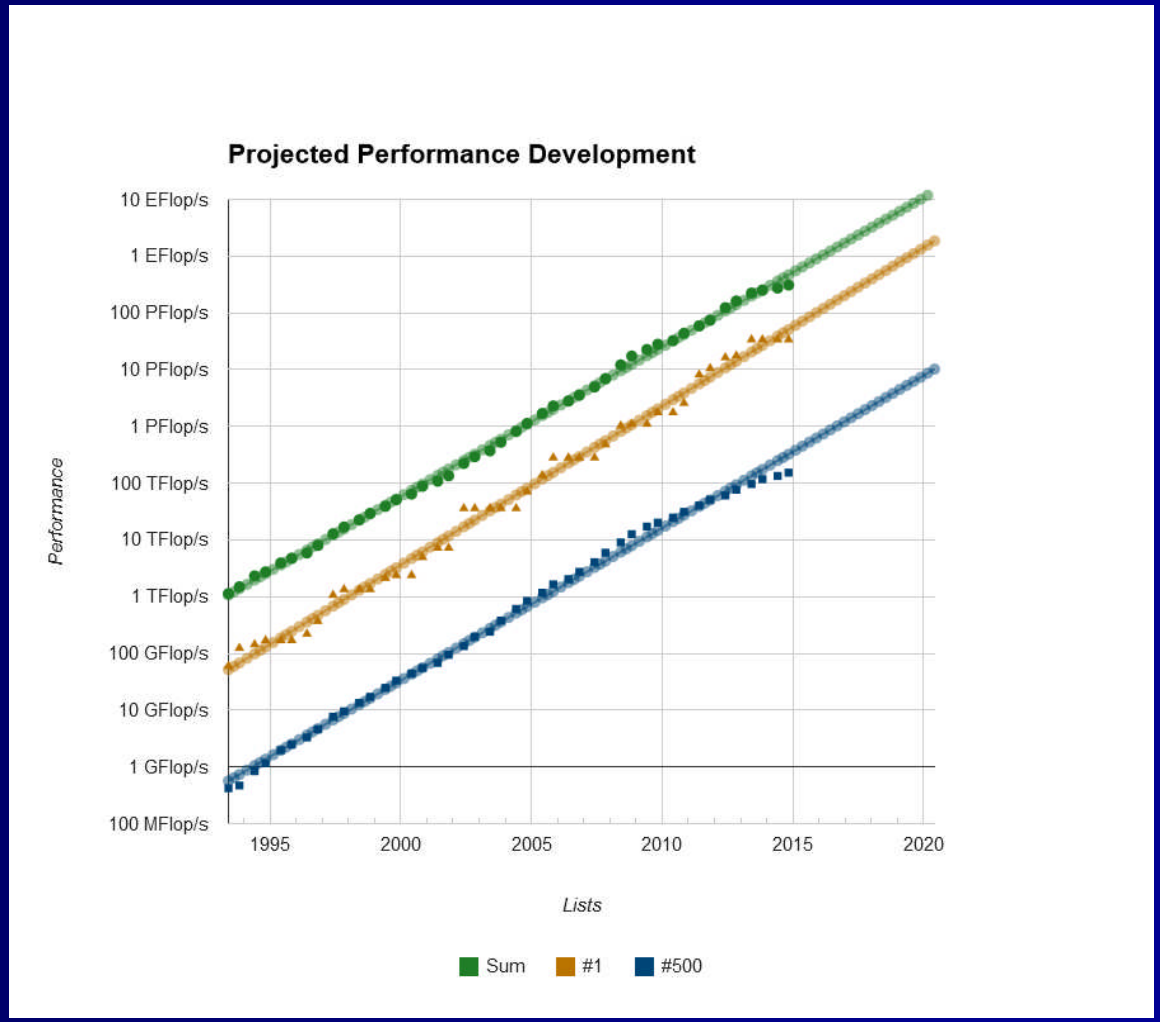
Active Memory Cube

Ravi Nair
IBM T. J. Watson Research Center
December 14, 2014

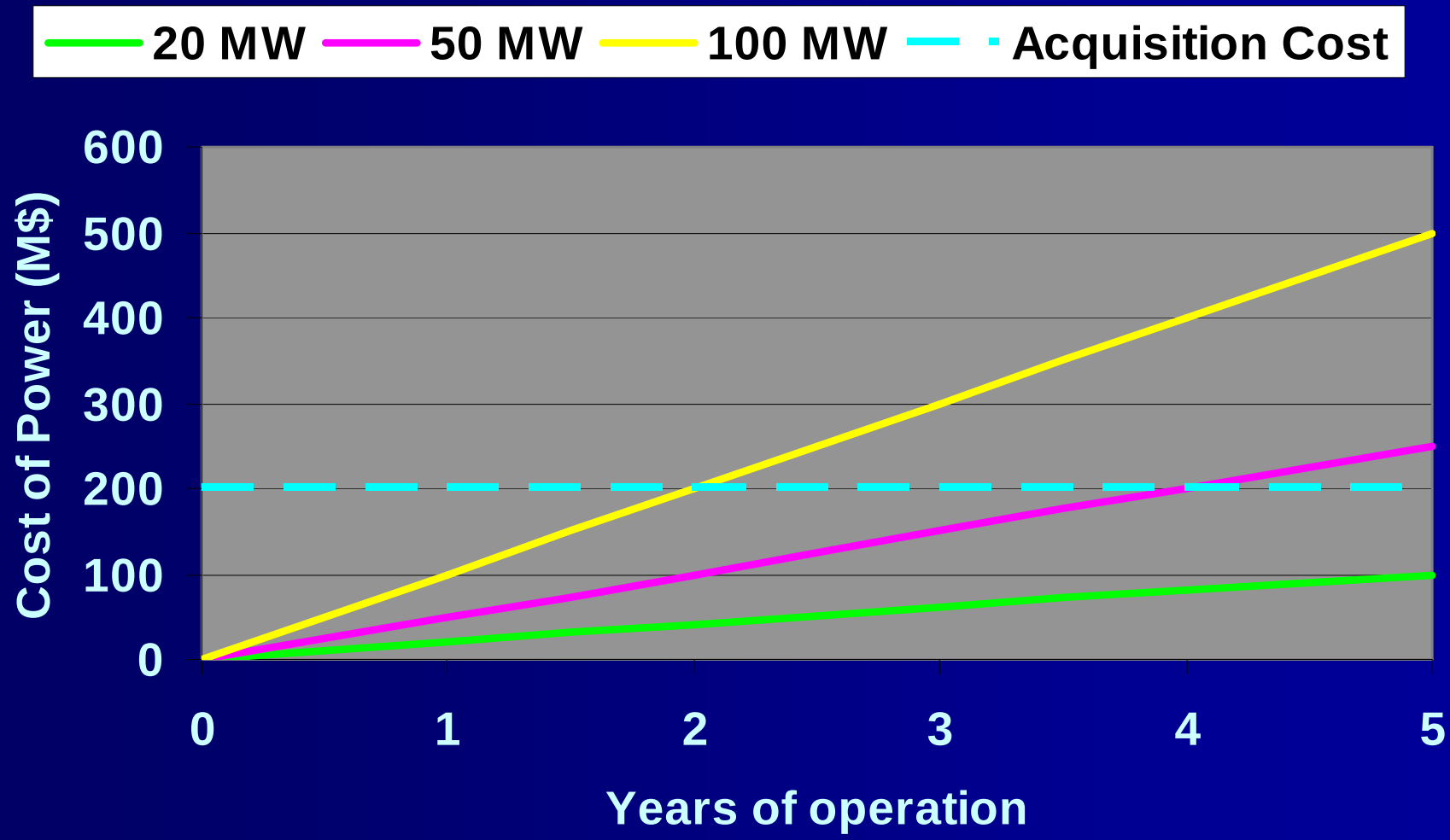
The Top 500 Supercomputers



The Power Story

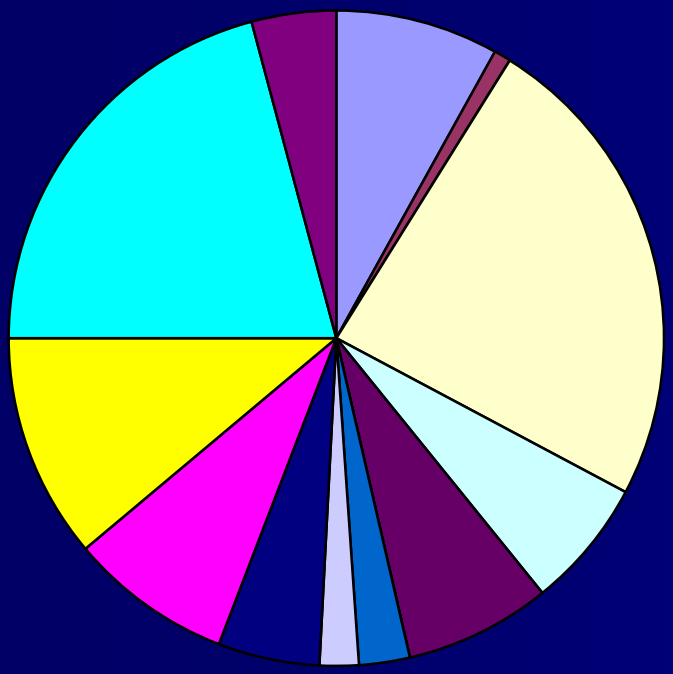
	Linpack TFlops	Power (KW)	MW per Exaflops
Tianhe-2	33,862	17,808	526
Titan	17,590	8,209	467
BG/Q	17,173	7,890	459
K-Computer	10,510	12,660	1205
1 MW of power costs roughly $1M per year			

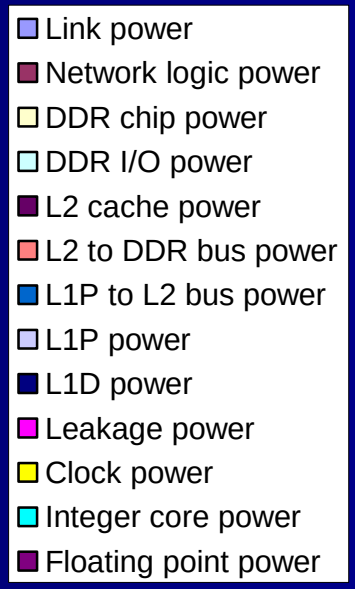
The Cost of Power



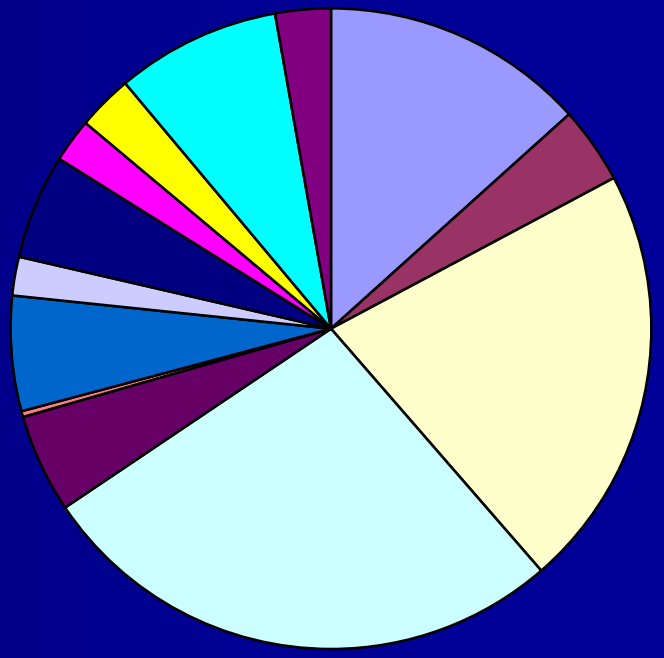
Power Scaling

20 PF/s - 2012



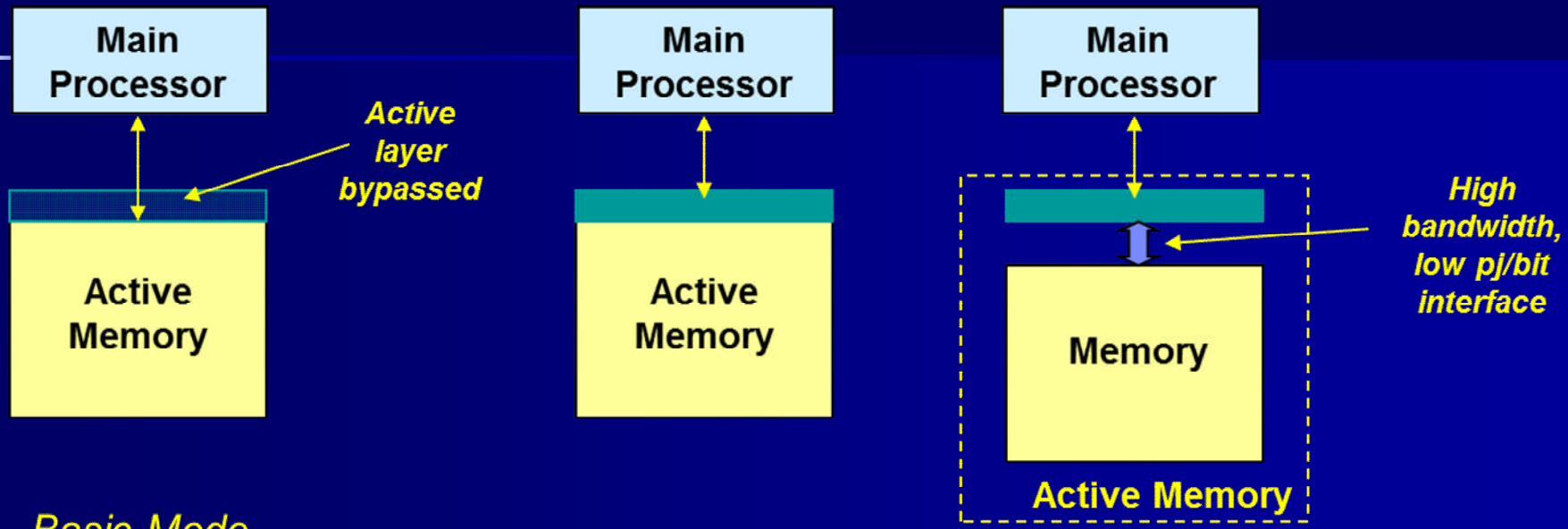


1 EF/s - 2018



Memory power and I/O to memory become major contributors

Active Memory



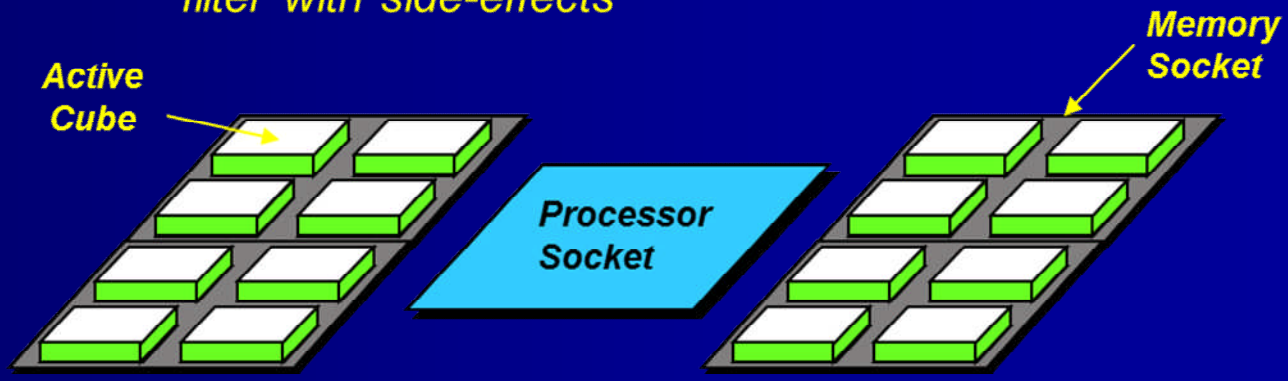
Basic Mode

Memory looks like a traditional memory

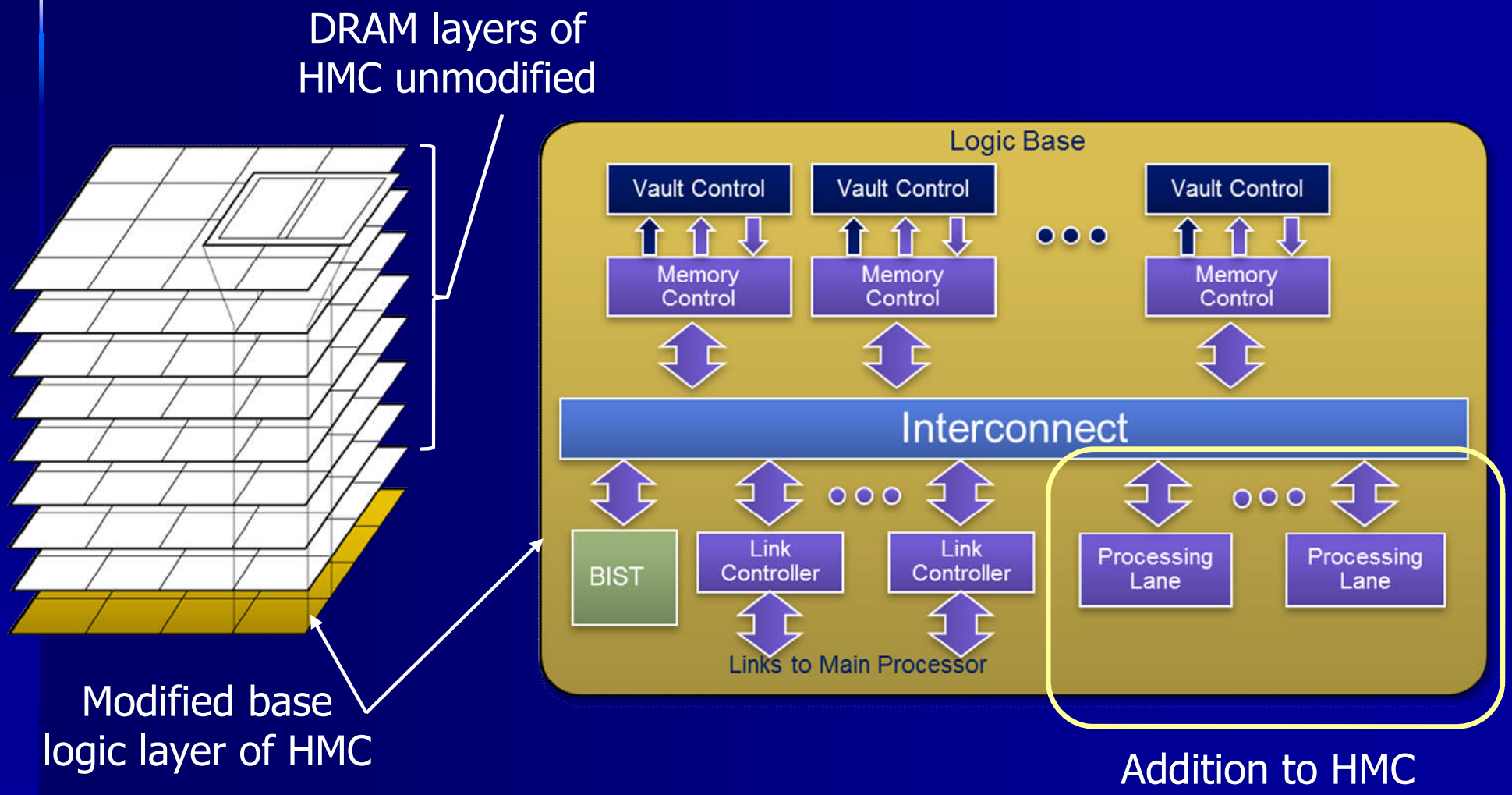
Active Mode

Memory acts as a filter with side-effects

Leverage proximity to memory to perform streaming calculations in a power-efficient manner



Active Memory Cube



Example Exascale Node

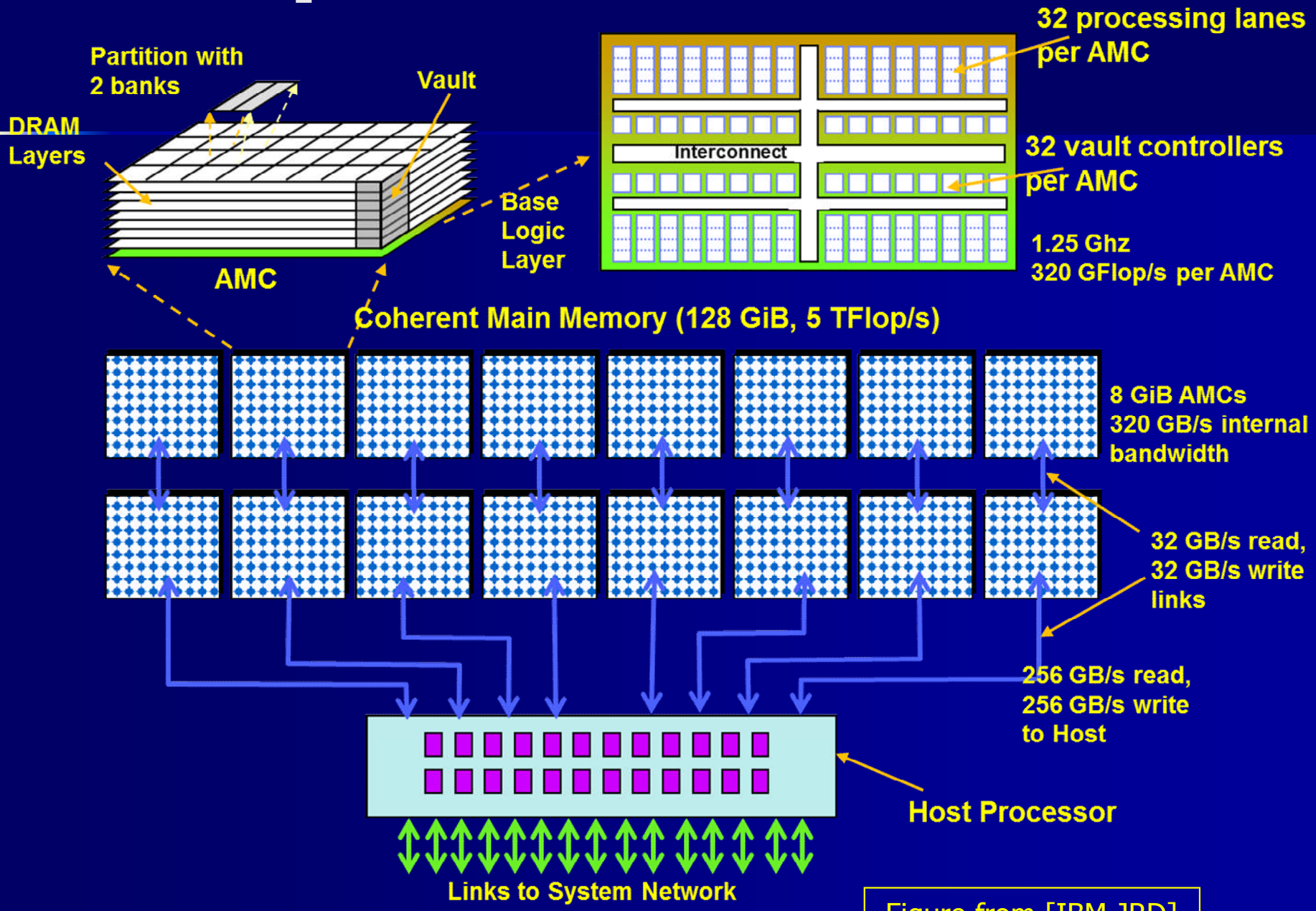
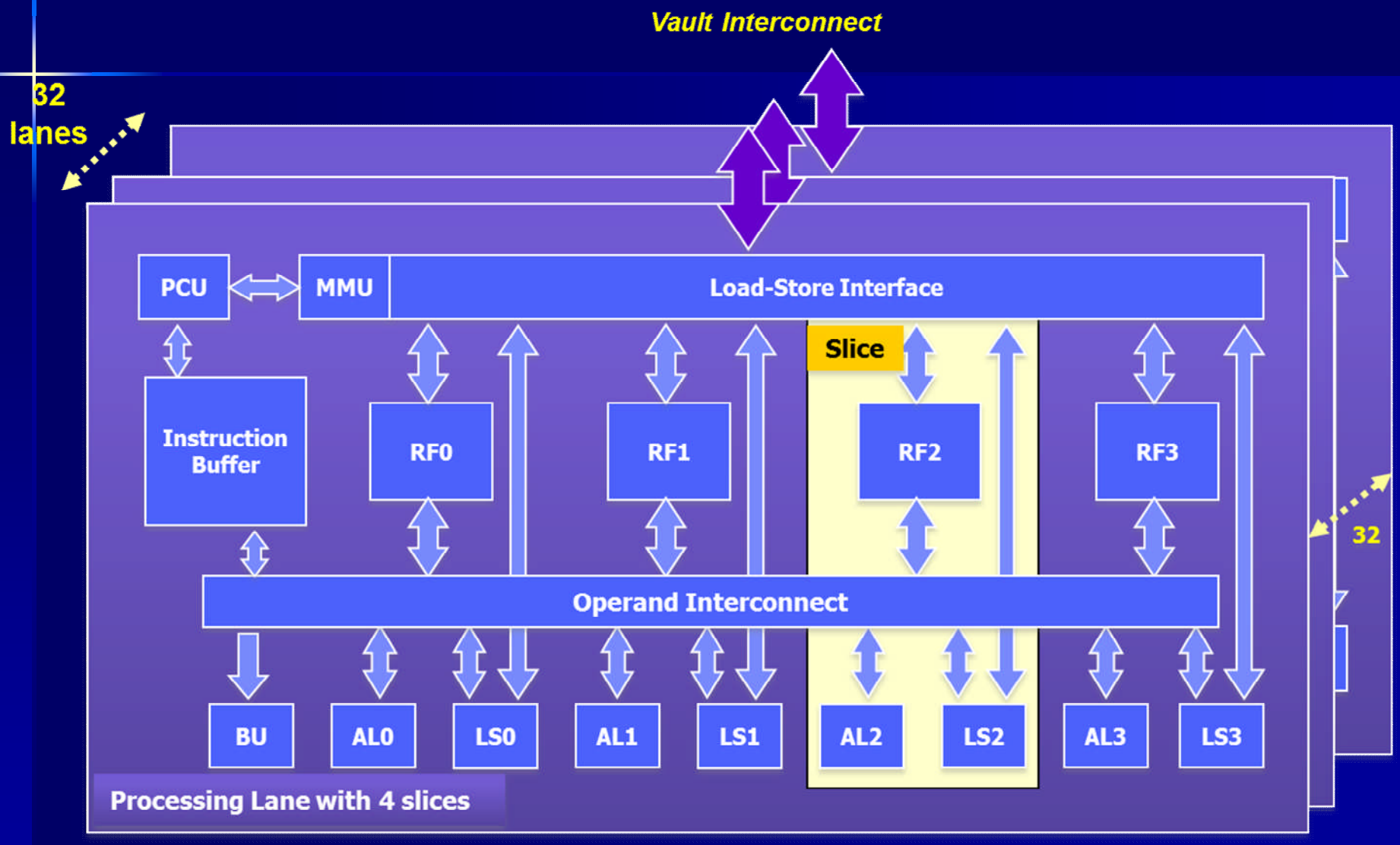
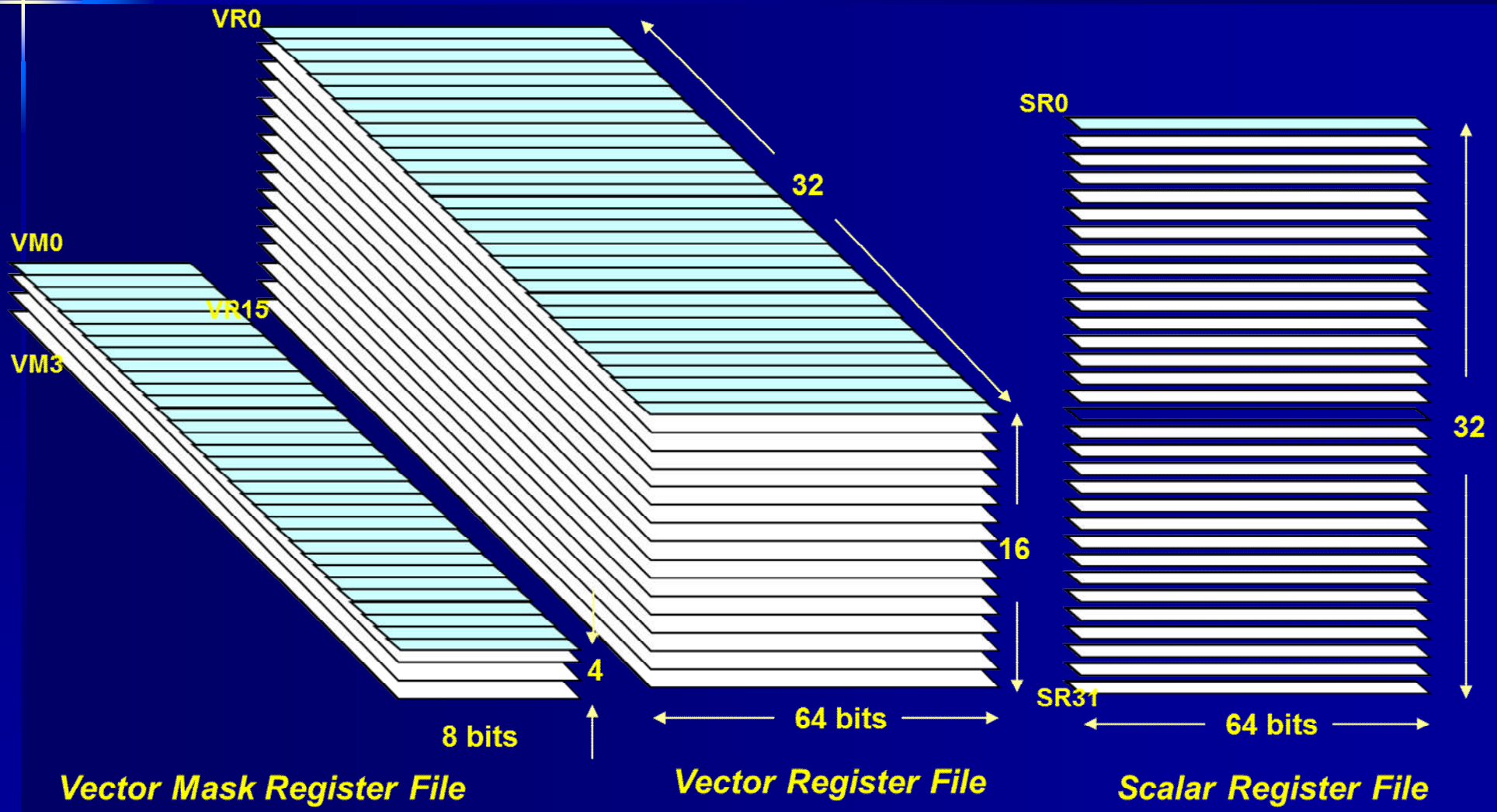


Figure from [IBM JRD]

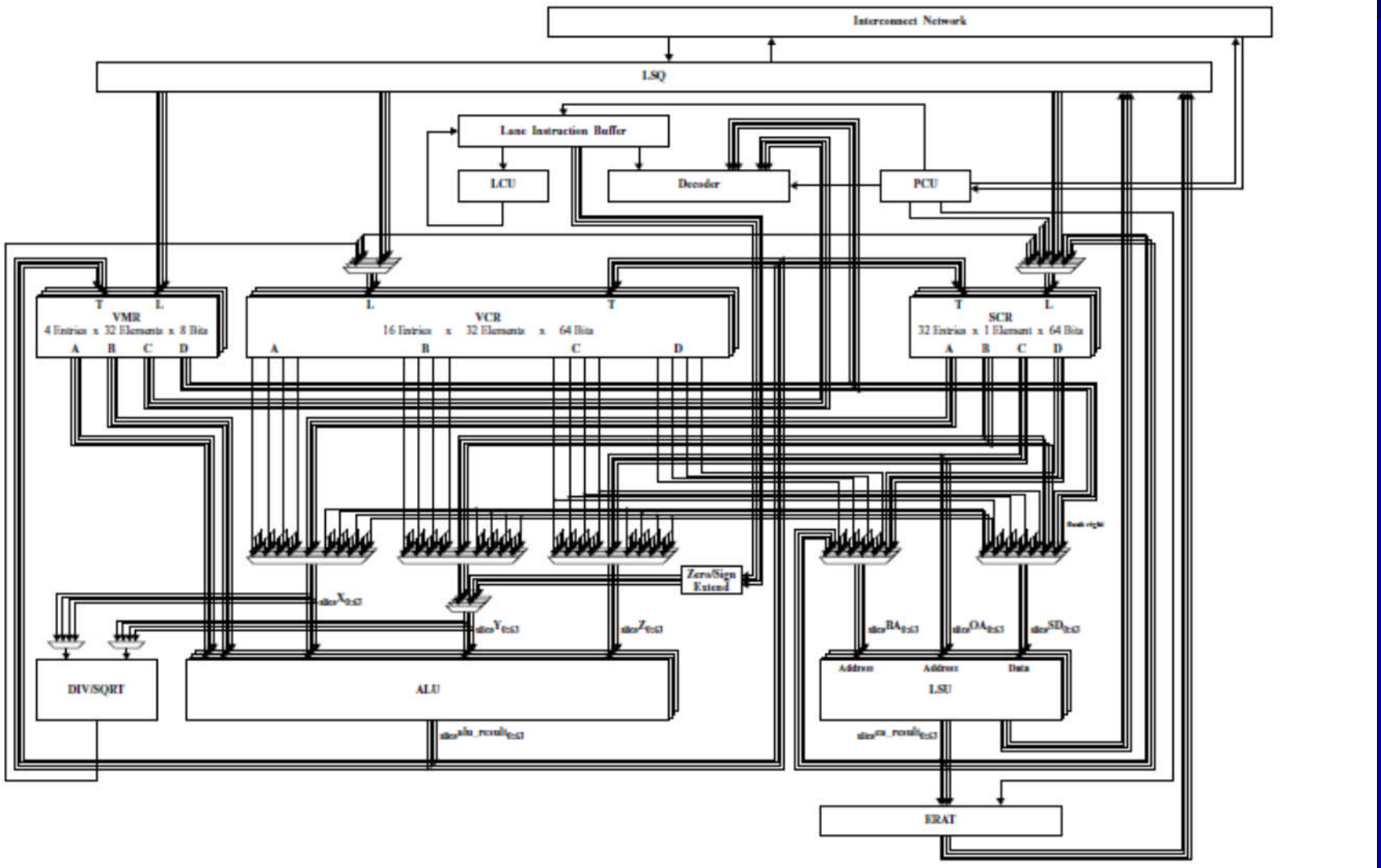
Processing Lane



Vector, Scalar and Mask Registers



Processing Lane



Power-Efficient Microarchitecture

- Direct path from memory to registers
- Short latency to memory
- Low pj/bit TSV interconnection between memory and processing lanes
- No dynamic scheduling of instructions (pipeline exposed to program)
- Explicit parallelism
- Efficient vector register file
- Chained operations to avoid intermediate writes to register file
- Largely predecoded commands to various execution units, similar to horizontal microcoding
- No instruction cache
- Aggressive gating of unused paths

Instruction Set

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP		D1COPY	D8SEL	D1ROTL	X1CMPEQ	X1ADD				F1NEG	F1RIP	F1ADD	F1DDIV	F1CMPEQ	
1			MFSPR		D2ROTL	X2CMPEQ	X2ADD				F2NEG	F2RIP	F2ADD	F1SDIV	F2CMPEQ	
2			MTSPR		D4ROTL	X4CMPEQ	X4ADD				F1ABS	F1RIM	F1SUB	F1DSQRT	F1CMPGT	
3					D8ROTL	X8CMPEQ	X8ADD				F2ABS	F2RIM	F2SUB	F1SSQRT	F2CMPGT	
4					D1SHL	X1CMPGTU	X1SUB				F1NABS	F1CTIDS	F1MUL	F1DISR	F1CMPGTEQ	
5					D2SHL	X2CMPGTU	X2SUB				F2NABS	F2CTIWS	F2MUL	F1SISR	F2CMPGTEQ	
6					D4SHL	X4CMPGTU	X4SUB				F1CPSGN	F1CTIDSZ	F1MADD		F1TSTNAN	
7					D8SHL	X8CMPGTU	X8SUB				F2CPSGN	F2CTIWSZ	F2MADD		F2TSTNAN	
8	D1LOGIC	M1LOGIC	D1CLZ		D1SHRU	X1CMPGTS	X1WMULU	X1WMADDU				F1CTIDU	F1MSUB		F1MAX	
9			D2CLZ		D2SHRU	X2CMPGTS	X1WLHMULU	X2WMADDU				F2CTIWU	F2MSUB		F2MAX	
A			D4CLZ		D4SHRU	X4CMPGTS	X2WMULU	X4HMADDU			F1RSP	F1CTIDUZ	F1NMADD		F1MIN	
B			D8CLZ		D8SHRU	X8CMPGTS	X4HMULU	X1WMADDS			F1CSTD	F2CTIWUZ	F2NMADD		F2MIN	
C			D1CO		D1SHRS		X1WMULS	X2WMADDS			F1RIN	F1CFIDS	F1NMSUB			
D			D2CO		D2SHRS		X1WLHMULS	X4HMADDS			F2RIN	F2CFIWS	F2NMSUB			
E			D4CO		D4SHRS		X2WMULS	X1WHLMSAU			F1RIZ	F1CFIDU	F2XMADD			
F			D8CO		D8SHRS		X4HMULS	X1WHLMSAS			F2RIZ	F2CFIWU	F2XNMSUBMADD			

	0	1	2	3
0	NOP			
1	LD8	LD8U	ST8	ST8U
2	LD4Z	LD4ZU	ST4	ST4U
3	LD4S	LD4SU		
4	LD4P	LD4PU	ST4P	ST4PU
5	LD2Z	LD2ZU	ST2	ST2U
6	LD2S	LD2SU		
7	LD2P	LD2PU	ST2P	ST2PU
8	LD1Z	LD1ZU	ST1	ST1U
9	LD1S	LD1SU		
A	LD1P	LD1PU	ST1P	ST1PU
B	LD1M	LD1MU	ST1M	ST1MU
C			AFA	
D				
E			ASTFA	ASTFAU
F			WAKEUP	

Bit ops

Integer ops: byte, halfword, word, doubleword

Floating-point ops: double precision, single precision

Load store ops: 1, 2, 4 and 8 bytes Single-element and packed

Instructions operate on scalar or vector operands

Sync ops

Sample AMC Lane Instruction

[32] nop {f1mul vr3,vr2,vr1; nop} {x2add vr5,vr7,vr9; nop} {nop; nop} {nop; nop}

Perform operation on next 32 elements. Do not branch at end of instruction – just continue to next instruction

DP multiply element from vector register 2 of slice 0 with element in vector register 1 of slice 0. Result in vector register 3 of slice 0. Bump register pointers to next element in each vector

32-bit integer add 2 elements from vector register 7 of slice 1 with 2 elements in vector register 9 of slice 1. Results in vector register 5 of slice 1. Bump register pointers to next element in each vector

[1] nop {f1mul sr3,sr2,sr1; nop} {x2add sr5,sr7,sr9; nop} {nop; nop} {nop; nop}

Same operations as above on registers in the scalar register file. Compiler cannot place dependent operation in next instruction.

LSU Commands, including strides, gather, scatter

[32] nop **{nop; l8u vr3,sr2,sr1}** **{nop; l8 vr5,vr7,sr9}** **{nop; nop}** **{nop; nop}**

Perform operation on next 32 elements

STRIDED LOAD:
Load DP element into vector register 3 of slice 0 from address obtained by adding scalar register 2 with scalar register 1. Next load will use the updated value for s1

GATHER:
Load DP element into vector register 5 of slice 1 from address provided by element of vector register 7 incremented by scalar value in sr9. Bump vector register to point to next element for next address.

Bandwidth and Latency (Sequential Access)

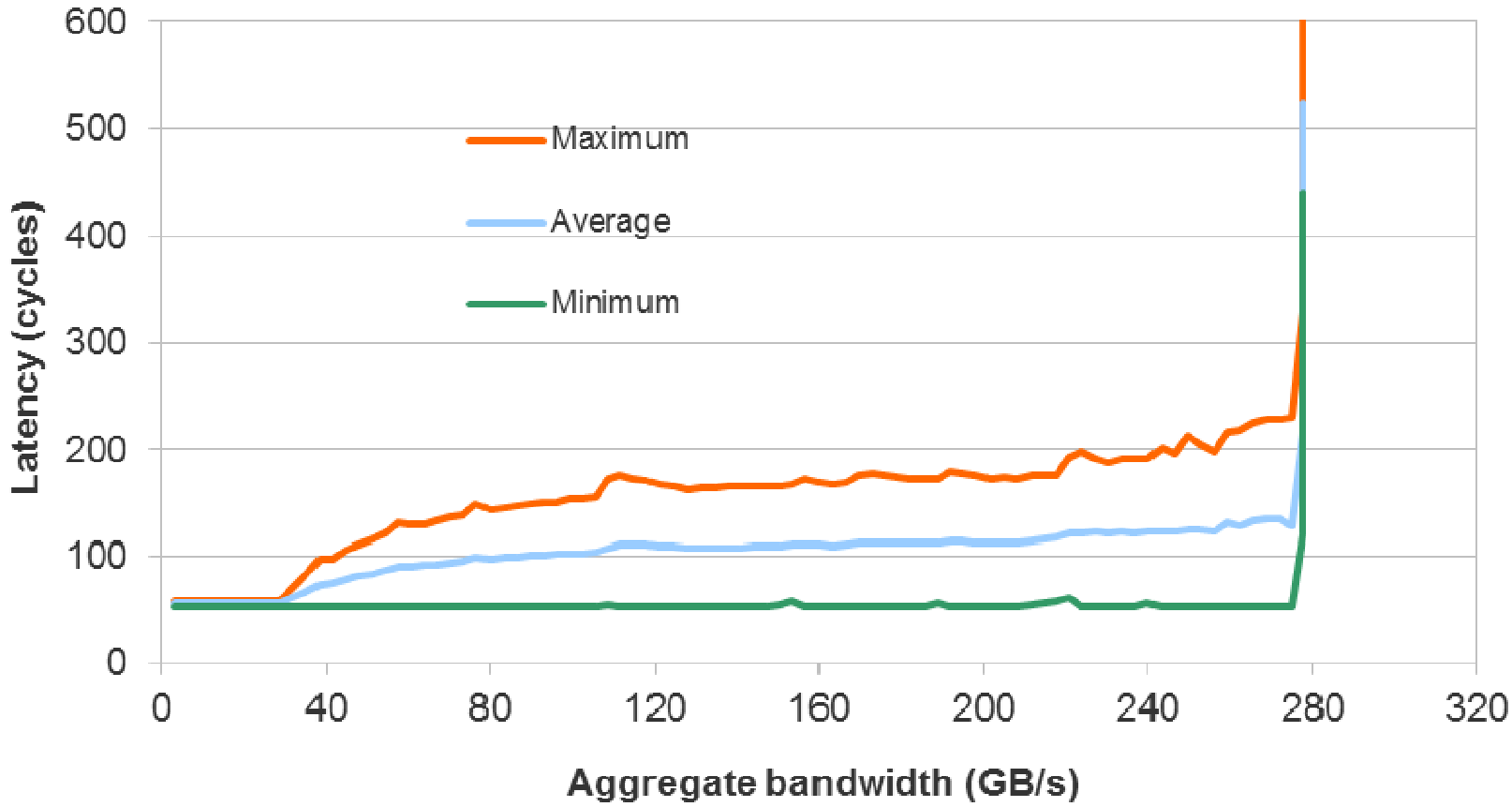


Figure from [IBM JRD]

Improvement with Open-Page Mode

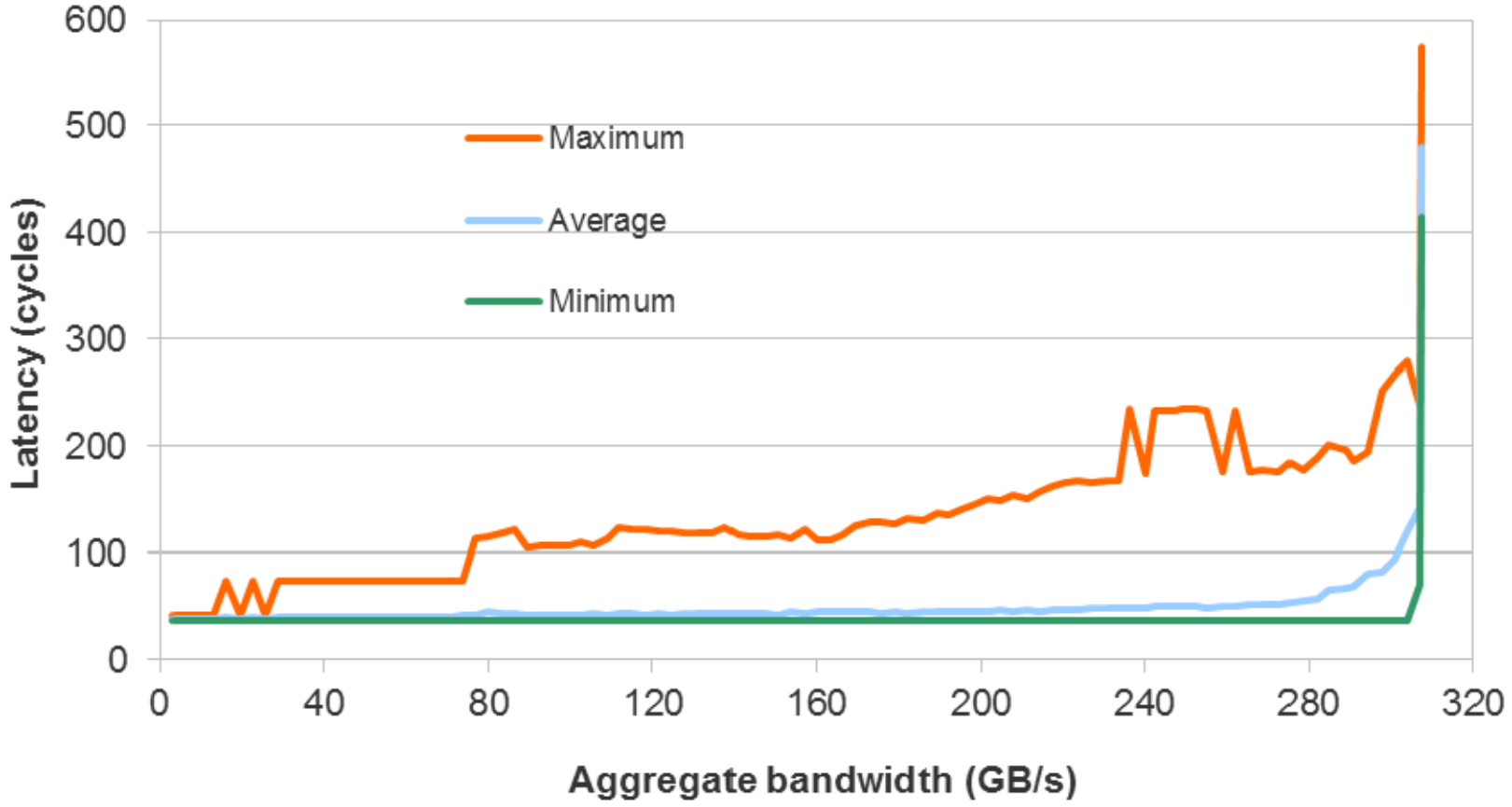
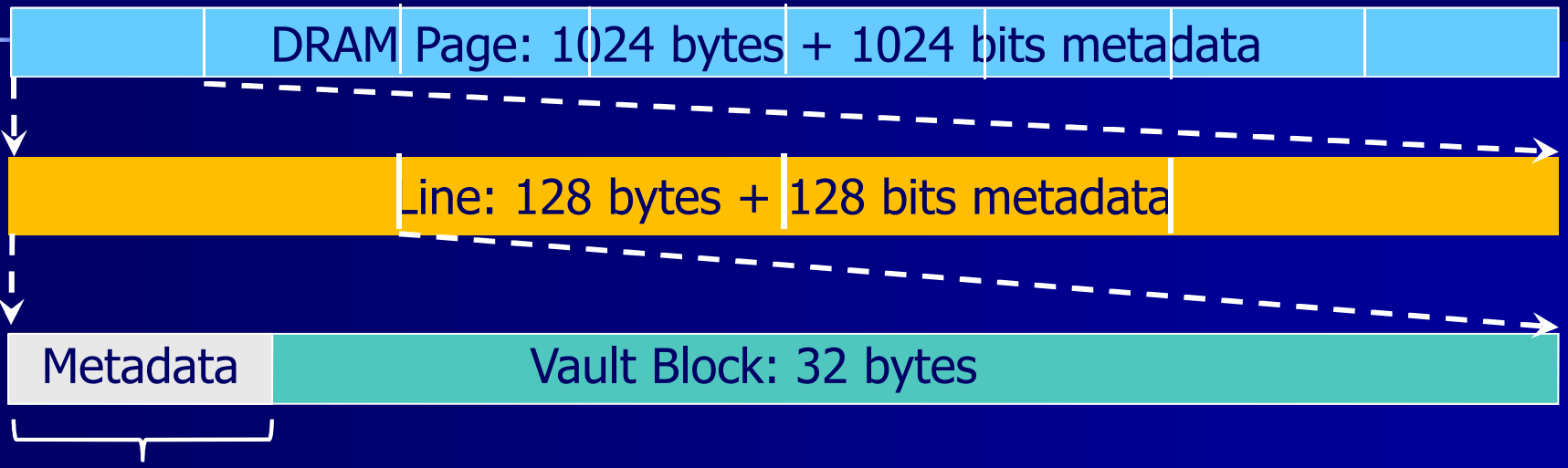


Figure from [IBM JRD]

Coherence



- No caches within AMC
- Granularity of transfer to lane
  - 8 bytes to 256 bytes
- Coherence bit checked on access
  - Overhead only on small-granularity stores

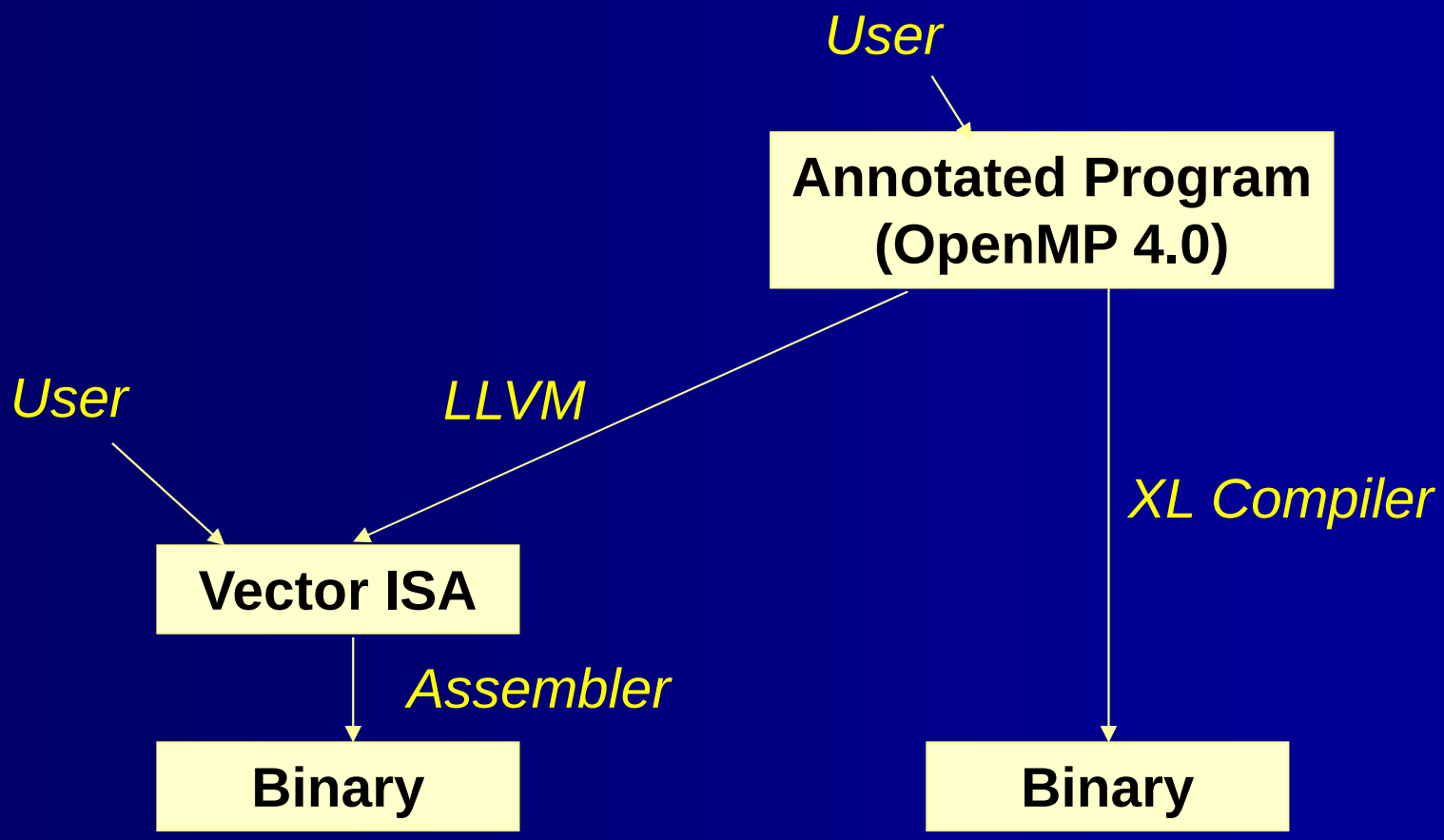
AMC API

- Goal:
  - Expose all hardware resources of interest to user
  - Allow efficient sharing of AMC execution units
  - Allow efficient user-mode access
- Logical naming of resources: Physical mapping by OS
  - OS is minimal kernel, Compute Node Kernel (CNK), employed on BG/Q
- Memory Placement: Allocation and relocation of memory
- AMC Interface Segment – for lane code and initialized data
- Data areas (with addresses in special-purpose registers)
  - Common Area: for data common to all lanes
  - Lane Stack: for data specific to a single lane
- Lane Completion Mechanism
  - Atomic fetch and decrement
  - Wakeup host

AMC Simulator

- Supports all features of Instruction Set Architecture
- Timing-correct at the AMC level, including internal interconnect and vaults
- Functionally correct at the host level, including privileged mode
- OS and runtime accurately simulated
- Provides support for power and reliability calculation
- Provides support for experimenting with different mix of resources and with new features

Compiling to the AMC



AMC Compiler

- Supports C, C++, Fortran front ends
- User annotates program using OpenMP 4.0 accelerator directives
  - Identify code section to run on AMC
  - Identify data region accessed by AMC

Compiled Applications

- DGEMM
- DAXPY
- Determinant
- LULESH
- SNAP
- Nekbone
- UMT
- CoMD

Challenges/Opportunities for Compiler Exploitation

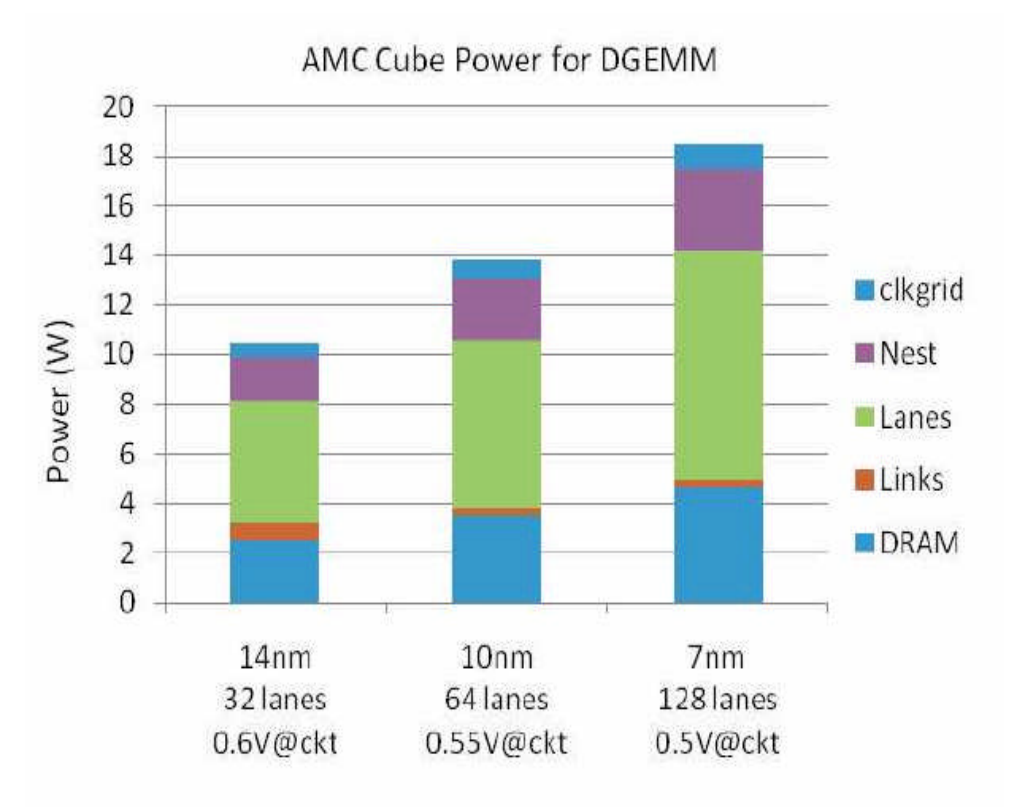
- Heterogeneity
- Programmable length vectors
- Gather-scatter
- Slice-mapping
- Predication using mask registers
- Scalar-vector combination code
- Memory latency hiding
- Instruction Buffer size

Techniques Used

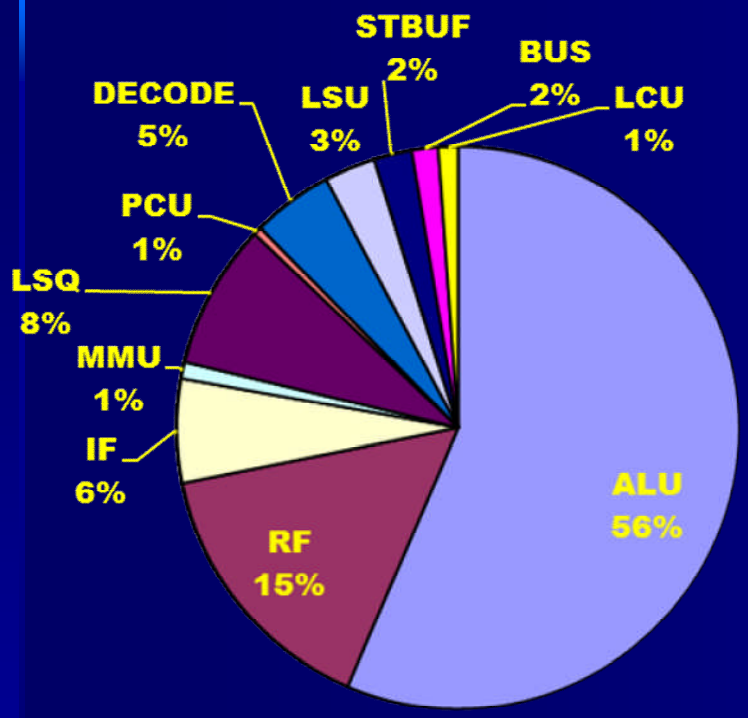
- Polyhedral framework
- Loop nests analyzed for vector exploitation
  - Loop blocking, loop versioning and loop unrolling applied in integrated manner
- Software I-cache
- Limited source code transformation
  - LULESH Kernel 1: Distribute kernel into two loops to help scheduling
  - Nekbone: Manually inline multiply nested procedure calls
  - Will be automated eventually

Performance

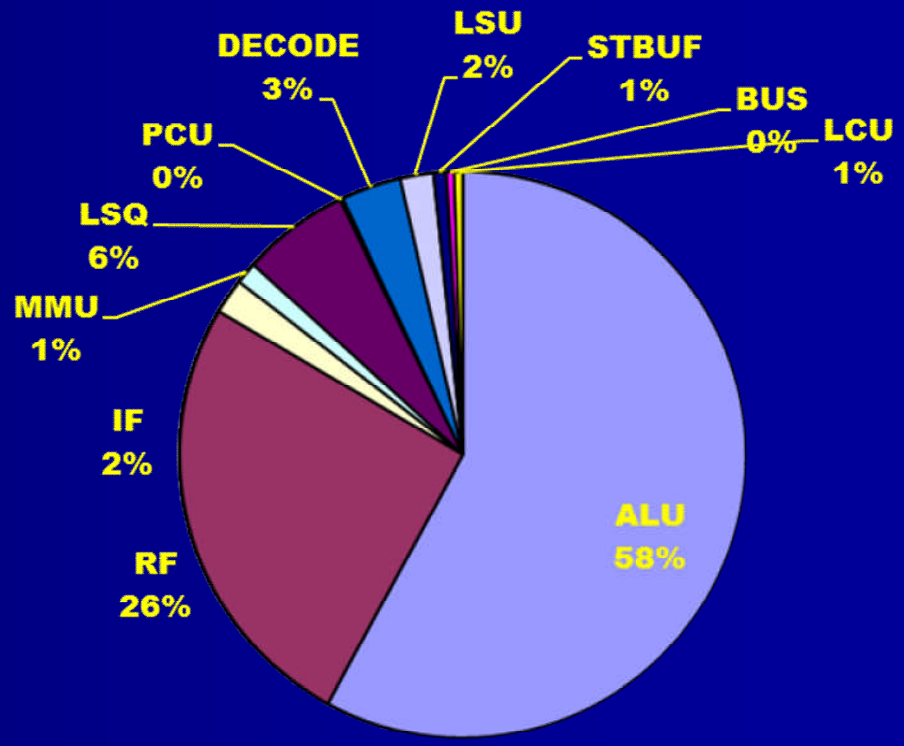
- DGEMM
  - **C** = **C** – **A** x **B**
  - 83% peak performance: 266 GF/AMC
    - Hand assembled
    - 77% through compiler
  - 10 W power in 14 nm technology
  - Roughly 20 GF/W at system level
  - 7 nm projection:
    - 56 GF/W at AMC
    - Roughly at target at system level



Breakdown of Resources



Area Breakdown

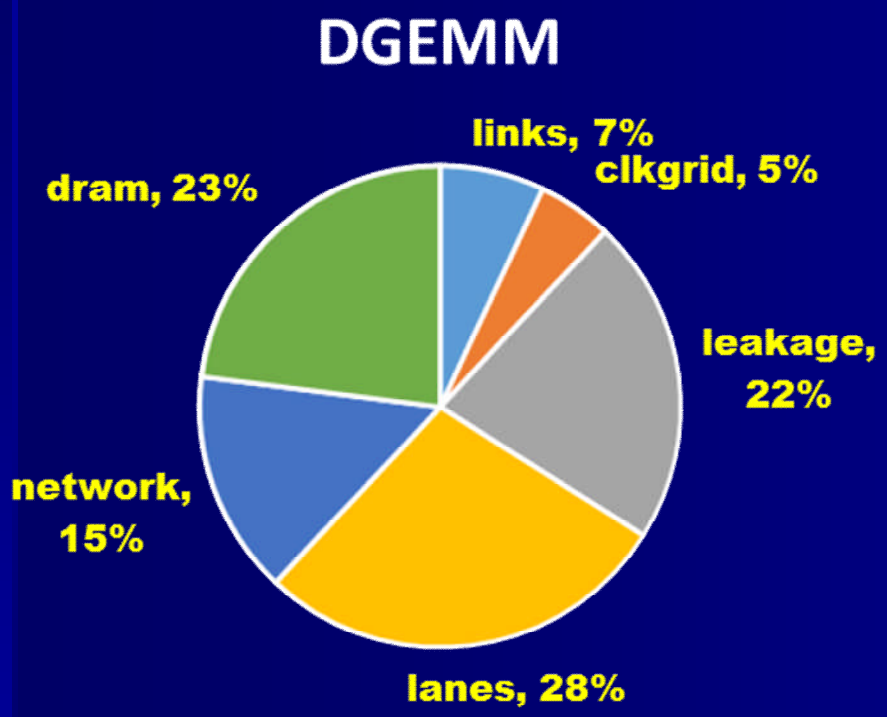


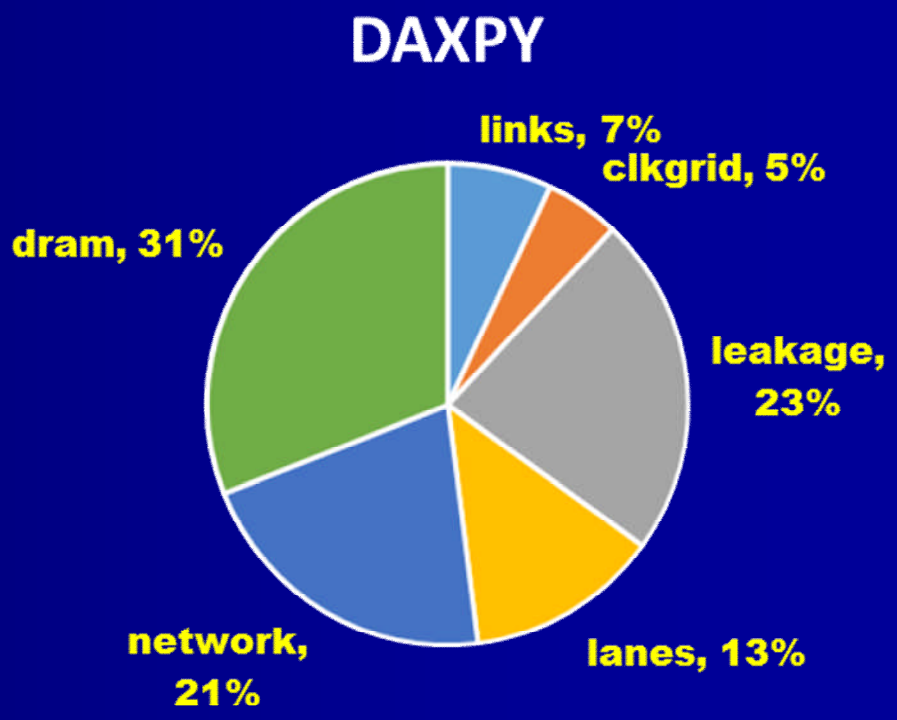
Power Breakdown

Bandwidth

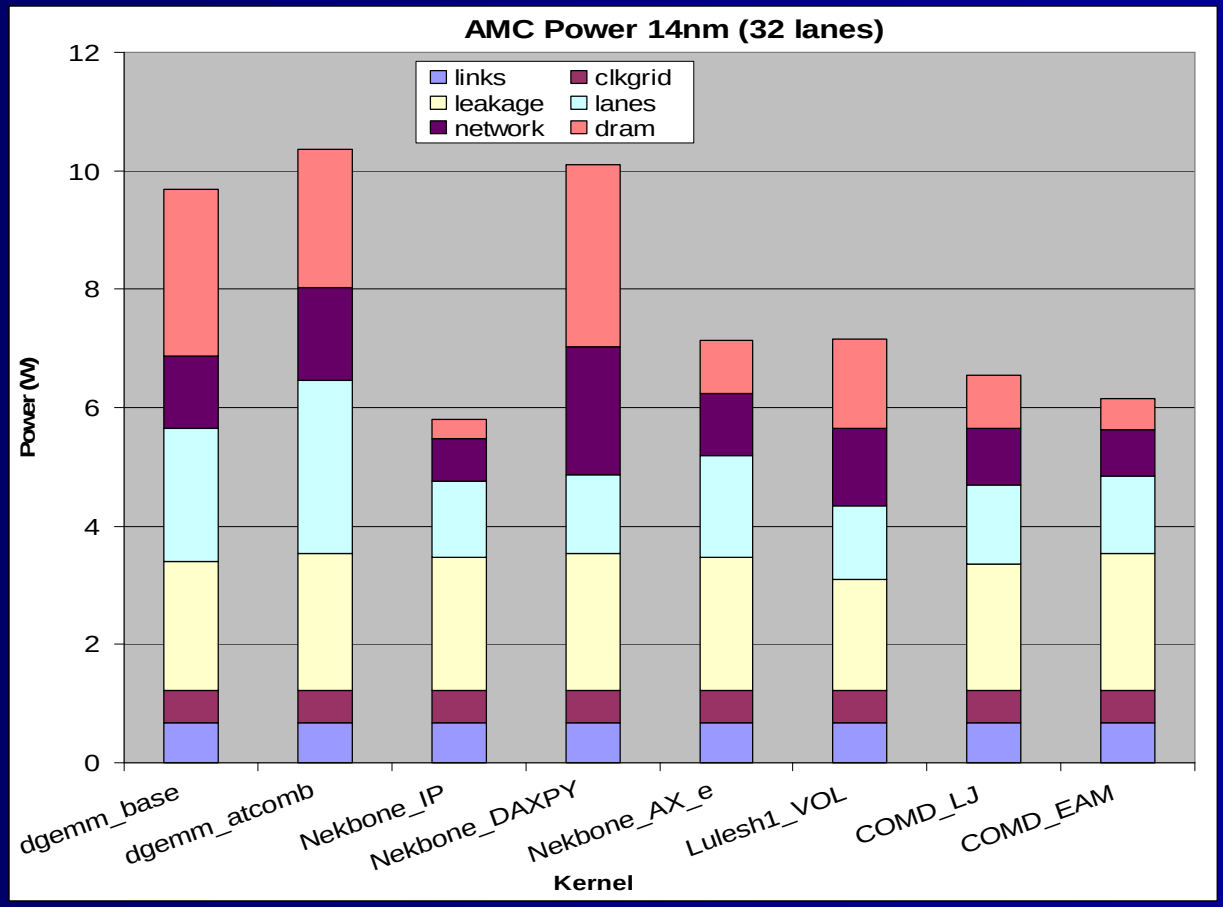
- DAXPY (**Y** = **Y** + *a* x **X**)
- Performance depends on data striping
- 4.95 computations per cycle per AMC when striped across all lanes
  - Out of 32 possible
  - Low because of bandwidth limitation
- Improves to 7.66 computations per cycle when data is blocked within vault
- Bandwidth utilized
  - 153.2 GB/s (read), 76.6 GB/s (write) per AMC
  - 2.4 TB/s (read), 1.2 TB/s (write) across node

Power Distribution





Power Consumption Across Applications



- Real applications tend to be less power-constrained

Concluding Remarks

- 3D technology is reviving interest in Processing-in-Memory
- The AMC design demonstrates a way to meet DoE Exascale requirements of 1 Exaflops in 20 MW
  - An AMC-like PIM approach may be the only way to achieve this target in 7 nm CMOS technology
- Widespread use of the AMC technology will depend on commodity adoption of such 3D memory+logic stacks

Thank you!

- This work was supported in part by Department of Energy Contract B599858 under the FastForward initiative
  - PIs: Ravi Nair and Jaime Moreno

- Collaborators
  - S. F. Antao, C. Bertolli, P. Bose, J. Brunheroto, T. Chen, C.-Y. Cher, C. H. A. Costa, J. Doi, C. Evangelinos, B. M. Fleischer, T. W. Fox, D. S. Gallo, L. Grinberg, J. A. Gunnels, A. C. Jacob, P. Jacob, H. M. Jacobson, T. Karkhanis, C. Kim, J. K. O'Brien, M. Ohmacht, Y. Park, D. A. Prener, B. S. Rosenburg, K. D. Ryu, O. Sallenave, M. J. Serrano, P. Siegl, K. Sugavanam, Z. Sura
- Upcoming Paper

[IBM JRD] R. Nair et al, "Active Memory Cube: A Processing-in-Memory Architecture for Exascale Systems", in *IBM Journal of Research & Development*, vol. 59, no. 2/3, 2015.