www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

High Performance Computing Aspects of Deep Learning

Marc Casas (marc.casas@bsc.es)

PART1: INTRODUCTION TO HIGH-PERFORMANCE COMPUTING

What is High Performance Computing (HPC)?

- HPC is the biggest and fastest computing right this minute
- A Supercomputer is one of the biggest and fastest computers right this minute
- So, the definitions of HPC and Supercomputers are constantly changing.
- A supercomputer is typically 100 times faster than a PC

Cray-I Computer (1975)

Piz Daint (2016)

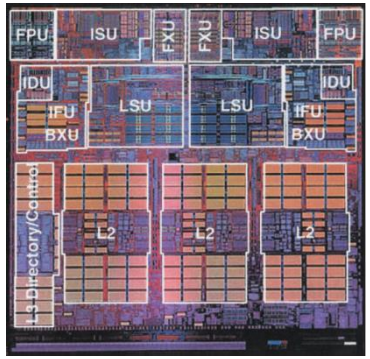
Architecture of Supercomputers

- Supercomputers have been designed in many different ways across history:
  - In the 70's and the 80's, they were vector processors specifically designed for HPC;
    - Cray1 (1975) eight vector registers, which held sixty-four 64-bit words each. Vector instructions applied between registers. Reached 160 MFLOP/s.
  - In the 90's, supercomputers started integrating 10's or 100's processors
  - At the end of the 20th century, they started to use large amounts of commodity hardware.
    - ASCI Red (1997) 18,000 Pentium Pro processors. Reached 1 TFLOP/s.
- Currently, they are mainly composed of 3 components:
  - Compute Nodes
  - High-Speed Interconnecting Network
  - Disk Storage
- Examples:
  - MareNostrum4 153,216 CPU's, 512-bits SIMD. Reaches 6PFLOP/s

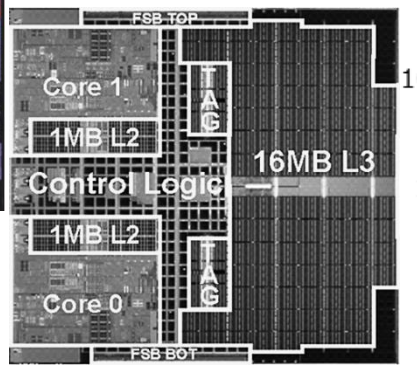
Compute Nodes based on Multicore Architectures

- Moore's Law + Memory Wall + Power Wall

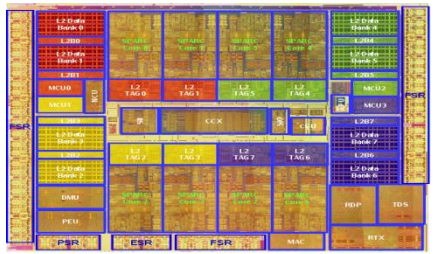
Chip MultiProcessors (CMPs)



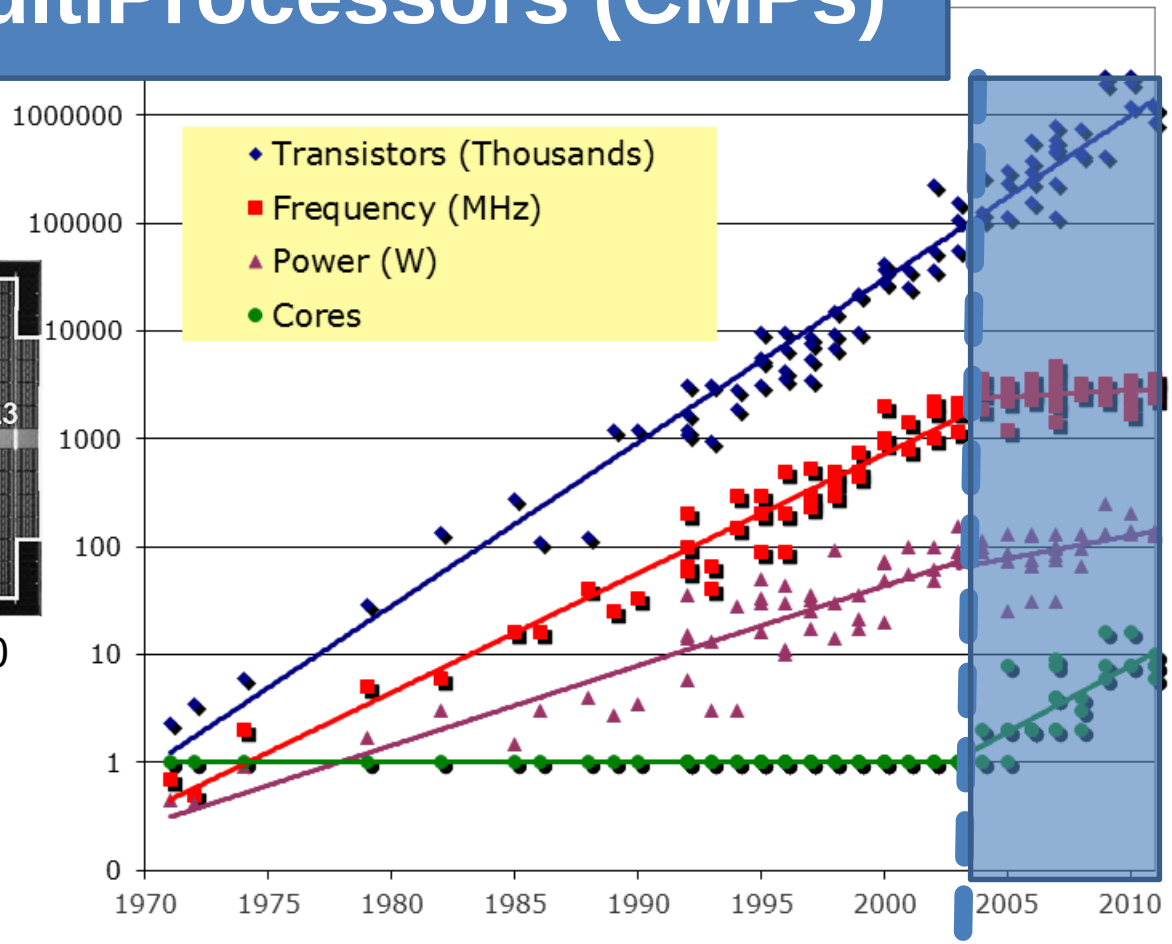
POWER4 (2001)



Intel Xeon 7100 (2006)



UltraSPARC T2 (2007)



How are the Multicore architectures designed?

IBM Power4 (2001)

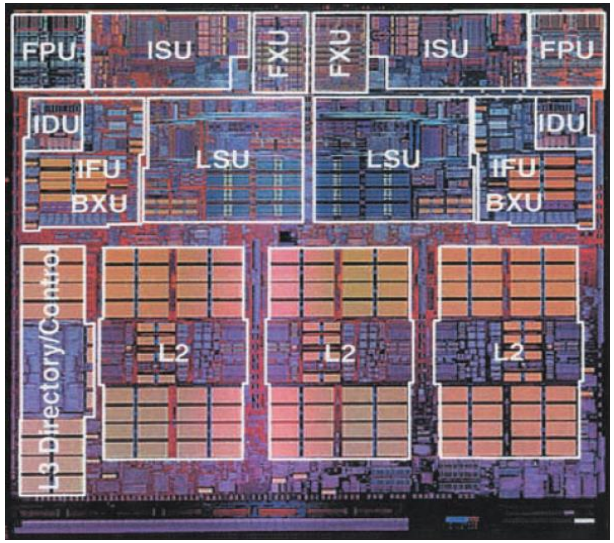
- 2 cores, ST
- 0.7 MB/core L2, 16MB/core L3 (off-chip)
- 115W TDP
- 10GB/s mem BW

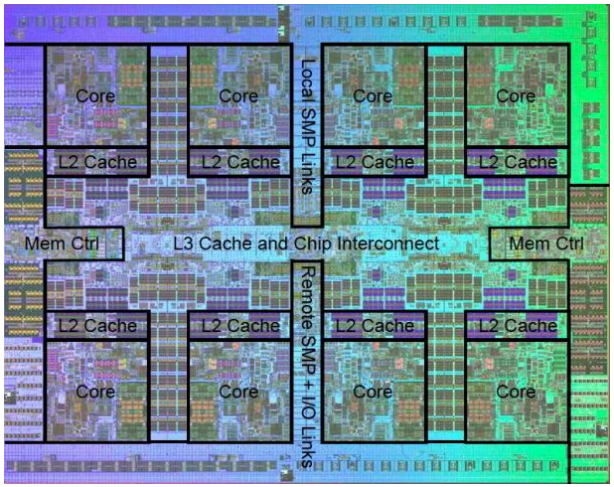
IBM Power7 (2010)

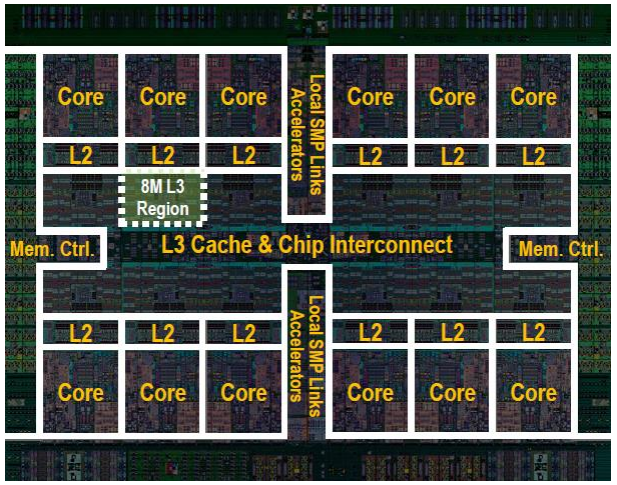
- 8 cores, SMT4
- 256 KB/core L2 16MB/core L3 (on-chip)
- 170W TDP
- 100GB/s mem BW

IBM Power8 (2014)

- 12 cores, SMT8
- 512 KB/core L2 8MB/core L3 (on-chip)
- 250W TDP
- 410GB/s mem BW

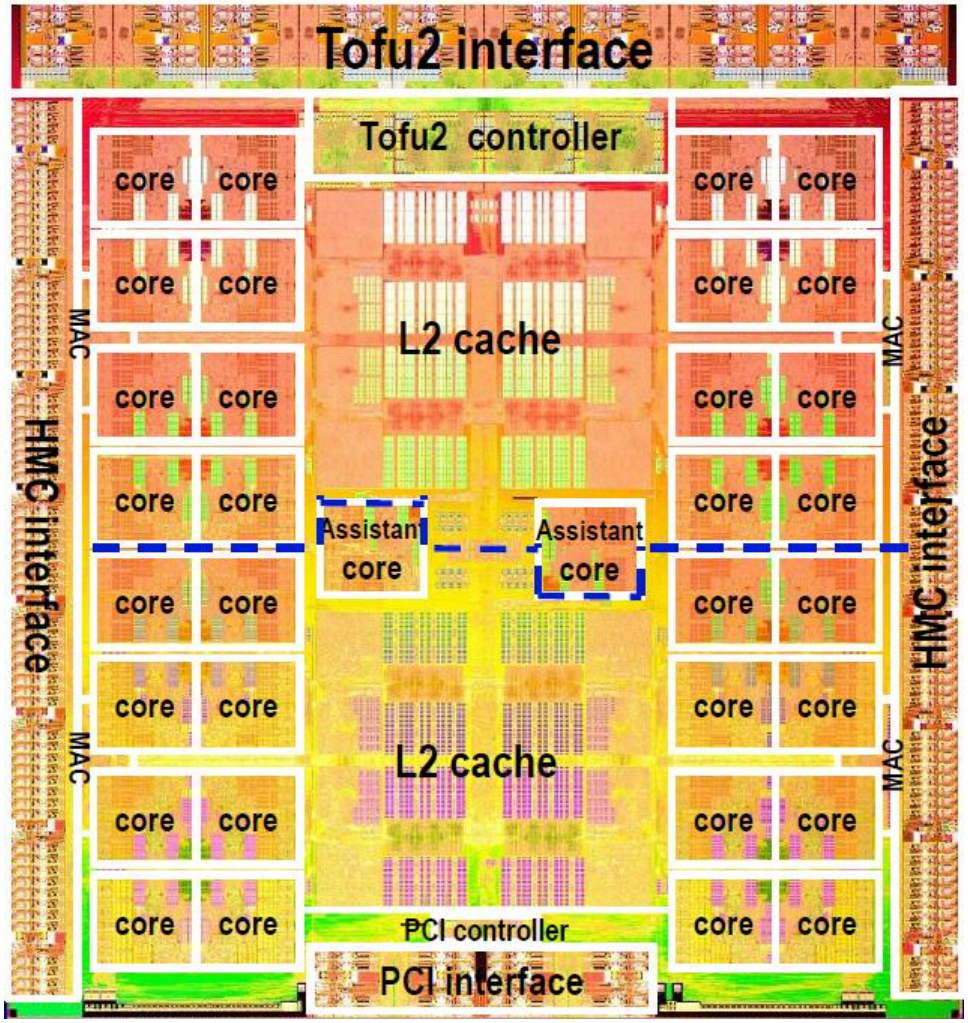




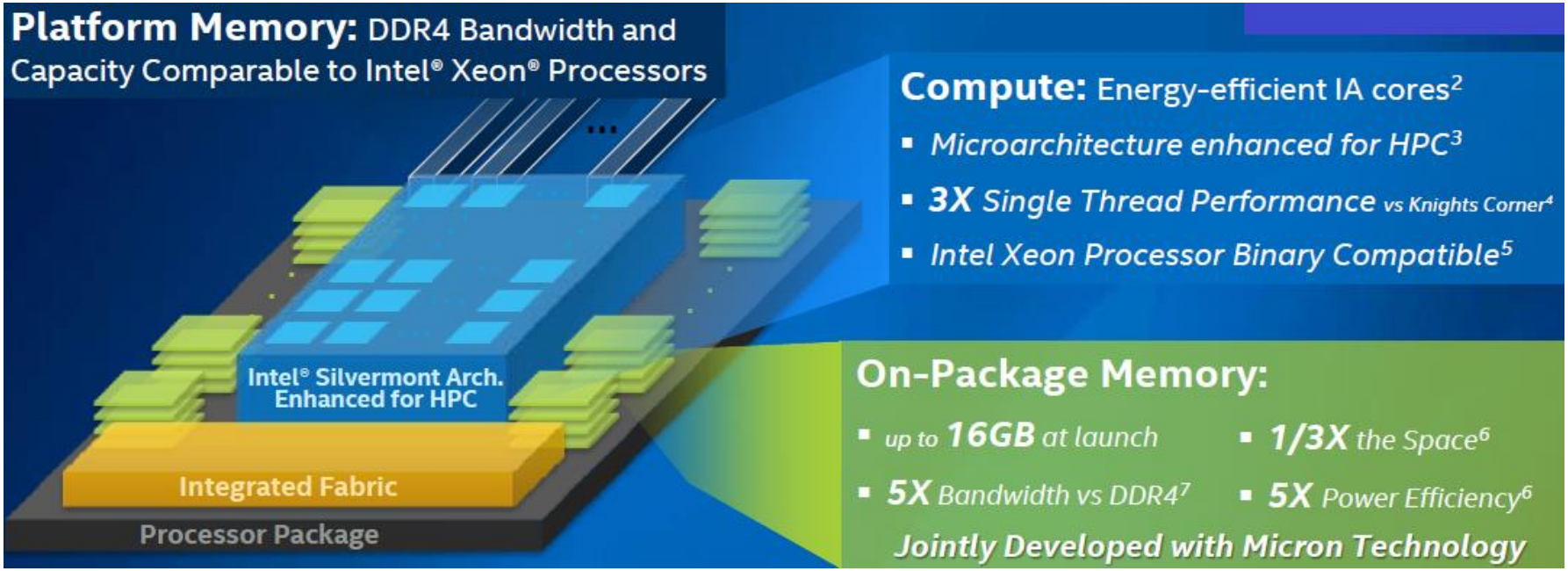


Fujitsu SPARC64 Xifx (2014)

- 32 computing cores (single threaded) + 2 assistant cores
- 24MB L2 sector cache
- 256-bit wide SIMD
- 20nm, 3.75M transistors
- 2.2GHz frequency
- 1.1TFlops peak performance
- High BW interconnects
  - HMC (240GB/s x 2 in/out)
  - Tofu2 (125GB/s x 2 in/out)



Intel Knights Landing (2016)

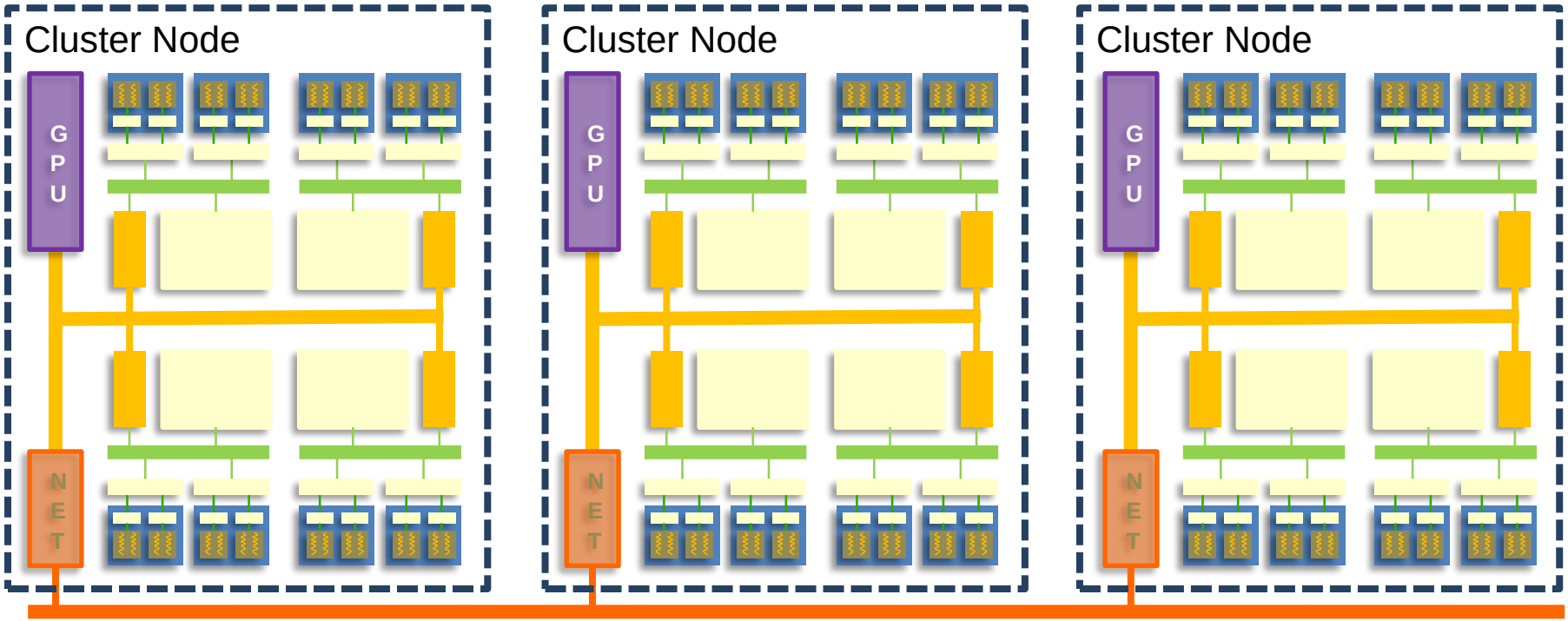


From Intel Corporation

- Intel's Knights Landing has a hybrid and reconfigurable memory hierarchy

Cluster Machines

- SM or DSM machines interconnected
  - Distributed Memory → Multiple Address Spaces
  - Communication through interconnection network
- Usually allows multiple levels of parallelism

