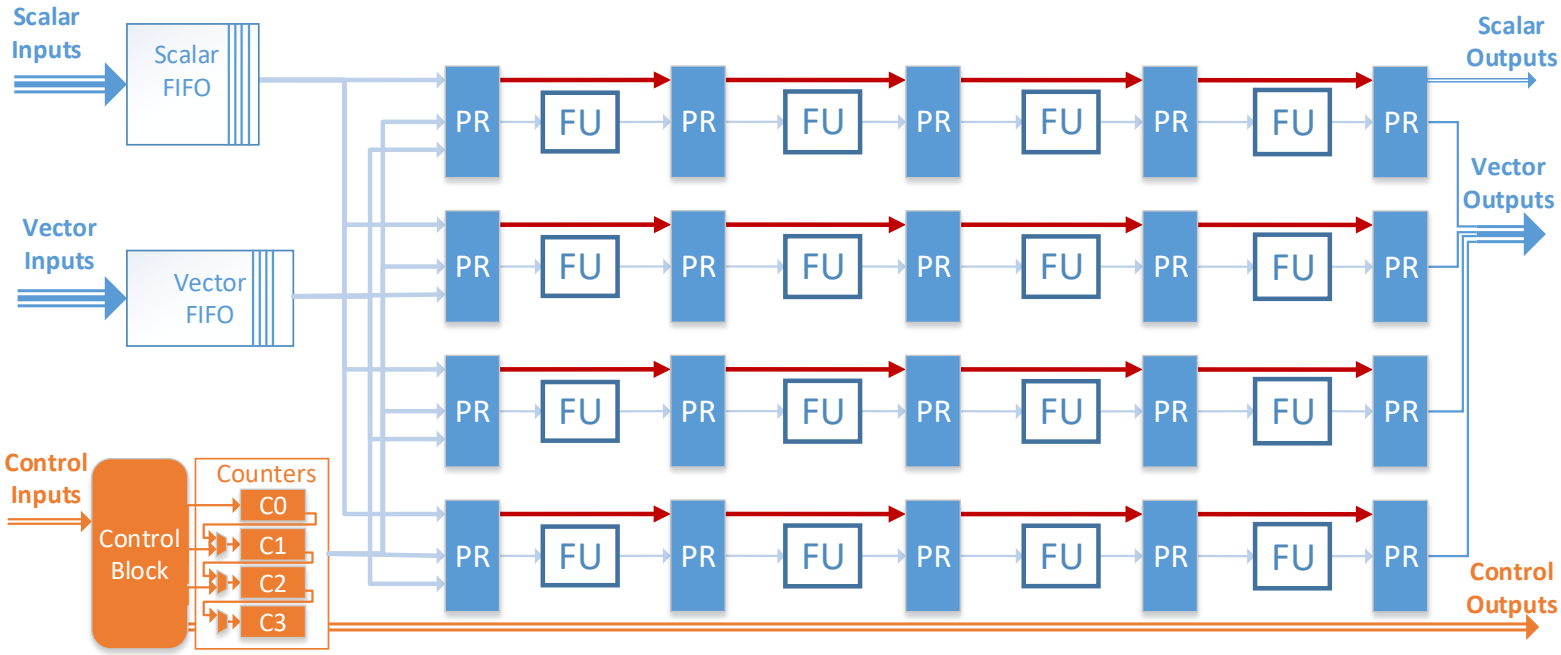
Pattern Compute Unit (PCU)



