www.bsc.es


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

# High Performance Computing Aspects of Deep Learning

Marc Casas Guix (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 10000 times faster than a PC

Cray-I Computer (1975)


Piz Daint (2016)

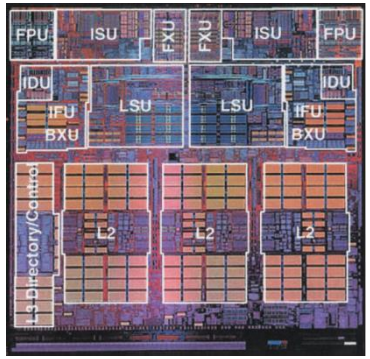
# Architecture of Supercomputers

- Supercomputers have been designed in many different ways across history:
  - In the 70's and the 80's, they were vector processors 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 20$^{th}$ 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

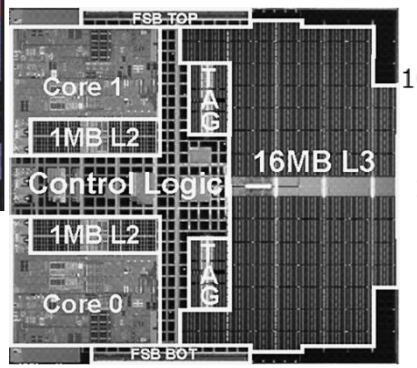
# Compute Nodes based on Multicore Architectures

- Moore's Law + Memory Wall + Power Wall

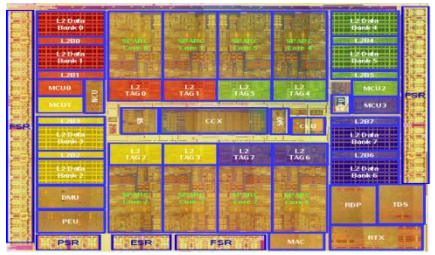
## Chip MultiProcessors (CMPs)



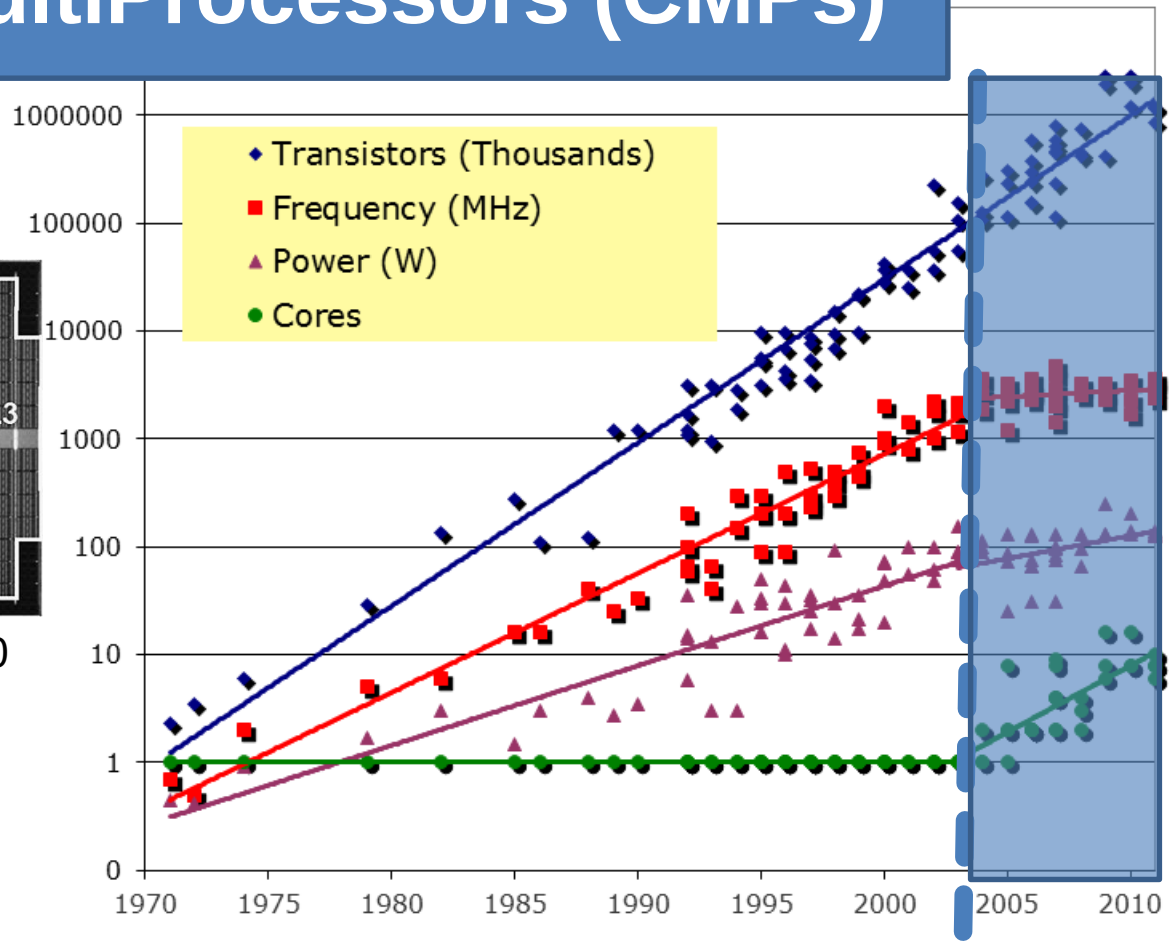

POWER4 (2001)




Intel Xeon 7100 (2006)



UltraSPARC T2 (2007)



# How are the Multicore architectures designed?

## IBM Power4 (2001)

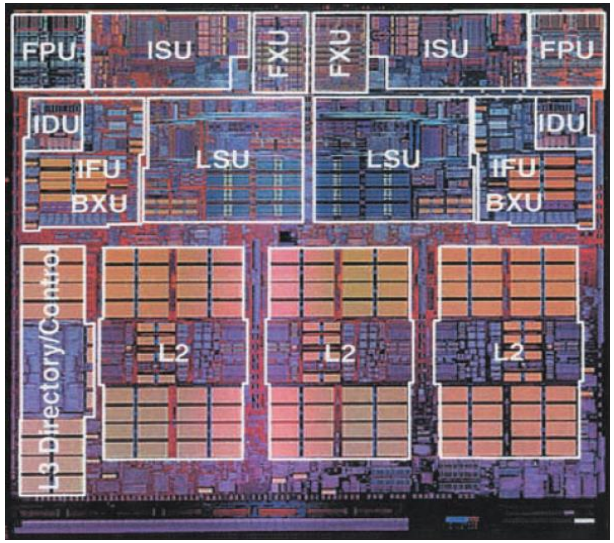
- 2 cores, ST
- 0.7 MB/core L2, 16MB/core L3 (off-chip)
- 115W TDP
- 10GB/s mem BW

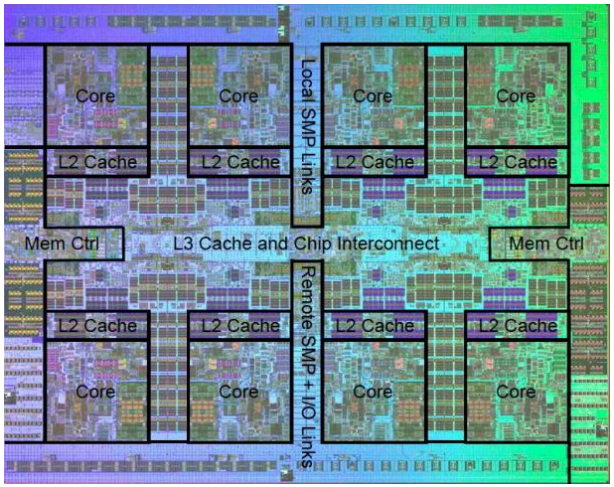
## IBM Power7 (2010)

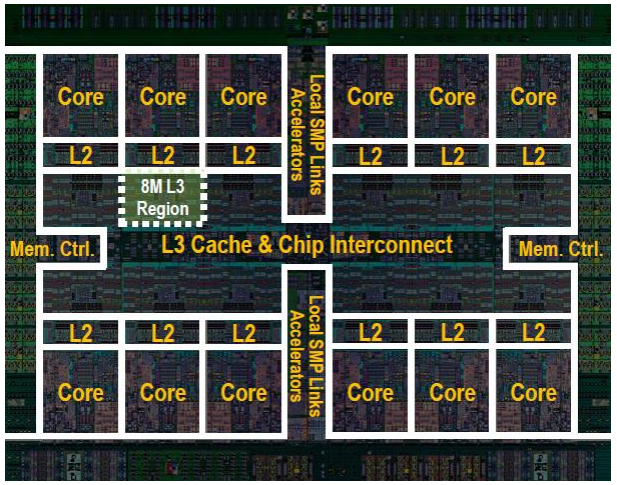
- 8 cores, SMT4
- 256 KB/core L2 16MB/core L3 (on-chip)
- 170W TDP
- 100GB/s mem BW

## IBM Power8 (2014)

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

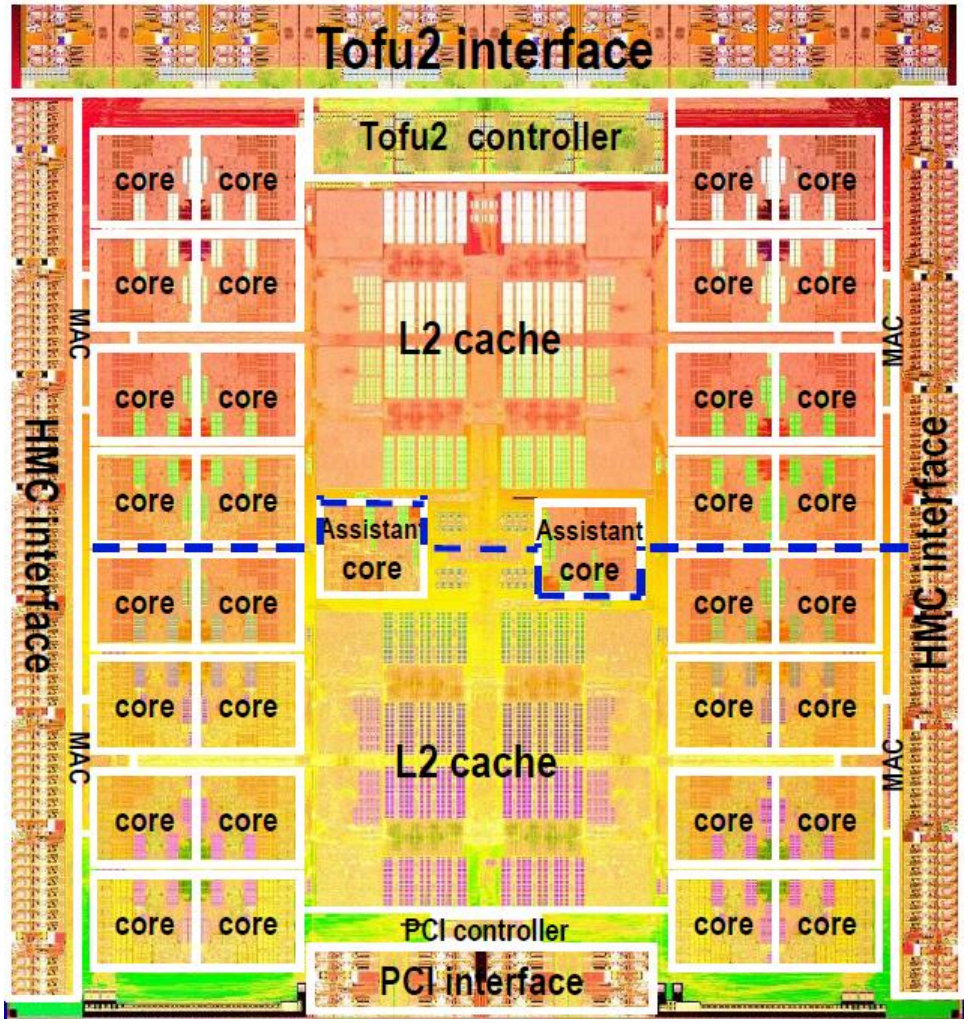


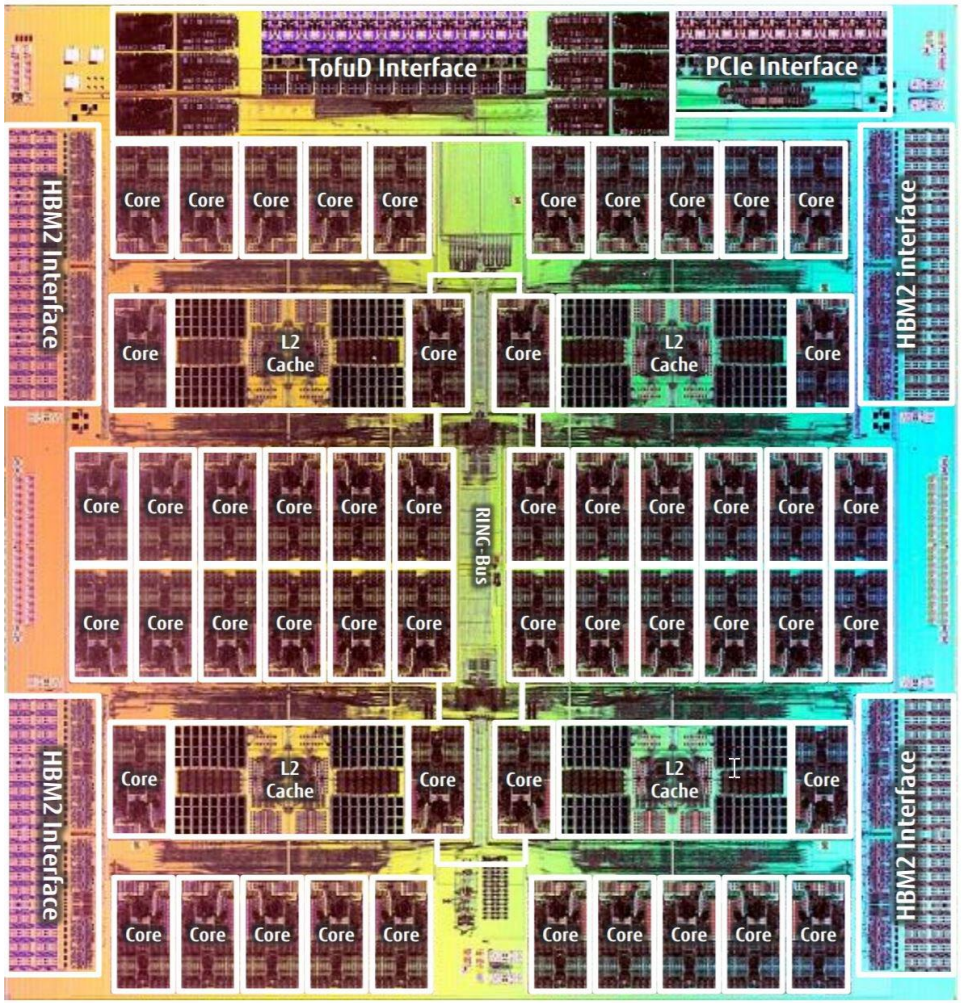
# Fujitsu SPARC64 Xifx (2014)

- 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)



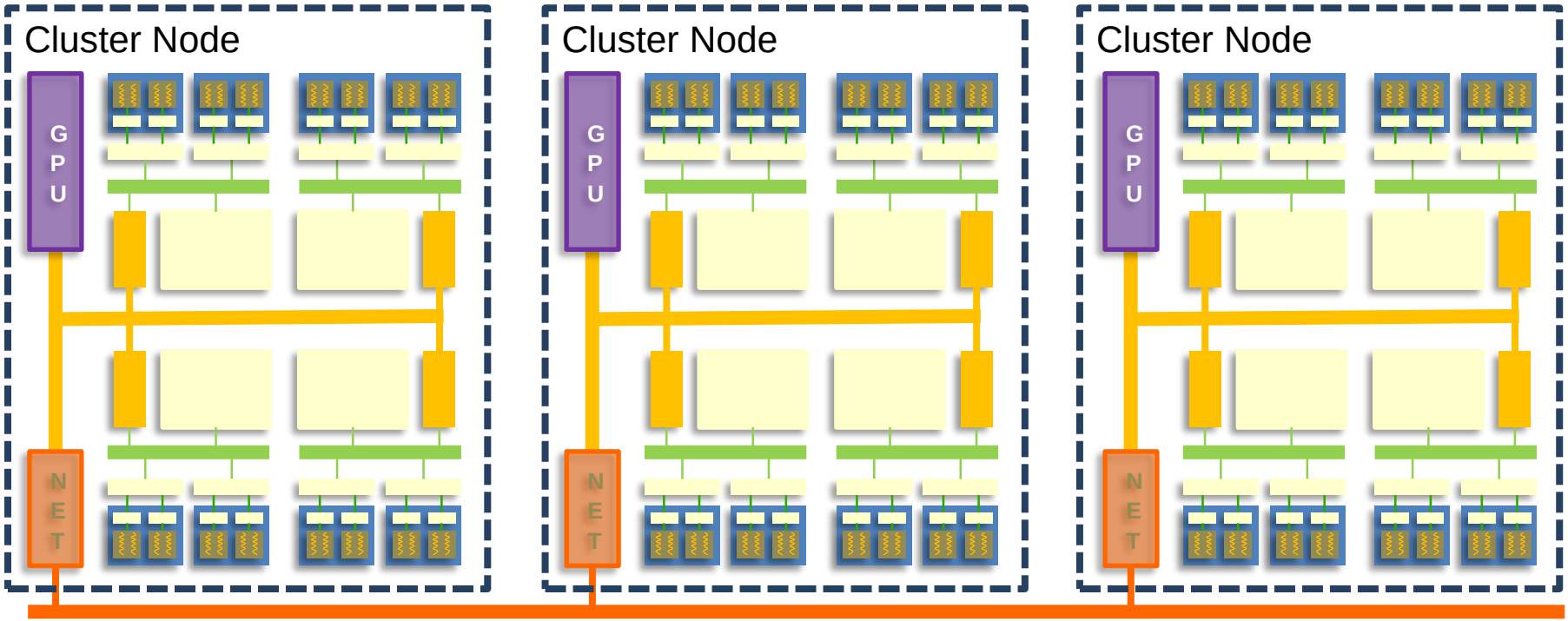
# Fujitsu A64FX Fugaku (2020)

- 4 sockets and 12 CPUs/socket
- Total 48 CPUs per node
- 32MiB (8MiB x 4) L2 cache
- 512-wide SIMD units
- 7nm, 8,786M transistors
- 2.2GHz peak frequency
- 3.4TFlops peak performance
- Memory BW
  - 256GB/s per HBM2 package
    - 4 HMB2 packages
- Memory capacity
  - 8GiB per HMB2 package