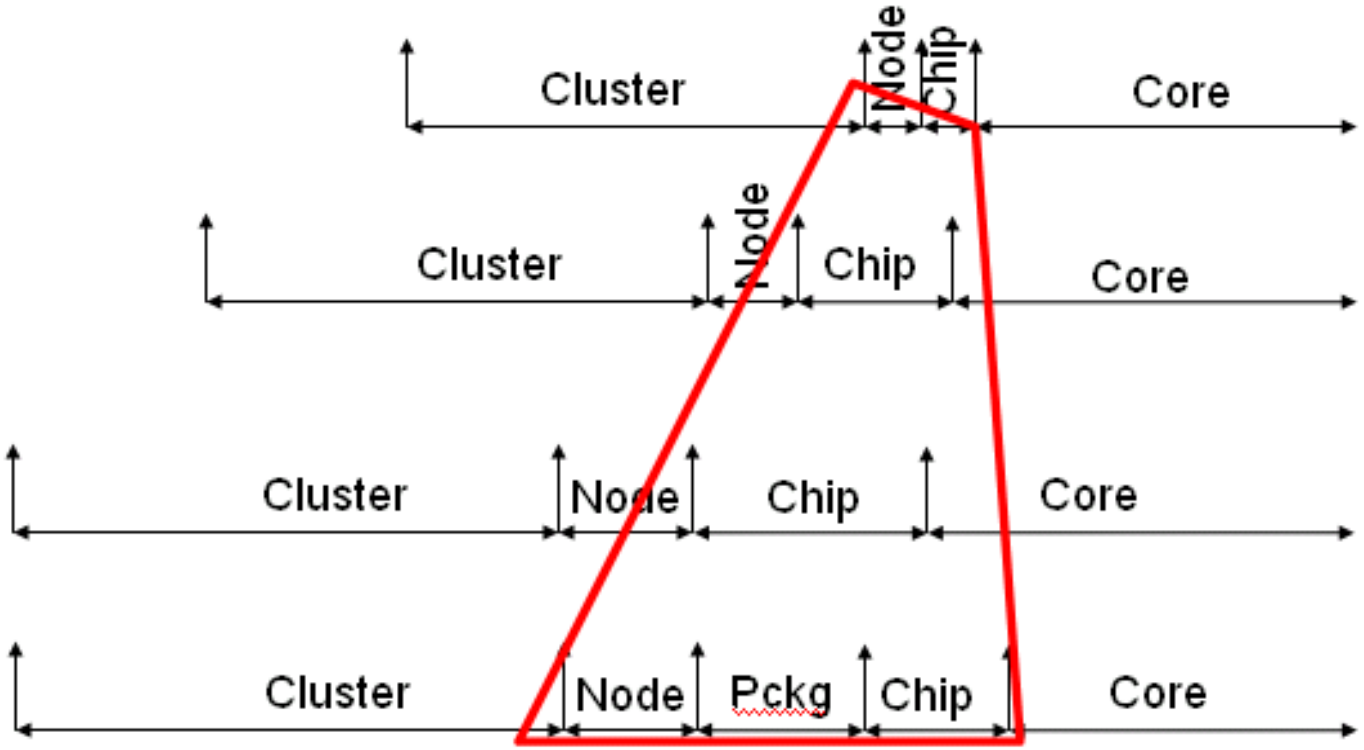


The Path to ExaFlop Computers

E P T G

1000000000000000000

Floating Point Ops / Sec

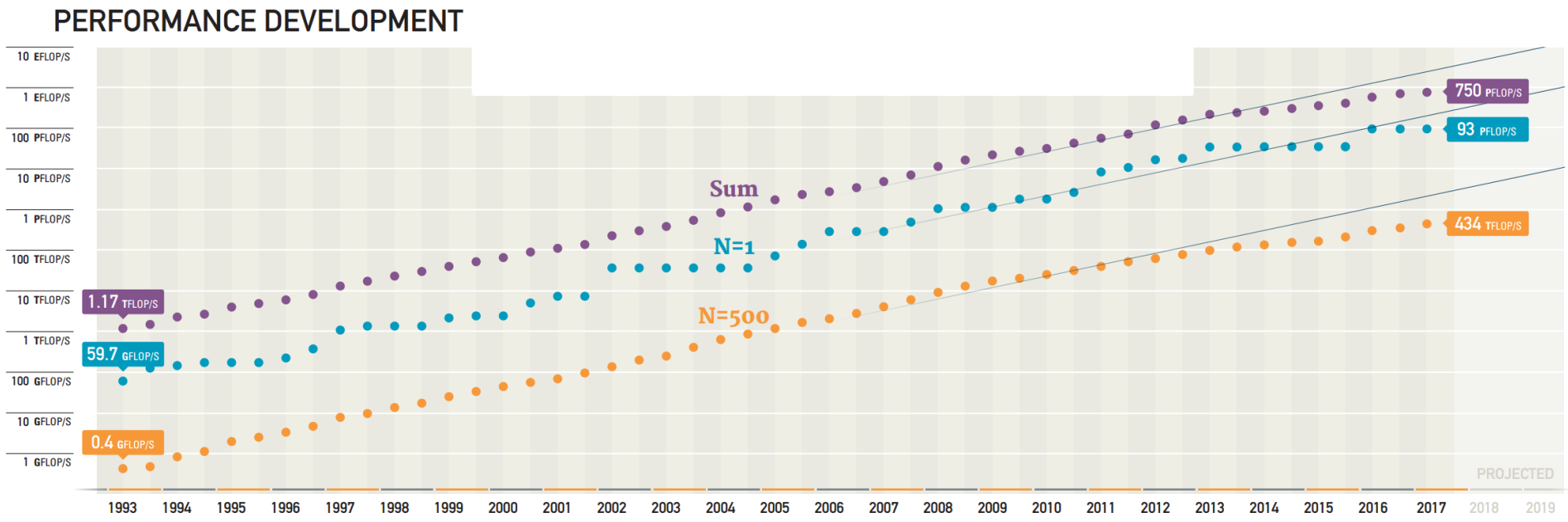


The Path to ExaFlop Computers

E P T G

1000000000000000000

Floating Point Ops / Sec



Source: Top500

Top500 Supercomputers List

Sunway TaihuLight

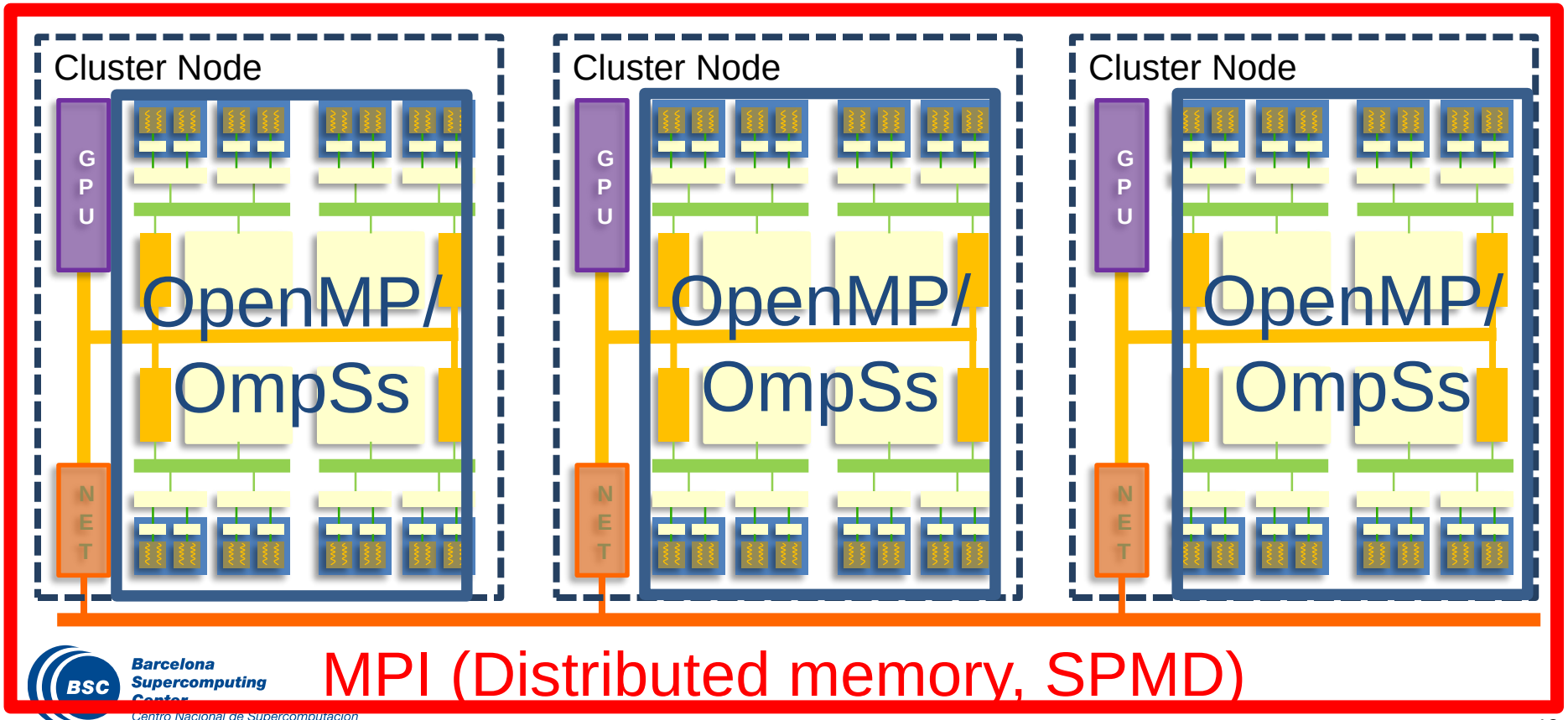
Piz Daint

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	Sunway TaihuLight - Sunway National Supercomputing Ce China	10649600	93014	125435	15371
2	Tianhe-2 (MilkyWay-2) - TH- National Super Computer Ce China	3120000	33862	54902	17808
3	Piz Daint - Cray XC50, Xeon E Swiss National Supercomput Switzerland	361760	19590	25326	2272
4	Gyoukou - ZettaScaler-2.2 H Japan Agency for Marine-Ea Japan	19860000	19135	28192	1350
5	Titan - Cray XK7, Opteron 62 DOE/SC/Oak Ridge National United States	560640	17590	27112	8209
6	Sequoia - BlueGene/Q, Powe DOE/NNSA/LLNL United States	1572864	17173	20132	7890
7	Trinity - Cray XC40, Intel Xeo DOE/NNSA/LANL/SNL United States	979968	14137	43902	3843
8	Cori - Cray XC40, Intel Xeon DOE/SC/LBNL/NERSC United States	622336	14014	27880	3939
9	Oakforest-PACS - PRIMERGY Joint Center for Advanced H Japan	556104	13554	24913	2718
10	K computer, SPARC64 VIIIfx RIKEN Advanced Institute fo Japan	705024	10510	11280	12660
16	MareNostrum - Lenovo SD5 Barcelona Supercomputing C Spain	153216	6471	10296	1632

Source: Top500

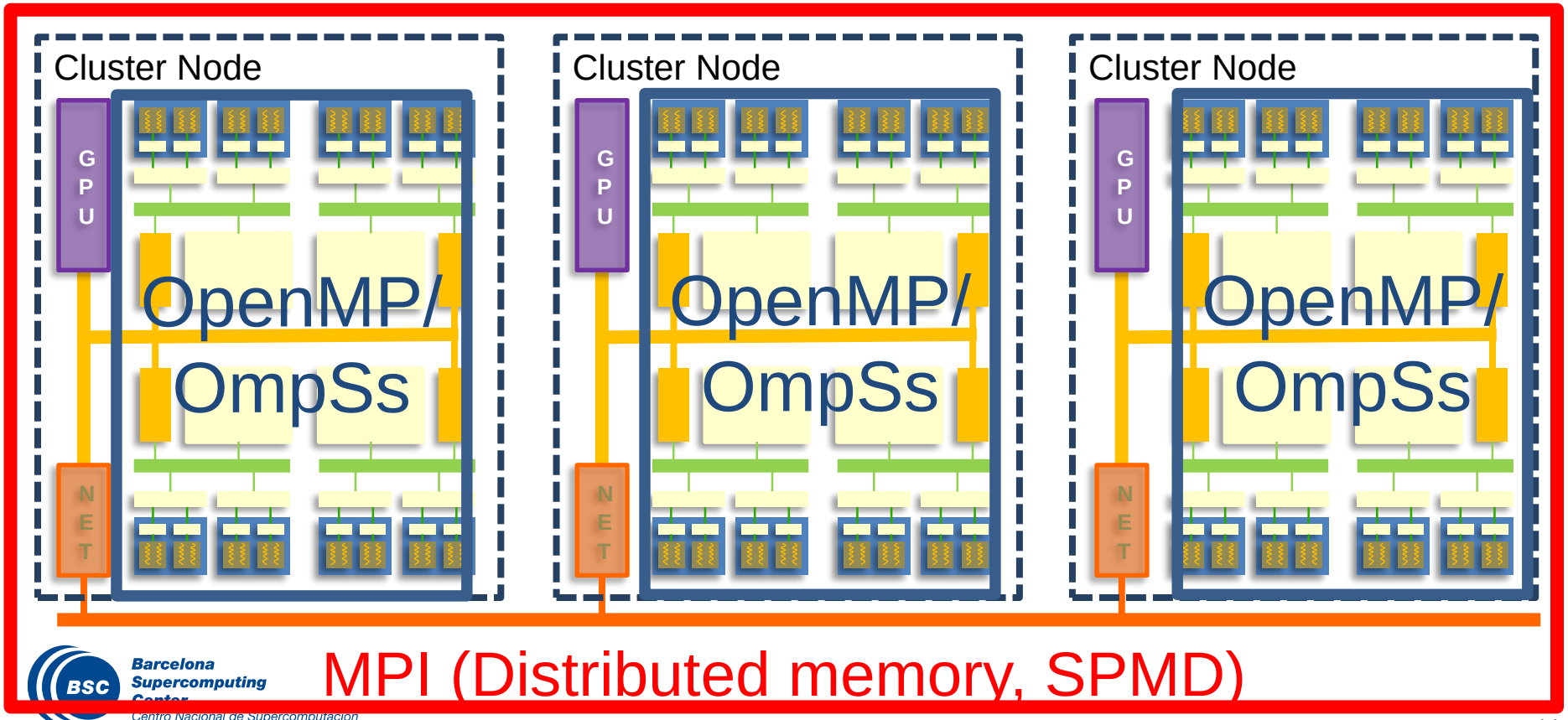
Programming HPC Machines

- Distributed Memory Level: Message Passing Interface (MPI)
- Shared Memory Level: OpenMP (OmpSs)



Programming HPC Machines

- Distributed Memory Level: Message Passing Interface (MPI)
- Shared Memory Level: OpenMP (OmpSs)



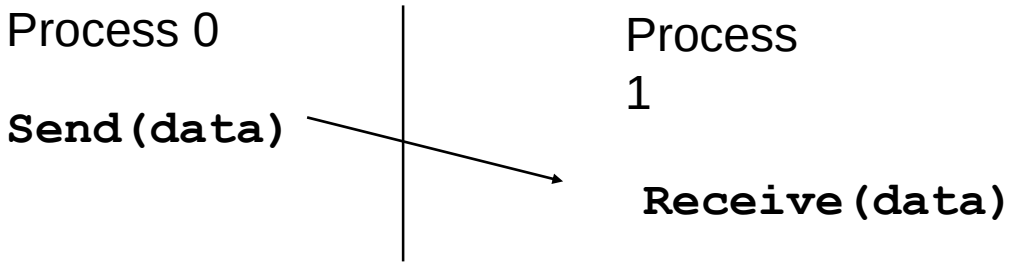
Communication Across Nodes: Message Passing

- Types of Parallel Computing Models
  - Data Parallel - the same instructions are carried out simultaneously on multiple data items (SIMD)
  - Task Parallel - different instructions on different data (MIMD)
  - SPMD (single program, multiple data) not synchronized at individual operation level
  - SPMD is equivalent to MIMD since MIMD program can be SPMD (similarly for SIMD, but not in practical sense.)
- The Message-Passing Model
  - Message passing (and MPI) is for MIMD/SPMD parallelism.
  - A *process* is (traditionally) a program counter and address space.
  - Processes may have multiple *threads* sharing a single address space. Interprocess communication consists of
    - Synchronization
    - Movement of data from one process's address space to another's.

Based on "An Introduction to MPI" William Gropp, Ewing Lusk

MPI: Cooperative Communication

- The message-passing approach makes the exchange of data *cooperative*.
- Data is explicitly *sent* by one process and *received* by another.
- An advantage is that any change in the receiving process's memory is made with the receiver's explicit participation.
- Communication and synchronization are combined.



Based on "An Introduction to MPI" William Gropp, Ewing Lusk

MPI: Collective Operations

- Collective operations are called by all processes in a communicator.
- **MPI_BCAST** distributes data from one process (the root) to all others in a communicator.
- **MPI_REDUCE** combines data from all processes in communicator and returns it to one process.
- In many numerical algorithms, **SEND/RECEIVE** can be replaced by **BCAST/REDUCE**, improving both simplicity and efficiency.

Based on “An Introduction to MPI” William Gropp, Ewing Lusk

Blocking function to Send a Message

- int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
  - Input Parameters:
    - Buf: initial address of send buffer (choice)
    - Count: number of elements in send buffer (nonnegative integer)
    - Datatype: datatype of each send buffer element (handle)
    - Dest: rank of destination (integer)
    - Tag: message tag (integer)
    - Comm: communicator (handle)

Blocking function to Receive a Message

- int MPI_Recv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)
  - Output Parameters
    - Buf: initial address of receive buffer (choice)
    - Status: status object (Status)
  - Input Parameters
    - Count maximum number of elements in receive buffer (integer)
    - Datatype: datatype of each receive buffer element (handle)
    - Source: rank of source (integer)
    - Tag: message tag (integer)
    - comm: communicator (handle)

Message matching

```
MPI_Send(send_buf_p, send_buf_sz, send_type, dest, send_tag,
         send_comm);
```

r

MPI_Send

src = q

MPI_Recv

dest = r

```
MPI_Recv(recv_buf_p, recv_buf_sz, recv_type, src, recv_tag,
         recv_comm, &status);
```

q

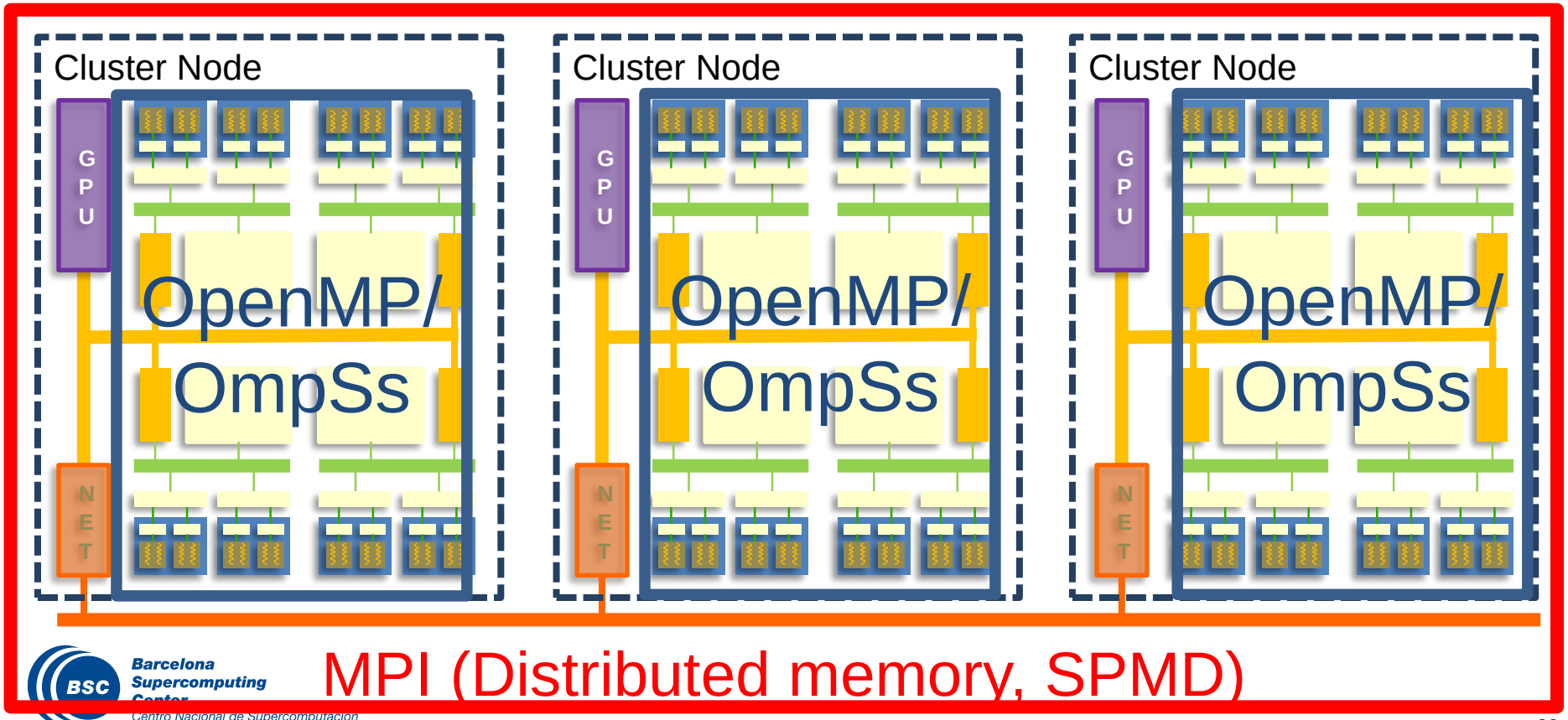
Our first MPI program

```
1  #include <stdio.h>
2  #include <string.h>  /* For strlen             */
3  #include <mpi.h>     /* For MPI functions, etc */
4
5  const int MAX_STRING = 100;
6
7  int main(void) {
8     char        greeting[MAX_STRING];
9     int         comm_sz;  /* Number of processes */
10    int         my_rank;  /* My process rank     */
11
12    MPI_Init(NULL, NULL);
13    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
14    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
15
16    if (my_rank != 0) {
17       sprintf(greeting, "Greetings from process %d of %d!",
18             my_rank, comm_sz);
19       MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0,
20             MPI_COMM_WORLD);
21    } else {
22       printf("Greetings from process %d of %d!\n", my_rank, comm_sz);
23       for (int q = 1; q < comm_sz; q++) {
24          MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q,
25             0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
26          printf("%s\n", greeting);
27       }
28    }
29
30    MPI_Finalize();
31    return 0;
32 }  /* main */
```

From Peter S. Pacheco

Programming HPC Machines

- Distributed Memory Level: Message Passing Interface (MPI)
- Shared Memory Level: OpenMP (OmpSs)



OmpSs: A Sequential Program …

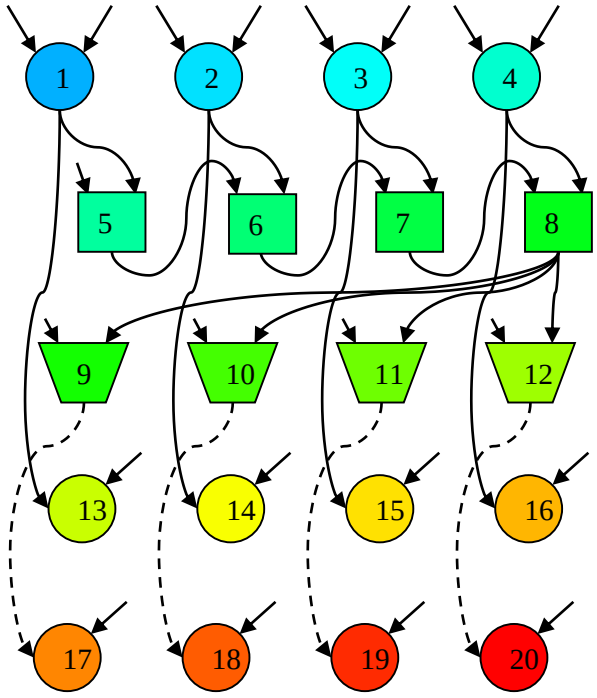
```
void vadd3 (float A[BS], float B[BS],
            float C[BS]);

void scale_add (float sum, float A[BS],
                float B[BS]);

void accum (float A[BS], float *sum);
```

```
for (i=0; i<N; i+=BS)                // C=A+B
   vadd3 ( &A[i], &B[i], &C[i]);
...
for (i=0; i<N; i+=BS)               //sum(C[i])
   accum (&C[i], &sum);
...
for (i=0; i<N; i+=BS)               // B=sum*A
   scale_add (sum, &E[i], &B[i]);
...
for (i=0; i<N; i+=BS)               // A=C+D
   vadd3 (&C[i], &D[i], &A[i]);
...
for (i=0; i<N; i+=BS)               // E=G+F
   vadd3 (&G[i], &F[i], &E[i]);
```

OmpSs: … Taskified …

```
#pragma css task input(A, B) output(C)
void vadd3 (float A[BS], float B[BS],
            float C[BS]);
#pragma css task input(sum, A) inout(B)
void scale_add (float sum, float A[BS],
                float B[BS]);
#pragma css task input(A) inout(sum)
void accum (float A[BS], float *sum);
```

```
for (i=0; i<N; i+=BS)              // C=A+B
   vadd3 ( &A[i], &B[i], &C[i]);
...
for (i=0; i<N; i+=BS)             //sum(C[i])
   accum (&C[i], &sum);
...
for (i=0; i<N; i+=BS)             // B=sum*A
   scale_add (sum, &E[i], &B[i]);
...
for (i=0; i<N; i+=BS)             // A=C+D
   vadd3 (&C[i], &D[i], &A[i]);
...
for (i=0; i<N; i+=BS)             // E=G+F
   vadd3 (&G[i], &F[i], &E[i]);
```



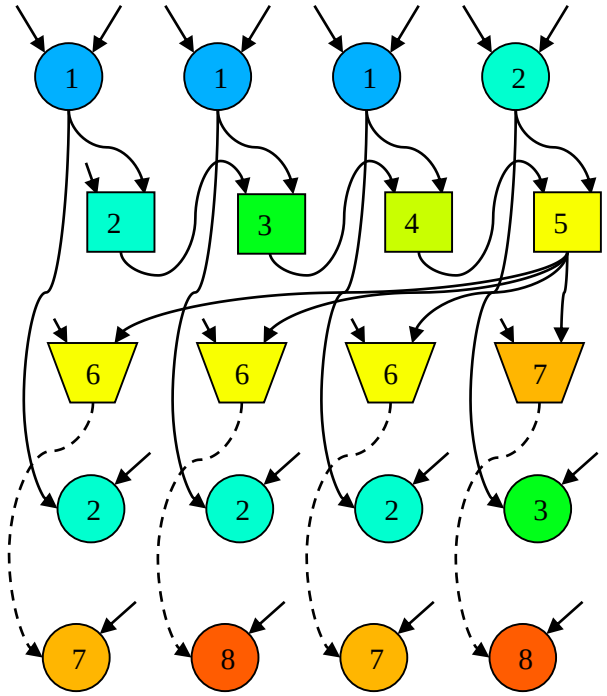
Color/number: order of task instantiation
Some antidependences covered by flow dependences not drawn

OmpSs: … and Executed in a Data-Flow Model

```
#pragma css task input(A, B) output(C)
void vadd3 (float A[BS], float B[BS],
            float C[BS]);
#pragma css task input(sum, A) inout(B)
void scale_add (float sum, float A[BS],
                float B[BS]);
#pragma css task input(A) inout(sum)
void accum (float A[BS], float *sum);
```

```
for (i=0; i<N; i+=BS)                 // C=A+B
   vadd3 ( &A[i], &B[i], &C[i]);
...
for (i=0; i<N; i+=BS)                 //sum(C[i])
   accum (&C[i], &sum);
...
for (i=0; i<N; i+=BS)                 // B=sum*A
   scale_add (sum, &E[i], &B[i]);
...
for (i=0; i<N; i+=BS)                 // A=C+D
   vadd3 (&C[i], &D[i], &A[i]);
...
for (i=0; i<N; i+=BS)                 // E=G+F
   vadd3 (&G[i], &F[i], &E[i]);
```

Decouple
how we write
form
how it is executed

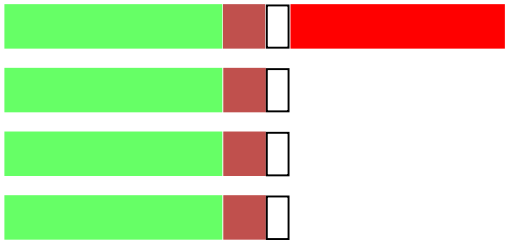


Color/number: a possible order of task execution

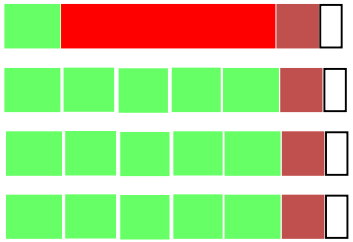
OmpSs/OpenMP4.0: Data-flow and asynchronous

Four loops/routines
Sequential program order

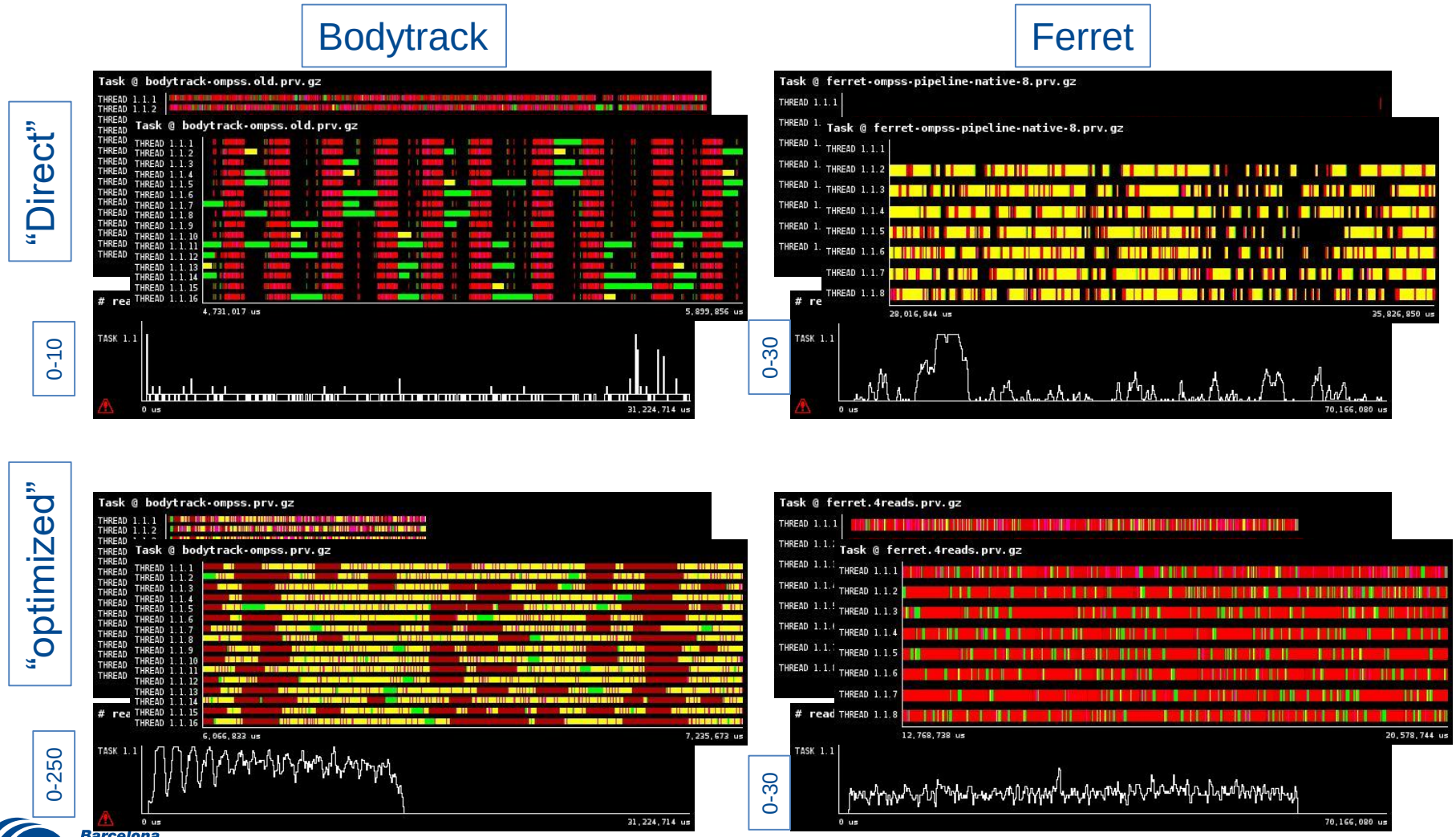
OpenMP 2.5
not parallelizing one loop



OmpSs/OpenMP4.0
not parallelizing one loop



Initial port from Pthreads to OmpSs and optimization

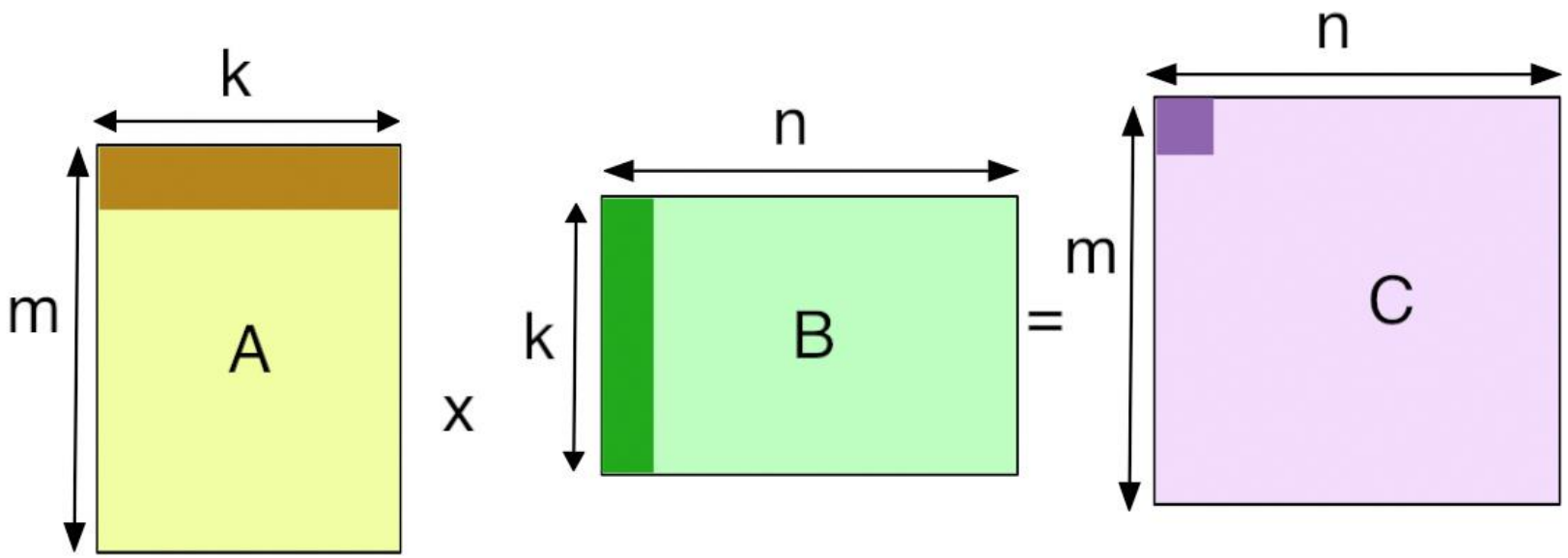


"Exploring the Impact of Task Parallelism in the PARSEC benchmark suite". TACO'2016

PART2: ACCELERATING DEEP LEARNING WORKLOADS IN THE HPC CONTEXT

The Fundamental DL Numerical Kernel: GEMM

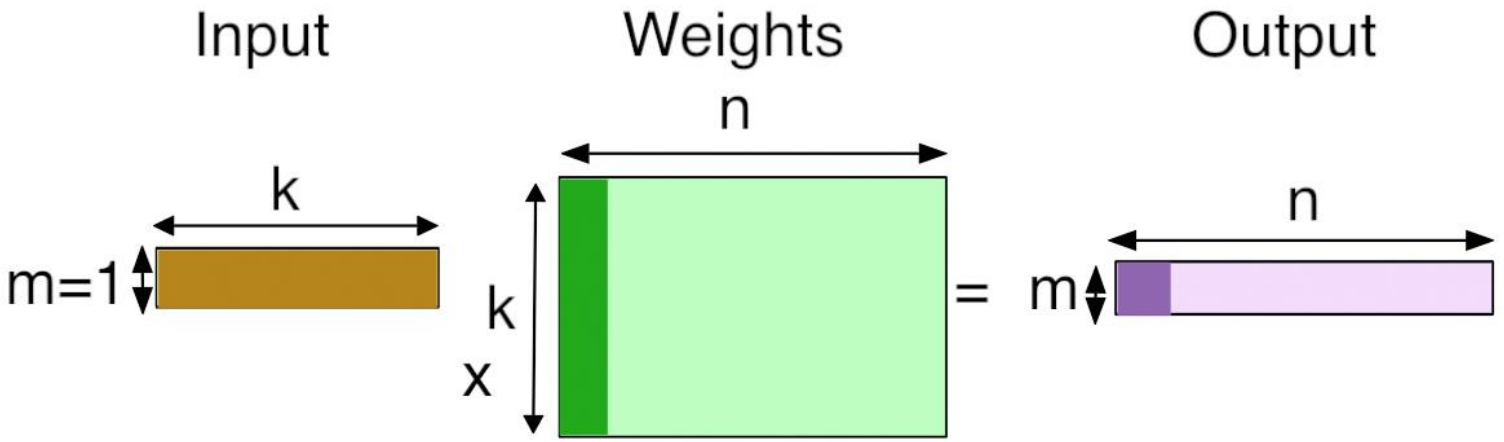
- GEMM stands for GEneral Matrix to Matrix Multiplication



Picture Taken from Pete Warden's blog

The Fundamental DL Numerical Kernel: GEMM

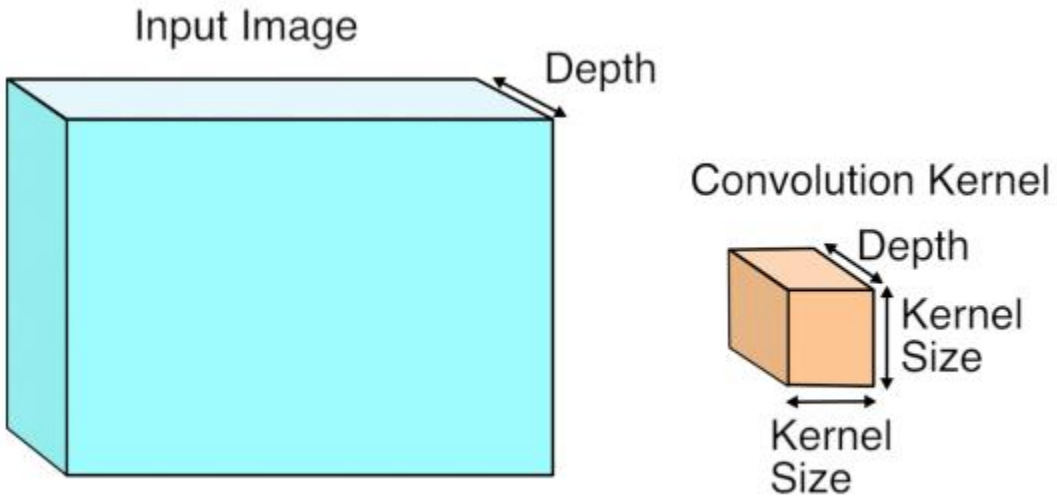
- GEMM stands for GEneral Matrix to Matrix Multiplication
- Fully-Connected layers (FC):
  - Each output value
    - looks at each value in the input layer
    - multiplies them all by the corresponding weight
    - sums the results to get its value



Picture Taken from Pete Warden's blog

The Fundamental DL Numerical Kernel: GEMM

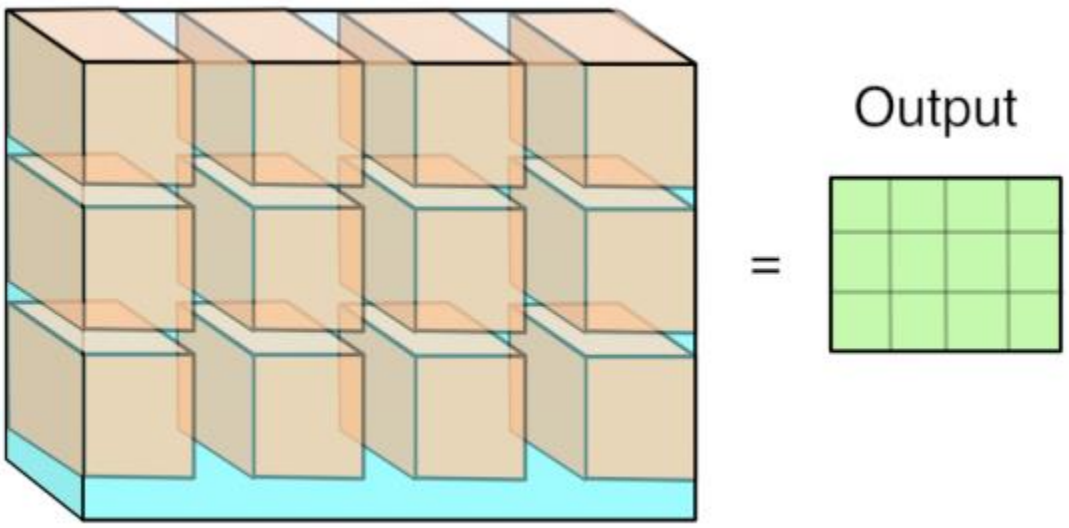
- GEMM stands for GEneral Matrix to Matrix Multiplication
- Convolutional Layers (CL)
  - treat inputs as a two dimensional image, with a number of channels for each pixel, much like a classical image with width, height, and depth.
  - The convolution operation produces its output by taking a number of ‘kernels’ of weights. and applying them across the image



Picture Taken from Pete Warden’s blog

The Fundamental DL Numerical Kernel: GEMM

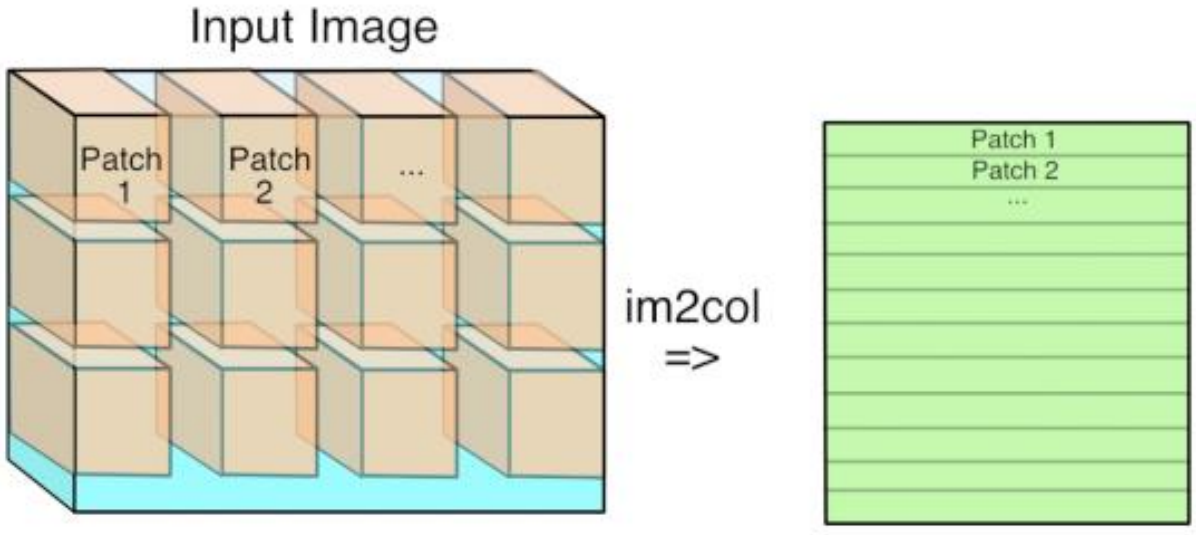
- GEMM stands for GEneral Matrix to Matrix Multiplication
- Convolutional Layers (CL)
  - Each kernel is another three-dimensional array of numbers, with the depth the same as the input image.
  - At each point the kernel is applied, all input values and weights are multiplied and then summed to produce a single output value at that point.



Picture Taken from Pete Warden's blog

The Fundamental DL Numerical Kernel: GEMM

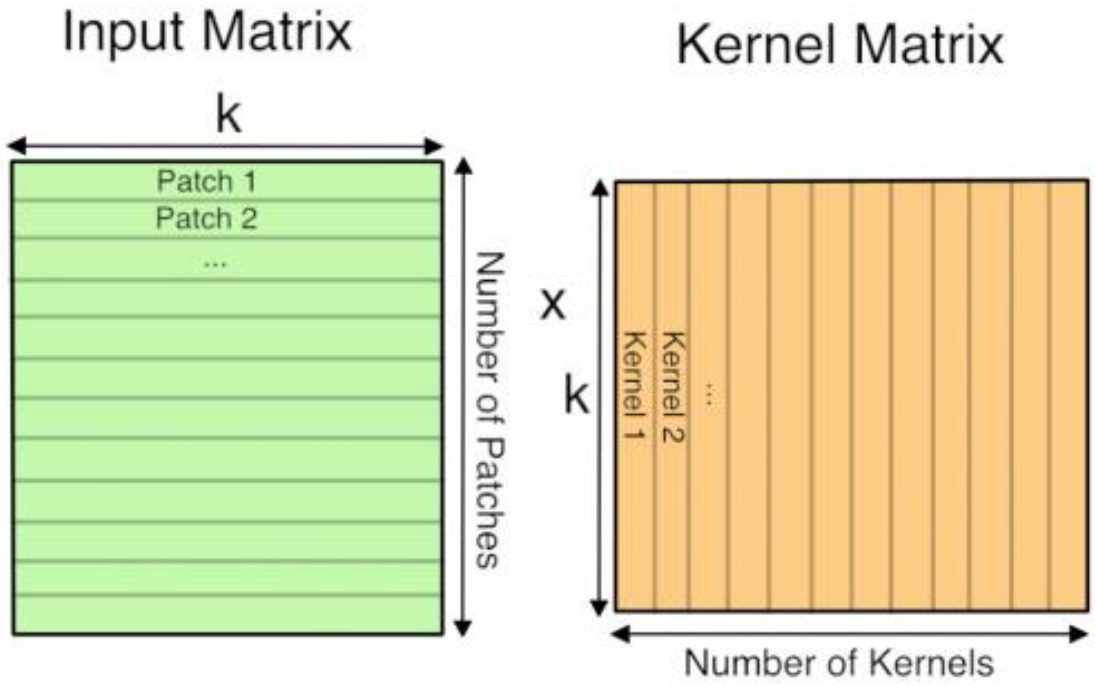
- GEMM stands for GEneral Matrix to Matrix Multiplication
- Convolutional Layers (CL)
  - Each kernel is another three-dimensional array of numbers, with the depth the same as the input image.
  - At each point the kernel is applied, all input values and weights are multiplied and then summed to produce a single output value at that point.



Picture Taken from Pete Warden's blog

The Fundamental DL Numerical Kernel: GEMM

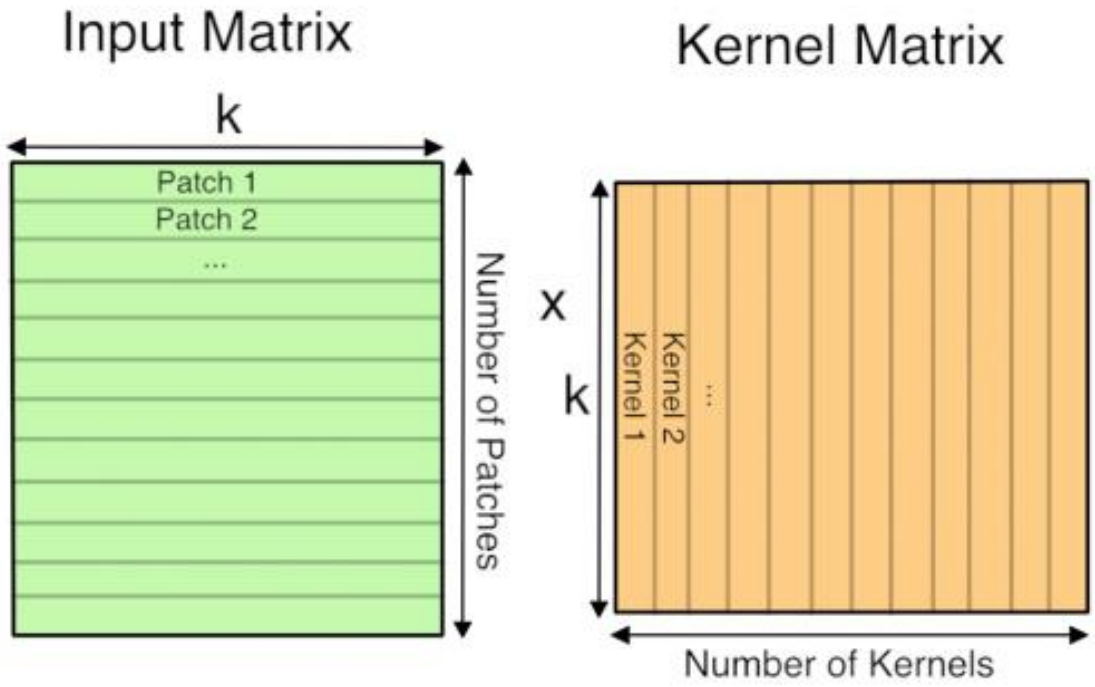
- GEMM stands for GEneral Matrix to Matrix Multiplication
- Convolutional Layers (CL)
  - Each kernel is another three-dimensional array of numbers, with the depth the same as the input image.
  - At each point the kernel is applied, all input values and weights are multiplied and then summed to produce a single output value at that point.



Picture Taken from Pete Warden's blog

The Fundamental DL Numerical Kernel: GEMM

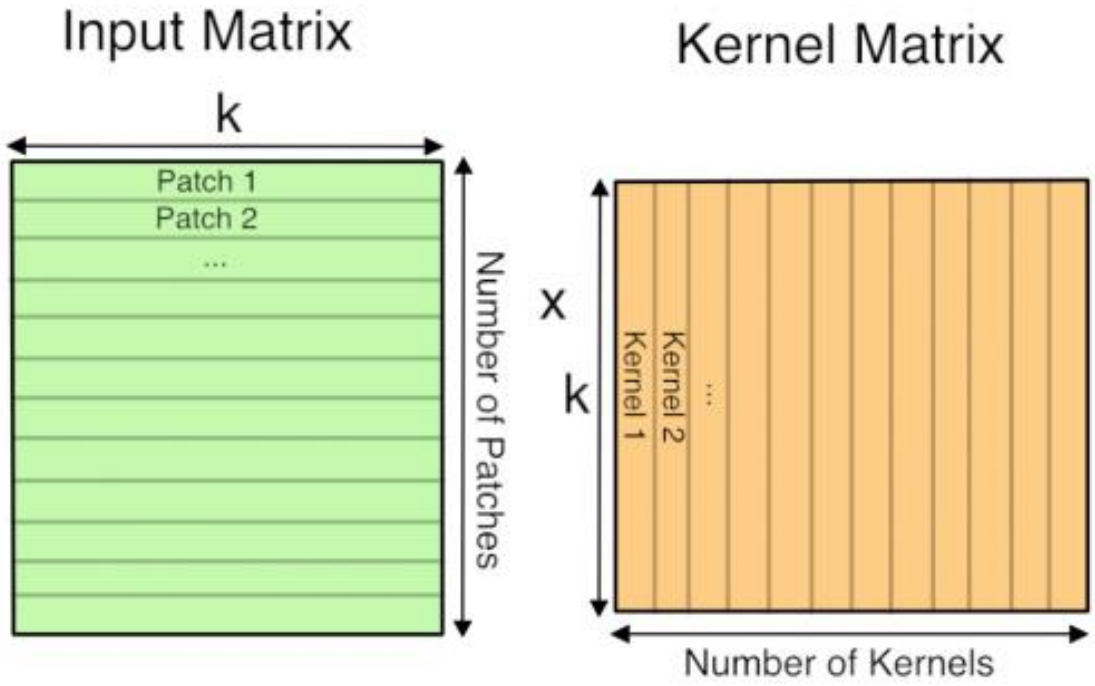
- GEMM stands for GEneral Matrix to Matrix Multiplication
- Why does GEMM work for Convolutional Layers (CL)?
  - Cons:
    - It is not the only possible mathematical formulation.
    - Pixels that are included in overlapping kernel sites will be duplicated in the matrix, which seems inefficient.