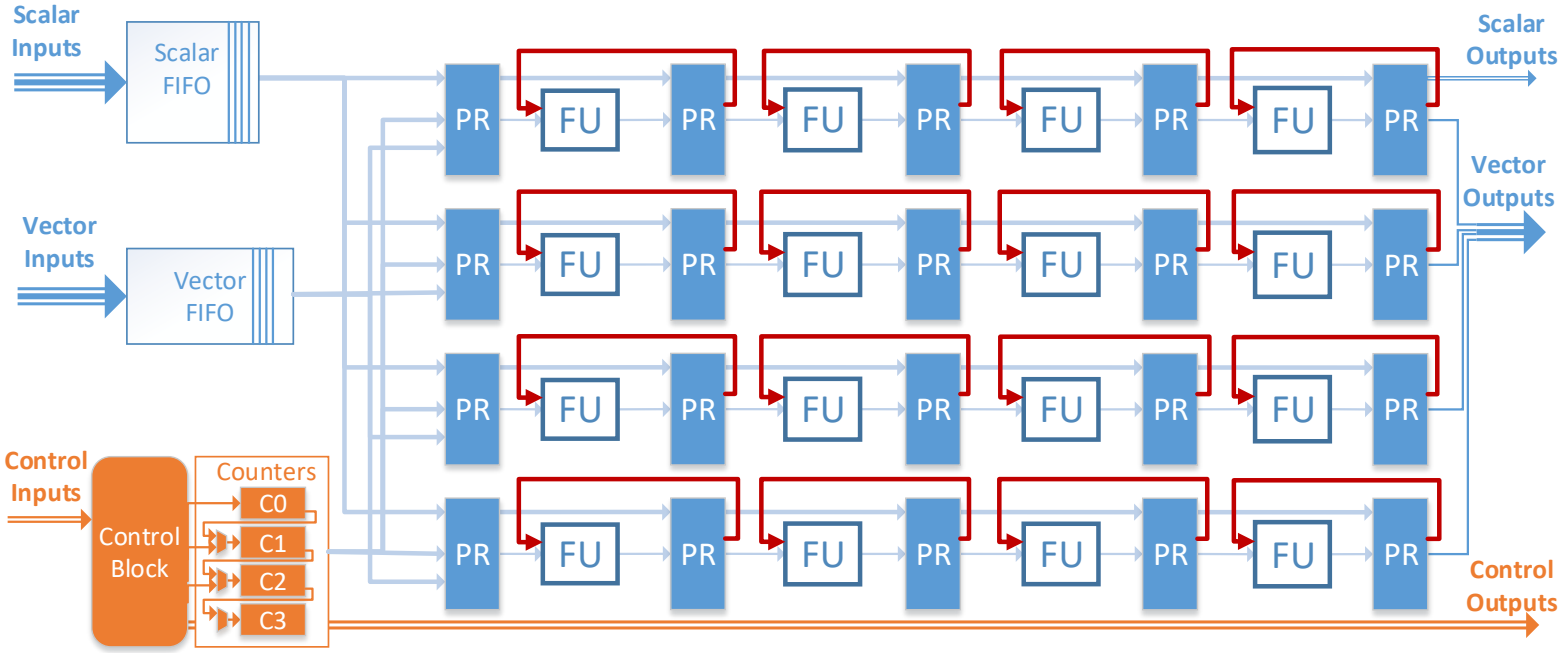
PCU: Pipeline Network



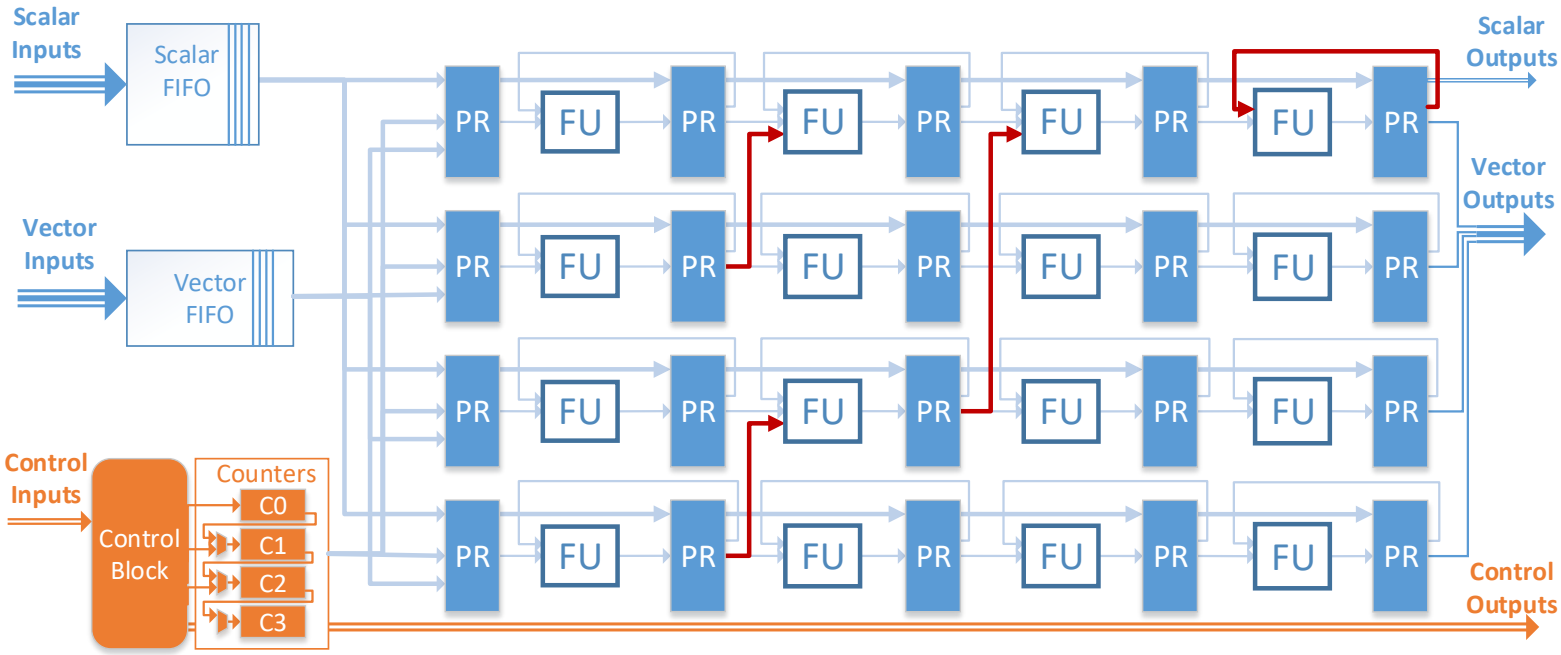
PCU: Forwarding Network



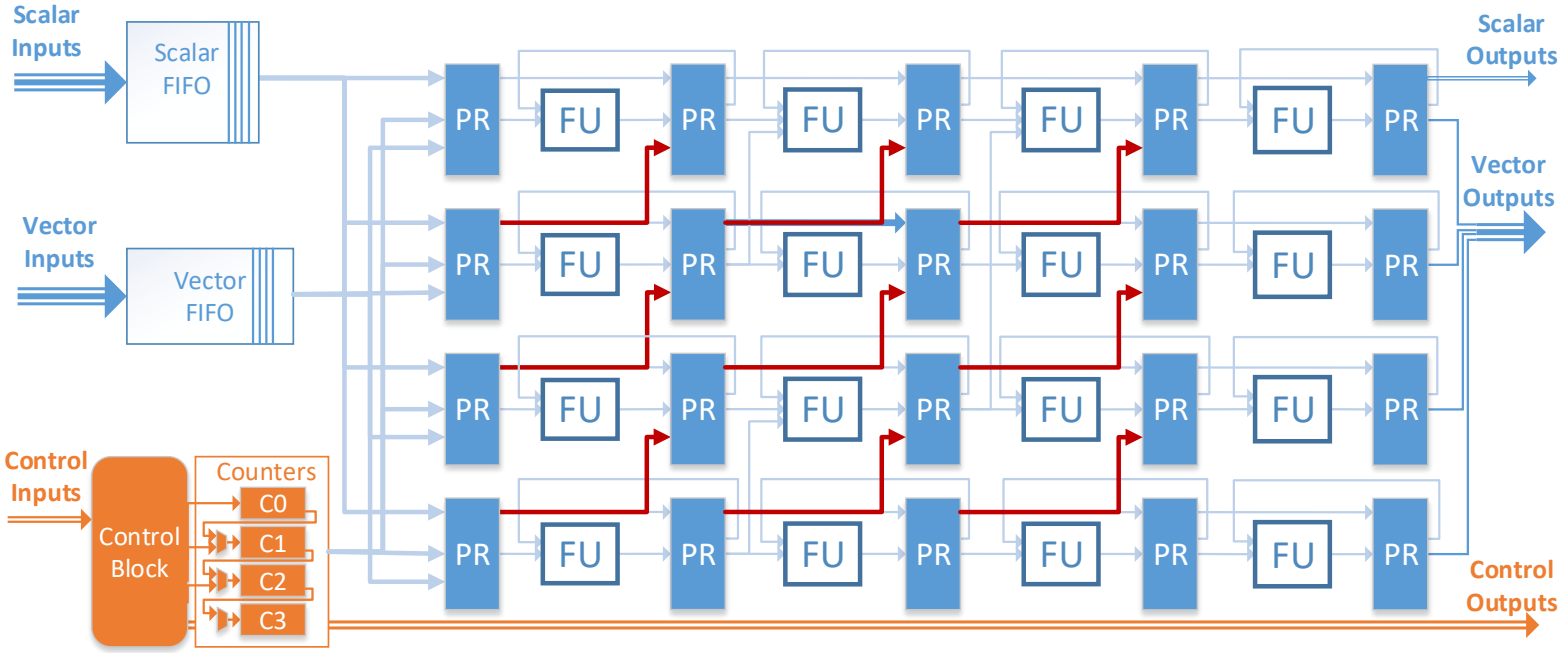
PCU: Feedback Paths



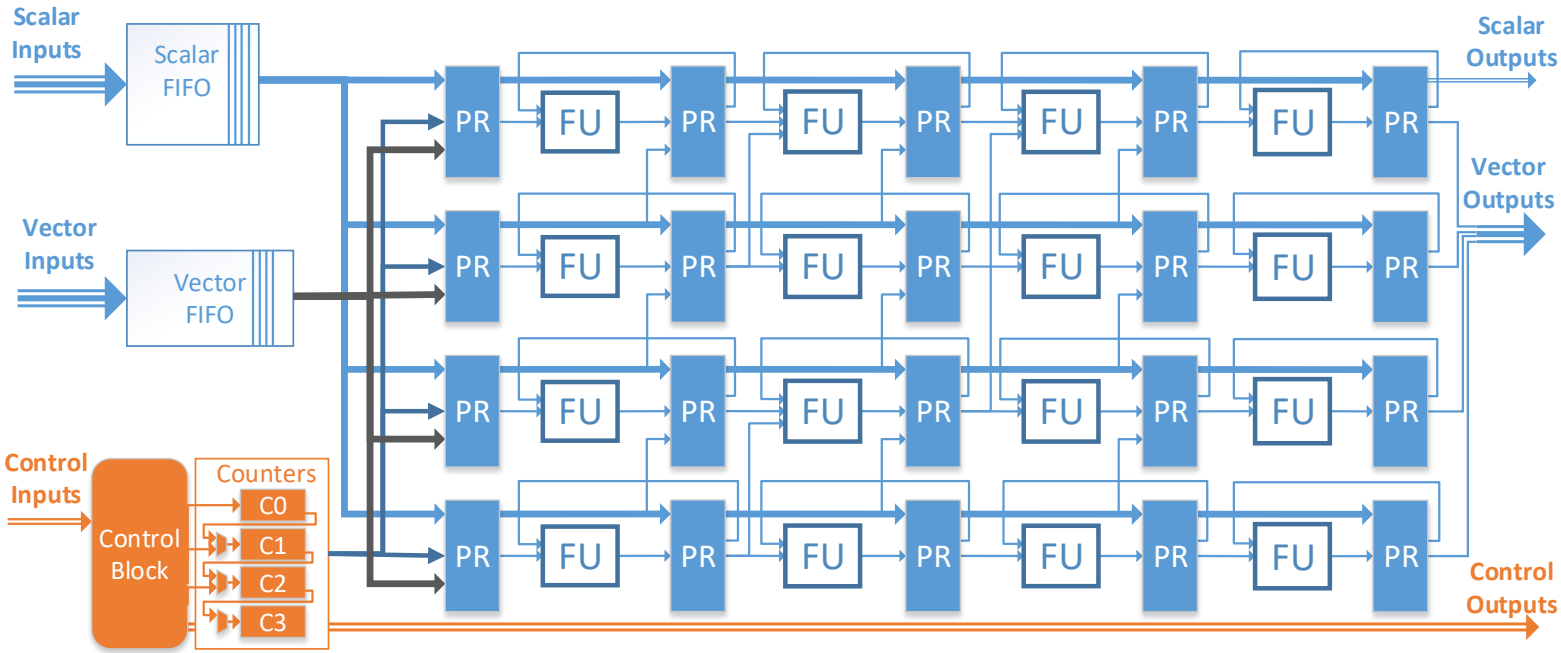
PCU: Reduction Network



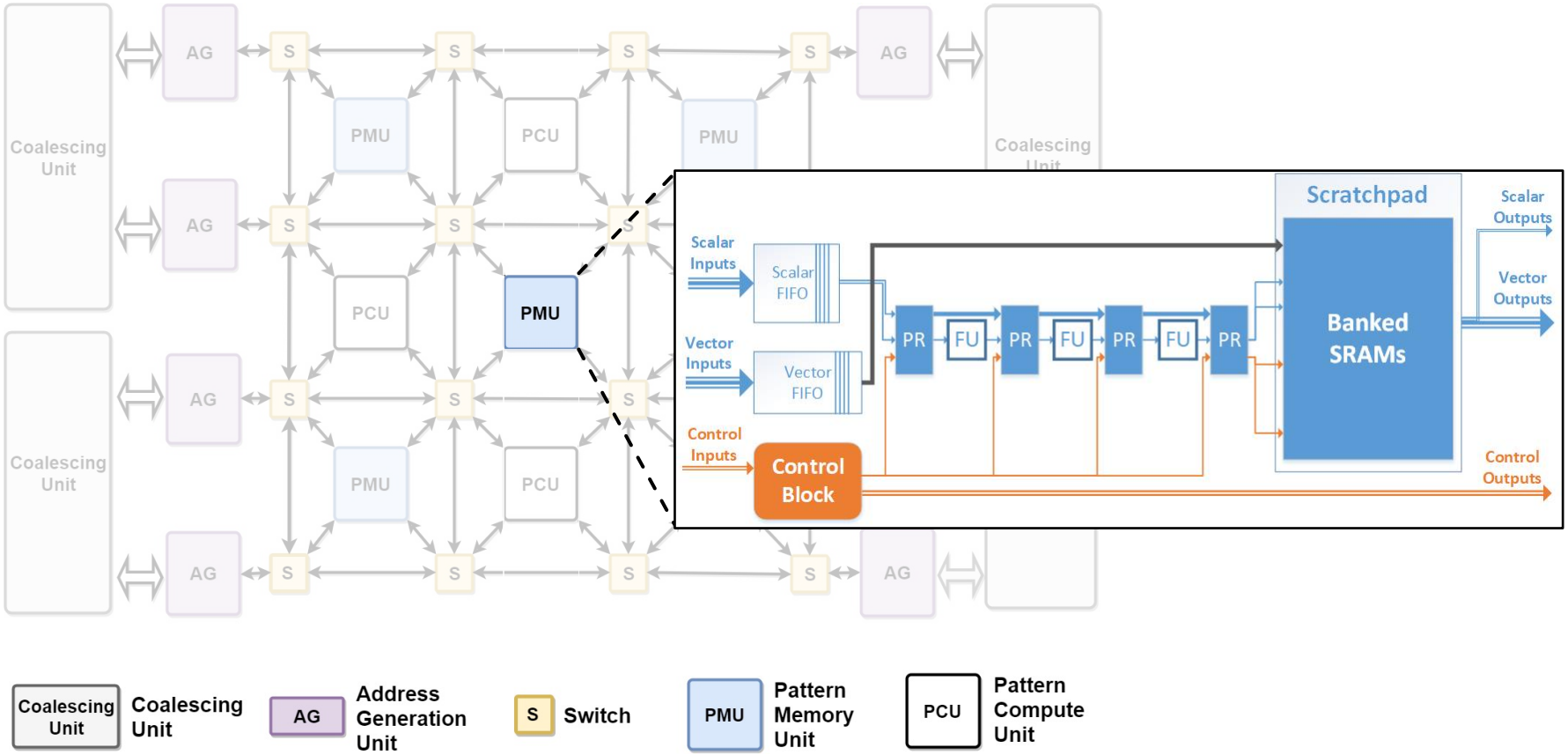
PCU: Shift Network



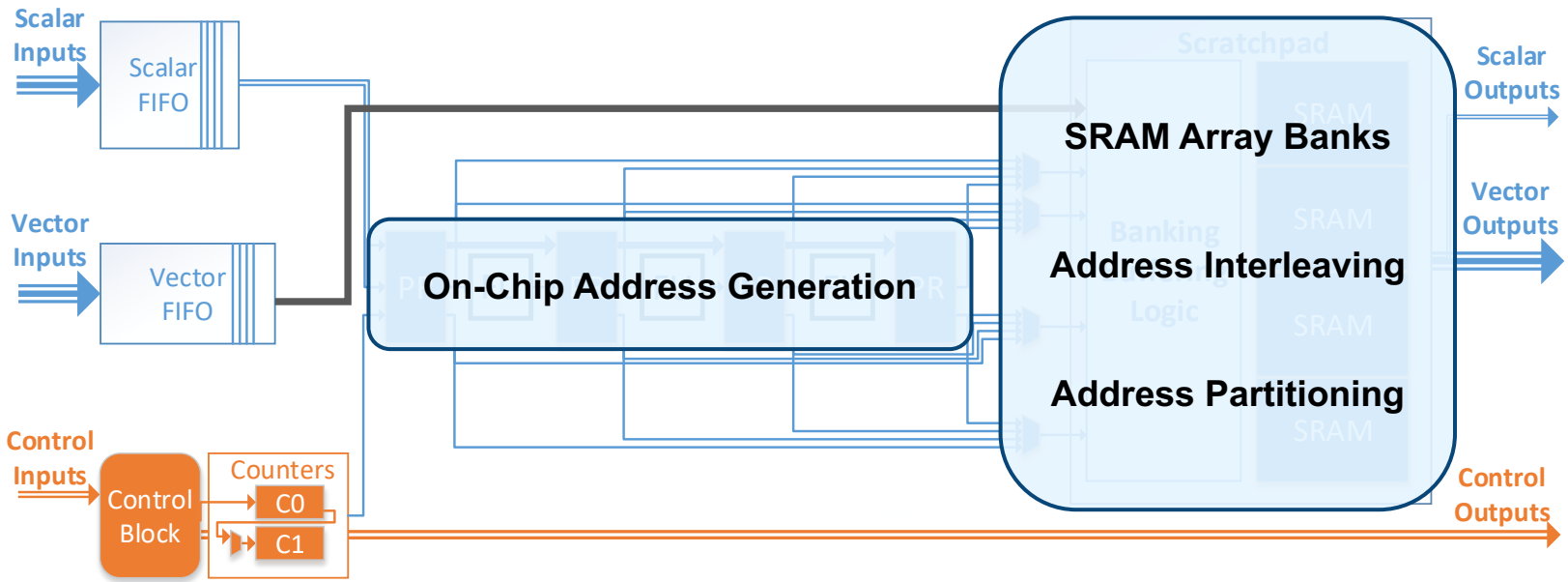
PCU: All


Scalar Inputs
Scalar FIFO
Vector Inputs
Vector FIFO
Control Inputs
Control Block
Counters
C0
C1
C2
C3
PR
FU
Scalar Outputs
Vector Outputs
Control Outputs

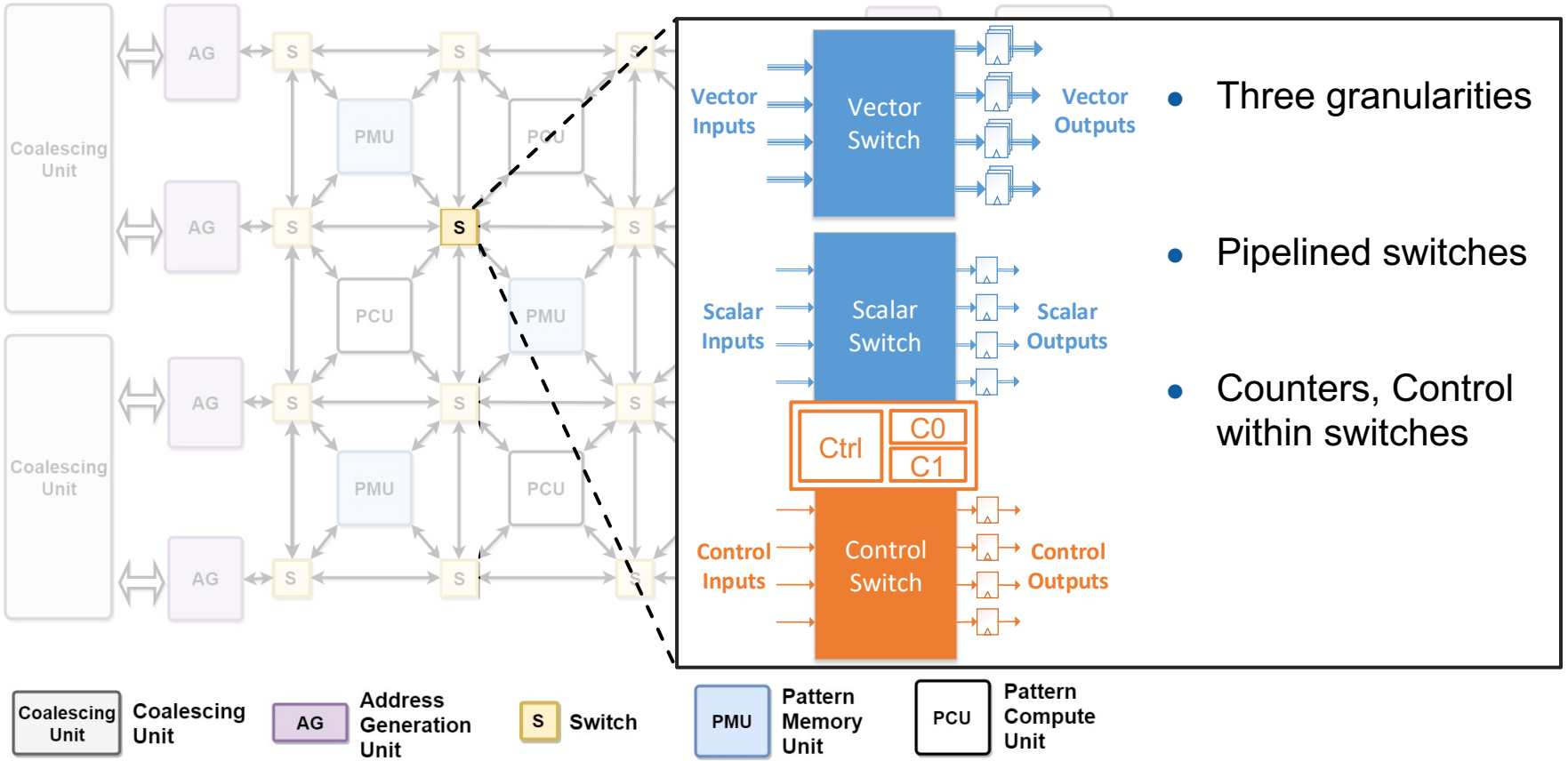
Plasticine: PMU



Pattern Memory Unit (PMU)

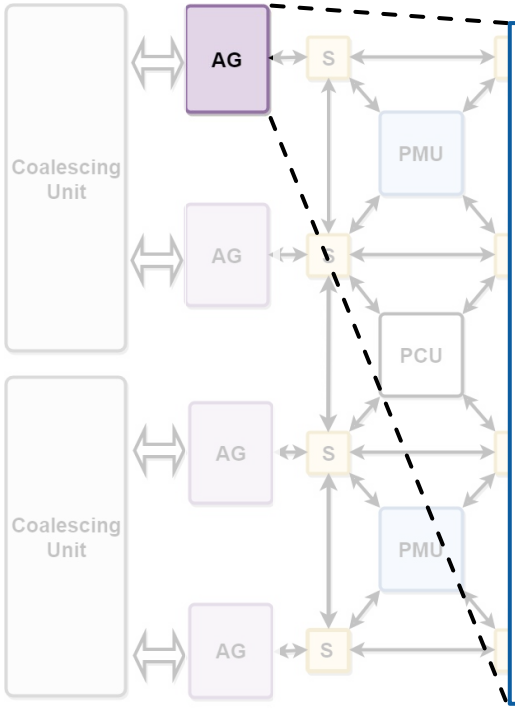


Plasticine: Interconnect



- Three granularities
- Pipelined switches
- Counters, Control within switches

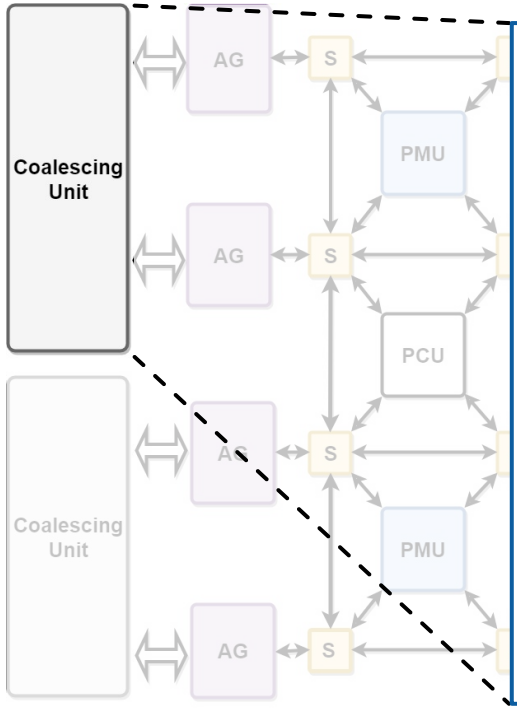
Plasticine: Address Generators



- Integer FUs to generate DRAM requests
- **Dense requests**: Burst mode operation
- **Sparse requests**: Address stream

Coalescing Unit — Coalescing Unit
AG — Address Generation Unit
S — Switch
PMU — Pattern Memory Unit
PCU — Pattern Compute Unit

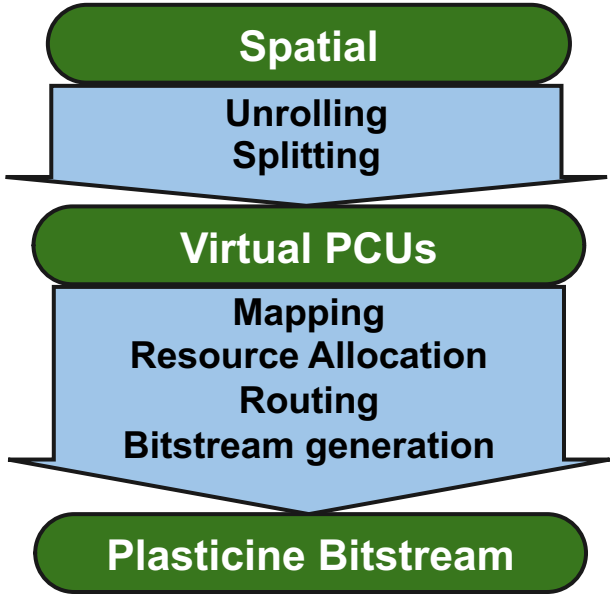
Plasticine: Coalescing Units



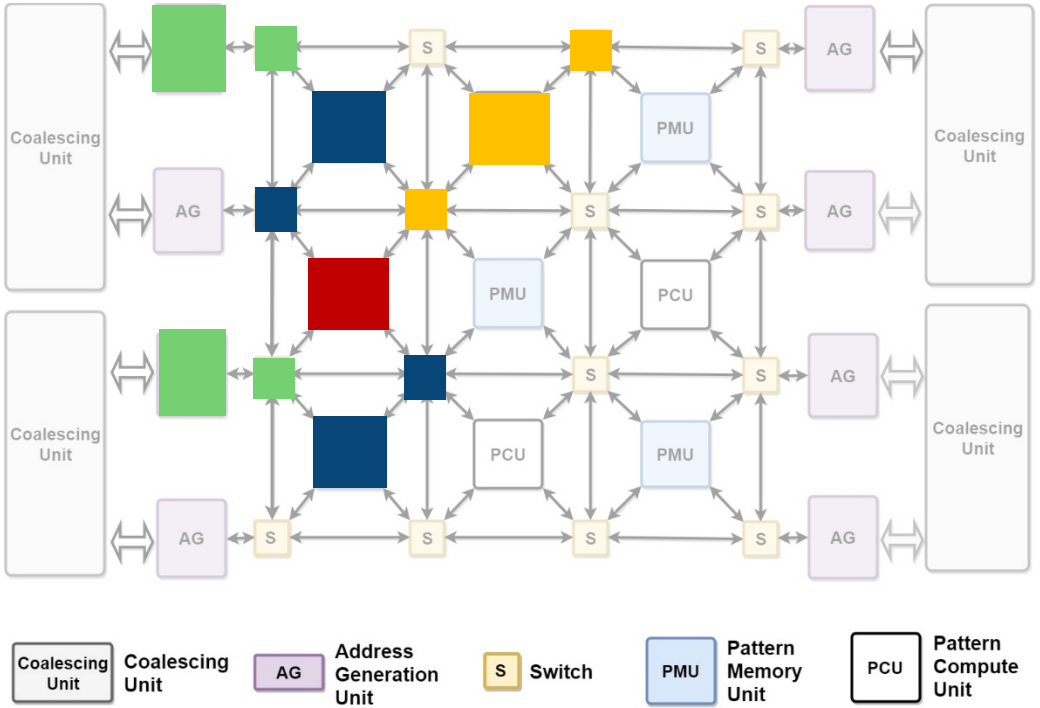
- **Arbitration between multiple address streams**
  - Shares physical DRAM channel between multiple AGs
- **Scatter-Gather Unit**
  - Coalescing cache maintains sparse request metadata
  - Combines requests to same DRAM burst
  - Allows large number of outstanding requests

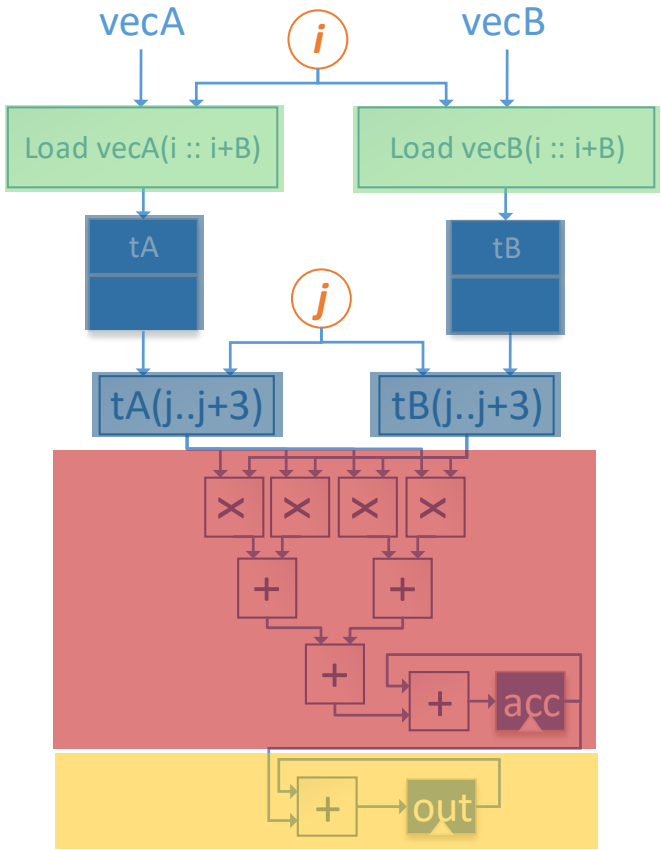
Coalescing Unit — Coalescing Unit
AG — Address Generation Unit
S — Switch
PMU — Pattern Memory Unit
PCU — Pattern Compute Unit

Application Mapping



Example: Dot Product

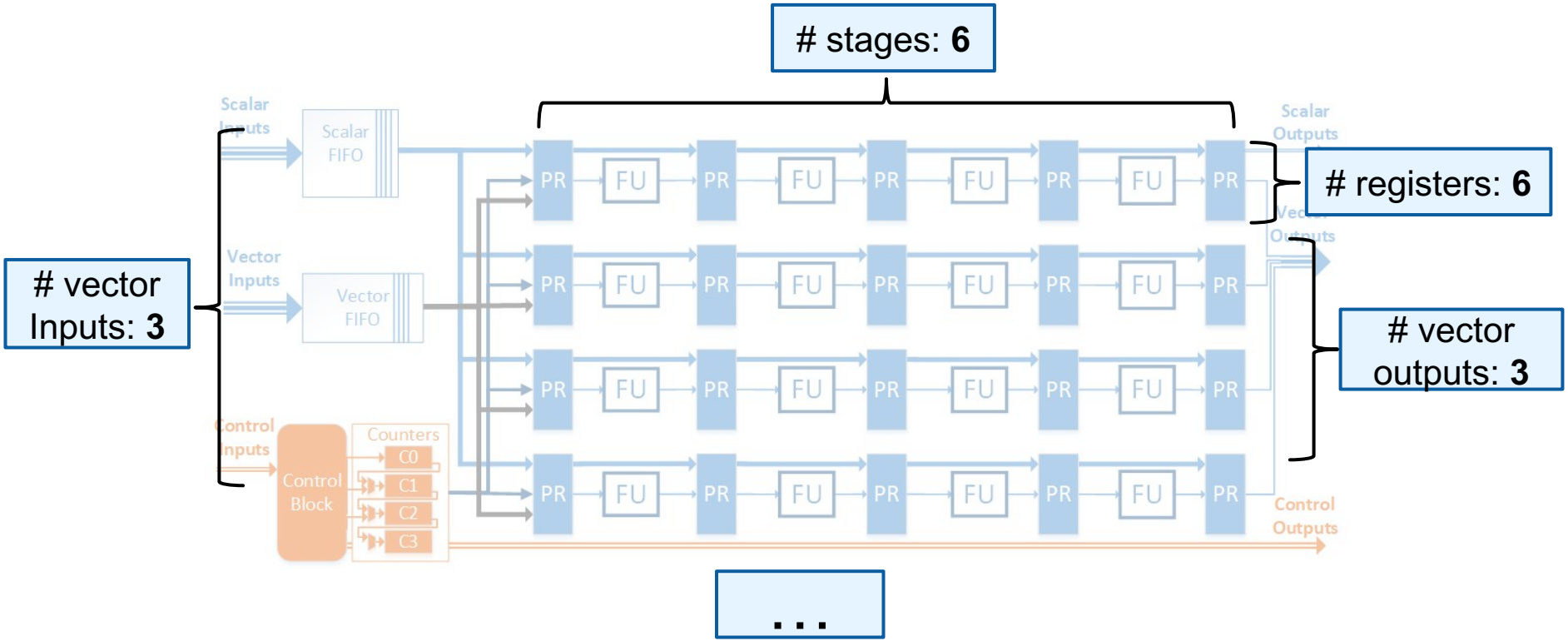




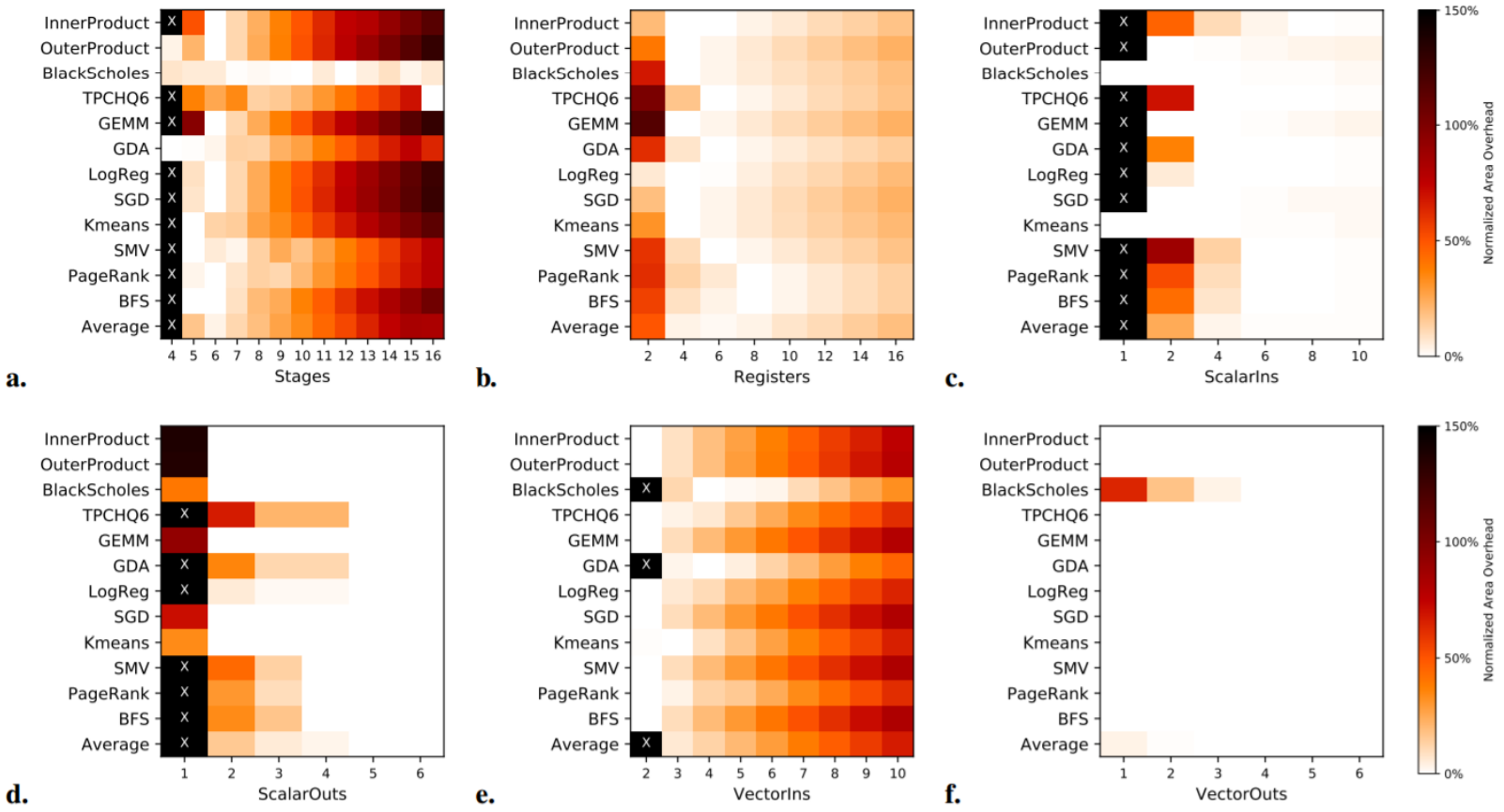
Evaluation

Sizing, Area, Power, Performance, Perf / W

PCU Sizing


stages: 6
registers: 6
vector Inputs: 3
vector outputs: 3
Scalar Inputs
Scalar FIFO
Vector Inputs
Vector FIFO
Control Inputs
Control Block
Counters
C0
C1
C2
C3
PR
FU
Scalar Outputs
Control Outputs
...

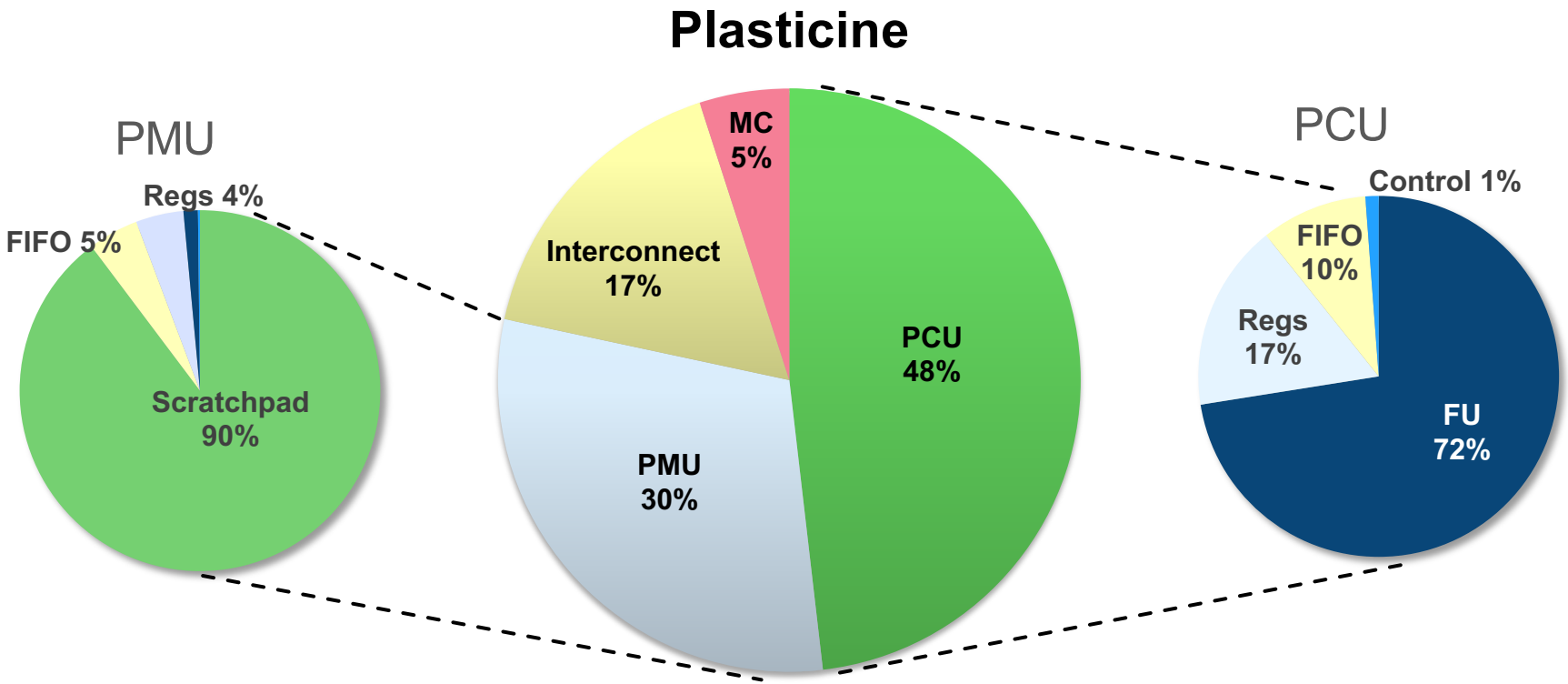
PCU Sizing


InnerProduct
OuterProduct
BlackScholes
TPCHQ6
GEMM
GDA
LogReg
SGD
Kmeans
SMV
PageRank
BFS
Average
a.
Stages
b.
Registers
c.
ScalarIns
d.
ScalarOuts
e.
VectorIns
f.
VectorOuts
Normalized Area Overhead
150%
100%
50%
0%

Plasticine Clock, Area, and Power

Architecture Dimensions	16 x 8: 64 PCUs, 64 PMUs
Total On-Chip Memory	16 MB (256 KB / PMU)
Technology Node	28nm
Clock Frequency	1 GHz
Total Area	112.77 mm^2
Total Power	49 W

Area Breakdown

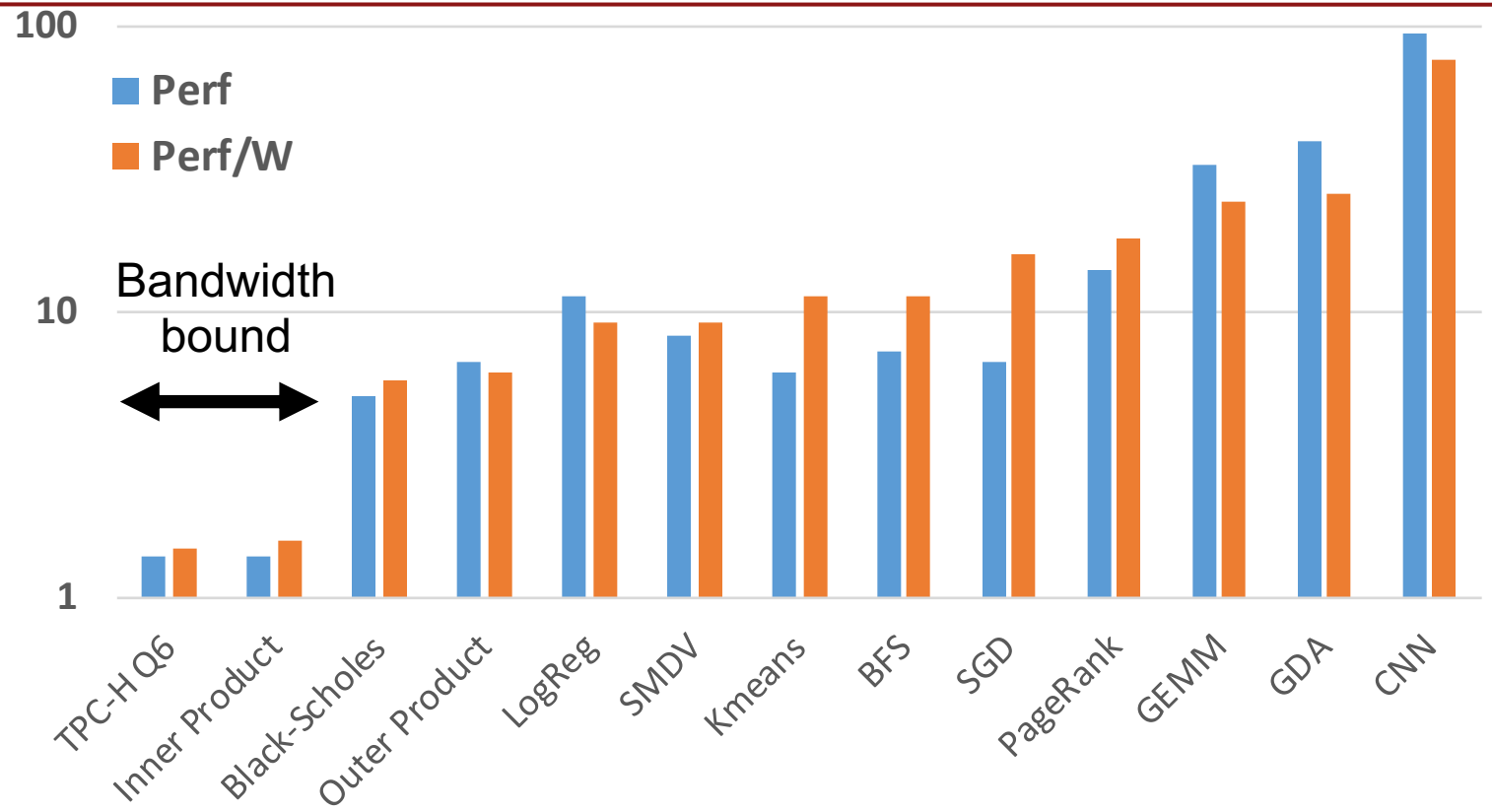


Experimental Setup

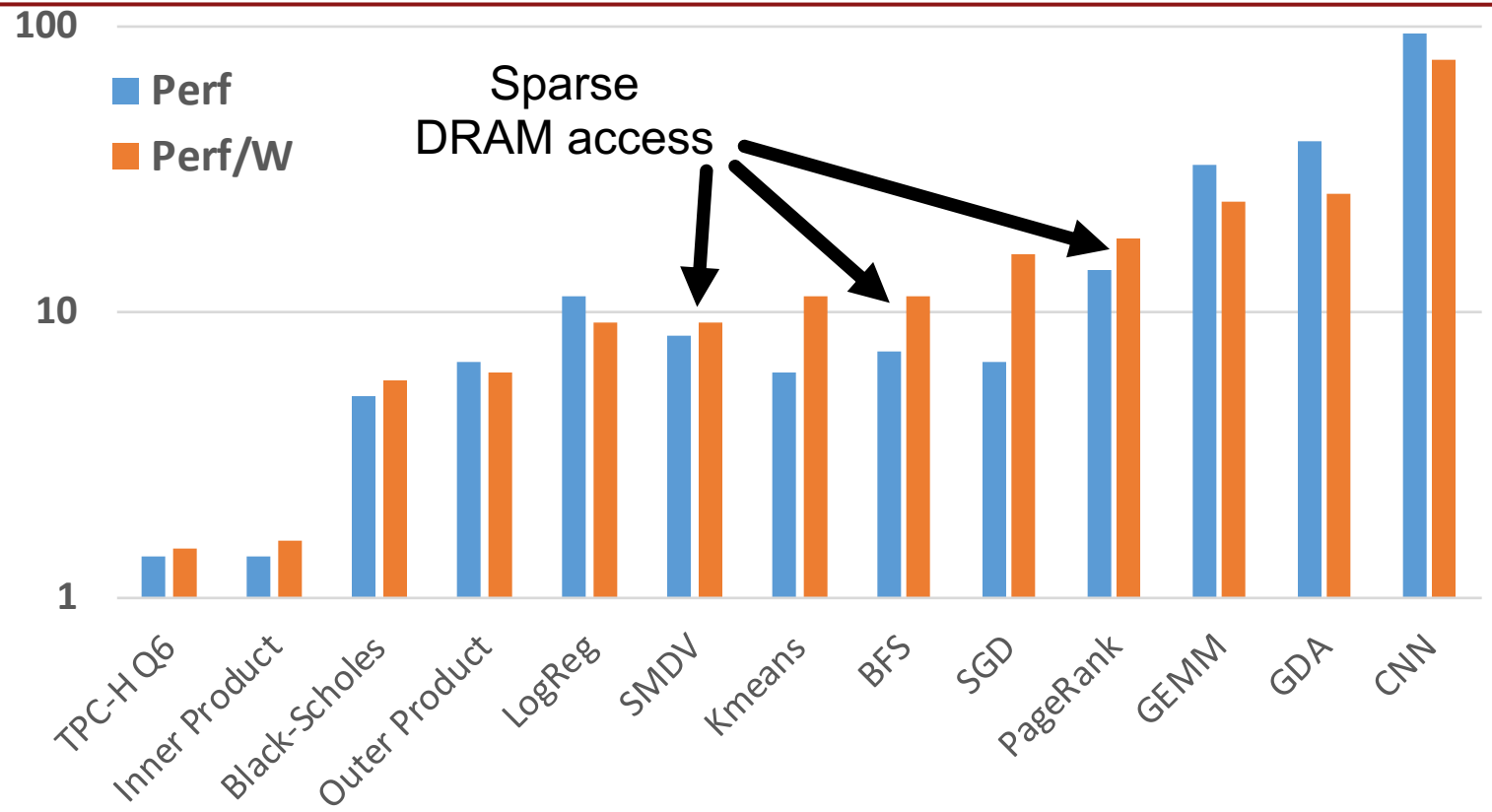
- **Plasticine:**
  - Implemented in Chisel
  - RTL synthesized with 28nm library
  - 4 DDR3-1600 DRAM channels, peak memory bandwidth = 51.2 GB/s
  - 1 GHz clock

- **FPGA:**
  - Altera Stratix V, 28 nm technology
  - 6 DDR3-800 DRAM channels, peak memory bandwidth = 37.5 GB/s
  - 150 MHz clock

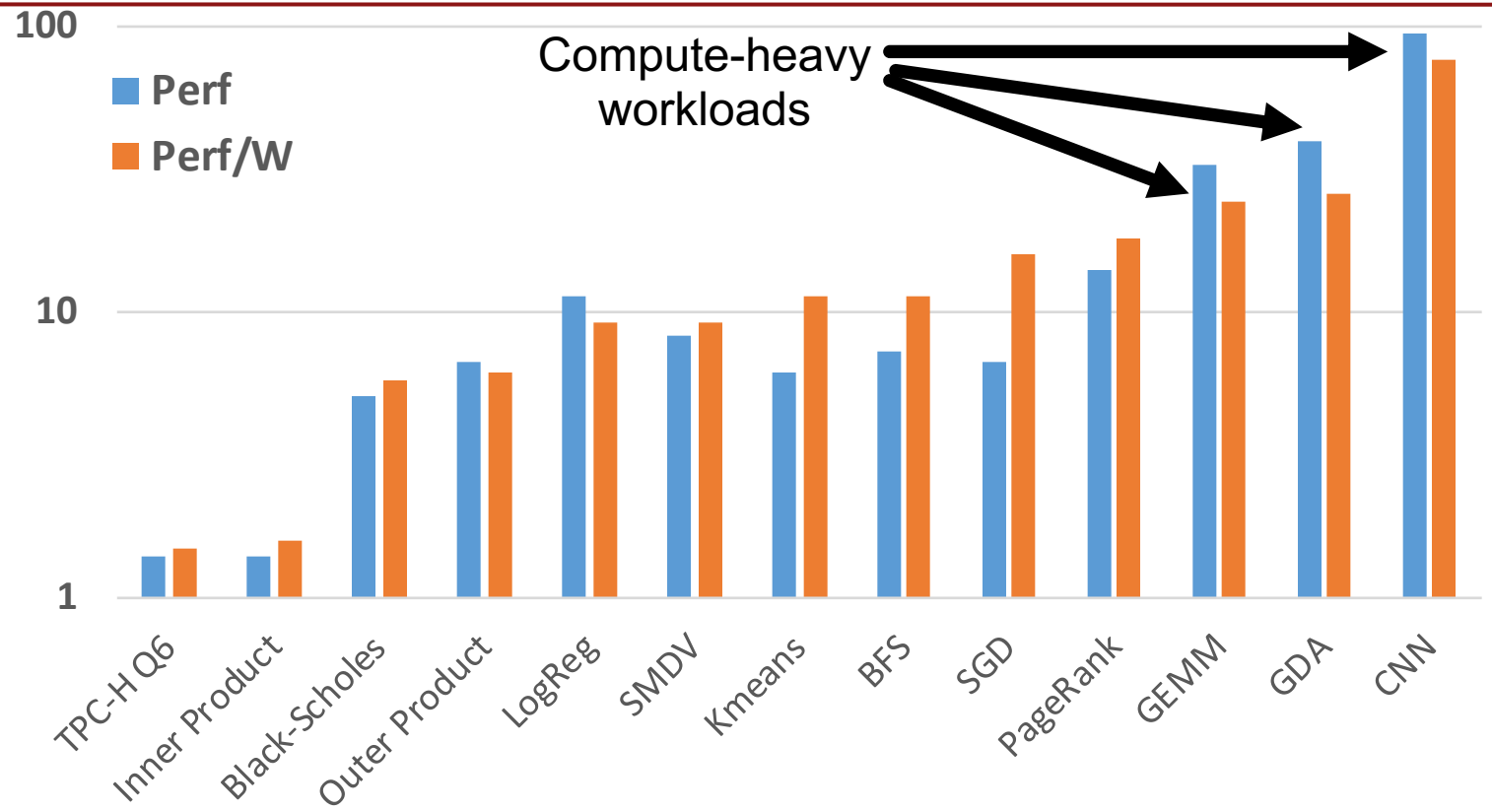
Plasticine vs FPGA (Stratix V)



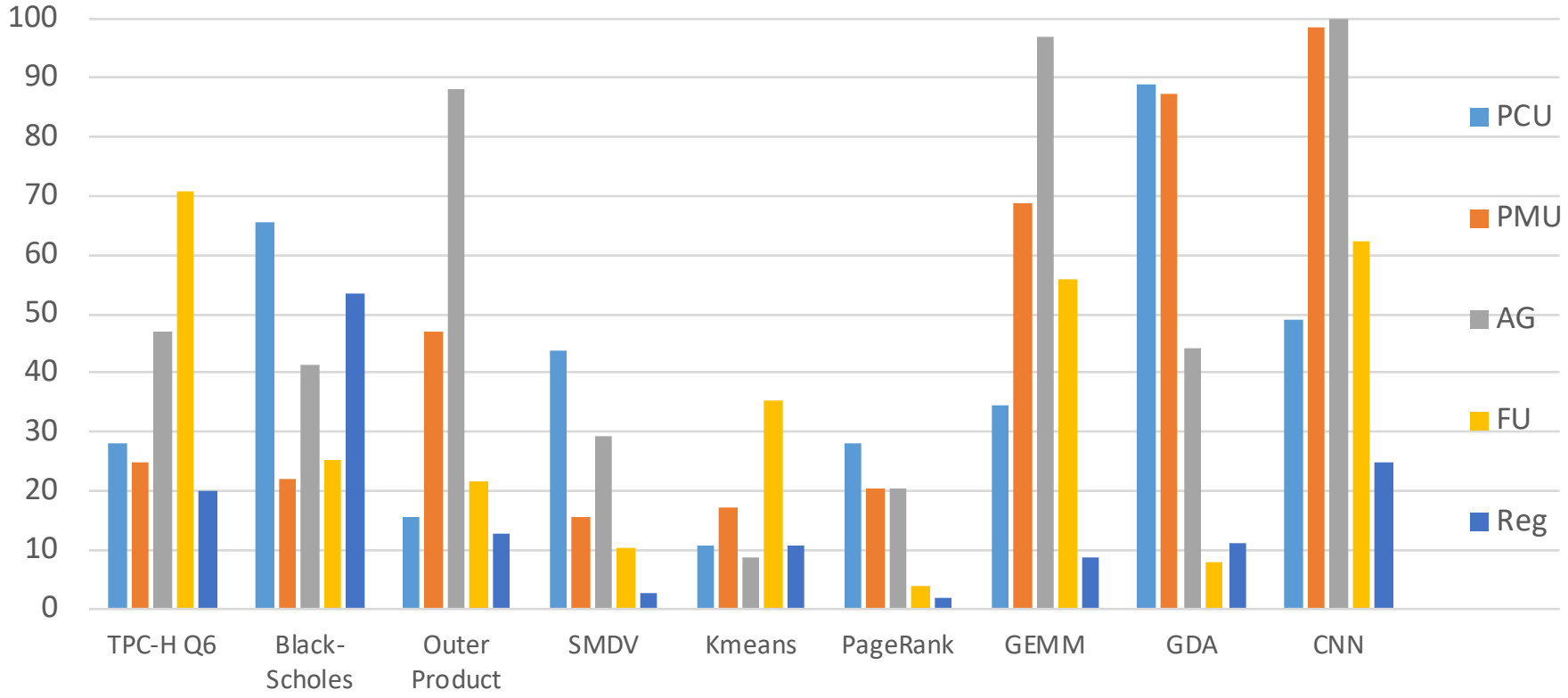
Plasticine vs FPGA (Stratix V)



Plasticine vs FPGA (Stratix V)



Resource Utilization



Summary

- Plasticine efficiently exploits **nested parallelism** using a **hierarchical datapath**
- Plasticine captures **data locality** and sustains high compute throughput using flexible scratchpads with configurable **banking** and **buffering** modes
- Supports dense and sparse DRAM requests using dedicated **address generators** and **scatter-gather units**
- Up to **95x** improvement in performance, **77x** improvement in perf/W over FPGA in similar process technology, with an area of 113mm^2