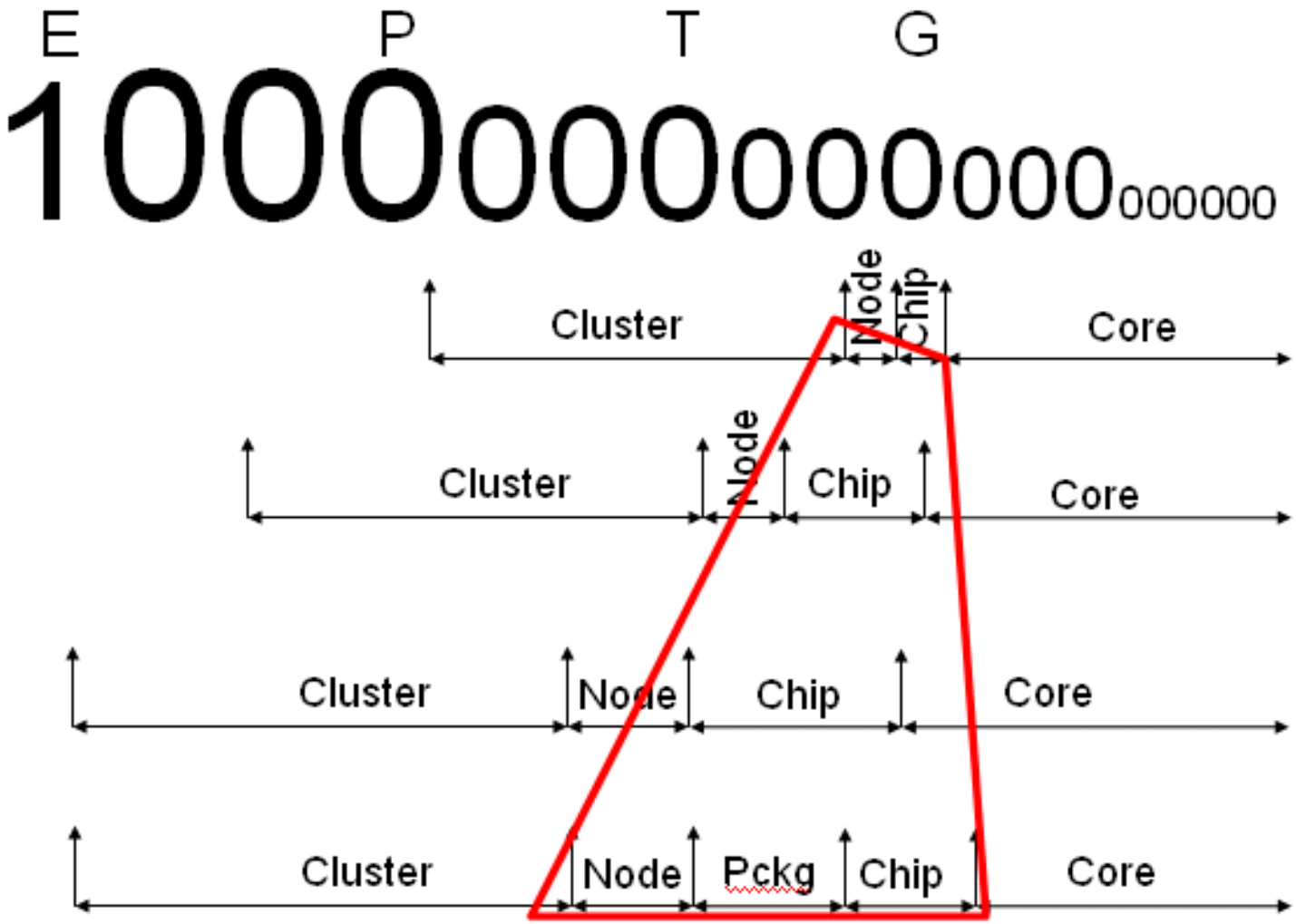


# Cluster Machines

- 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

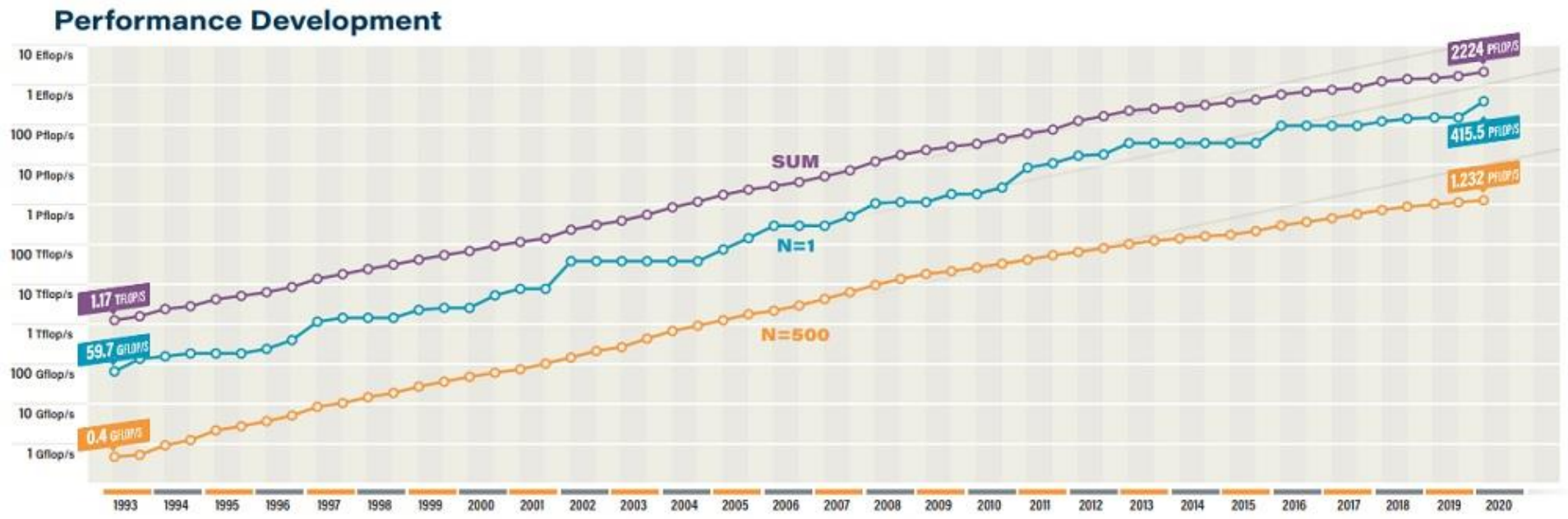


# The Path to ExaFlop Computers

E P T G

1000000000000000000

Floating Point Ops / Sec




Source: Top500

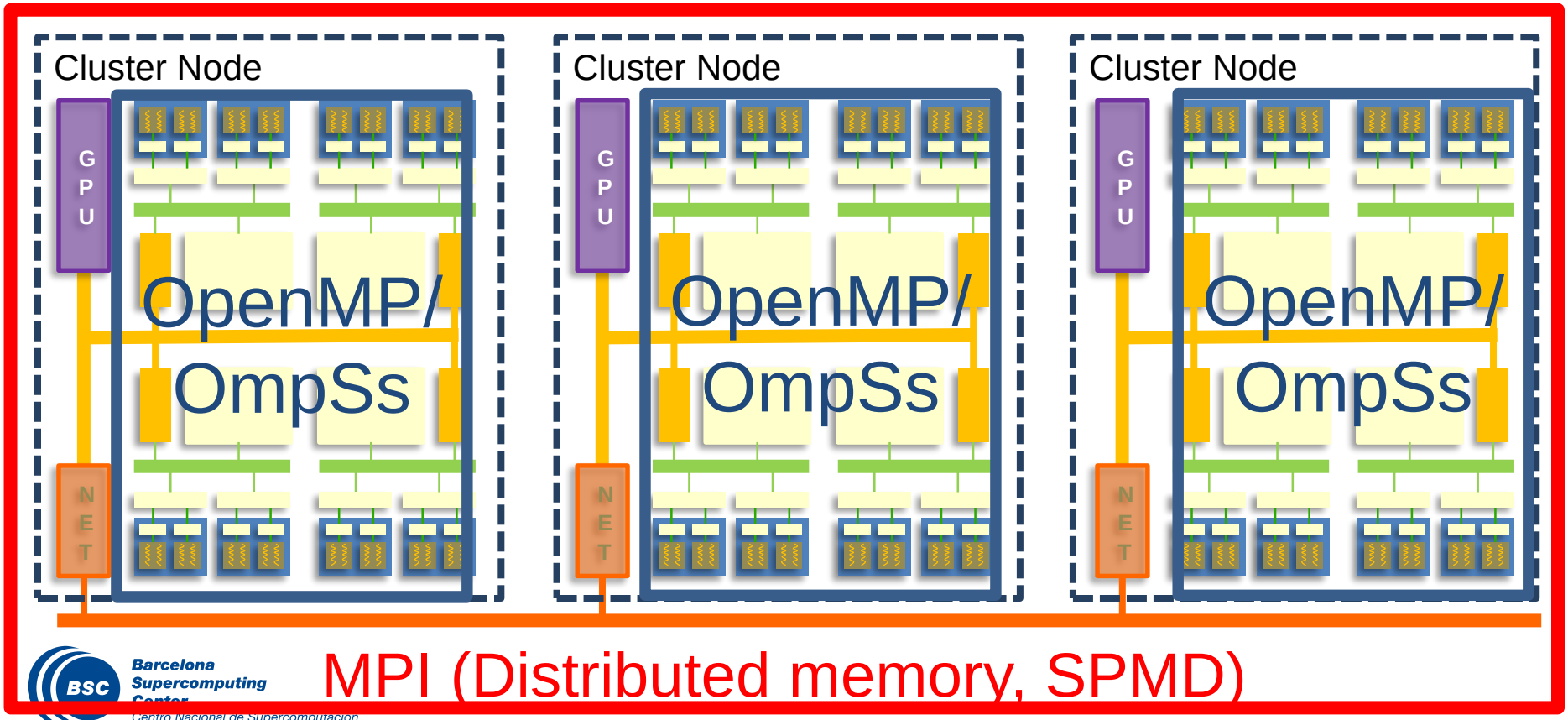
# Top500 Supercomputers List

| | System | Peak Petaflops | HPL Petaflops | Compute Efficiency | Concurrent Cores+SMs | Cores+SMs 1 Exaflops HPL | Compute Node Configuration CPU+Accelerator | Interconnect |
|---|---|---|---|---|---|---|---|---|
| | *NSC/Tianjin "Tianhe-3"* | *2,050.0* | *1,567.6* | *76.5%* | *???* | *???* | *2 * Phytium Arm + Matrix 3000* | *400 Gb/sec TH-Express 3 (IB)* |
| | *NSC/Wuxi "OceanLight"* | *1,500.0* | *1,220.0* | *81.3%* | *41,930,000* | *34,368,852* | *1 * Sunway SW26010-Pro* | *Custom InfiniBand* |
| 1 | Oak Ridge "Frontier" | 1,679.8 | 1,194.0 | 71.1% | 8,699,904 | 7,286,352 | 1 * AMD Trento Epyc + 4 * AMD MI250X | 200 Gb/sec Slingshot-11 |
| 2 | Argonne "Aurora" | 1,059.3 | 585.3 | 55.3% | 4,742,808 | 8,102,655 | 2 * Intel Xeon Max 9470 + 6 * Intel GPU Max 9470 | 200 Gb/sec Slingshot-11 |
| 3 | Microsoft Azure "Eagle" | 846.8 | 561.2 | 66.3% | 1,123,200 | 2,001,426 | 2 * Intel Xeon 8480C + 8 * Nvidia H100 | 400 Gb/sec NDR InfiniBand |
| 4 | RIKEN "Fugaku" | 537.2 | 442.0 | 82.3% | 7,630,848 | 17,263,971 | 1 * Fujitsu A64FX | 56 Gb/sec Tofu D |
| 5 | CSC "LUMI" | 531.5 | 379.7 | 71.4% | 2,725,704 | 7,178,573 | 1 * AMD Trento Epyc + 4 * AMD MI250X | 200 Gb/sec Slingshot-11 |
| 6 | CINECA "Leonardo" | 304.5 | 238.7 | 78.4% | 1,824,768 | 7,644,608 | 1 * Intel Xeon 8358 + 4 * Nvidia A100 | 100 Gb/sec HDR InfiniBand |
| 7 | Oak Ridge "Summit" | 200.8 | 148.6 | 74.0% | 2,414,592 | 16,248,937 | 2 * IBM Power9 + 6 * Nvidia V100 | 100 Gb/sec EDR InfiniBand |
| 8 | BSC "MareNostrum 5 ACC" | 234.0 | 138.2 | 59.1% | 680,960 | 4,927,352 | 1 * Intel Xeon 8460Y + 4 * Nvidia H100 | 200 Gb/sec NDR InfiniBand |
| 9 | Nvidia "Eos" | 188.7 | 121.4 | 64.4% | 485,888 | 4,002,372 | 2 * Intel Xeon 8480C + 8 * Nvidia H100 | 400 Gb/sec NDR InfiniBand |
| 10 | Lawrence Livermore "Sierra" | 125.7 | 94.6 | 75.3% | 1,572,480 | 16,615,385 | 2 * IBM Power9 + 4 * Nvidia V100 | 100 Gb/sec EDR InfiniBand |
| 11 | NSC/Wuxi "TaihuLight" | 125.4 | 93.1 | 74.2% | 10,649,600 | 114,388,829 | 1 * Sunway SW26010 | Custom InfiniBand |
| 12 | Lawrence Berkeley "Perlmutter" | 113.0 | 79.2 | 70.1% | 888,832 | 11,218,377 | 1 * AMD Epyc 7763 + 4 * Nvidia A100 | 200 Gb/sec Slingshot-11 |
| 13 | Nvidia "Selene" | 79.2 | 63.5 | 80.1% | 555,520 | 8,753,861 | 2 * AMD Epyc 7742 + 8 * Nvidia A100 | 100 Gb/sec HDR InfiniBand |
| 14 | NSC/Guangzhou "Tianhe-2A" | 100.7 | 61.4 | 61.0% | 4,981,760 | 81,083,333 | 2 * Intel Xeon 2692 + 3 * Matrix 2000 | TH-Express 2+ Custom InfiniBand |
| 15 | Microsoft Azure "Explorer-WUS3" | 87.0 | 54.0 | 62.0% | 445,440 | 8,255,004 | 1 * AMD Epyc 7V12 + 4 * AMD MI250X | 400 Gb/sec NDR InfiniBand |
| 16 | Nebius AI "ISEG" | 86.8 | 46.5 | 53.6% | 218,880 | 4,703,051 | 1 * Intel Xeon 8468 + 4 * Nvidia H100 | 400 Gb/sec NDR InfiniBand |
| 17 | GENCI-CINES "Adastra" | 61.6 | 46.1 | 74.8% | 319,072 | 6,921,302 | 1 * AMD Trento Epyc + 4 * AMD MI250X | 200 Gb/sec Slingshot-11 |
| 18 | FZJ "JEWELS Booster Module" | 71.0 | 44.1 | 62.2% | 449,280 | 10,183,137 | 2 * AMD Epyc 7402 + 4 * Nvidia A100 | 100 Gb/sec HDR InfiniBand |
| 19 | BSC "MareNostrum 5 GPP" | 46.4 | 40.1 | 86.5% | 725,760 | 18,098,753 | 2 * Intel Xeon 03H-LC/8480+ | 200 Gb/sec NDR InfiniBand |
| 20 | King Abdullah "Shaheen III" | 39.6 | 35.7 | 90.0% | 877,824 | 24,616,489 | 2 * AMD Epyc 9654 | 200 Gb/sec Slingshot-11 |
| 21 | Eni "HPC5" | 51.7 | 35.5 | 68.5% | 669,760 | 18,893,089 | 2 * Intel 6252 + 4* Nvidia V100 | 100 Gb/sec HDR InfiniBand |
| 22 | Naver Corp "Sejong" | 40.8 | 33.0 | 80.9% | 277,760 | 8,424,628 | 1 * AMD Epyc 7742 + 4 * Nvidia A100 | 100 Gb/sec HDR InfiniBand |
| 23 | Microsoft Azure "Voyager-EUS2" | 39.5 | 30.1 | 76.0% | 253,440 | 8,433,943 | 2 * AMD Epyc 7V12 + 8 * Nvidia A100 | 100 Gb/sec HDR InfiniBand |
| 24 | Los Alamos "Crossroads" | 40.2 | 30.0 | 74.7% | 660,800 | 22,004,662 | 2 * Intel Xeon CPU Max 9480 | 200 Gb/sec Slingshot-11 |
| 25 | Pawsey Supercomputing "Setonix" | 35.0 | 27.2 | 77.6% | 181,248 | 6,673,343 | 1 * AMD Trento Epyc + 4 * AMD MI250X | 200 Gb/sec Slingshot-11 |
| 26 | ExxonMobil "Discovery 5" | 31.0 | 26.2 | 84.4% | 232,000 | 8,871,893 | 1 * AMD Epyc 7543 + 4 * Nvidia A100 | 200 Gb/sec Slingshot-11 |
| 27 | Argonne "Polaris" | 34.2 | 25.8 | 75.6% | 256,592 | 9,941,573 | 1 * AMD Epyc 7532 + 4 * Nvidia A100 | 200 Gb/sec Slingshot-10 |
| 28 | Samsung "SSC-21" | 31.8 | 25.2 | 79.3% | 204,160 | 8,108,022 | 1 * AMD Epyc 7543 + 4 * Nvidia A100 | 200 Gb/sec HDR InfiniBand |
| 29 | TACC "Frontera" | 38.8 | 23.5 | 60.7% | 448,448 | 19,066,667 | 2 * Intel Xeon 8280 | 100 Gb/sec HDR InfiniBand |
| 30 | CEA "CEA-HF" | 31.8 | 23.2 | 73.2% | 810,240 | 34,864,028 | 2 * AMD Epyc 7763 | Atos BXI V2 |

November 2023 List, taken from “The Next Platform”

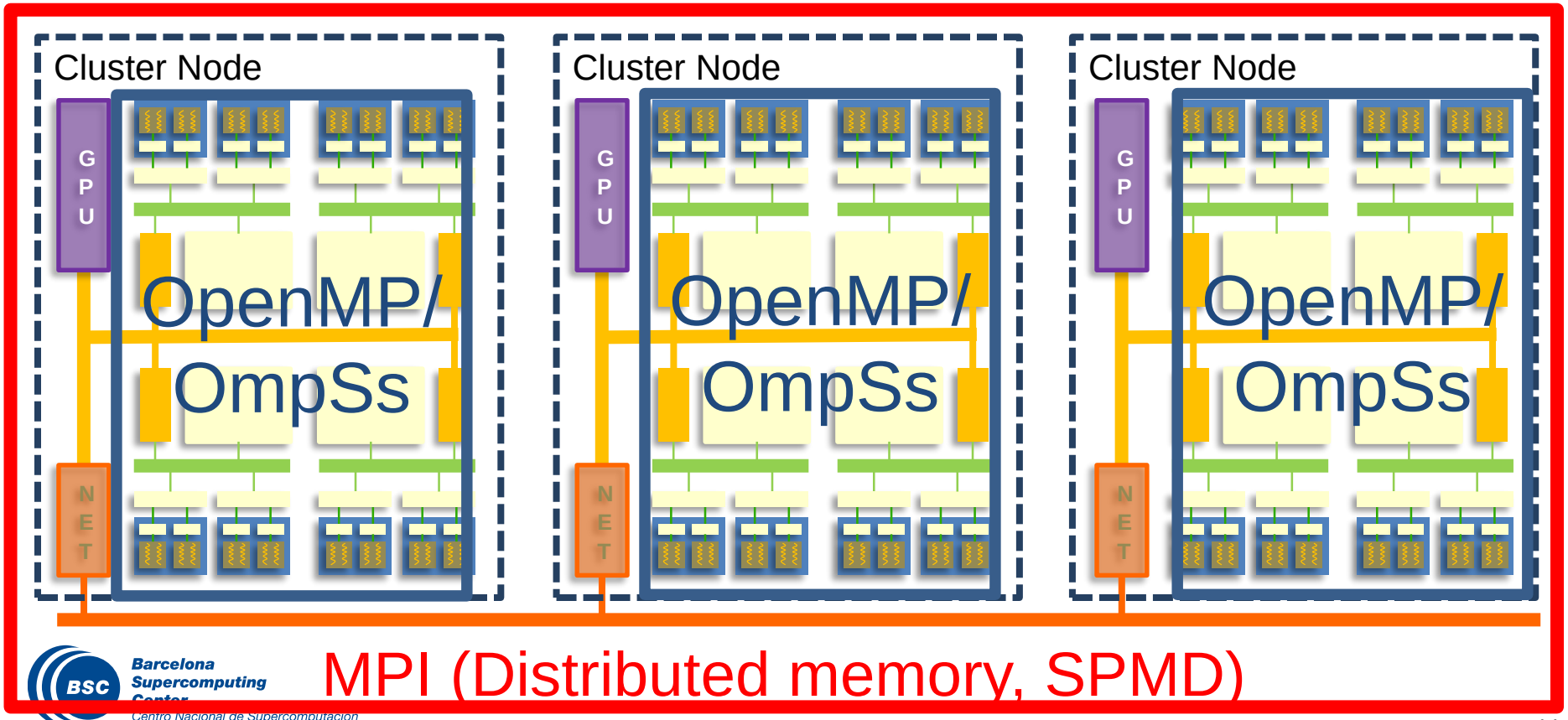
# Programming HPC Machines

- 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)



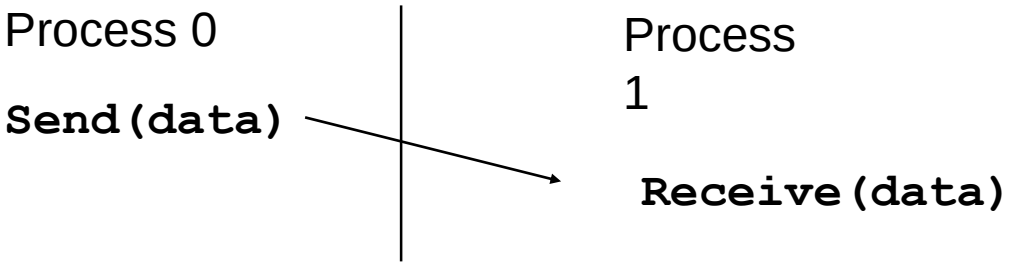
# Communication Across Nodes: Message Passing

- 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*

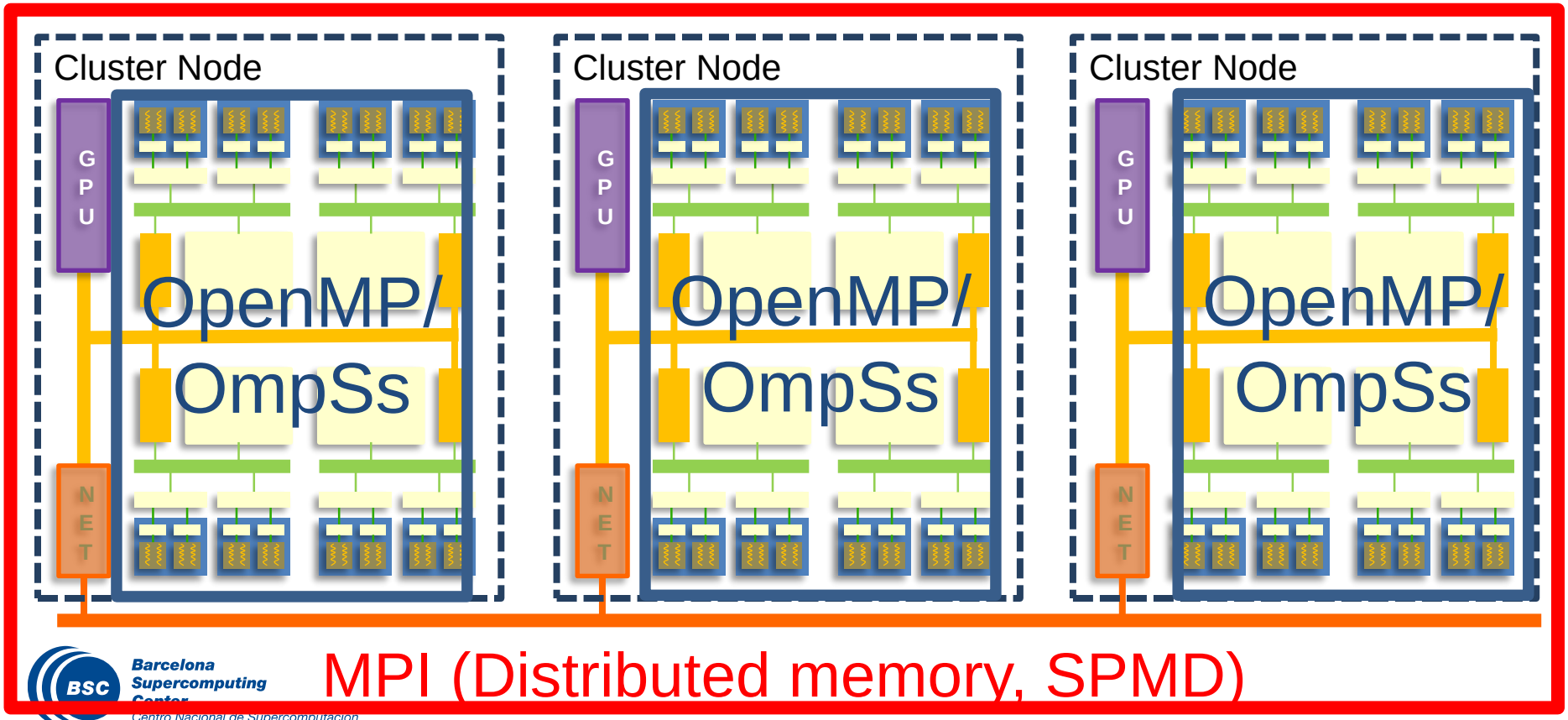
# Our first MPI program

```
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 …

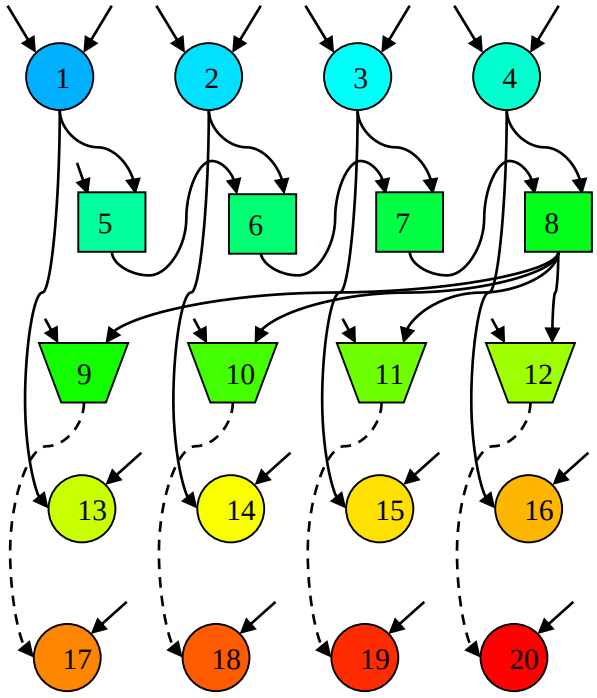
```
void vadd3 (float A[BS], float B[BS],
            float C[BS]);

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]);
```

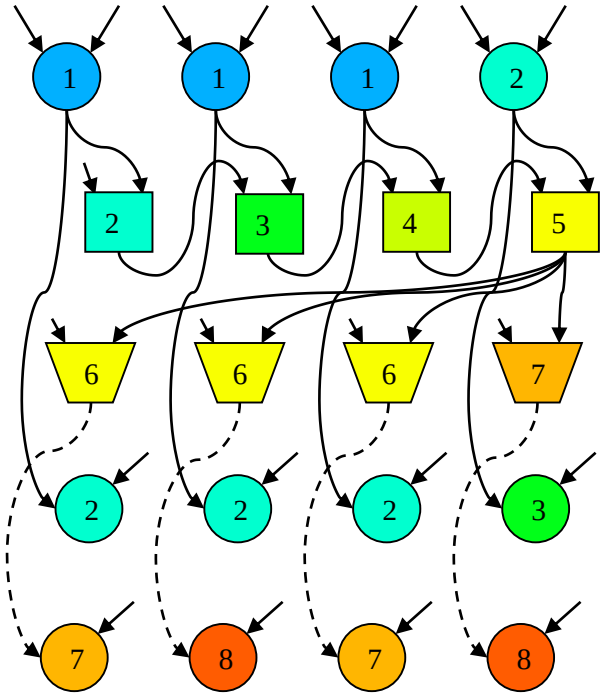

Color/number: order of task instantiation
Some antidependences covered by flow dependences not drawn

# 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

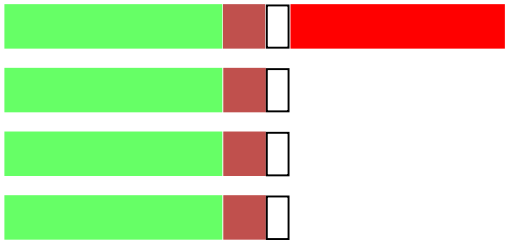




Color/number: a possible order of task execution

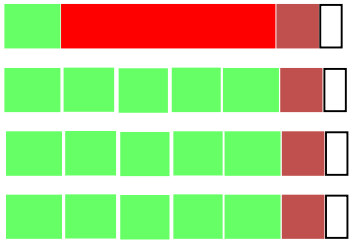
# OmpSs/OpenMP4.0: Data-flow and asynchronous

Four loops/routines
Sequential program order

OpenMP 2.5
not parallelizing one loop

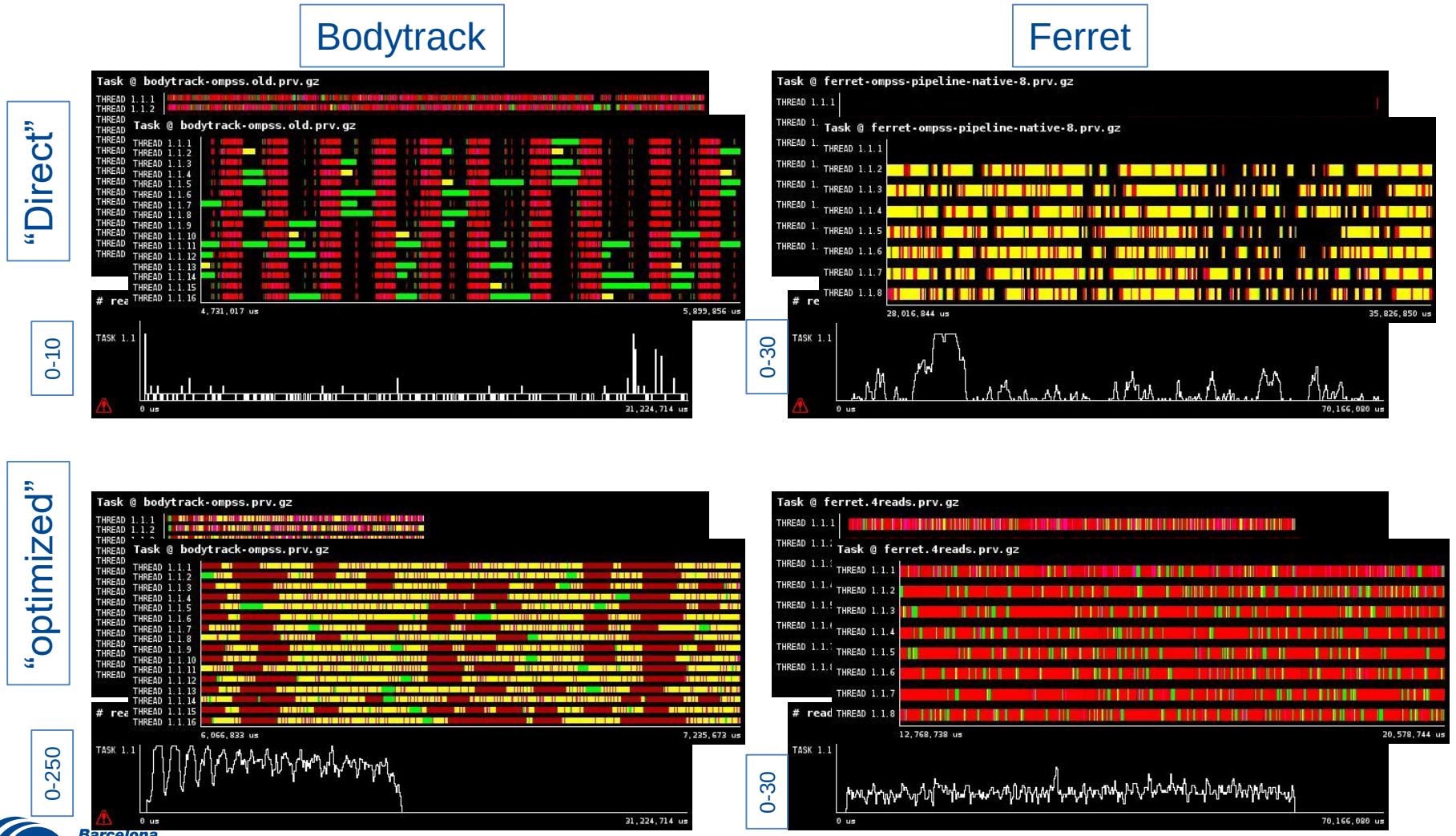


OmpSs/OpenMP4.0
not parallelizing one loop



# PARSEC

- Initial port from Pthreads to OmpSs and optimization





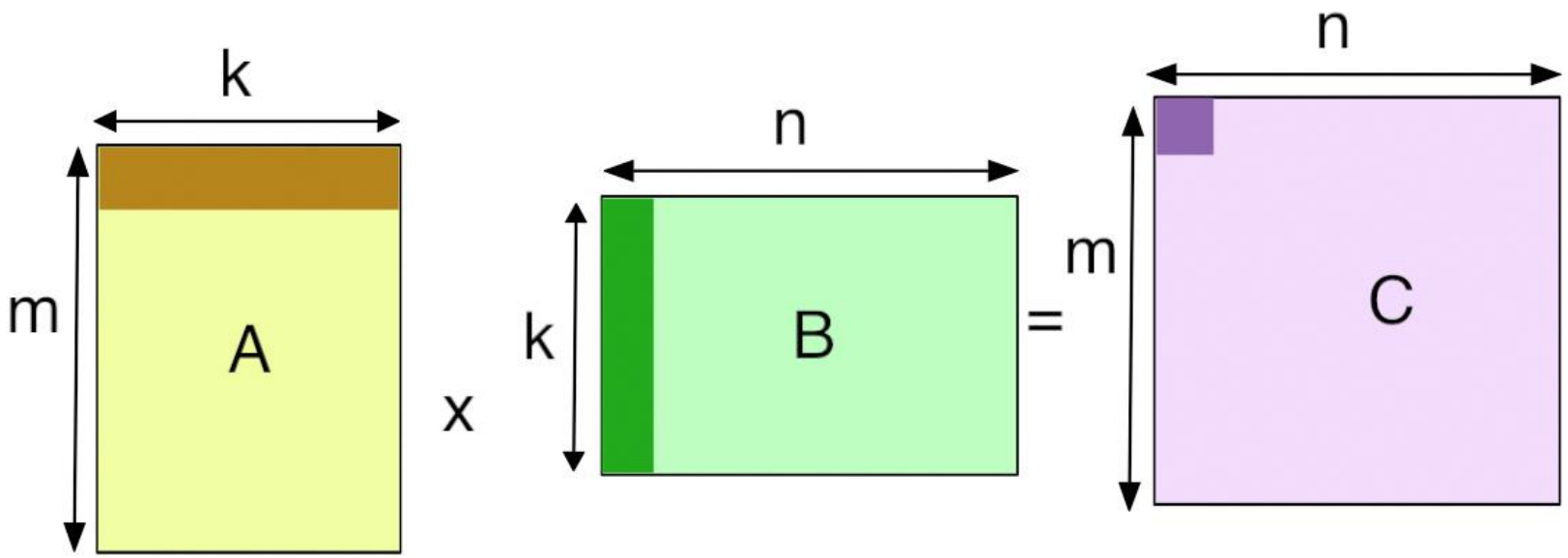

"Exploring the Impact of Task Parallelism in the PARSEC benchmark suite". TACO'2016

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

# PART2: ACCELERATING DEEP LEARNING WORKLOADS IN THE HPC CONTEXT

# The Fundamental DL Numerical Kernel: GEMM

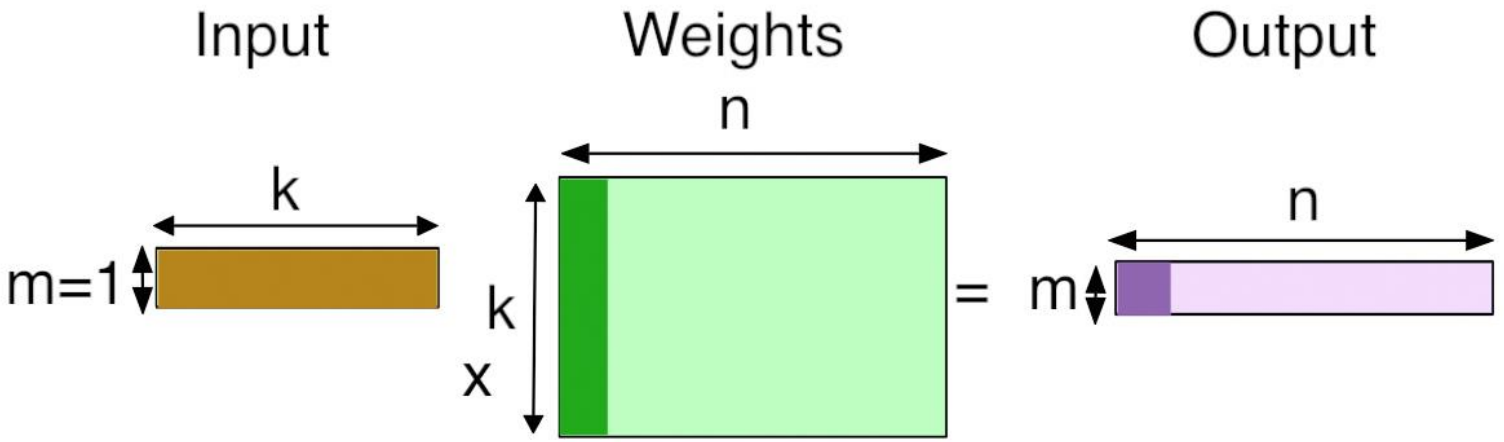
- GEMM stands for GEneral Matrix to Matrix Multiplication




Picture Taken from Pete Warden's blog

# The Fundamental DL Numerical Kernel: GEMM

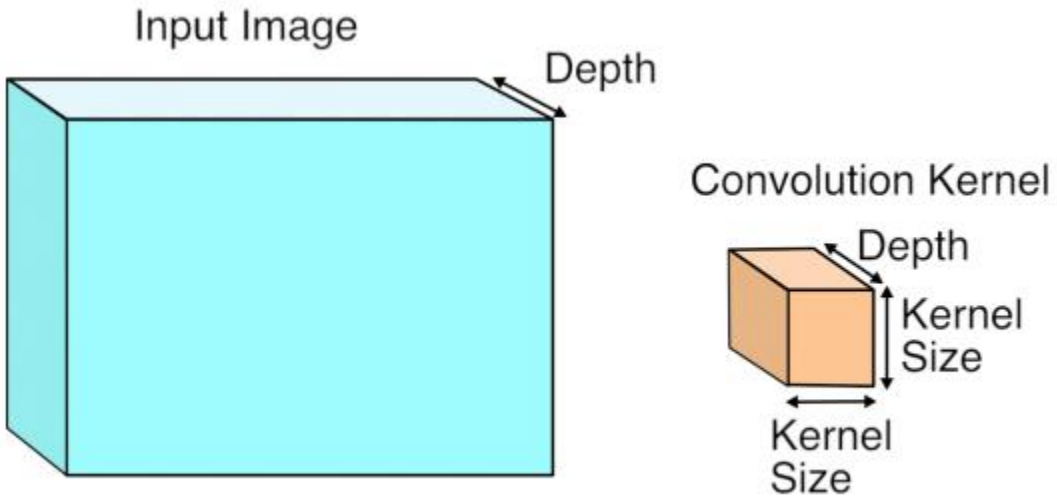
- GEMM stands for GEneral Matrix to Matrix Multiplication
- Fully-Connected layers (FC):
  - Each output value
    - looks at each value in the input layer
    - multiplies them all by the corresponding weight
    - sums the results to get its value




Picture Taken from Pete Warden's blog

# The Fundamental DL Numerical Kernel: GEMM

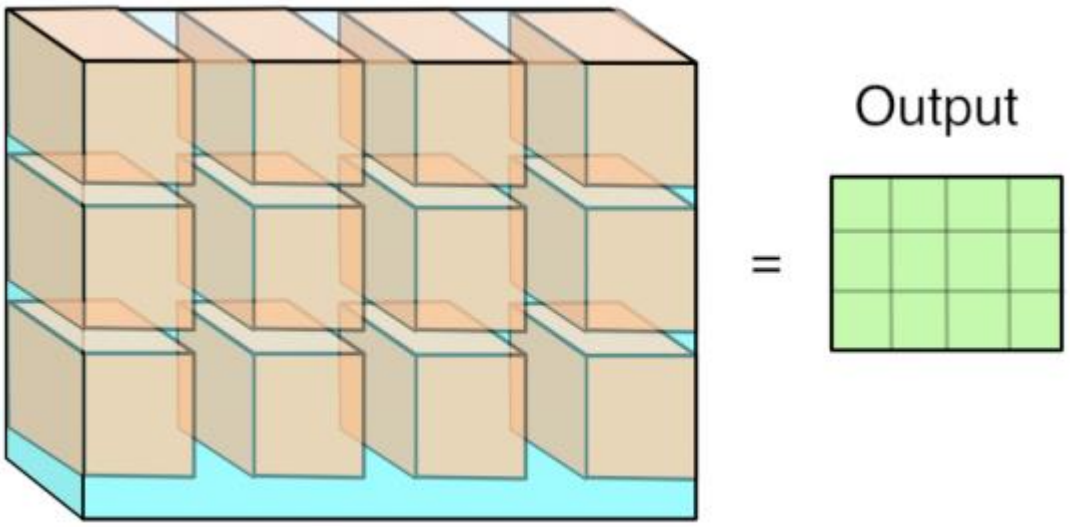
- GEMM stands for GEneral Matrix to Matrix Multiplication
- Convolutional Layers (CL)
  - treat inputs as a two dimensional image, with a number of channels for each pixel, much like a classical image with width, height, and depth.
  - The convolution operation produces its output by taking a number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




Picture Taken from Pete Warden’s blog

# The Fundamental DL Numerical Kernel: GEMM

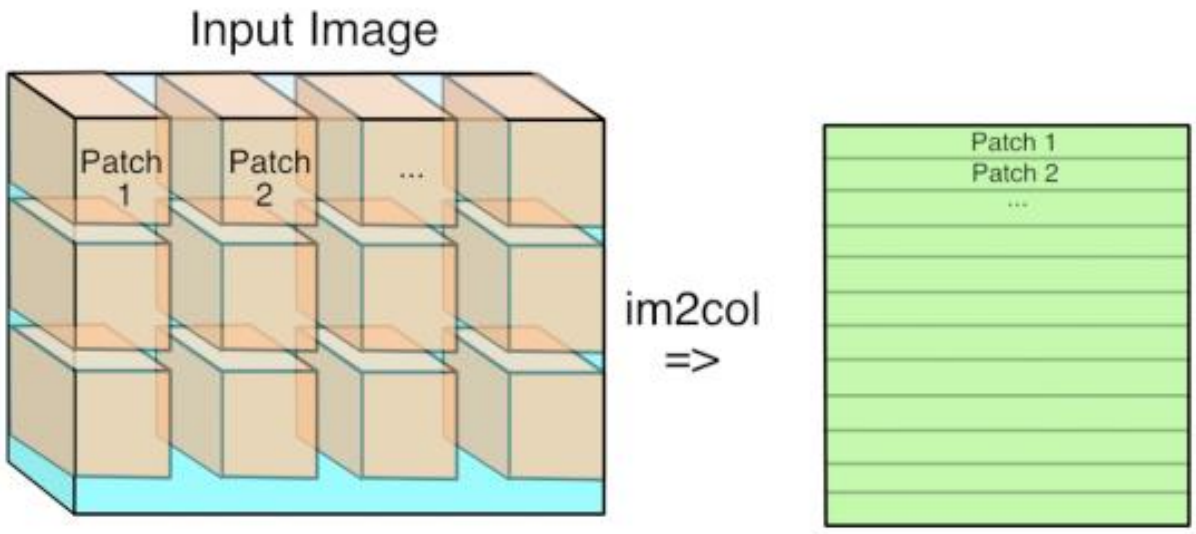
- GEMM stands for GEneral Matrix to Matrix Multiplication
- Convolutional Layers (CL)
  - Each kernel is another three-dimensional array of numbers, with the depth the same as the input image.
  - At each point the kernel is applied, all input values and weights are multiplied and then summed to produce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




Picture Taken from Pete Warden's blog

# The Fundamental DL Numerical Kernel: GEMM

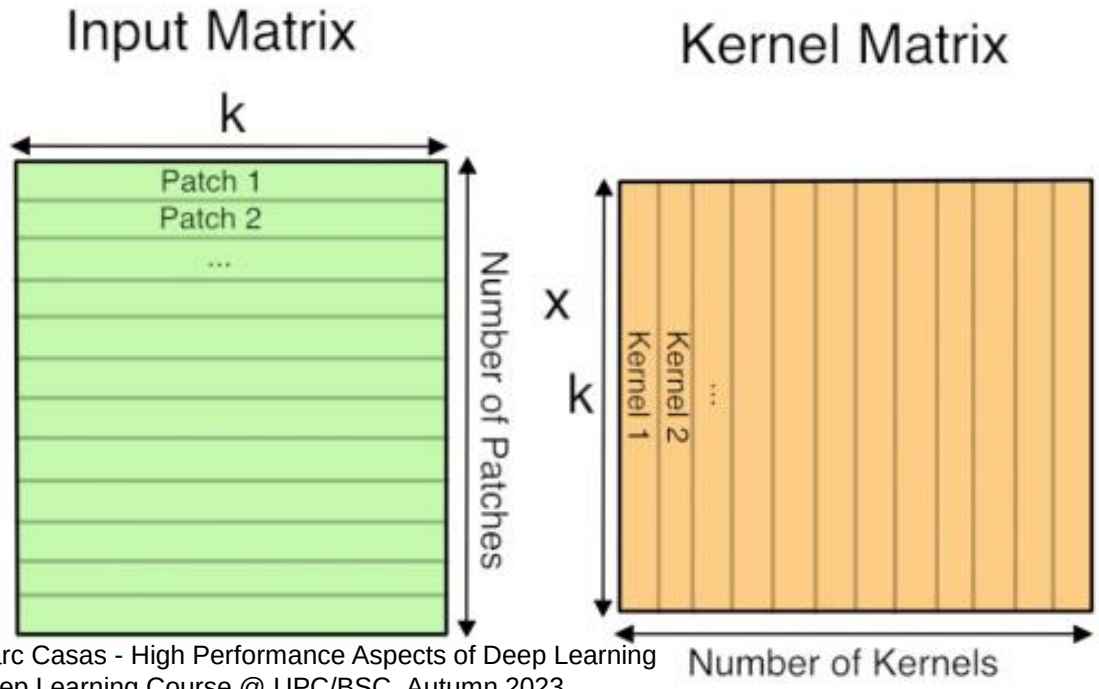
- GEMM stands for GEneral Matrix to Matrix Multiplication
- Convolutional Layers (CL)
  - Each kernel is another three-dimensional array of numbers, with the depth the same as the input image.
  - At each point the kernel is applied, all input values and weights are multiplied and then summed to produce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




Picture Taken from Pete Warden's blog

# The Fundamental DL Numerical Kernel: GEMM

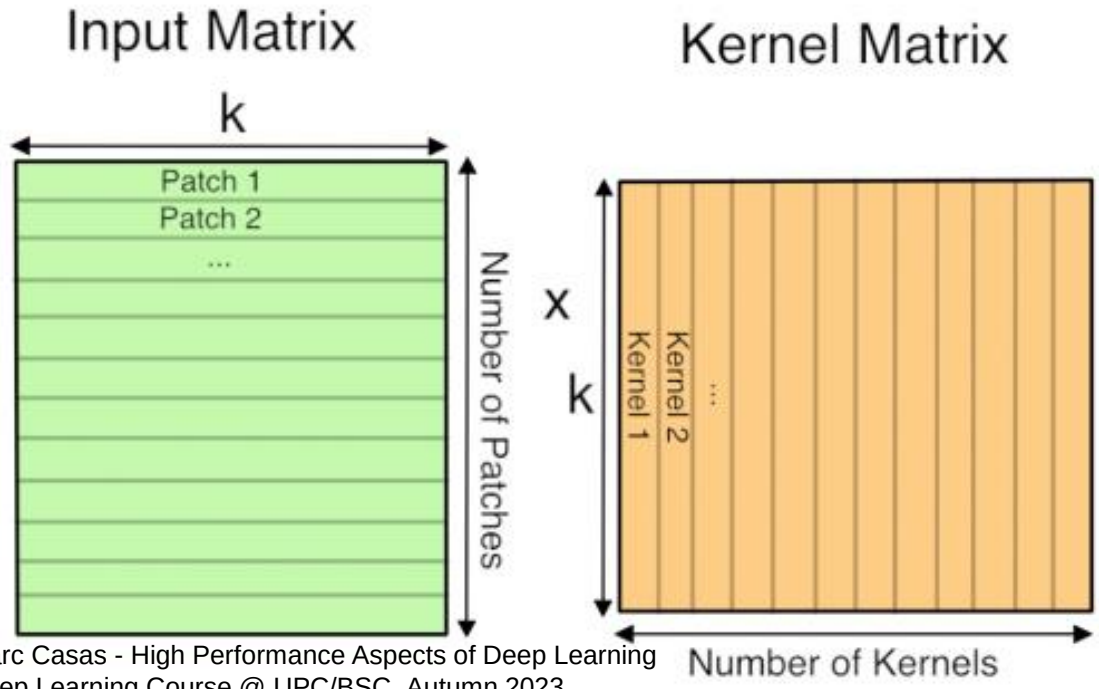
- GEMM stands for GEneral Matrix to Matrix Multiplication
- Convolutional Layers (CL)
  - Each kernel is another three-dimensional array of numbers, with the depth the same as the input image.
  - At each point the kernel is applied, all input values and weights are multiplied and then summed to produce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




Picture Taken from Pete Warden's blog

# The Fundamental DL Numerical Kernel: GEMM

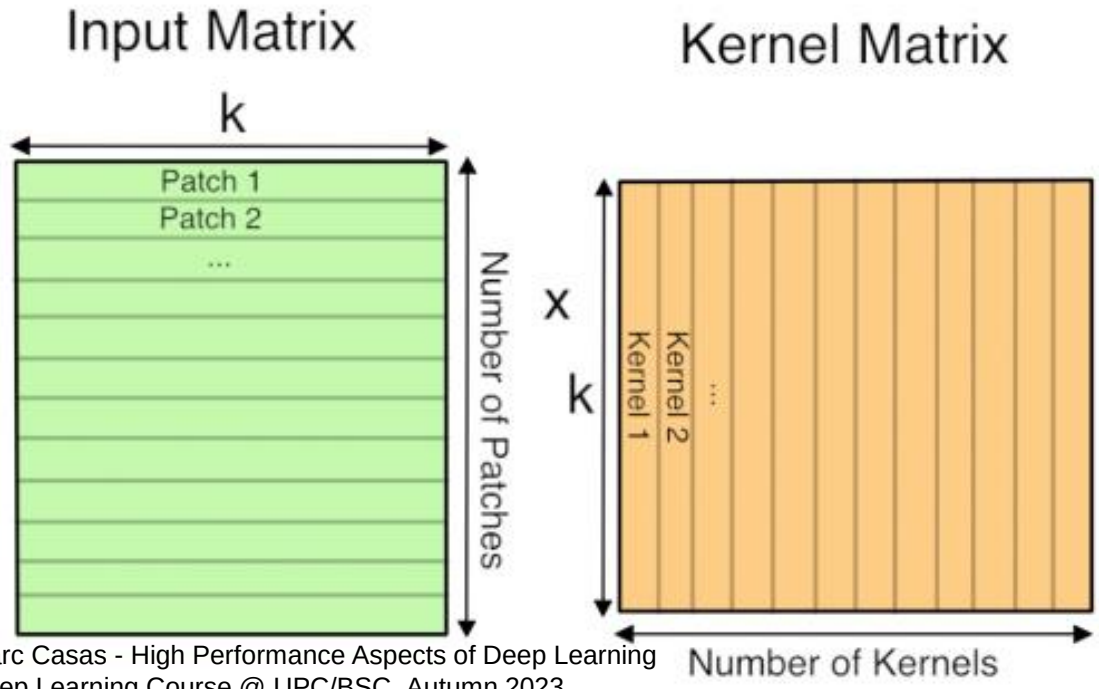
- GEMM stands for GEneral Matrix to Matrix Multiplication
- Why does GEMM work for Convolutional Layers (CL)?
  - Cons:
    - It is not the only possible mathematical formulation.
    - Pixels that are included in overlapping kernel sites will be duplicated in the matrix, which seems inefficient.




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

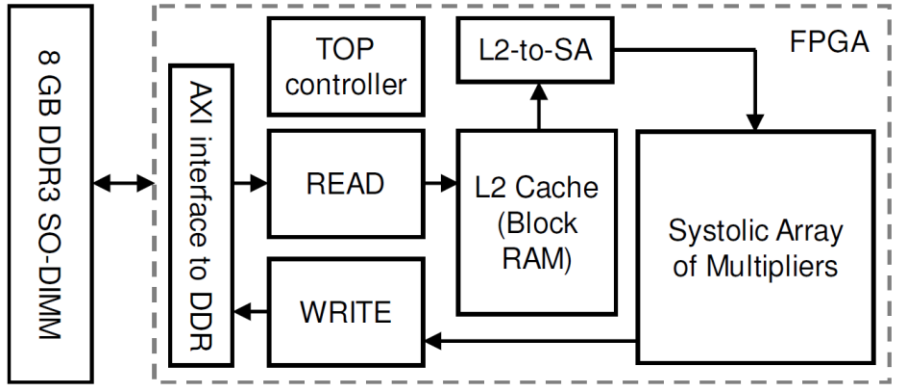
- DL on General-purpose hardware:
  - Classical HPC approaches based on MPI+OpenMP parallel schemes running on clusters
- GPU's:
  - Practical exercise on 24/11/2022
- GEMM-Specific Accelerators:
  - 2D systolic arrays
    - Google's TPU, Nervana's Lake Crest (Intel), etc.
- Hardware-implemented Neural Networks:
  - IBM True North

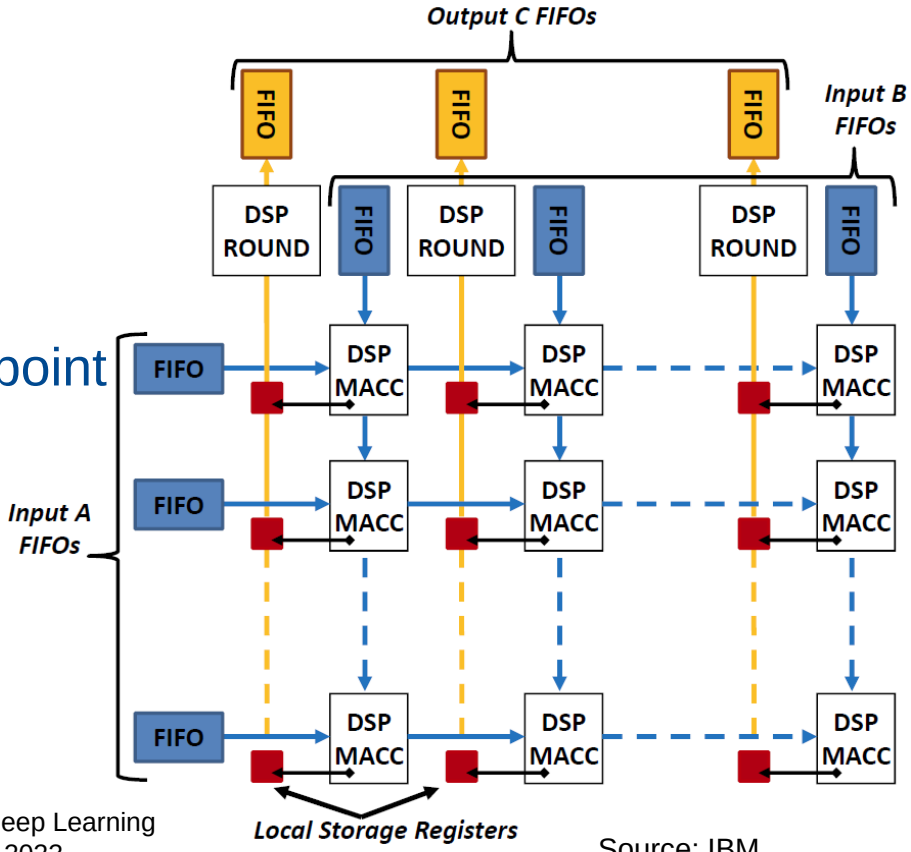
# Approaches for DL (i.e. GEMM) at HPC

- DL on General-purpose hardware:
  - Classical HPC approaches based on MPI+OpenMP parallel schemes running on clusters
- GPU's:
  - Practical exercise on 24/11/2022
- 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

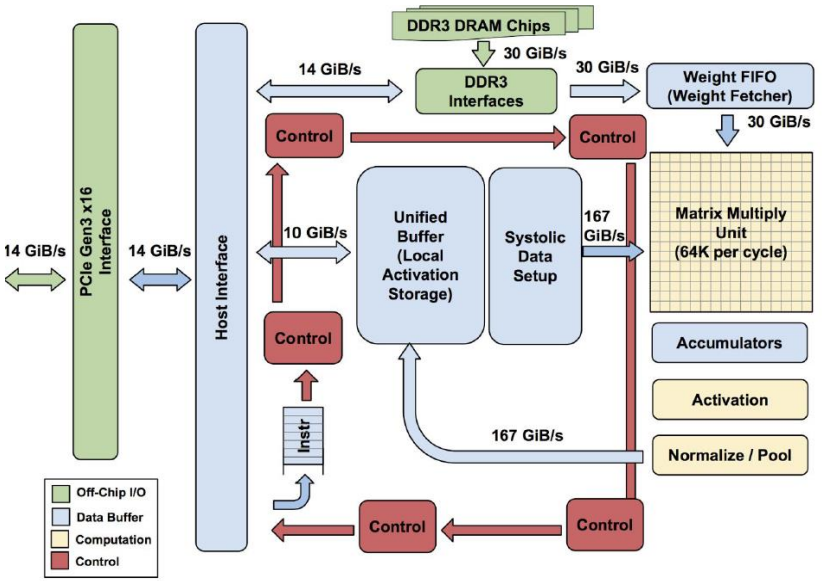


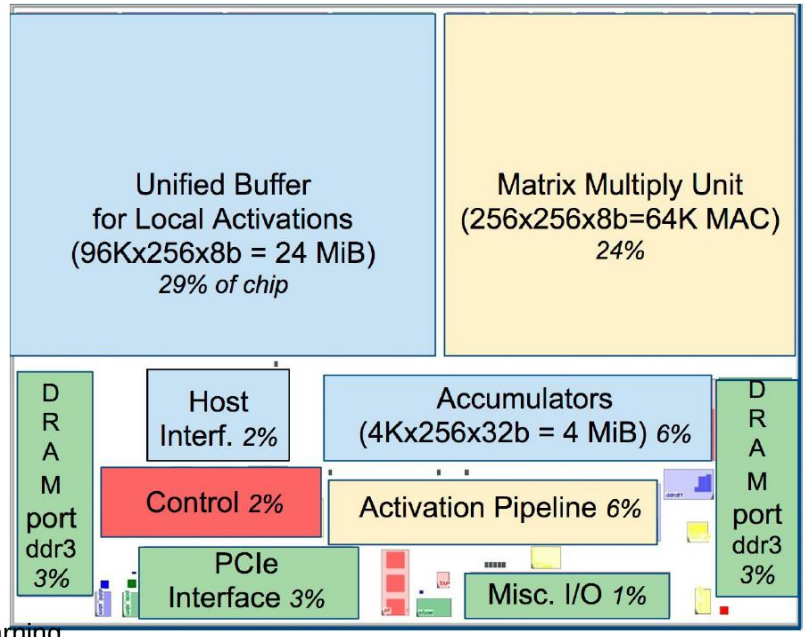




Source: IBM

# 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

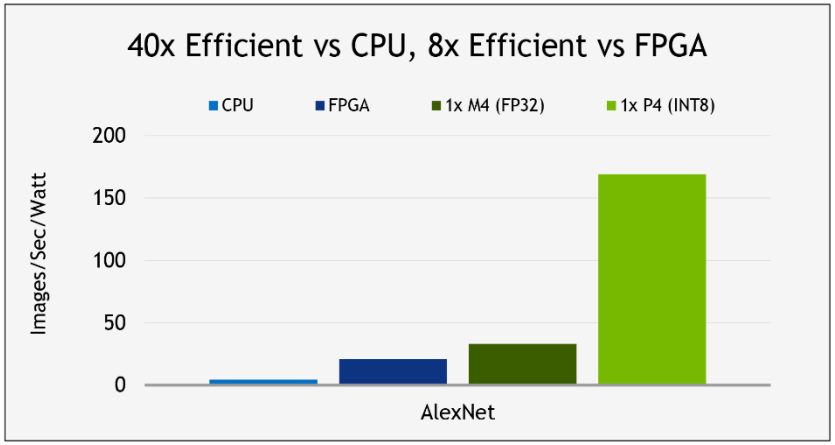


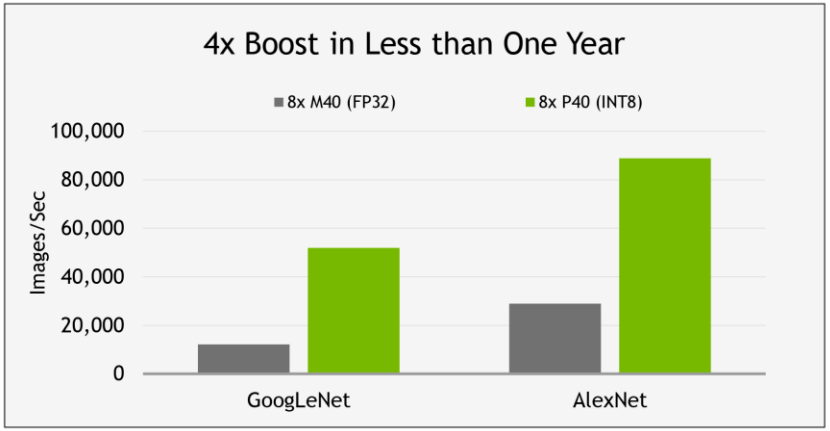




Source: google

# NVIDIA Tesla P4 and P40 GPU's (2016)

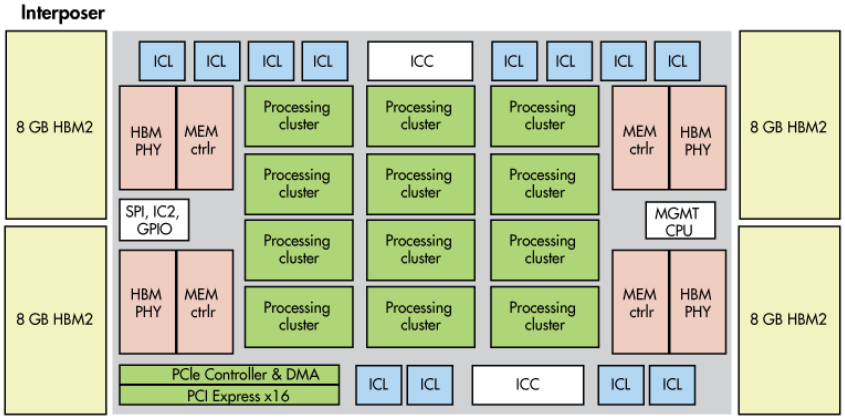
- Tesla P4
  - # CUDA cores: 2560 @ 1063MHz
  - Peak single precision: 5.5TFLOPS
  - Peak INT8: 22 TOPS
  - Low precision: 8-bit dot-product with 32-bit accumulate
  - VRAM: 8 GB GDDR5 @ 192 GB/s
  - TDP: ~75W
- Tesla P40
  - # CUDA cores: 2560 @ 1531MHz
  - Peak single precision: 12.0TFLOPS
  - Peak INT8: 47 TOPS
  - Low precision: 8-bit dot-product with 32-bit accumulate
  - VRAM: 24 GB GDDR5 @ 346GB/s
  - TDP: ~250W



# 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

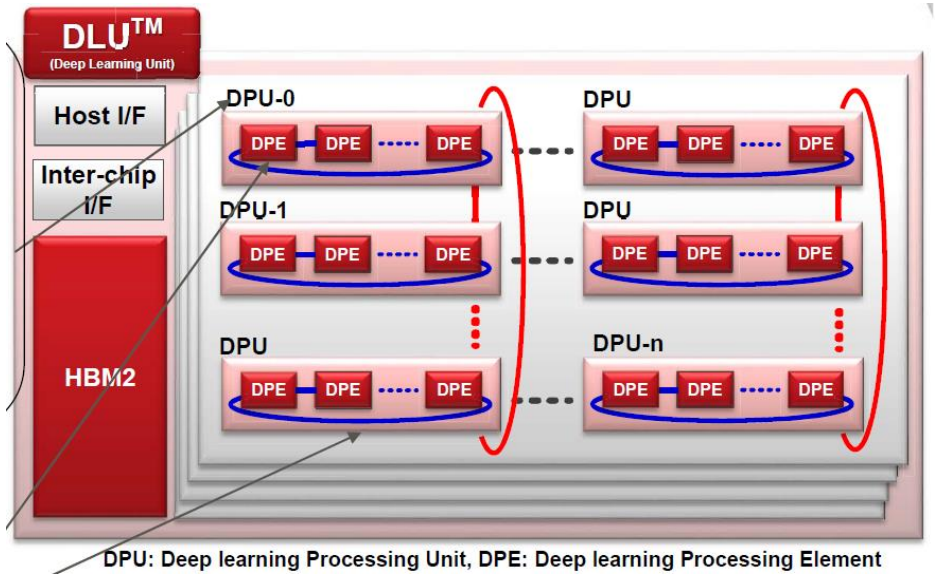


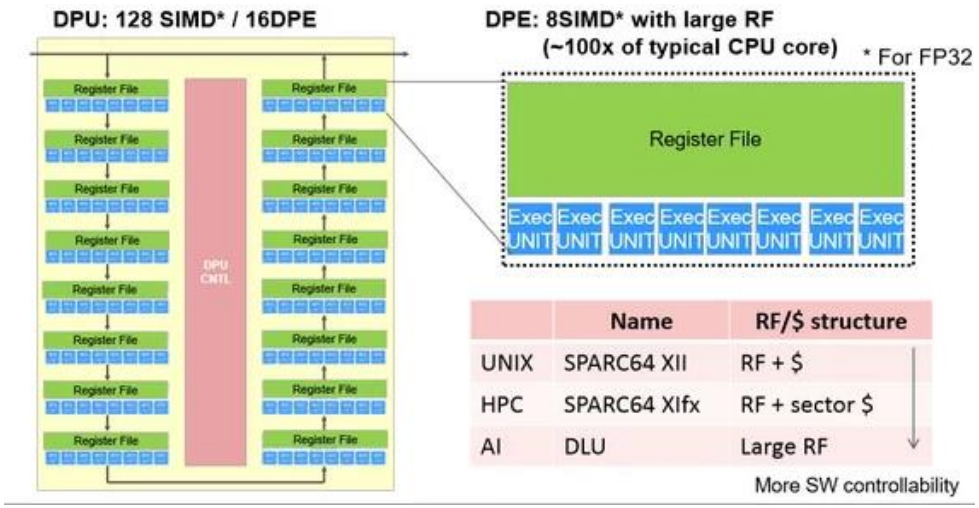



Source: elektroniknet

# Fujitsu Deep Learning Unit (2017)

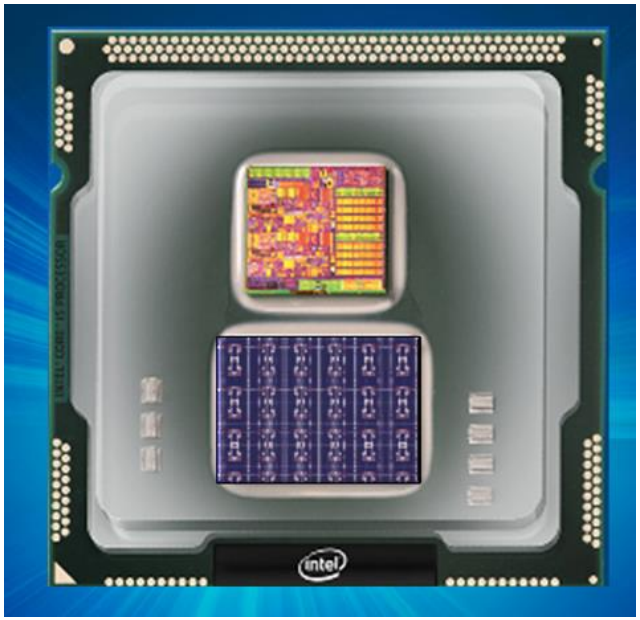
- DLU composed of many Deep Processing Units (DPU) connected via a NoC
- DPU consists of 16 Deep Processing Elements (DPE) connected with another NoC
- DPE includes wide SIMD execution units and large Register File (RF)
  - RF is exposed to the SW
- Deep Learning Integer provides 8- and 16- bits accuracies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



# Intel Loihi (2017)

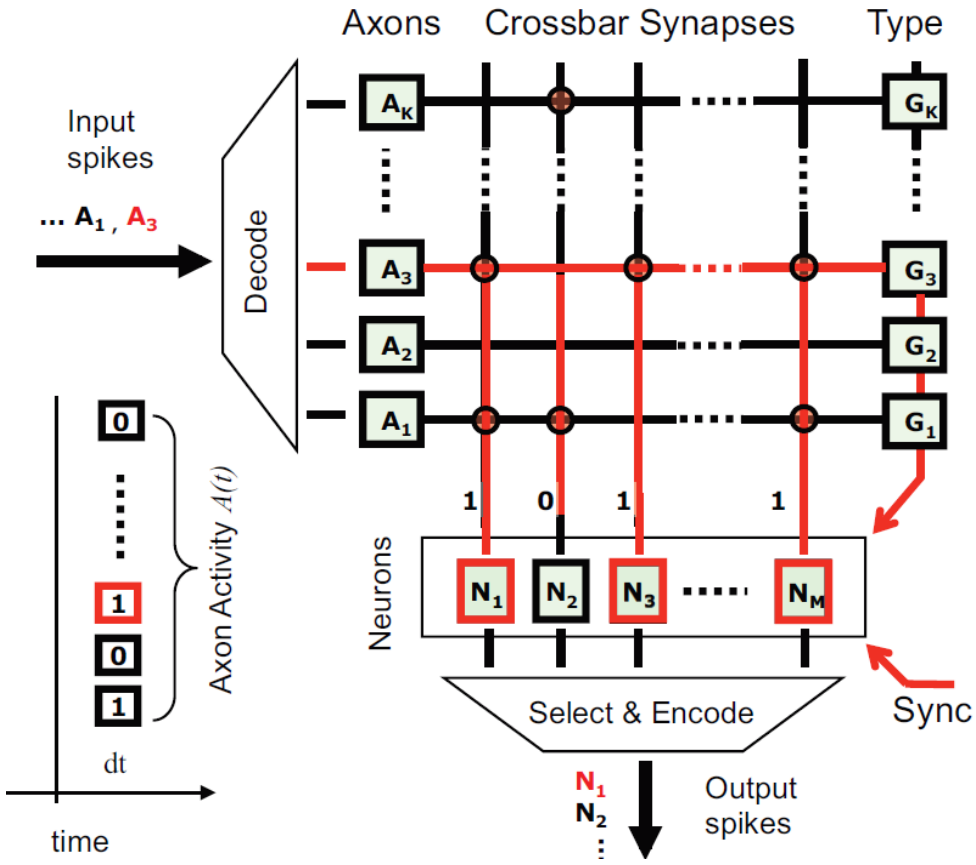
- Lohi is build with 14 nm technology and includes 130,000 neurons and 130 million synapses
- Loihi is composed of a many core mesh that supports:
  - sparse, hierarchical and recurrent neural network topologies
  - each neuron capable of communicating with thousands of other neurons
- Neuromorphic cores 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



# 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…

# Next Session will Describe the Practical Exercise

- It will take place on 04/05/2023.
  - I may upload the exercise before.

- Practical exercise is due on 31/05/2023.
  - Do not leave it for the very last day!


BSC
Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación