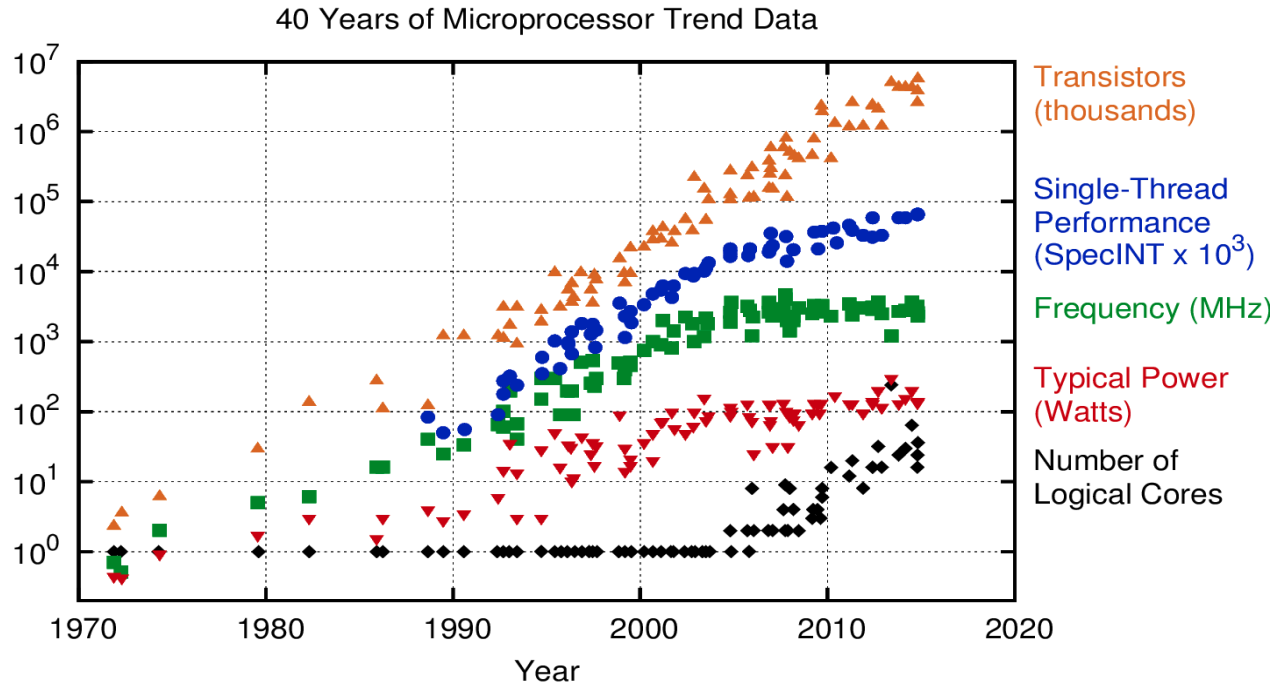


# SOSCIP GPU Platform

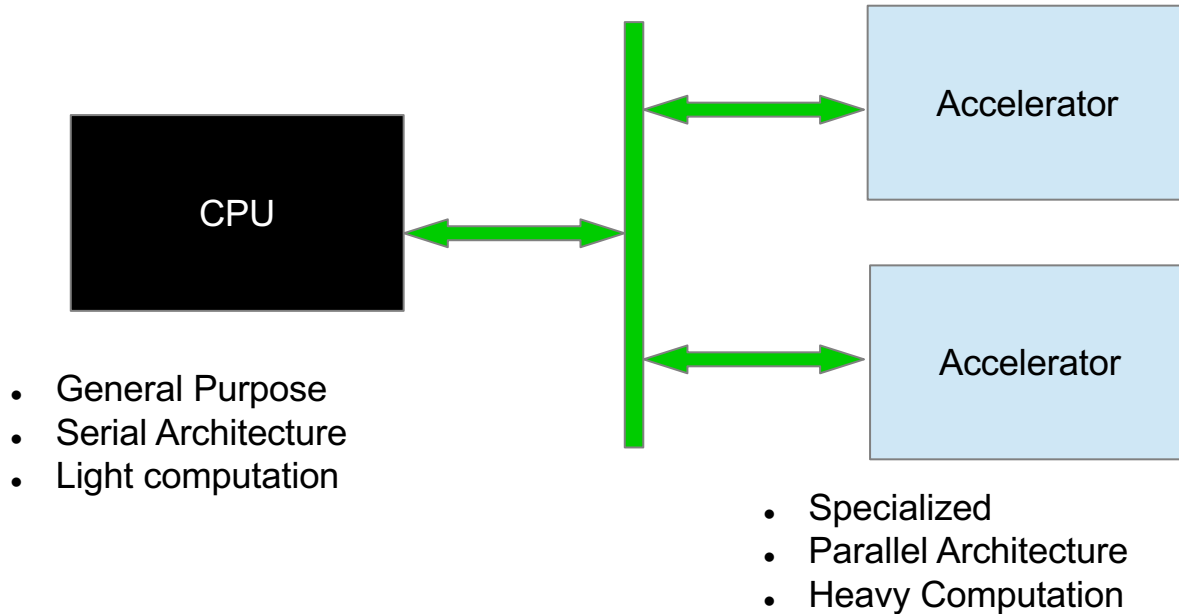


# The Problem

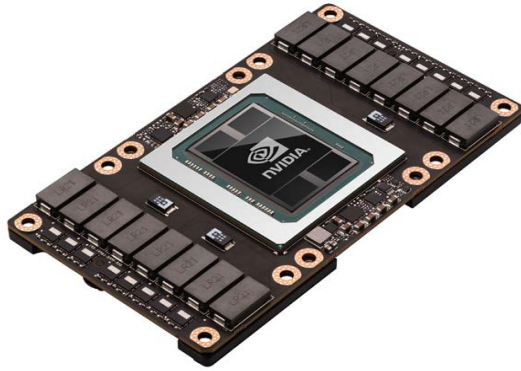


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten  
New plot and data collected for 2010-2015 by K. Rupp

# Heterogeneous Computing



# GPUs as Accelerators



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

# Agenda

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

# SOSCIP GPU Platform

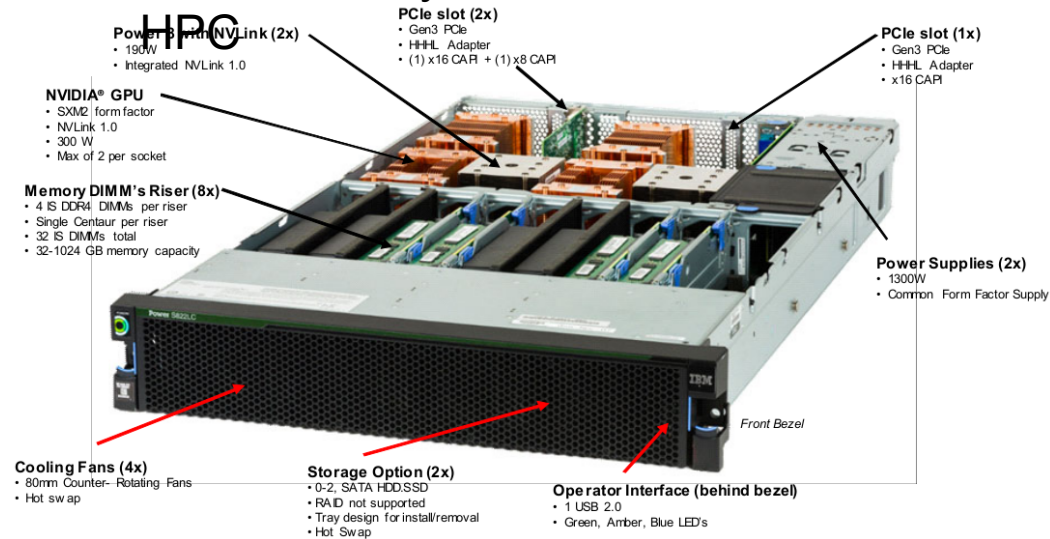


**System Rack** 1 x 7965-94Y  
**RDHX** 1 x 1164-95X

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

**Compute nodes** 14 x 8335-GTB

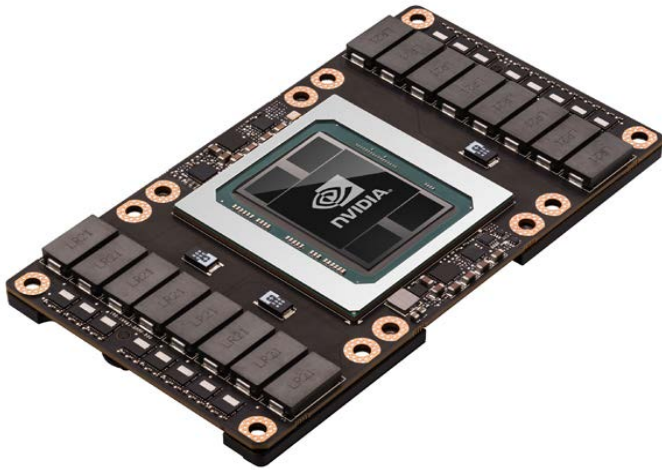
## IBM Power System S822LC for HPC



System TFLOPS (theoretical): 296 DP, 1187 HP

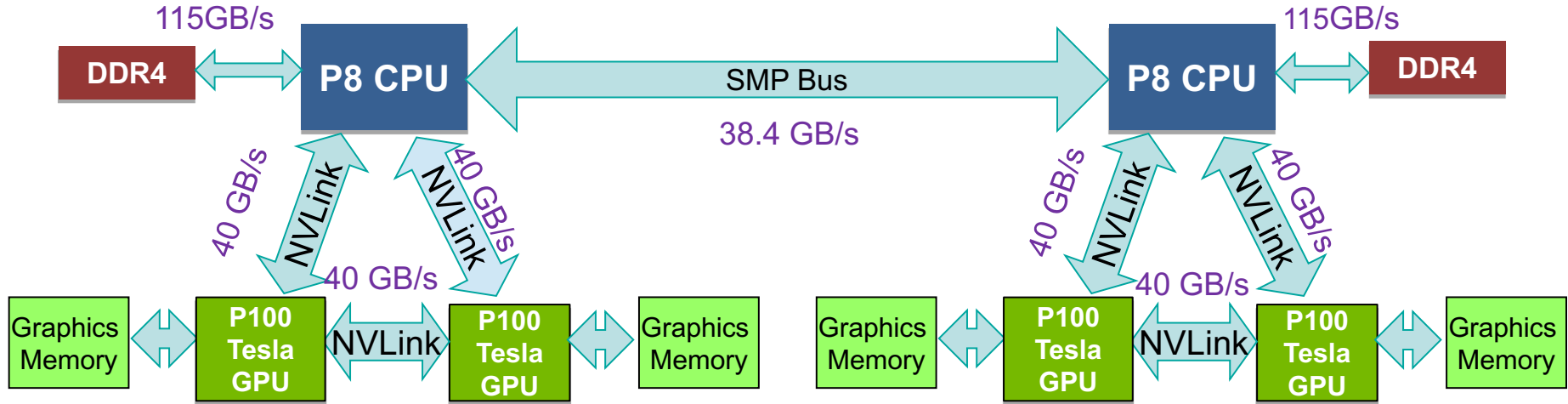


# NVIDIA P100 GPU



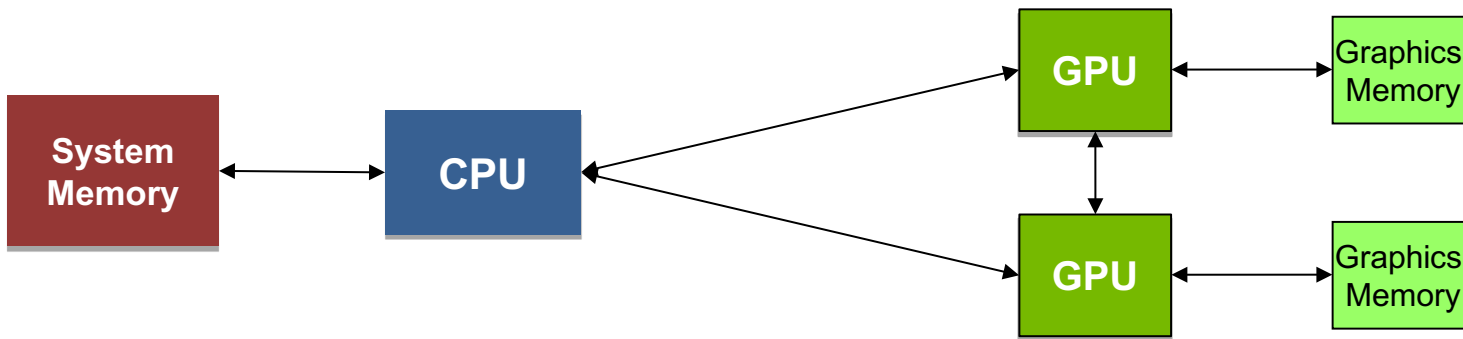
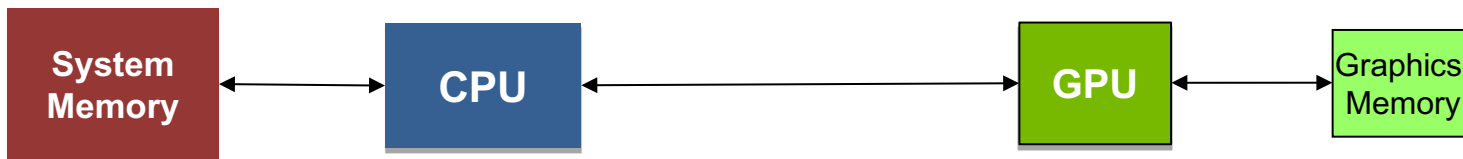
Cores	3584
Memory	16 GB HBM2
Memory Bandwidth	720 GB/s
FLOPS (sp)	10.6 TFLOPS
FLOPS (dp)	5.3 TFLOPS
FLOPS (hp)	21.2 TFLOPS
Power consumption	300 W
CUDA compute ability	6.0

# S822LC Architecture

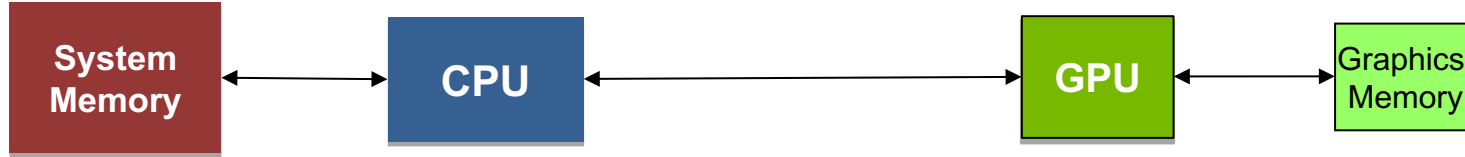




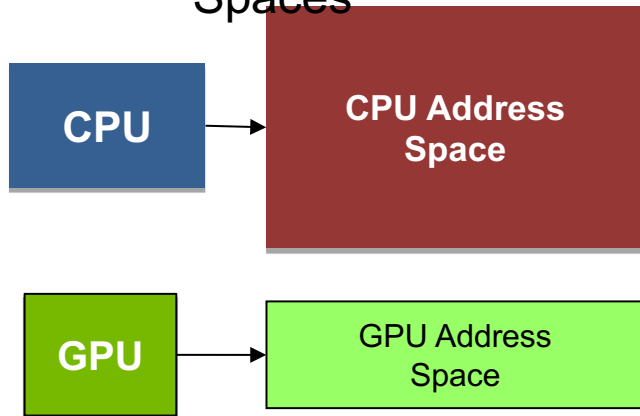
# Interconnect is important



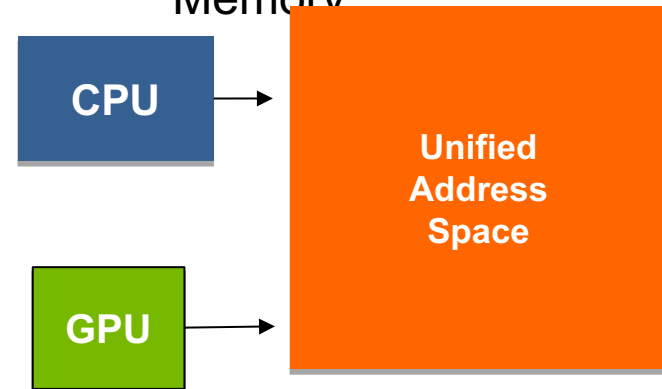
# Unified Memory



Separate Address Spaces

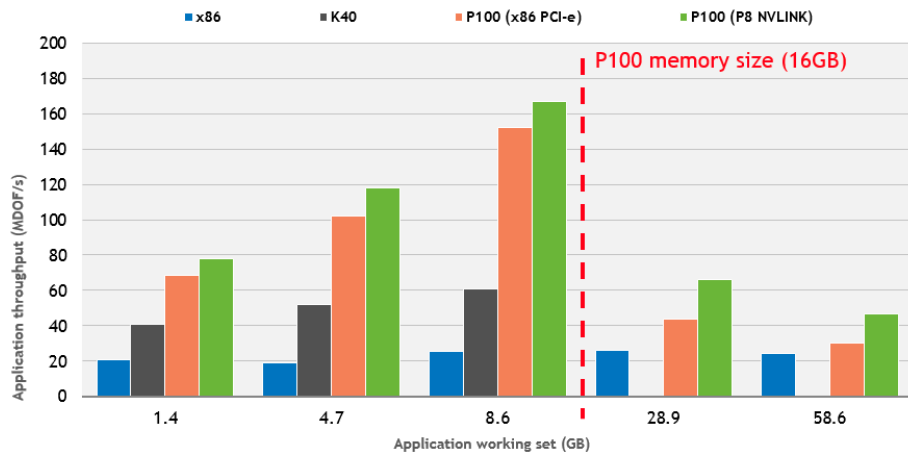


Unified Memory



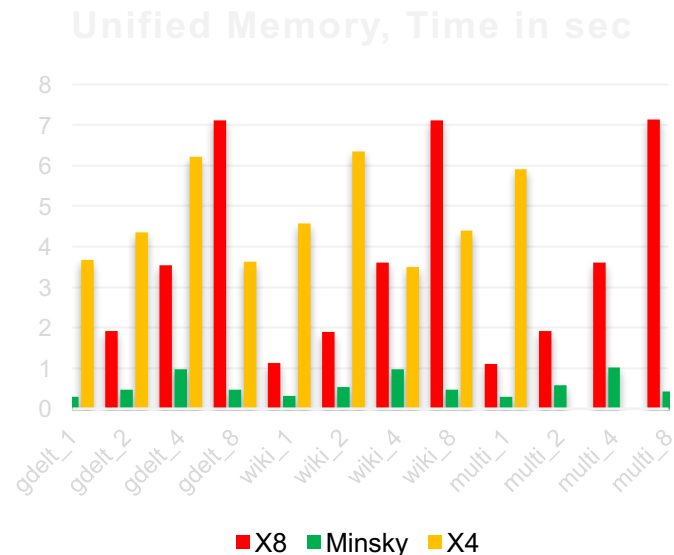
# Performance

## GMG AMR combustion simulation



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

## K-mean clustering in Spark MLlib



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

# Typical Applications

## Deep Learning

## Physical Simulation

Computer vision

Language modeling

Medical image analysis

Physics/Chemistry modeling

Computational Fluid Dynamics

Geomechanics

Molecular dynamics

Climate simulation

Accelerated DB

# Available Software

Physics/Chemistry:

**GROMACS**  
FAST. FLEXIBLE. FREE.



Deep Learning:

PowerAI

Dev Tools:

IBM XL C/C++/Fortran

IBM Advanced Toolchain



Libraries:

IBM MASS

IBM ESSL/PESSL



# Access and Login

User account => SciNet Account

Login process:

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

# Submitting Jobs

## Batch jobs

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

## Interactive jobs

```
salloc <myjob.script>
```

## Job script

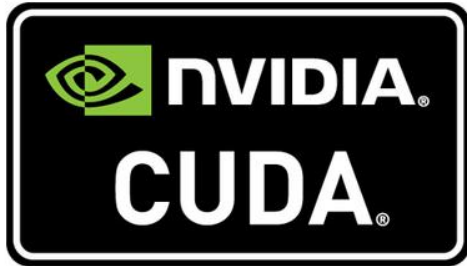
```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=20 # MPI tasks (needed for srun)
#SBATCH --time=00:10:00 # H:M:S
#SBATCH --gres=gpu:4 # Ask for 4 GPUs per node

hostname
nvidia-smi

./vectorAdd
```



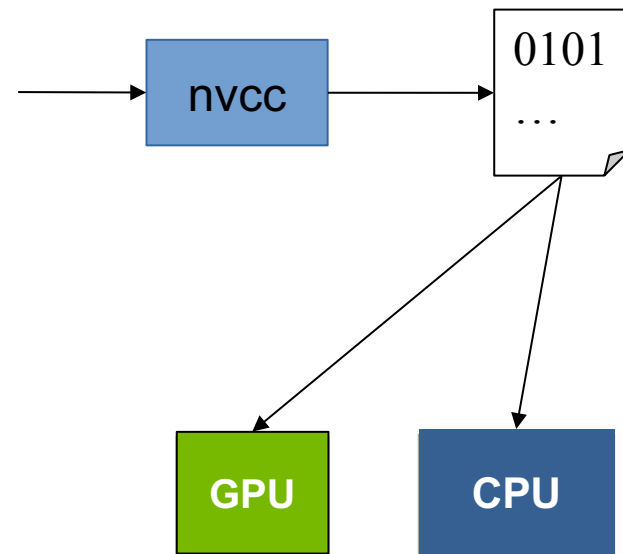
# Programming GPUs



# CUDA

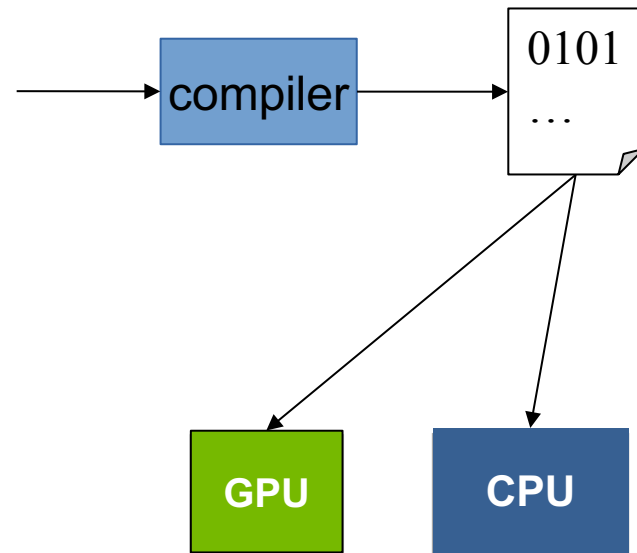
```
// Device code
__global__ void VecAdd_kernel(const float *A,
                             const float *B, float *C, int N)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < N)
        C[i] = A[i] + B[i];
}

// 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 ...
```

# 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]);
    ...
}
```

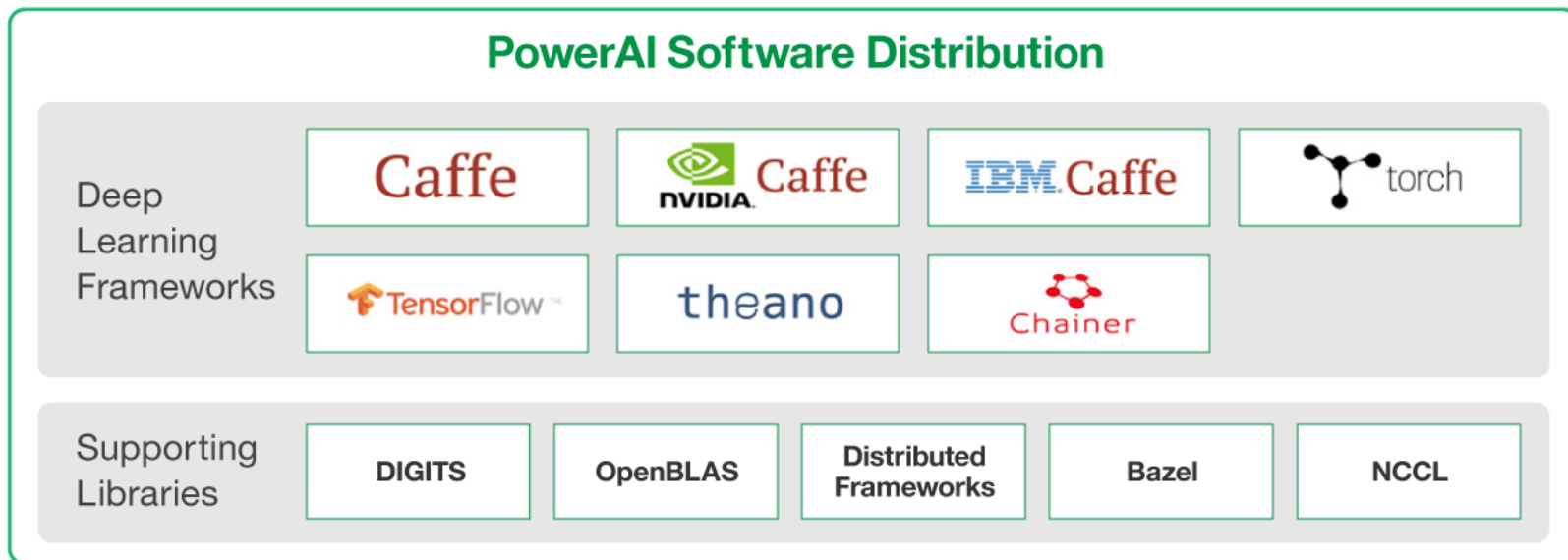
# Using Multiple GPUs: OpenMP

```
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

## IBM PowerAI Platform



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