Power Systems

IBM

Advanced High-Performance Computing Features of the OpenPOWER ISA

Workshop on RISC-V and OpenPOWER
International Conference on Supercomputing
June 20, 2020

José Moreira – *IBM Research*

Many thanks to: Jeff Stuecheli
Edmund Gieske
Brian Thompto

Open architectures for supercomputing

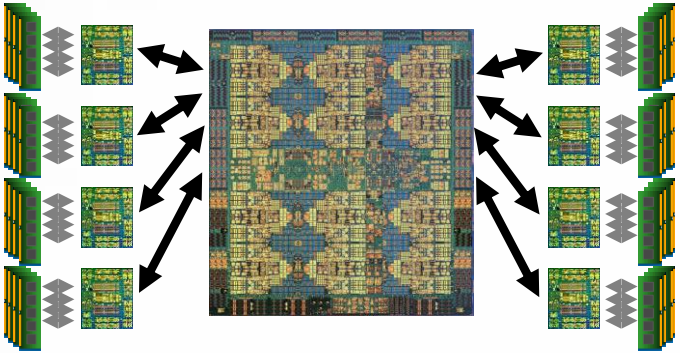
- Since the dawn of supercomputing with the CDC 6600, through the many generations of Seymour Cray machines, massively parallel processors like Blue Gene and the current champion Fugaku, high performance computing has been about three things:
  - Number crunching: perform as many arithmetic operations as possible
  - Memory bandwidth: read/write as much data from/to memory as possible
  - Interconnect: communicate between elements as fast as possible

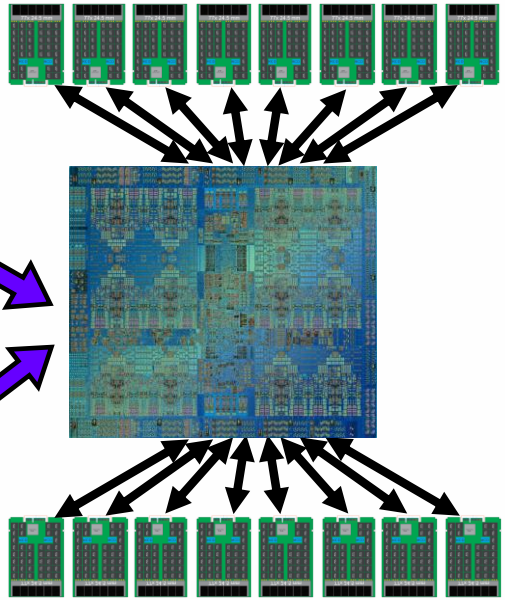
- Through the OpenPOWER initiative, IBM has made a variety of computing technologies openly available to the community, *including the three HPC-enabling technologies listed above*
- In this talk, we will focus on recent OpenPOWER developments in the areas of number crunching and memory bandwidth
- But don't forget the interconnect if you are building a supercomputer ☺

POWER family memory architecture

Scale Up
Buffered Memory

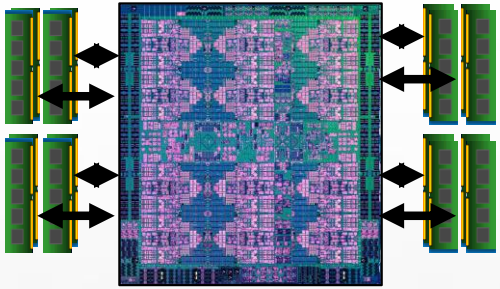
Superior RAS, High bandwidth, High Capacity
Agnostic interface for alternate memory innovations





Scale Out
Direct Attach Memory

Low latency access
Commodity packaging form factor

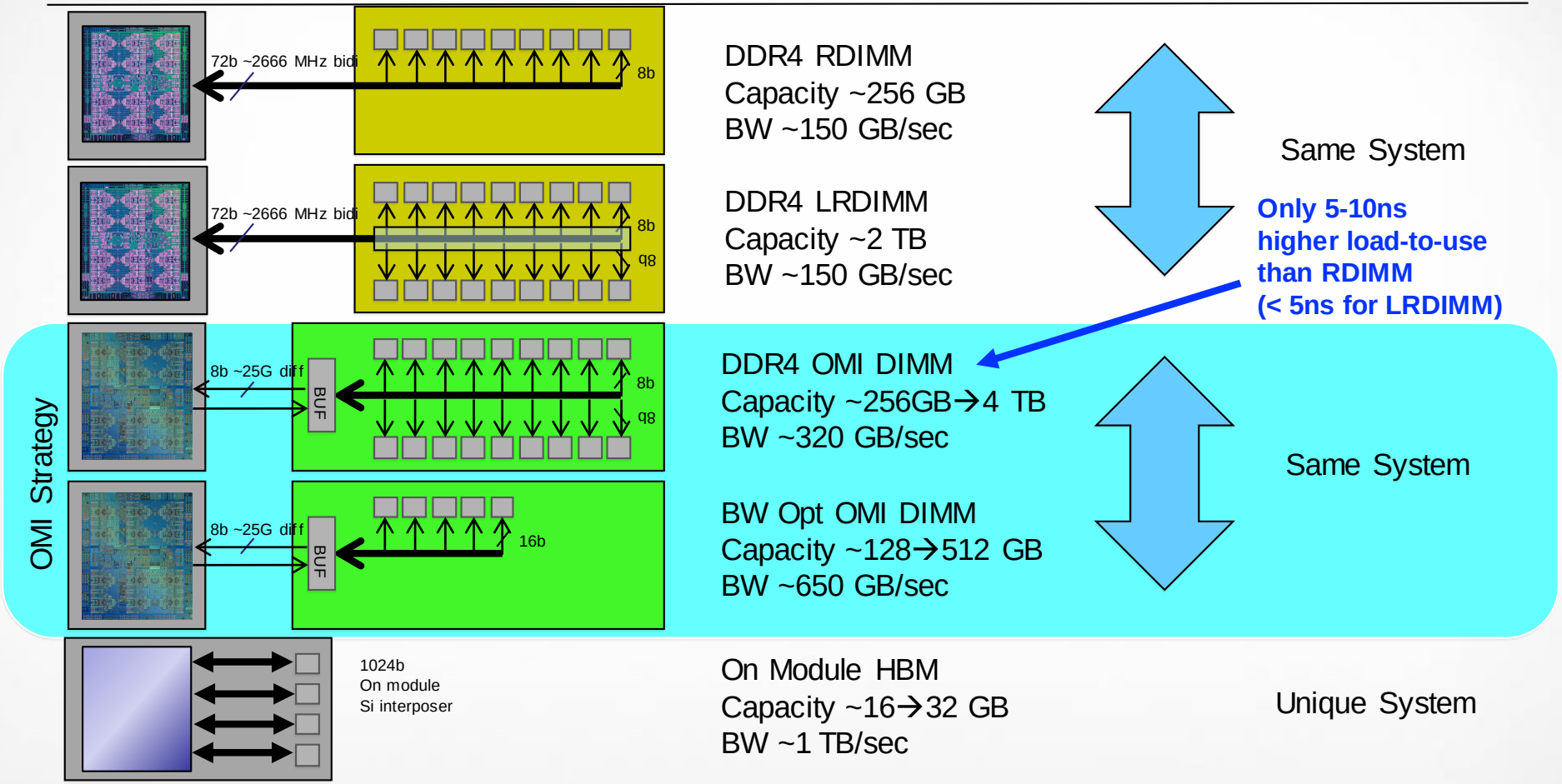


OpenCAPI Agnostic Buffered Memory (OMI)

Near Tier	Commodity	Enterprise	Storage Class
Extreme Bandwidth	Low Latency	RAS	Extreme Capacity
Lower Capacity	Low Cost	Capacity	Persistence
		Bandwidth	

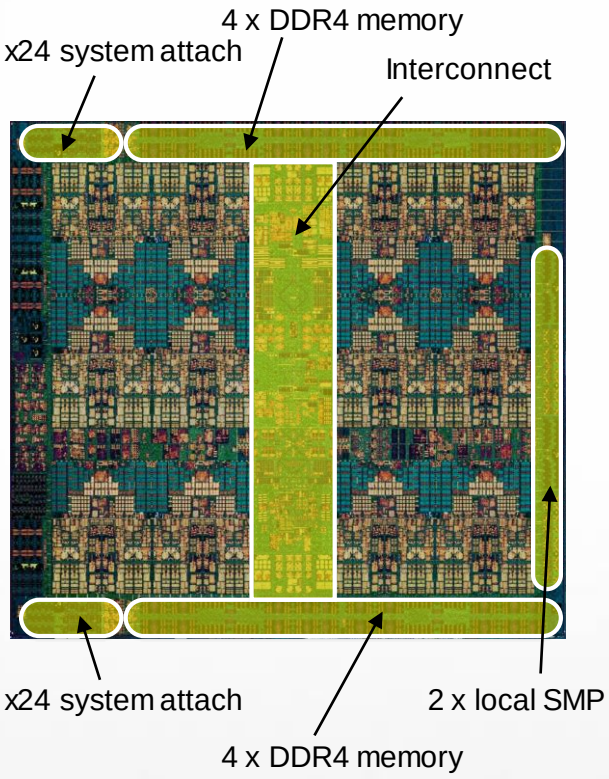
Same Open Memory Interface used for all Systems and Memory Technologies

Primary tier memory options

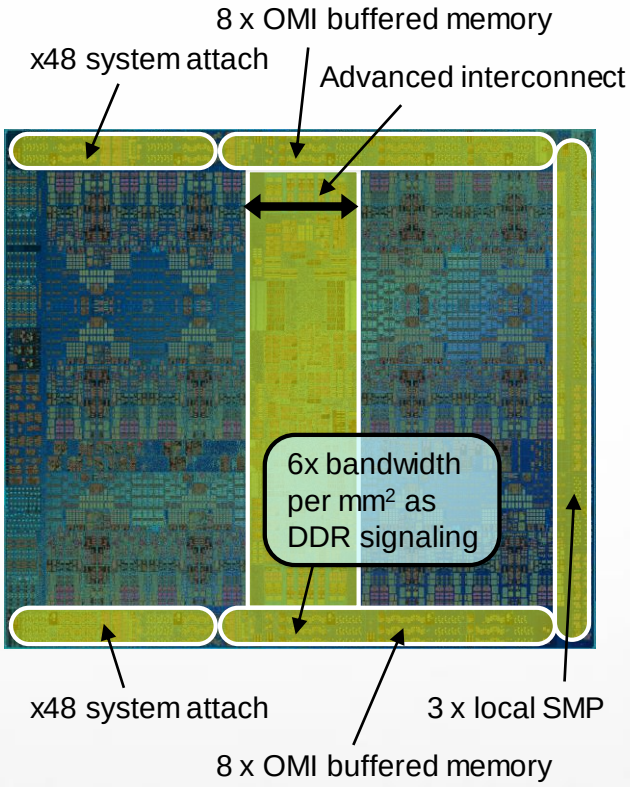


POWER connectivity variants

Direct Attach Memory
Max capacity: 2 TB
Max bandwidth: 150 GB/s
Limited system interconnect



OMI Buffered Memory
Max capacity: 4 TB
Max bandwidth: 650 GB/s
Enhanced system interconnect



DRAM DIMM comparison

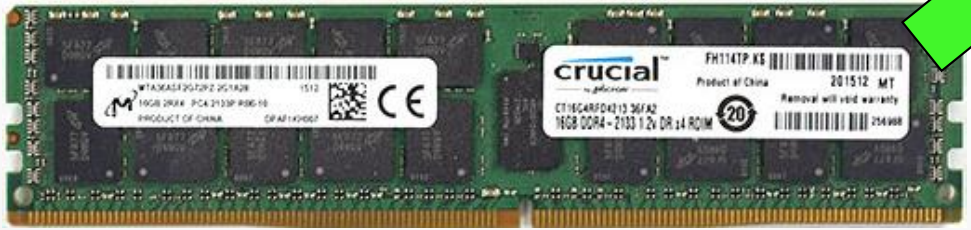
IBM Centaur DIMM

Ultra-scale

OMI DDIMM

Economy

JEDEC DDR DIMM



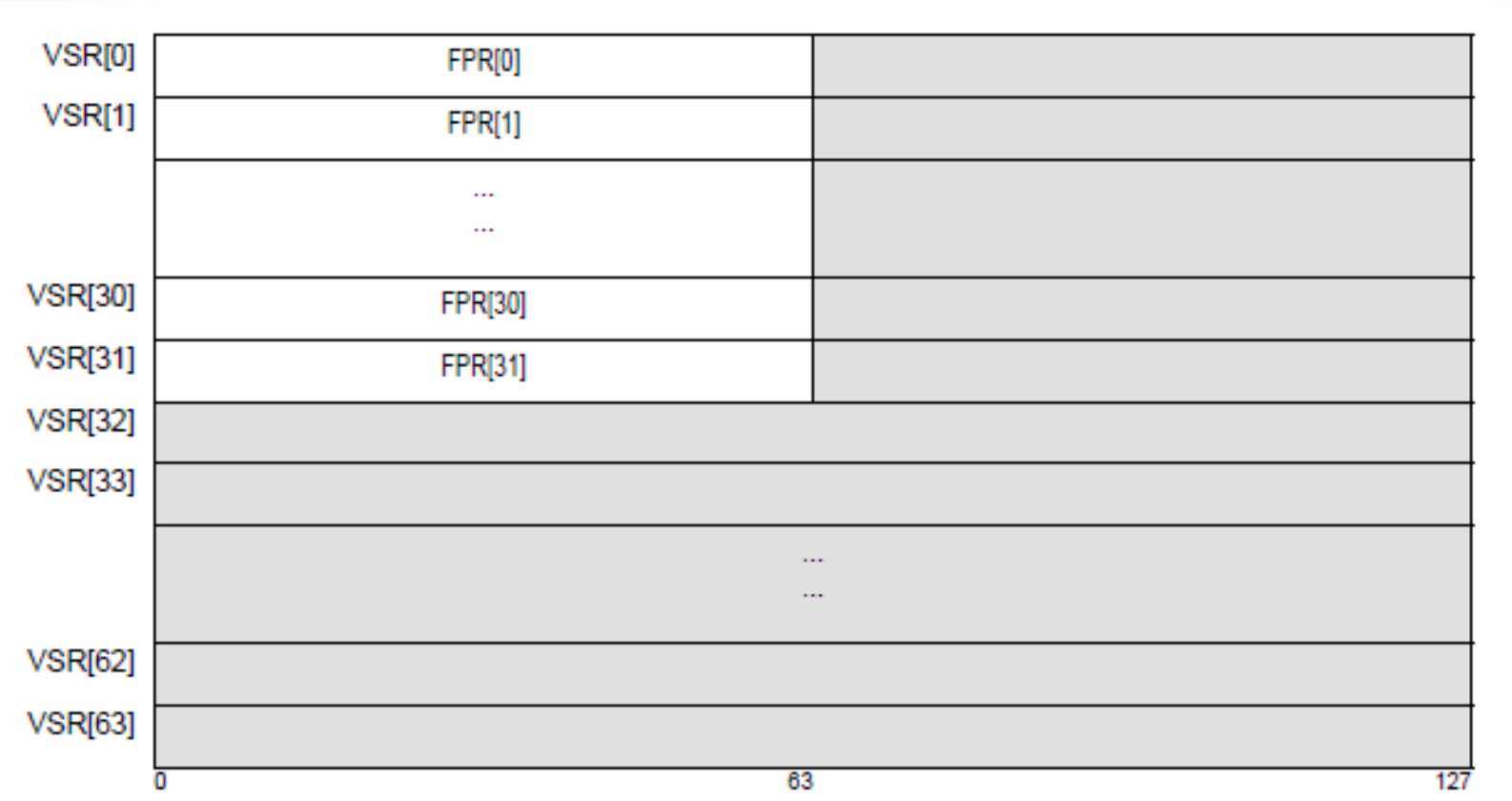
- Technology agnostic
- Low cost
- Ultra-scale system density
- Enterprise reliability
- Low-latency
- High bandwidth

Approximate Scale

Matrix-Multiply Assist (MMA) instructions

- The latest version of Power ISA (for POWER10) is now publicly available
- The Matrix-Multiply Assist instructions lead to very efficient implementations for high performance computing
- These instructions are a natural match for implementing dense numerical linear algebra computations
- We have also shown application to other computations such as convolution
- Various other computations require additional work and research, including arbitrary precision arithmetic, discrete Fourier transform, …

Power ISA Vector-Scalar Registers (VSRs)



Accumulators

- Accumulators are 4×4 arrays of 32-bit elements (we will briefly mention the 64-bit extension later)

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$

- The elements can be either 32-bit signed-integers (int32) or 32-bit single-precision floating-point numbers (fp32)
- Each accumulator is a software-managed shadow to a set of 4 consecutive VSRs (8 architected accumulators – ACC[0:7]) – must choose between using accumulator or associated VSRs
- State must be explicitly transferred between accumulators and VSRs using *VSX Move From Accumulator* (***xxmfacc***) and *VSX Move To Accumulator* (***xxmtacc***)

Outer-product (*xv*<type>*ger*<rank-k>) instructions

- Accumulators are updated by rank-k update instructions:
  - Input: 1 accumulator (A) + 2 VSRs (X, Y)
  - Output: 1 accumulator (same as input to reduce instruction encoding space)
- Operation: $A \leftarrow \{\pm\} A \{\pm\} XY^T$
  - For 32-bit data, X and Y are 4×1 arrays of elements
  - For 16-bit data, X and Y are 4×2 arrays of elements
  - For 8-bit data, X and Y are 4×4 arrays of elements
  - For 4-bit data, X and Y are 4×8 arrays of elements
- This way XY^T always has a 4×4 shape, compatible with accumulator

Instruction	A	X	Y^T	# of madds/instruction
xvf32ger	4×4 (fp32)	4×1 (fp32)	1×4 (fp32)	16
xvf16ger2	4×4 (fp32)	4×2 (fp16)	2×4 (fp16)	32
xvi16ger2	4×4 (int32)	4×2 (int16)	2×4 (int16)	32
xvi8ger4	4×4 (int32)	4×4 (int8)	4×4 (int8)	64
xvi4ger8	4×4 (int32)	4×8 (int4)	8×4 (int4)	128

Extension to 64-bit

- Accumulators are 4×2 arrays of 64-bit floating-point elements

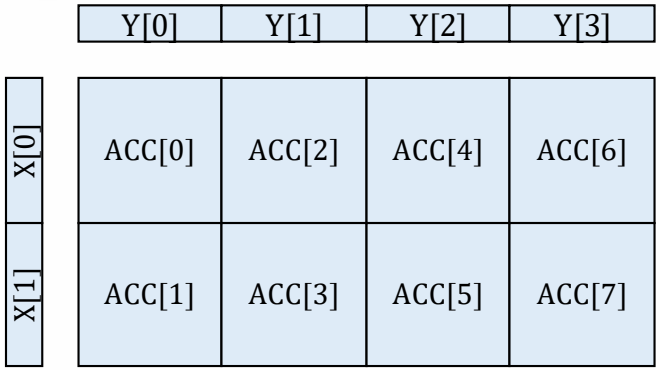
$$A = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \\ a_{20} & a_{21} \\ a_{30} & a_{31} \end{bmatrix}$$

- Accumulators are updated by outer-product instructions:
  - Input: 1 accumulator (A) + 3 VSRs (X_1, X_2, Y)
  - Output: 1 accumulator (same as input to reduce instruction encoding space)
- Operation: $A \leftarrow A + \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} Y^T$
  - X_1, X_2 and Y are 2×1 arrays of elements
  - $\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} Y^T$ always has a 4×2 shape, compatible with accumulator

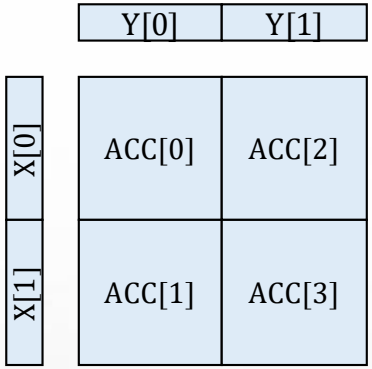
Instruction	A	X	Y^T	# of madds/instruction
xvf64ger	4×2 (fp64)	4×1 (fp64)	1×2 (fp64)	8

Load pressure and unit latency (32-bit results)

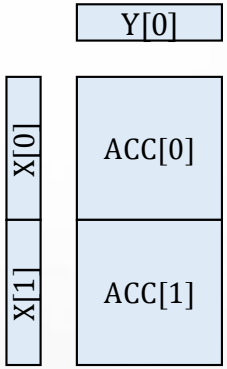
- 8 accumulators (8 × 16 result)
  - 6 VSR loads/8 ***xv_ger*** instructions
  - 0.75 VSR load/***xv_ger***
  - Tolerates the most latency



- 4 accumulators (8 × 8 result)
  - 4 VSR loads/4 ***xv_ger*** instructions
  - 1 VSR load/***xv_ger***
  - Could work well in SMT modes

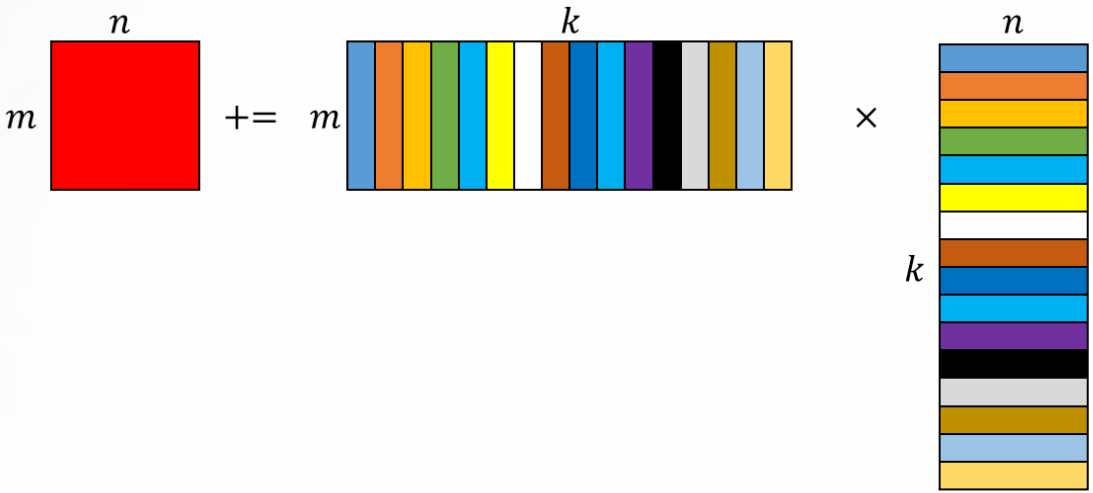


- 2 accumulators (8 × 4 result)
  - 3 VSR loads/2 ***xv_ger*** instructions
  - 1.5 VSR load/ ***xv_ger***

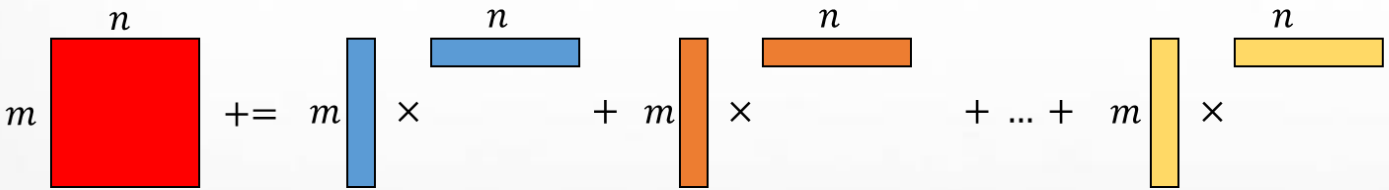


- 1 accumulator (4 × 4 result)
  - 2 VSR loads/1 ***xv_ger*** instruction
  - 2 VSR load/ ***xv_ger***

The micro-kernel of GEMM: $C_{m\times n} += A_{m\times k} \times B_{k\times n}$



- Load a small, “squarish” panel of C and keep it in registers
- Load one small column of A and one small row of B
- Outer-product and accumulate
- Repeat!



SGEMM micro-kernel using 8×8 virtual accumulator


```
00000000100006f0 sgemm_kernel_8x96x8:
…
100006fc:  01 00 05 f4     lxv 0, 0(5)
10000700:  21 00 25 f4     lxv 1, 32(5)
10000704:  41 00 45 f4     lxv 2, 64(5)
10000708:  61 00 65 f4     lxv 3, 96(5)
1000070c:  11 00 85 f4     lxv 4, 16(5)
10000710:  31 00 a5 f4     lxv 5, 48(5)
10000714:  51 00 c5 f4     lxv 6, 80(5)
10000718:  71 00 e5 f4     lxv 7, 112(5)
1000071c:  81 00 05 f5     lxv 8, 128(5)
10000720:  a1 00 25 f5     lxv 9, 160(5)
10000724:  c1 00 45 f5     lxv 10, 192(5)
10000728:  e1 00 65 f5     lxv 11, 224(5)
1000072c:  91 00 85 f5     lxv 12, 144(5)
10000730:  b1 00 a5 f5     lxv 13, 176(5)
10000734:  d1 00 c5 f5     lxv 14, 208(5)
10000738:  f1 00 e5 f5     lxv 15, 240(5)
```

load accumulators

```
1000073c:  62 01 01 7c     xxmtacc 0
10000740:  62 01 81 7c     xxmtacc 1
10000744:  62 01 01 7d     xxmtacc 2
10000748:  62 01 81 7d     xxmtacc 3
```

transfer to

```
1000074c:  00 00 23 18     lxvp 32, 0(3)
10000750:  00 00 24 1a     lxvp 48, 0(4)
10000754:  d6 80 00 ec     xvf32gerpp 0, 32, 48
10000758:  d6 88 80 ec     xvf32gerpp 1, 32, 49
1000075c:  d6 80 01 ed     xvf32gerpp 2, 33, 48
10000760:  d6 88 81 ed     xvf32gerpp 3, 33, 49
```

compute 1

```
10000764:  20 00 23 18     lxvp 32, 32(3)
10000768:  20 00 24 1a     lxvp 48, 32(4)
1000076c:  d6 80 00 ec     xvf32gerpp 0, 32, 48
10000770:  d6 88 80 ec     xvf32gerpp 1, 32, 49
10000774:  d6 80 01 ed     xvf32gerpp 2, 33, 48
10000778:  d6 88 81 ed     xvf32gerpp 3, 33, 49
```

compute 2

```
…
10001034:  e0 0b 23 18     lxvp 32, 3040(3)
10001038:  e0 0b 24 1a     lxvp 48, 3040(4)
1000103c:  d6 80 00 ec     xvf32gerpp 0, 32, 48
10001040:  d6 88 80 ec     xvf32gerpp 1, 32, 49
10001044:  d6 80 01 ed     xvf32gerpp 2, 33, 48
10001048:  d6 88 81 ed     xvf32gerpp 3, 33, 49
```

compute 96

```
1000104c:  62 01 00 7c     xxmfacc 0
10001050:  62 01 80 7c     xxmfacc 1
10001054:  62 01 00 7d     xxmfacc 2
10001058:  62 01 80 7d     xxmfacc 3
```

transfer from

```
1000105c:  05 00 05 f4     stxv 0, 0(5)
10001060:  25 00 25 f4     stxv 1, 32(5)
10001064:  45 00 45 f4     stxv 2, 64(5)
10001068:  65 00 65 f4     stxv 3, 96(5)
1000106c:  15 00 85 f4     stxv 4, 16(5)
10001070:  35 00 a5 f4     stxv 5, 48(5)
10001074:  55 00 c5 f4     stxv 6, 80(5)
10001078:  75 00 e5 f4     stxv 7, 112(5)
1000107c:  85 00 05 f5     stxv 8, 128(5)
10001080:  a5 00 25 f5     stxv 9, 160(5)
10001084:  c5 00 45 f5     stxv 10, 192(5)
10001088:  e5 00 65 f5     stxv 11, 224(5)
1000108c:  95 00 85 f5     stxv 12, 144(5)
10001090:  b5 00 a5 f5     stxv 13, 176(5)
10001094:  d5 00 c5 f5     stxv 14, 208(5)
10001098:  f5 00 e5 f5     stxv 15, 240(5)
…
```

store accumulators

Conclusions

- OpenPOWER delivers the three essentials of a supercomputer:
  - Number crunching: Through its SIMD and MMA instructions
  - Memory bandwitth: Through OMI
  - Interconnect: Through OpenCAPI (not discussed in this talk)
- The MMA instructions provide a new level of performance for dense linear algebra and related computations
- OMI provides a new level of memory bandwidth for computing systems while delivering low cost, versatility and capacity
- Both are scalable, offering room to growth in both dimensions
- Together with Open Source software, we see a clear road ahead for OpenHPC systems that combine the best innovation from the community!