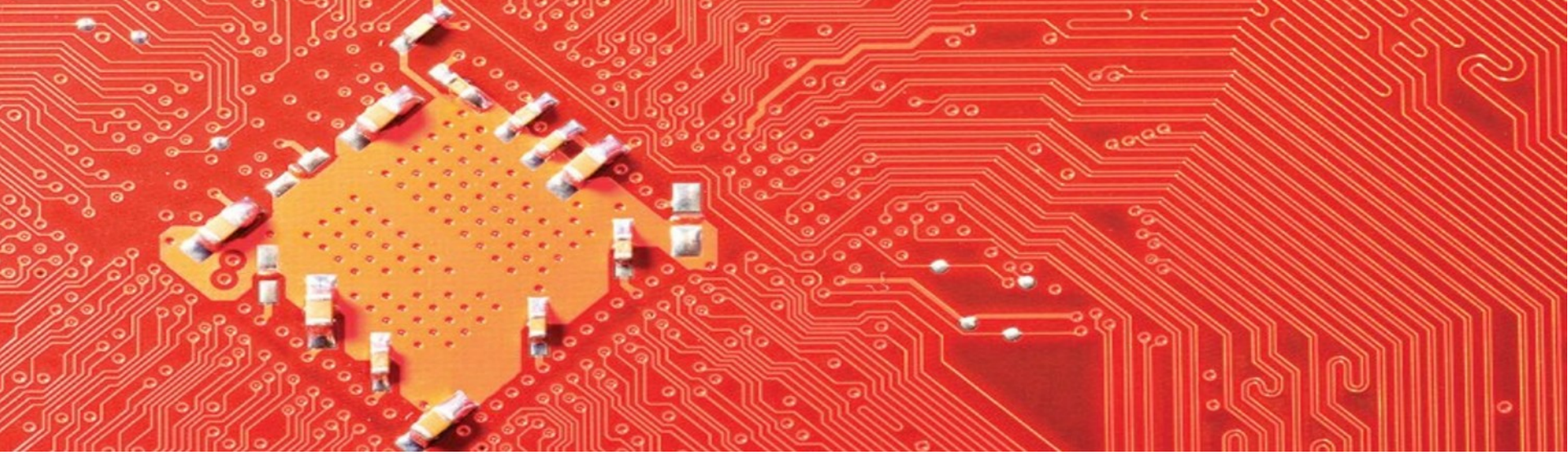
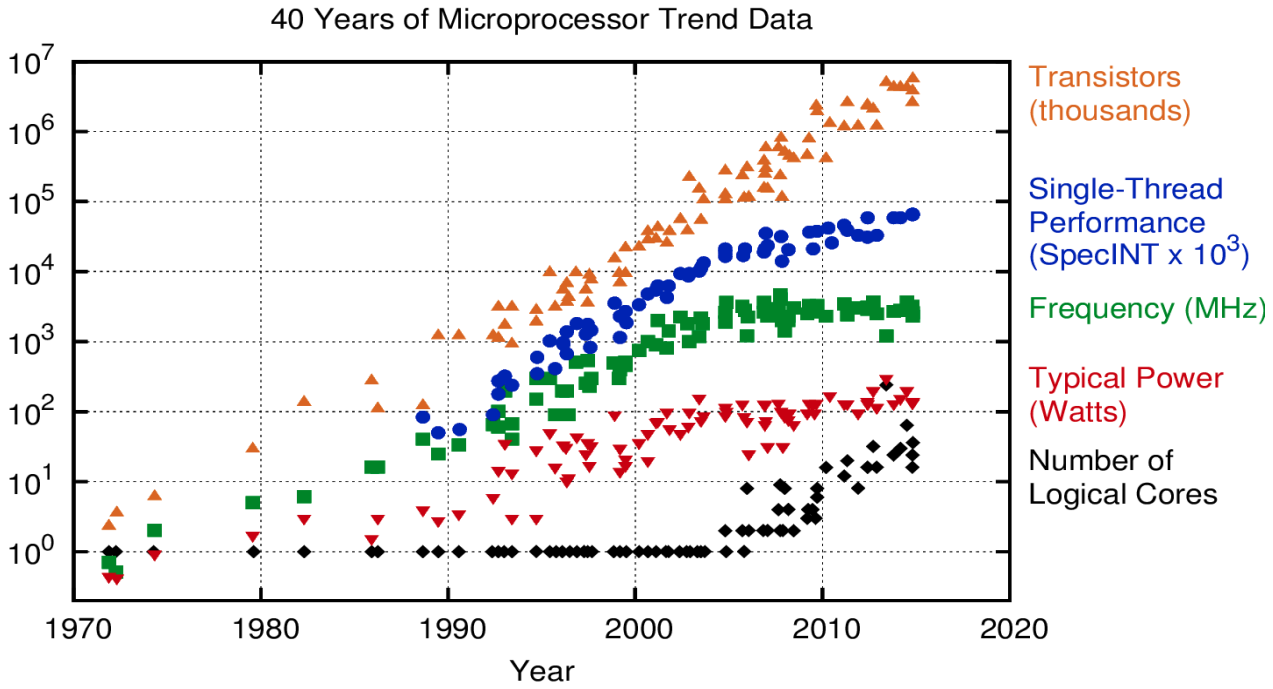


SOSCIP GPU Platform

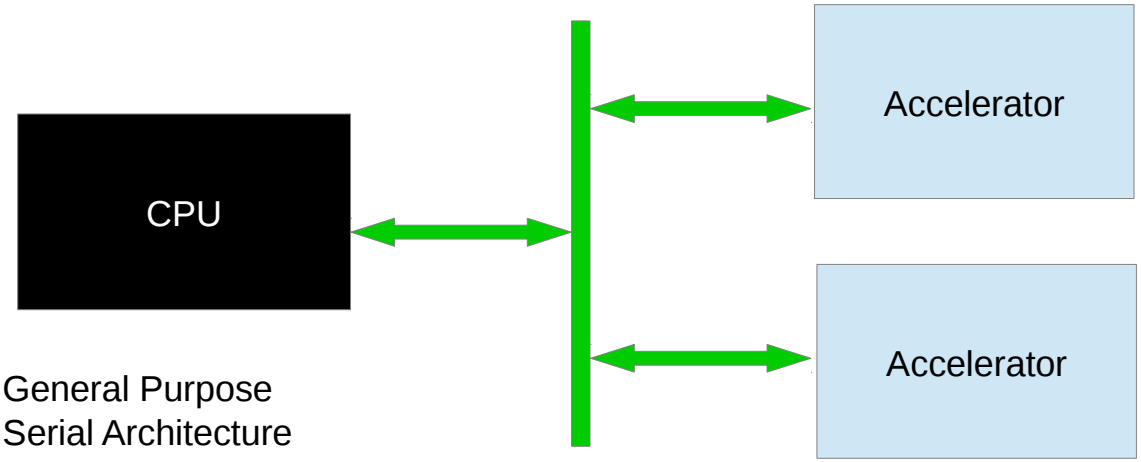
SOSCIP
SMART COMPUTING FOR INNOVATION

The Problem



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

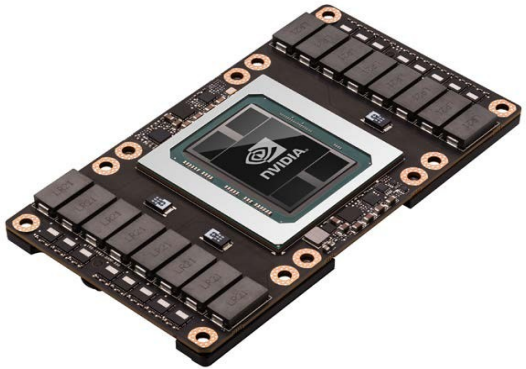
Heterogeneous Computing



- General Purpose
- Serial Architecture
- Light computation

- Specialized
- Parallel Architecture
- Heavy Computation

GPUs as Accelerators



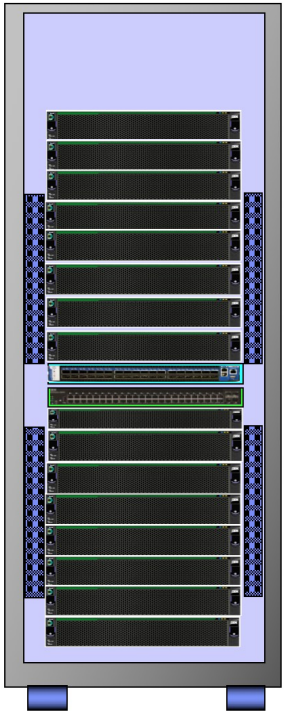
[Image source: NVIDIA whitepaper WP-08019-001_v01]

SCIP
SMART COMPUTING FOR INNOVATION

Agenda

- SOSCIP's GPU Platform
- Typical Applications
- System Access and Usage
- Software Development
- Available Pre-built Software

SOSCIP GPU Platform

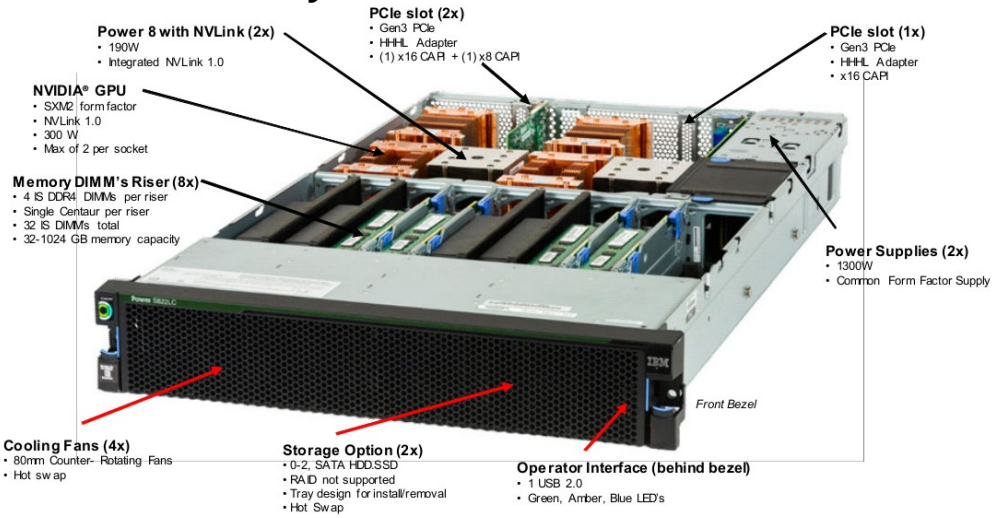


System Rack *1 x 7965-94Y*
RDHX *1 x 1164-95X*

InfiniBand TOR Switch *1 x 8828-E36*
Ethernet TOR Switch *1 x 7120-48L*
Login node *1 x 8001-12C*

Compute nodes *15 x 8335-GTB*

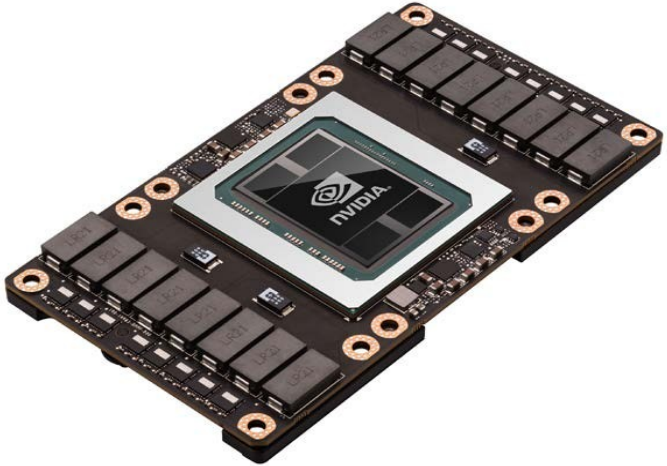
IBM Power System S822LC for HPC



System TFLOPS (theoretical): 318 DP, 1272 HP

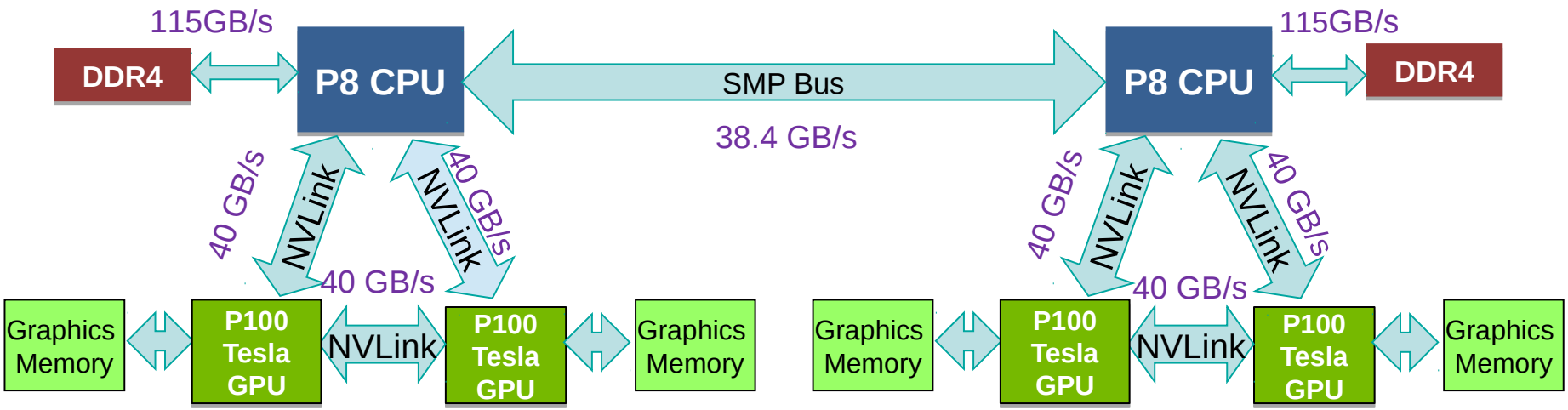
SOSCIP
SMART COMPUTING FOR INNOVATION

NVIDIA P100 GPU



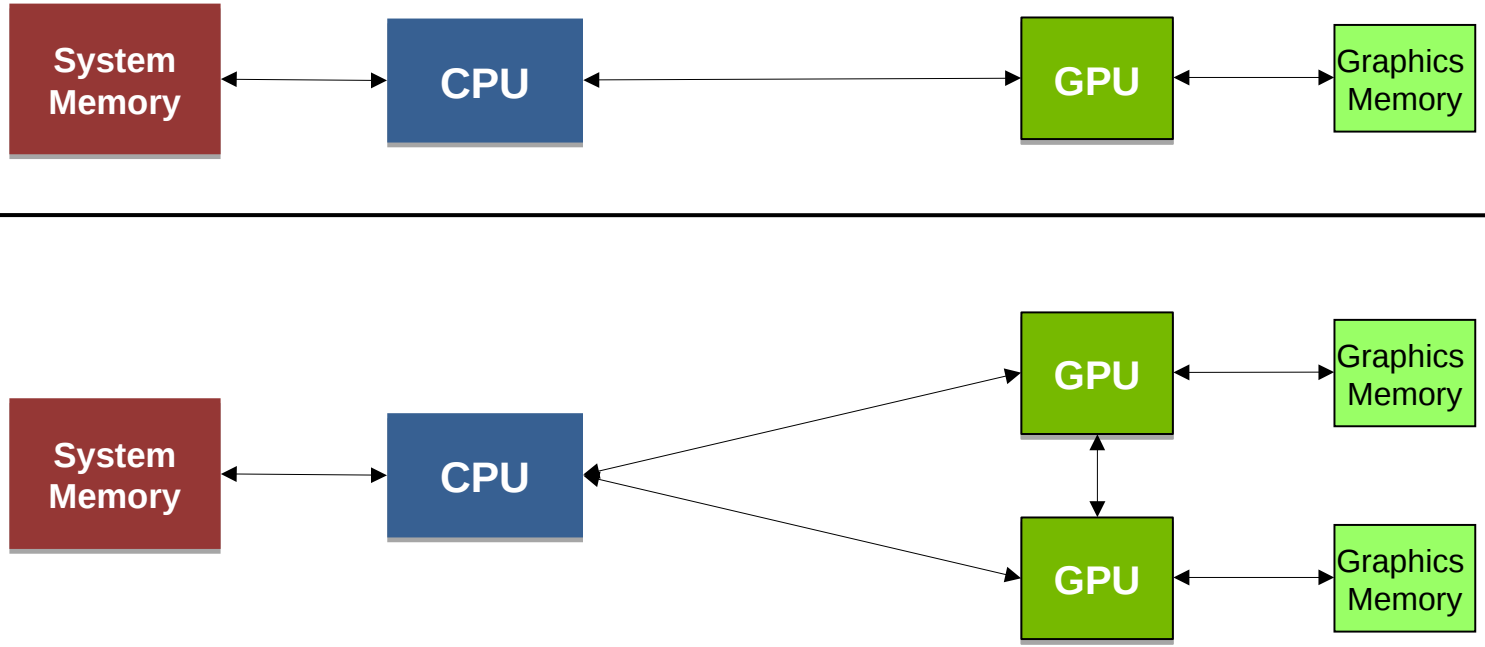
Cores	3584
Memory	16 GB HBM2
Memory Bandwidth	720 GB/s
FLOPS (sp)	10.6 TFLOPS
FLOPS (dp)	5.3 TFLOPS
FLOPS (hp)	21.2 TFLOPS
Power consumption	300 W
CUDA compute ability	6.0

S822LC Architecture



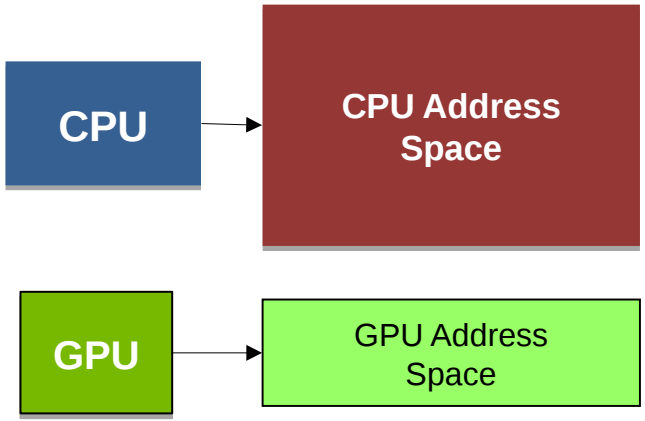
SCIP
SMART COMPUTING FOR INNOVATION

Interconnect is important

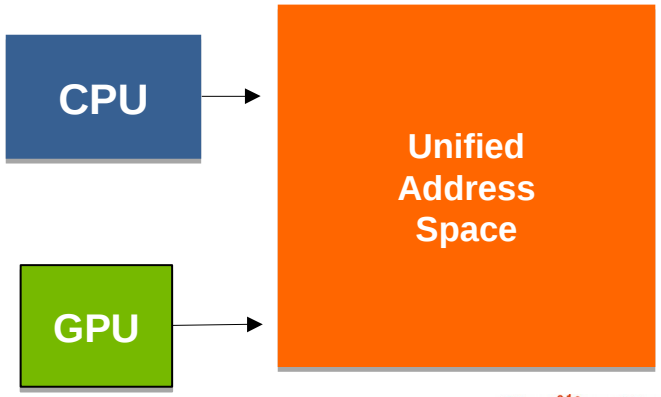


Unified Memory

Separate Address Spaces

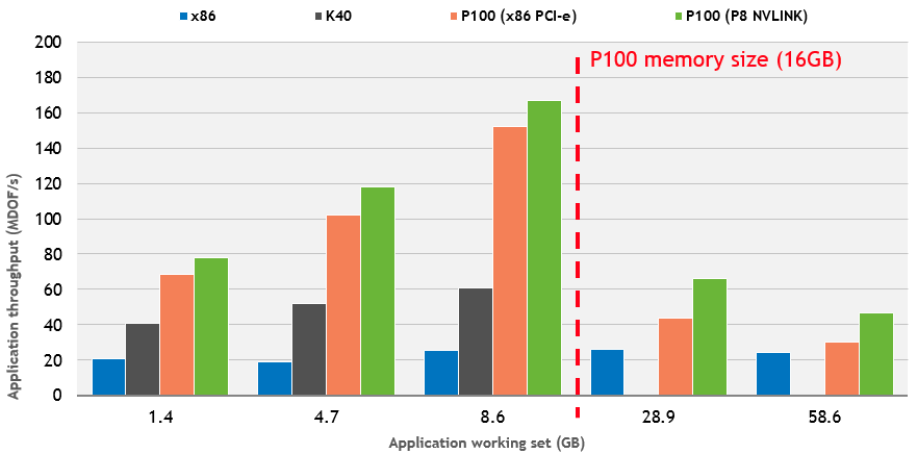


Unified Memory



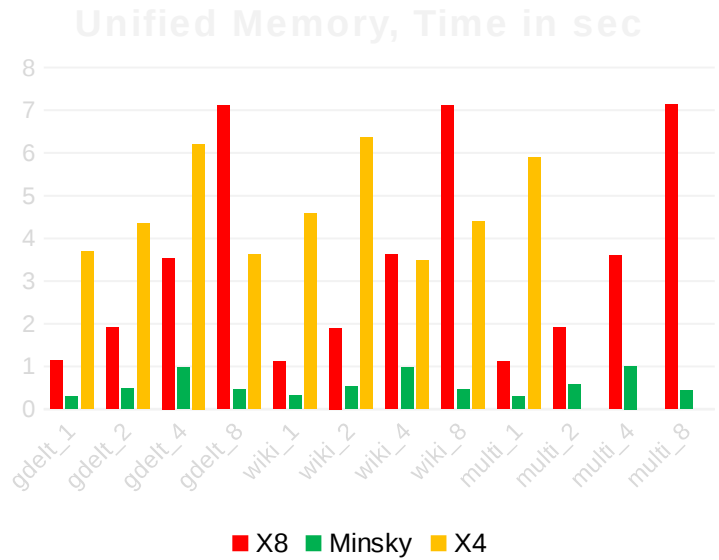
Performance

GMG AMR combustion simulation



[image source: N. Sakharnykh, nvidia.com, 2016]

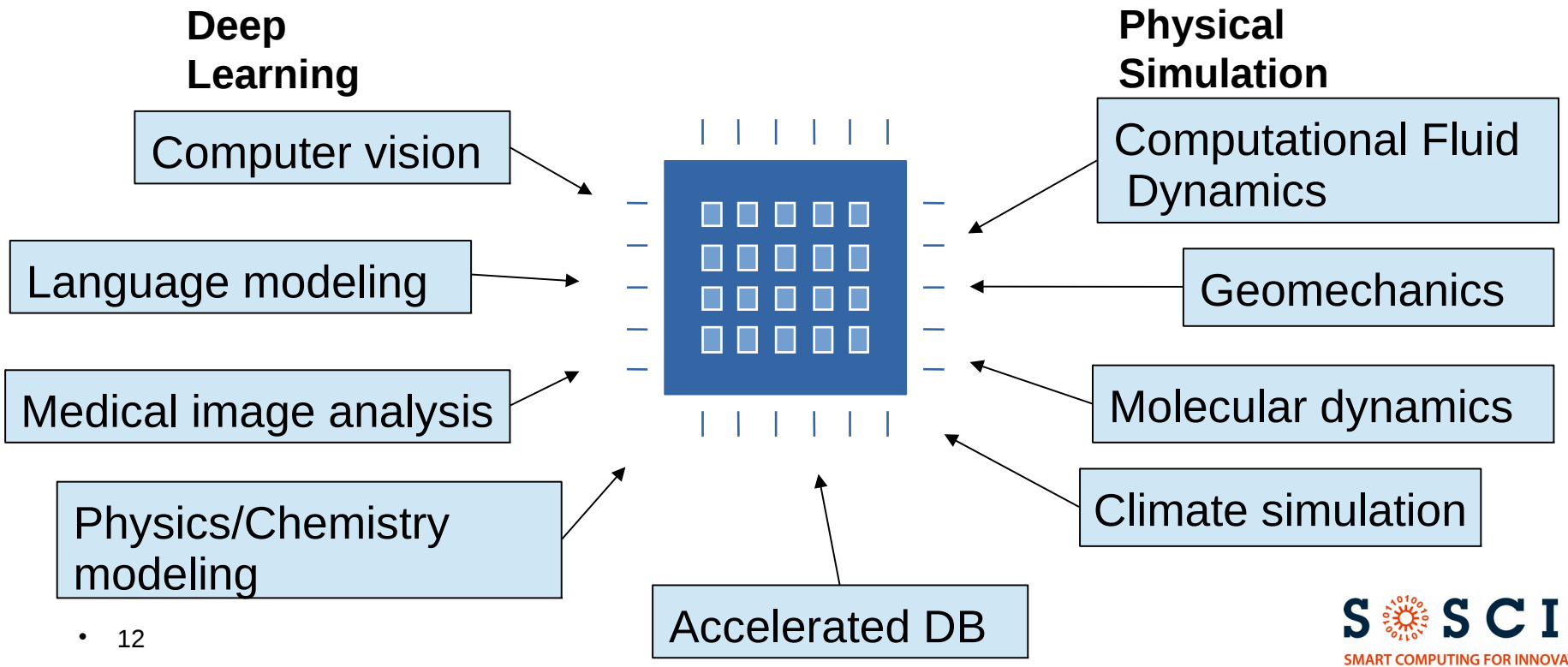
K-mean clustering in Spark MLlib



[image source: R. Bordewaker (IBM), GTC 2017]

SCIP
SMART COMPUTING FOR INNOVATION

Typical Applications



Available Software

Physics/Chemistry: GROMACS FAST. FLEXIBLE. FREE.

Deep Learning: PowerAI

Dev Tools: IBM XL C/C++/Fortran IBM Advanced Toolchain NVIDIA CUDA

Libraries: IBM MASS IBM ESSL/PESSL cuDNN

SCIP SMART COMPUTING FOR INNOVATION

Access and Login

User account => SciNet Account

Login process:

- `ssh <username>@bgqdev.scinet.utoronto.ca`
- Then, `ssh sgc01-ib0`

Submitting Jobs

Batch jobs

```
sbatch <myjob.script>
```

Interactive jobs

```
salloc --gres=gpu:4
```

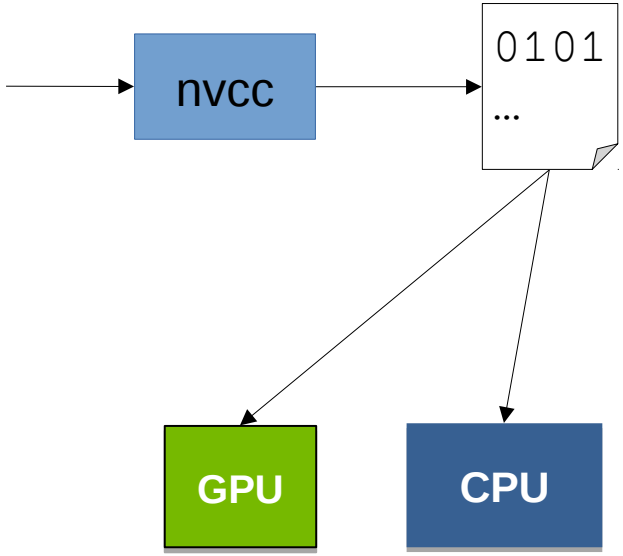
Job script

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=20  # MPI tasks (needed for srun)
#SBATCH --time=00:10:00  # H:M:S
#SBATCH --gres=gpu:4     # Ask for 4 GPUs per node

hostname
nvidia-smi

./vectorAdd
```

Programming GPUs

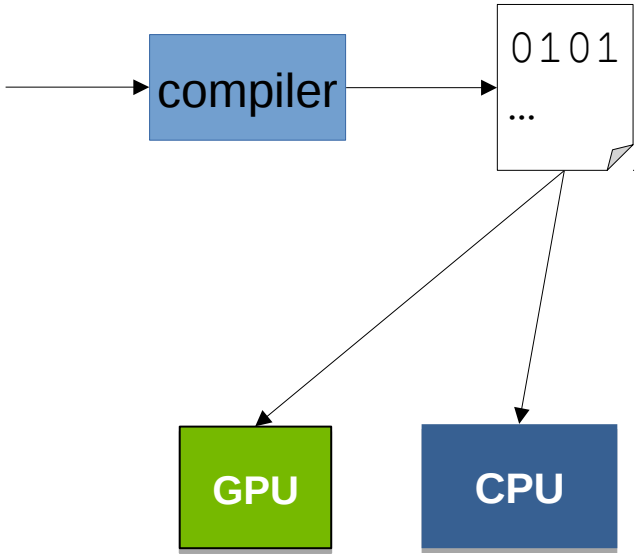
CUDA

```
// Device code
__global__ void VecAdd_kernel(const float *A,
           const float *B, float *C, int N)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < N)
        C[i] = A[i] + B[i];
}


// Host code
int main()
{
      float *a, *b, *c;
      cudaMallocManaged(&a, N*sizeof(float));
      ...
      VecAdd_kernel<<<1,N>>>(a, b, c, N);
      ...
      cudaFree(a);
      ...
}
```



OpenMP 4.5

```
void vectorAdd(     const float *x,
                    const float *y,
                    float *z,
                    int len)
{
  #pragma omp target map(to:x[:len]) map(to:y[:len]) map(from:z[:len])
  {
    #pragma omp parallel
    for (int i=0; i<len; i++)
    {
      z[i] = x[i] + y[i];
    }
  }
}
```



IBM XL Compilers

```
$> module load xlc
```


```
$> xlc -qsmp -qoffload ...
```


```
$> nvcc -ccbin xlC ...
```


SCIP
SMART COMPUTING FOR INNOVATION

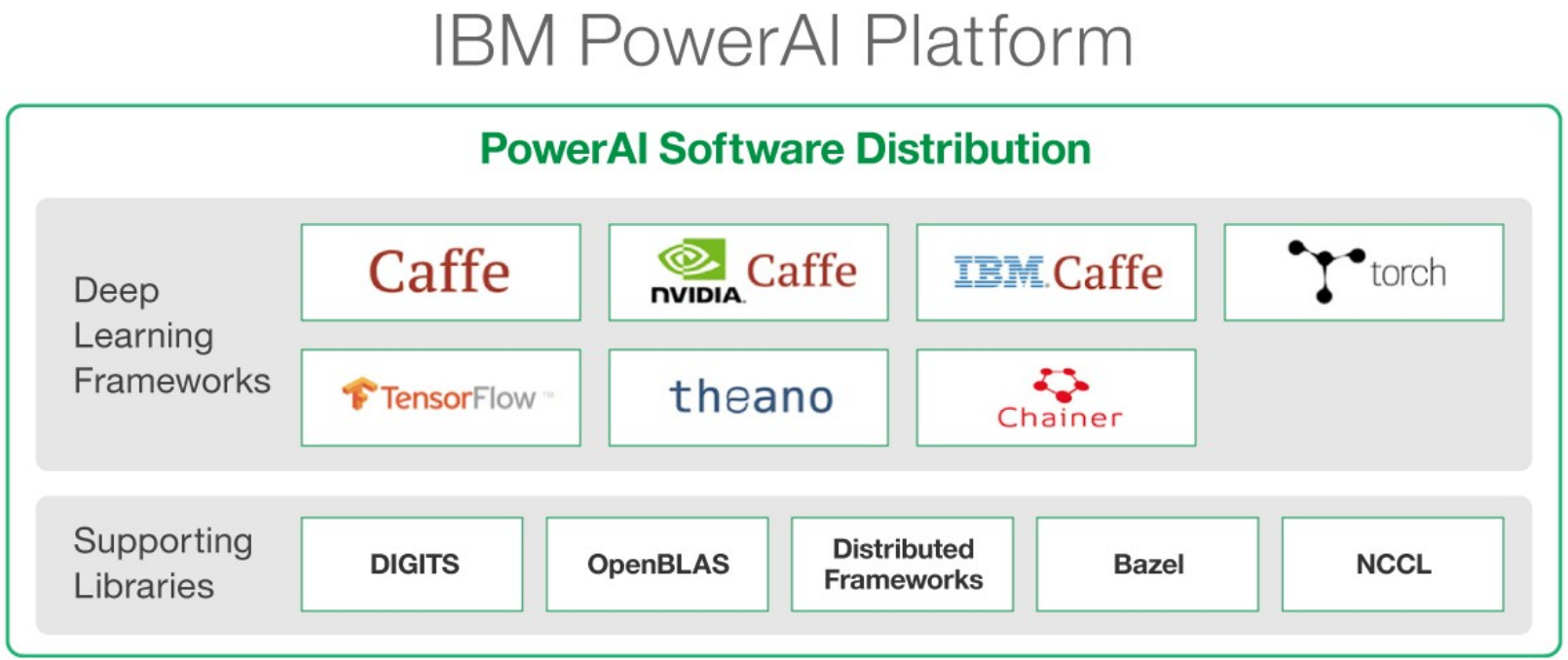
Using Multiple GPUs: CUDA

```
// Device code
__global__ void VecAdd_kernel(const float *A,
      const float *B, float *C, int N)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < N)
        C[i] = A[i] + B[i];
}


// Host code
int main()
{
      float *a[NUMGPU], *b[NUMGPU], *c[NUMGPU];
      ...
      cudaMallocManaged(&a[gpu], N*sizeof(float));
      ...
      cudaSetDevice(gpu);
      VecAdd_kernel<<<1,N>>>(a[gpu], b[gpu], c[gpu], N/NUMDEV);
      ...
      cudaFree(a[gpu]);
      ...
}
```

Using Multiple GPUs: OpenMP

```
void vectorAdd(     const float *x,
                    const float *y,
                    float *z,
                    int len)
{
  int num_gpu = omp_get_num_devices();
  int stride = len/num_gpu;


  for (int gpu=0; gpu < num_gpu; gpu++)
  {
    int begin = gpu*stride;
    int end = begin+stride-1;
    #pragma omp target device(gpu) \
        map(to:x[begin:end]) map(to:y[begin:end]) map(from:z[begin:end])
    {
      #pragma omp parallel
      for (int i=0; i<stride; i++)
      {
        z[i] = x[i] + y[i];
      }
    }
  }
}
```

PowerAI



Current release 4 includes IBM Distributed Deep Learning (DDL)!