Picture Taken from
Pete Warden's blog

The Fundamental DL Numerical Kernel: GEMM

- GEMM stands for GEneral Matrix to Matrix Multiplication
- Why does GEMM work for Convolutional Layers (CL)?
  - Pros:
    - The Scientific computing area has spent decades optimizing code to perform large matrix to matrix multiplications
    - The benefits from the very regular patterns of memory access outweigh the wasteful storage cost



Picture Taken from Pete Warden's blog

Approaches for DL (i.e. GEMM) at HPC

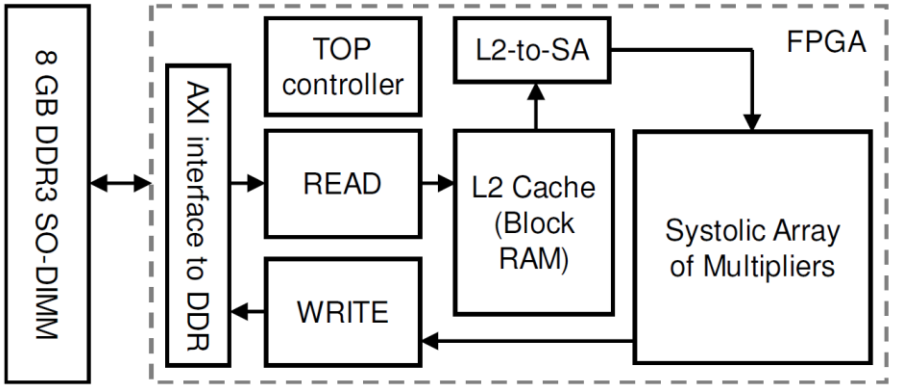
- DL on General-purpose hardware:
  - Classical HPC approaches based on MPI+OpenMP parallel schemes running on clusters
- GPU's:
  - Next weeks' practical exercise
- GEMM-Specific Accelerators:
  - 2D systolic arrays
    - Google's TPU, Nervana's Lake Crest (Intel), etc.
- Hardware-implemented Neural Networks:
  - IBM True North

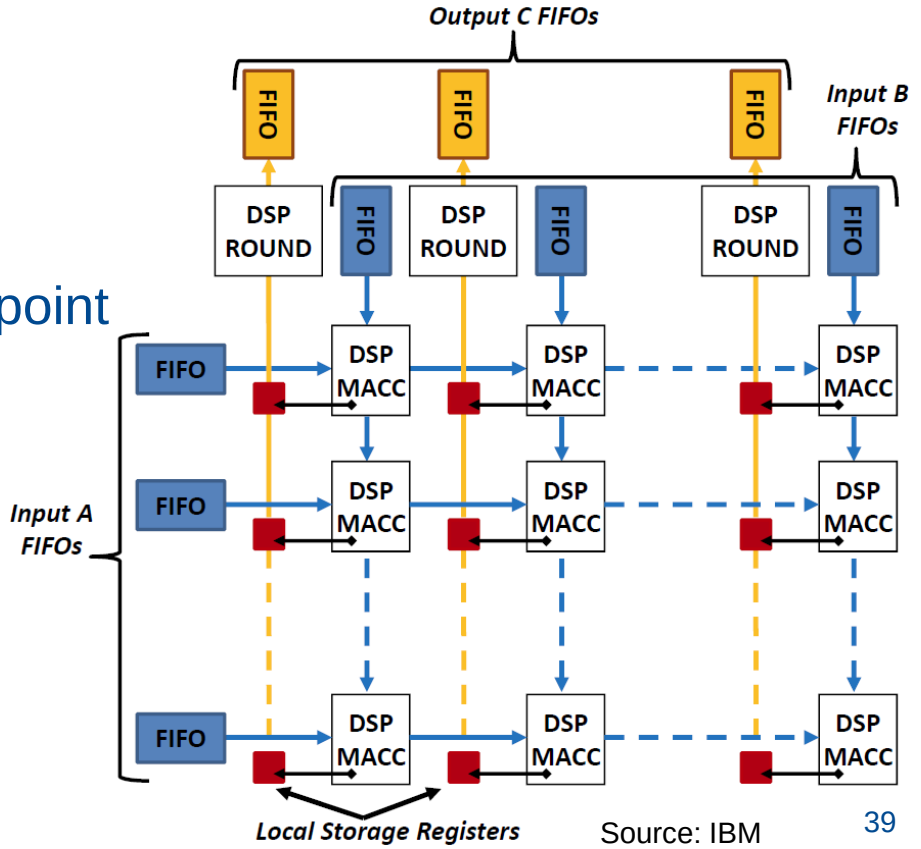
Approaches for DL (i.e. GEMM) at HPC

- DL on General-purpose hardware:
  - Classical HPC approaches based on MPI+OpenMP parallel schemes running on clusters
- GPU's:
  - Next weeks' practical exercise
- GEMM-Specific Accelerators:
  - 2D systolic arrays
    - Google's TPU, Nervana's Lake Crest (Intel), etc.
- Hardware-implemented Neural Networks:
  - IBM True North

IBM low-precision fixed-point with stoch. rounding (2015)

- 8 GB DDR3 @ 6.4 GB/s
- 2MB on-chip Block RAM
- Frequency 166MHz
- TDP 7W
- 2D Systolic Array:
  - 28x28 Multiply-ACCumulate (MACC) DSP units
  - 28 DSP stochastic rounding units
  - MACC operate with 48-bits fixed point
  - DSP round to 18-bits
- Peak Throughput: 260 GOPS
- Power Efficiency: 37GOPS/W

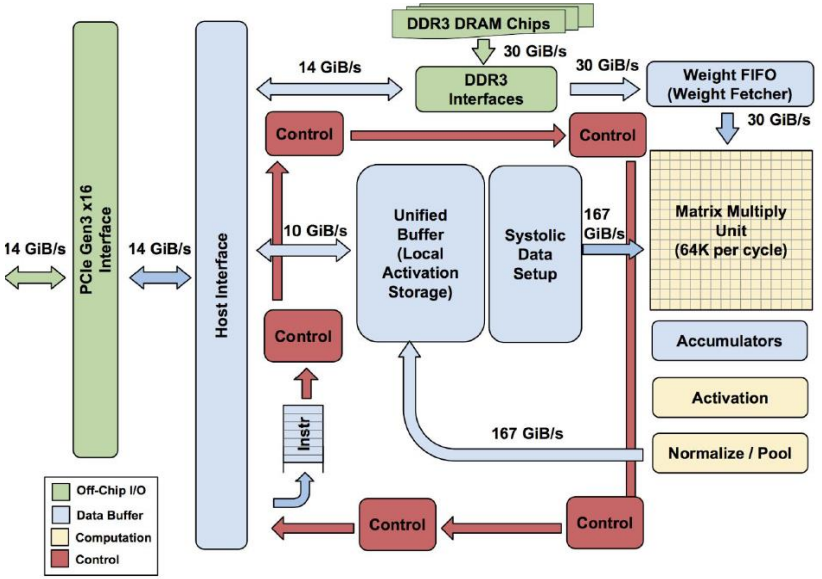


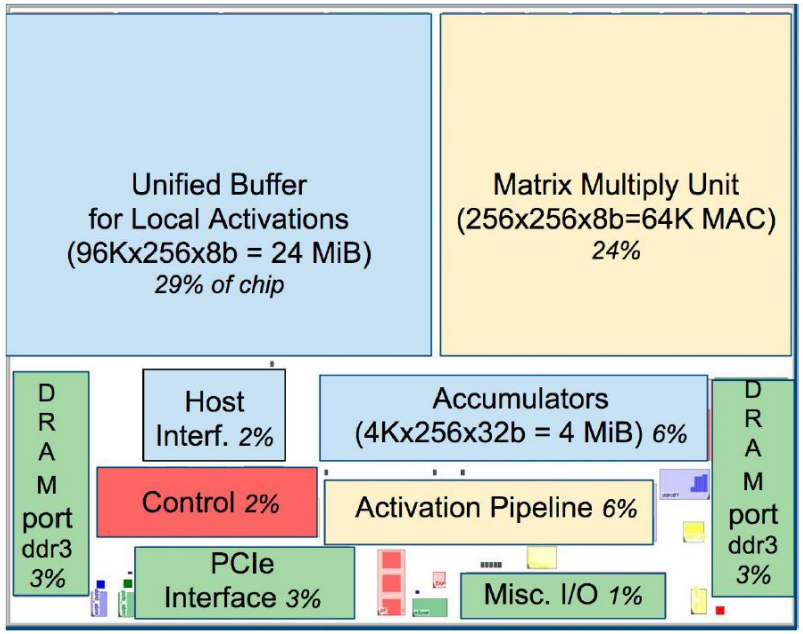


Source: IBM

Google Tensor Processing Unit (2015, published 2017)

- 34 GB/s off-chip memory
- 28MB on-chip memory
- Frequency 700MHz
- TDP 75W
- Matrix Multiply Unit
  - 256x256 MAC Units
  - 8-bit multiply and adds
  - 32-bit accumulators
- Peak Throughput: 92 TOPS/s
- Power Efficiency: 132 GOPS/W
- Seriously memory bandwidth limited

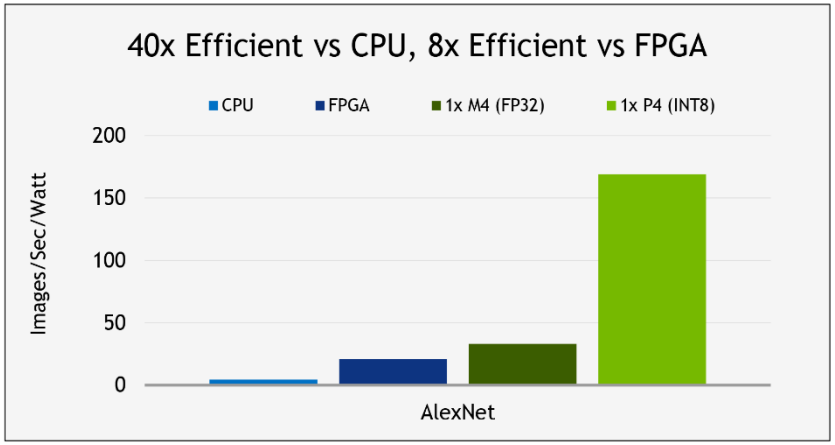




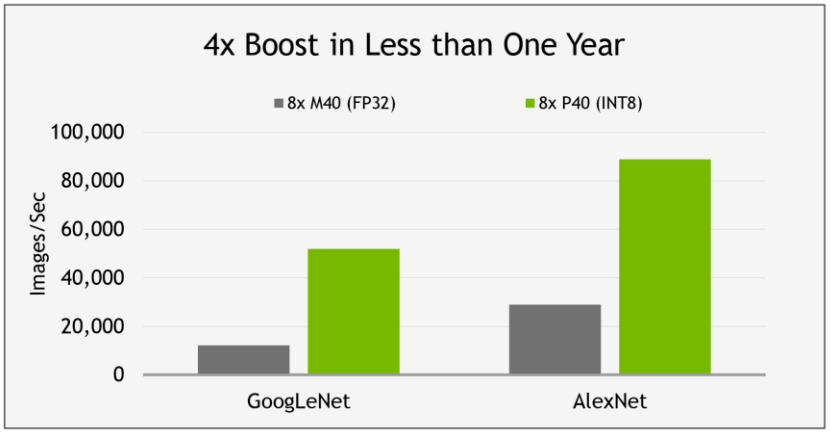
Source: google

NVIDIA Tesla P4 and P40 GPU's (2016)

- Tesla P4
  - # CUDA cores: 2560 @ 1063MHz
  - Peak single precision: 5.5TFLOPS
  - Peak INT8: 22 TOPS
  - Low precision: 8-bit dot-product with 32-bit accumulate
  - VRAM: 8 GB GDDR5 @ 192 GB/s
  - TDP: ~75W



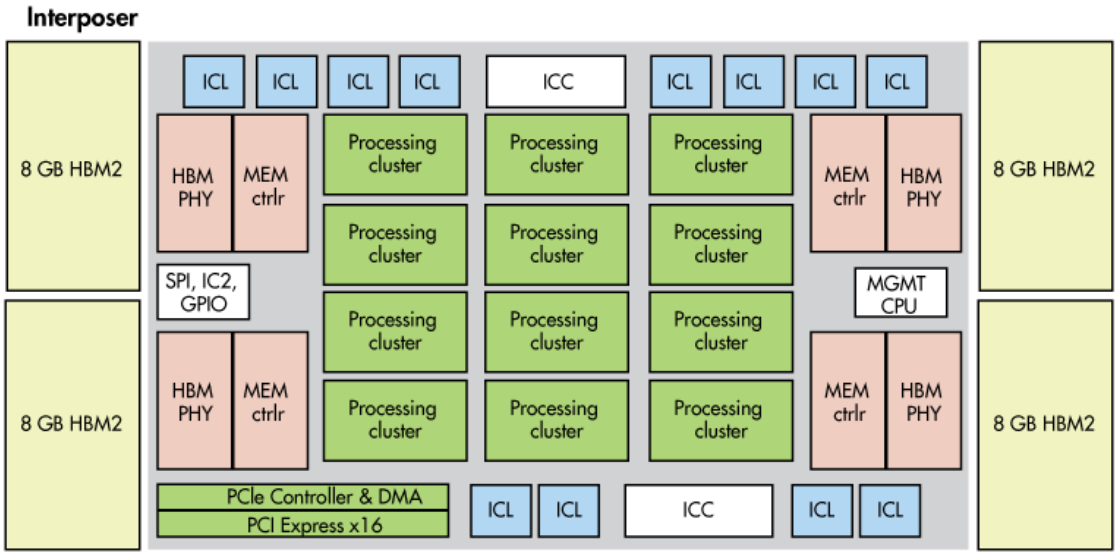
- Tesla P40
  - # CUDA cores: 2560 @ 1531MHz
  - Peak single precision: 12.0TFLOPS
  - Peak INT8: 47 TOPS
  - Low precision: 8-bit dot-product with 32-bit accumulate
  - VRAM: 24 GB GDDR5 @ 346GB/s
  - TDP: ~250W



Source: NVIDIA

Nervana's Lake Crest Deep Learning Architecture (2017)

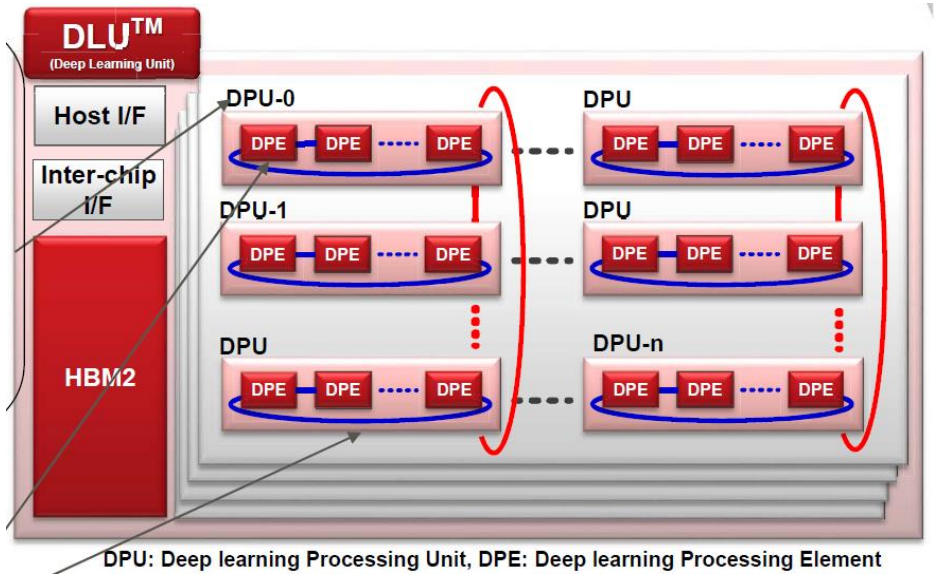
- The Lake Crest chip will operate as a Xeon Co-processor.
- Tensor-based (i. e. dense linear algebra computations)
- 4 8GB HBM2 at the same chip interposer@1TB/s
  - Each HBM has its own memory controller
- 12 Inter-Chip Links (ICL) 20x faster than PCI
- 12 Computing Nodes featuring several cores
- Intel's new "Flexpoint" architecture within the Nodes
  - Flexpoint enables 10x ILP increase and low power consumption

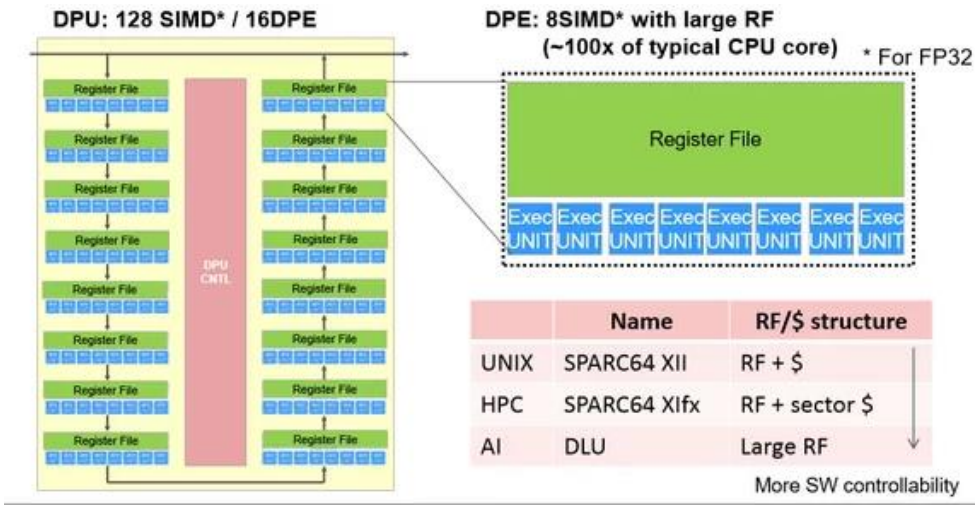


Source: elektroniknet

Fujitsu Deep Learning Unit (2017)

- DLU composed of many Deep Processing Units (DPU) connected via a NoC
- DPU consists of 16 Deep Processing Elements (DPE) connected with another NoC
- DPE includes wide SIMD execution units and large Register File (RF)
  - RF is exposed to the SW
- Deep Learning Integer provides 8- and 16- bits accuracies for DL operations

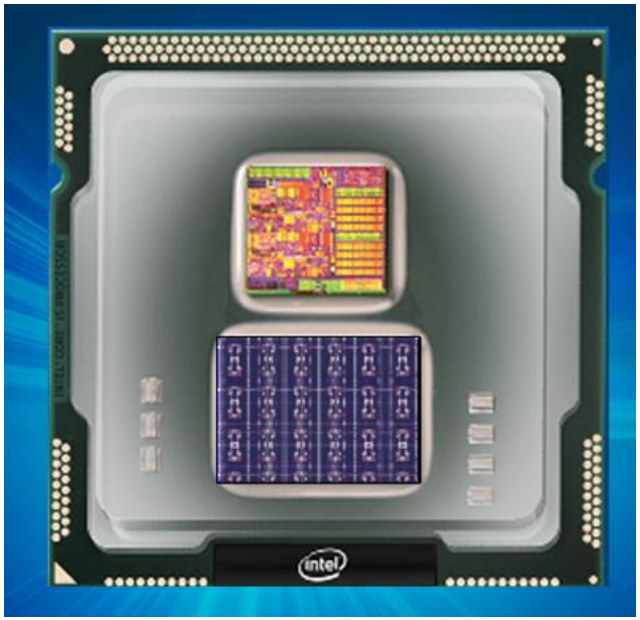




Source: Fujitsu

Intel Loihi (2017)

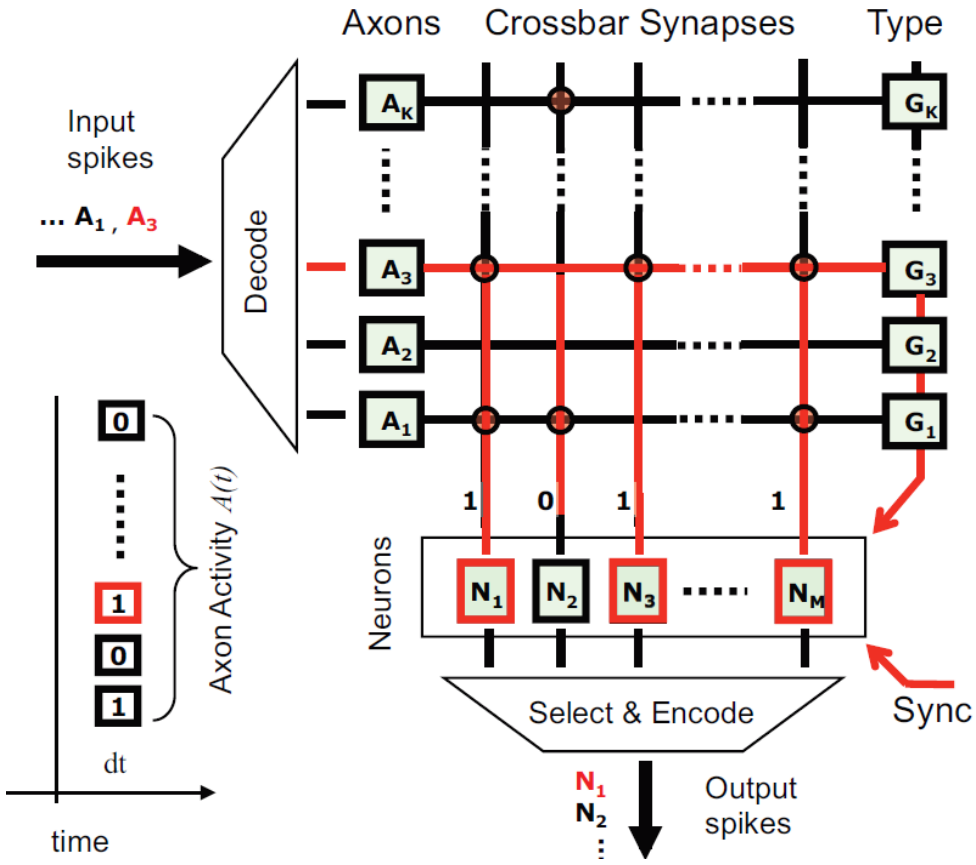
- Lohi is build with 14 nm technology and includes 130,000 neurons and 130 million synapses
- Loihi is composed of a many core mesh that supports:
  - sparse, hierarchical and recurrent neural network topologies
  - each neuron capable of communicating with thousands of other neurons
- Neuromorphic cores include a learning engine
  - It can be programmed to adapt network parameters during operation
  - It supports supervised, unsupervised and other learning paradigms.



Source: Intel

IBM TrueNorth (2014)

- 4096 cores, each one with
  - 256 neurons
  - 256 synapsis per neuron that convey signals between them
- 2^{68} synapsis in total
- 5,4 Billion Transistors
- 70 mW
- Neuron weights learned off-line and transformed into hardware format



Source: IBM

Research Opportunities

- Are you interested in working in topics involving AI, HPC and computer architecture?
  - Contact marc.casas@bsc.es
- We have ongoing research projects with top-level IT multinational companies, as well as collaborations with US universities.
- Possibilities for
  - Master Projects
  - PhD
  - Others…


BSC
Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación