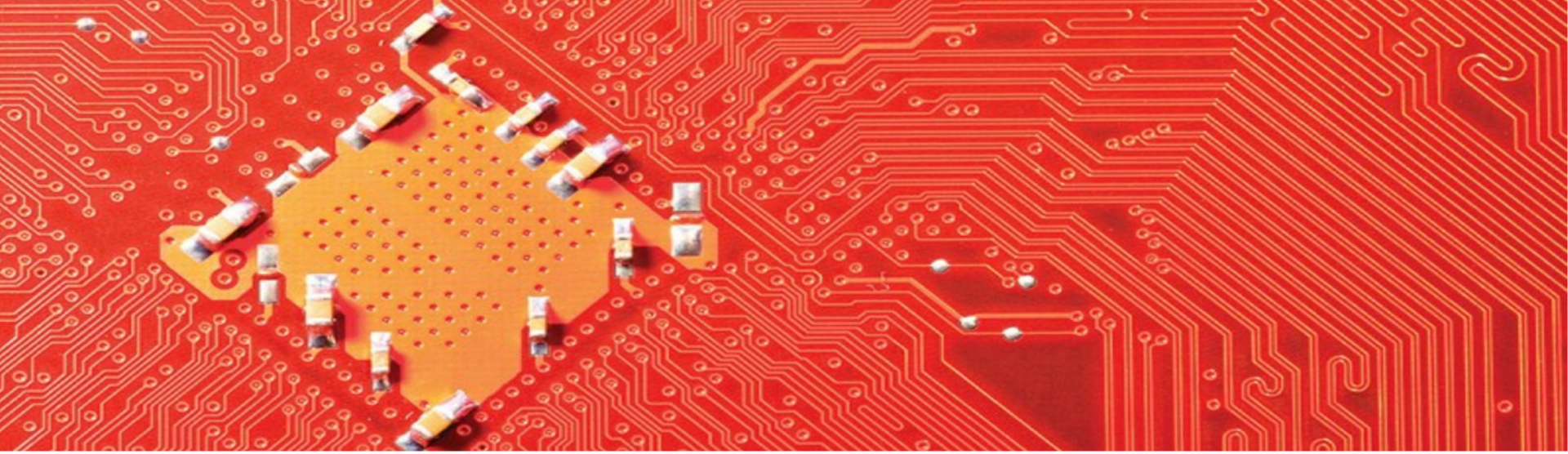
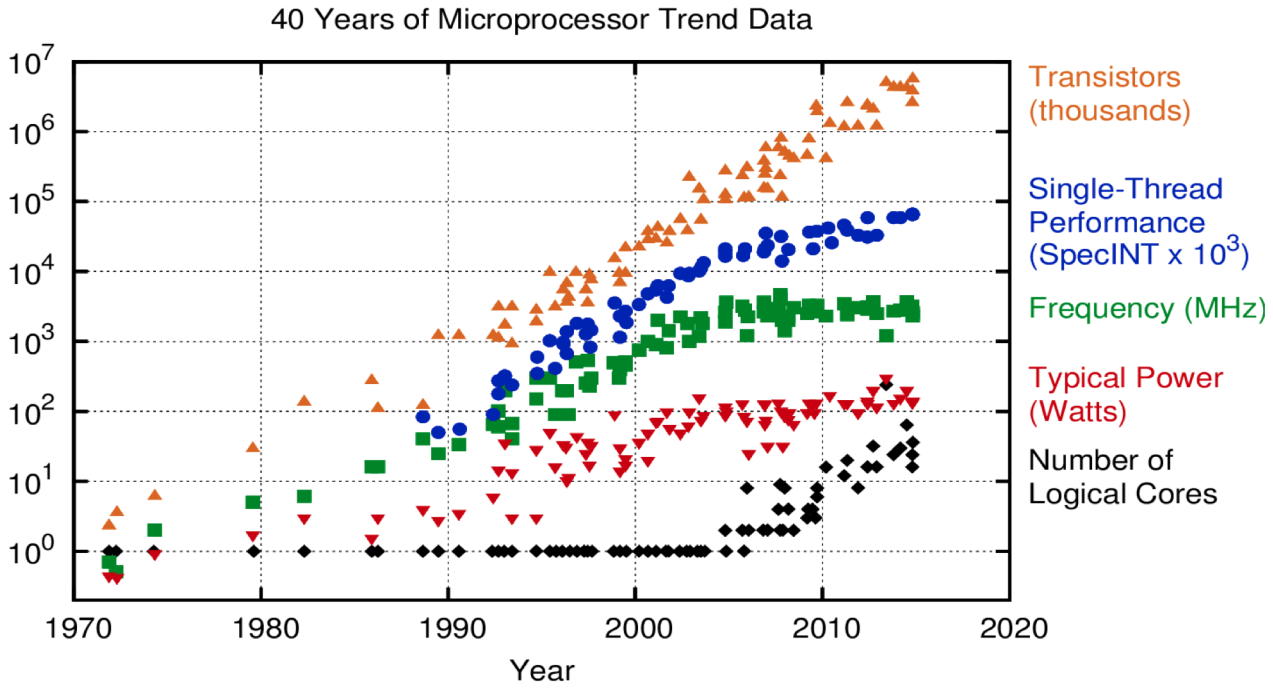


SOSCIP GPU Platform

SOSCIP
SMART COMPUTING FOR INNOVATION

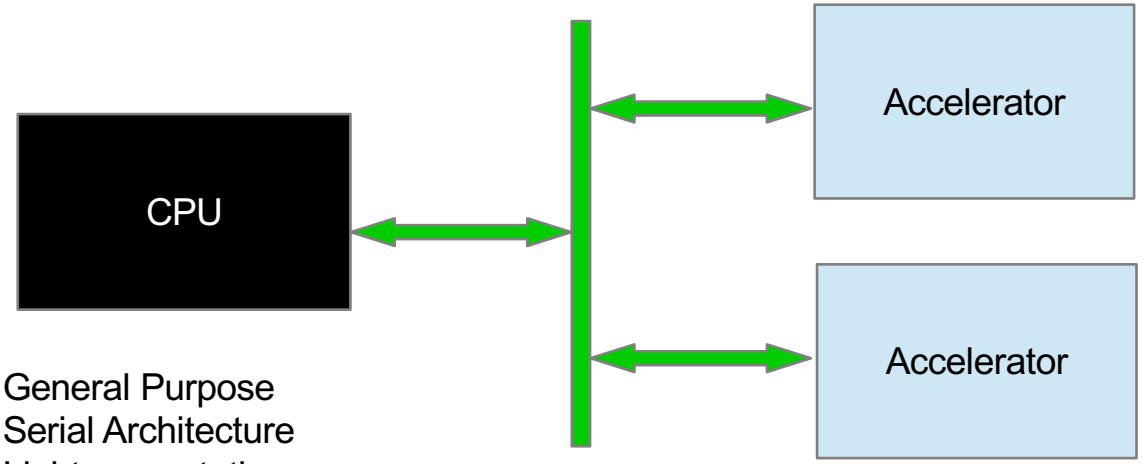
The Problem



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

SCIP
SMART COMPUTING FOR INNOVATION

Heterogeneous Computing



- General Purpose
- Serial Architecture
- Light computation

- Specialized
- Parallel Architecture
- Heavy Computation

GPUs as Accelerators

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

Agenda

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

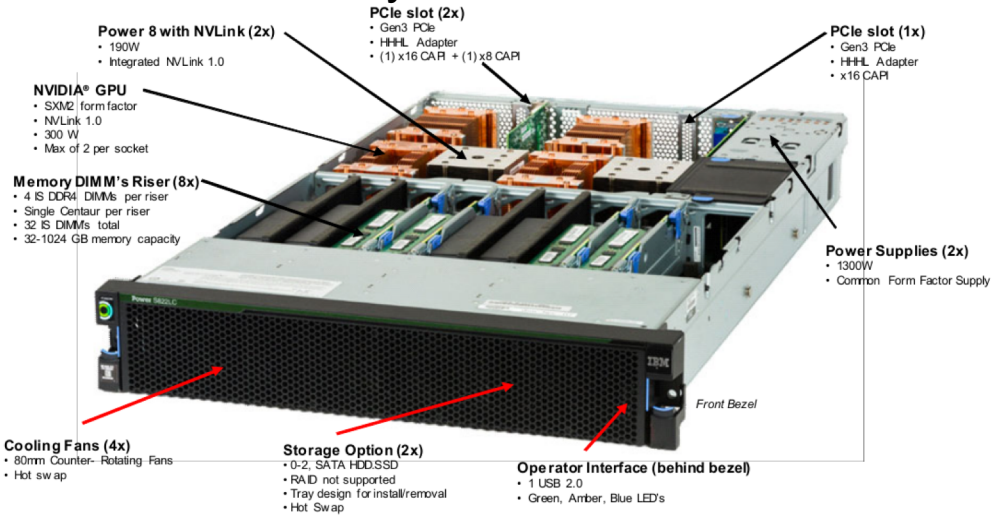
SOSCIP GPU Platform

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

15 x IBM Power System S822LC for HPC

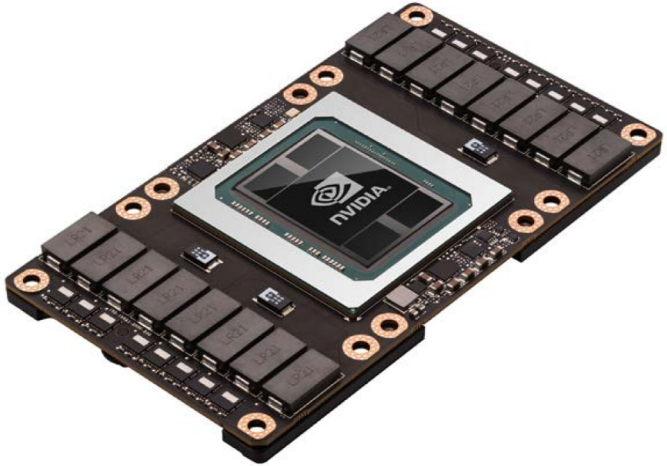


System TFLOPS (theoretical): 318 DP, 1272 HP

- 2x10 cores per node (8 SMT threads per core)
- 512 GB CPU RAM
- 4 x P100 GPUs per node

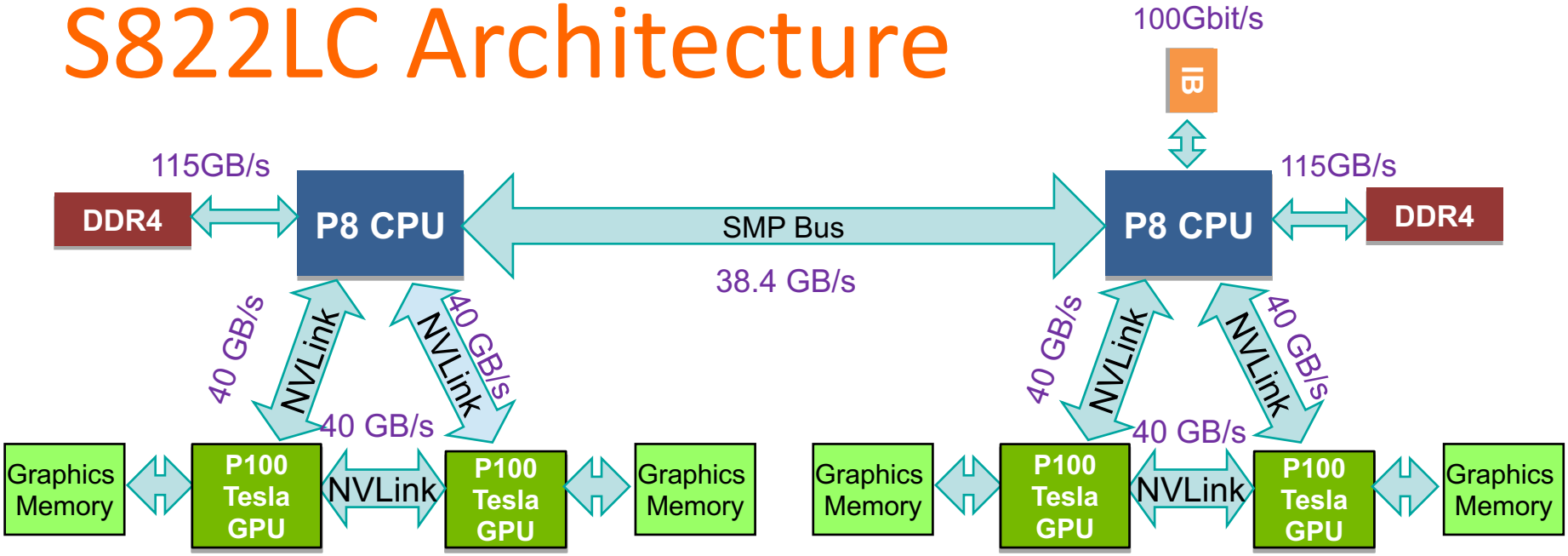
SOSCIP
SMART COMPUTING FOR INNOVATION

NVIDIA P100 GPU



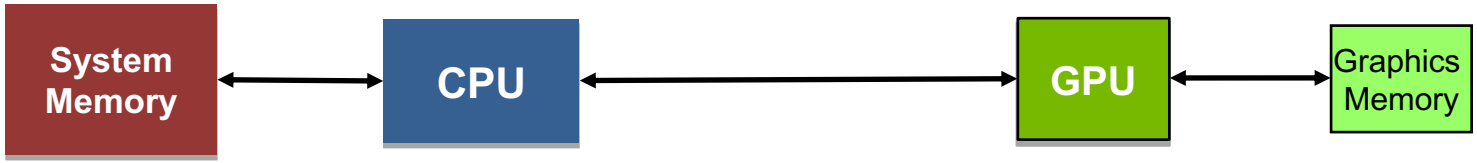
Cores	3584
Memory	16 GB HBM2
Memory Bandwidth	720 GB/s
FLOPS (sp)	10.6 TFLOPS
FLOPS (dp)	5.3 TFLOPS
FLOPS (hp)	21.2 TFLOPS
Power consumption	300 W
CUDA compute ability	6.0

S822LC Architecture

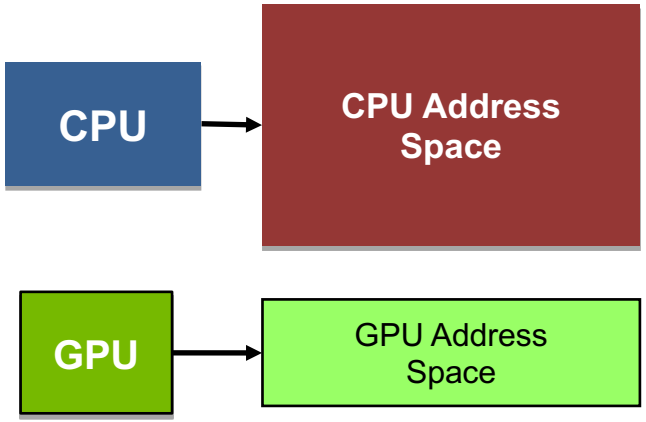


SCIP
SMART COMPUTING FOR INNOVATION

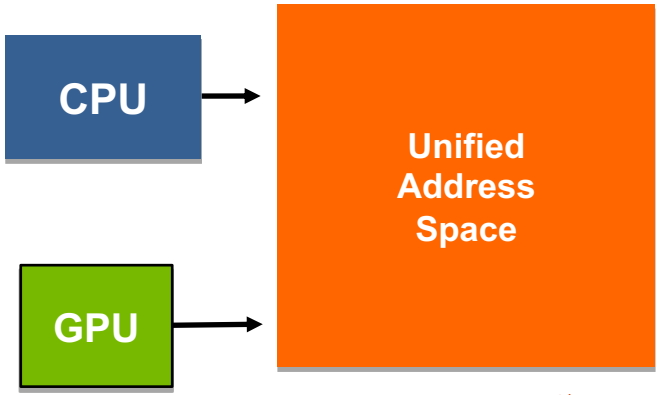
Unified Memory



Separate Address Spaces



Unified Memory

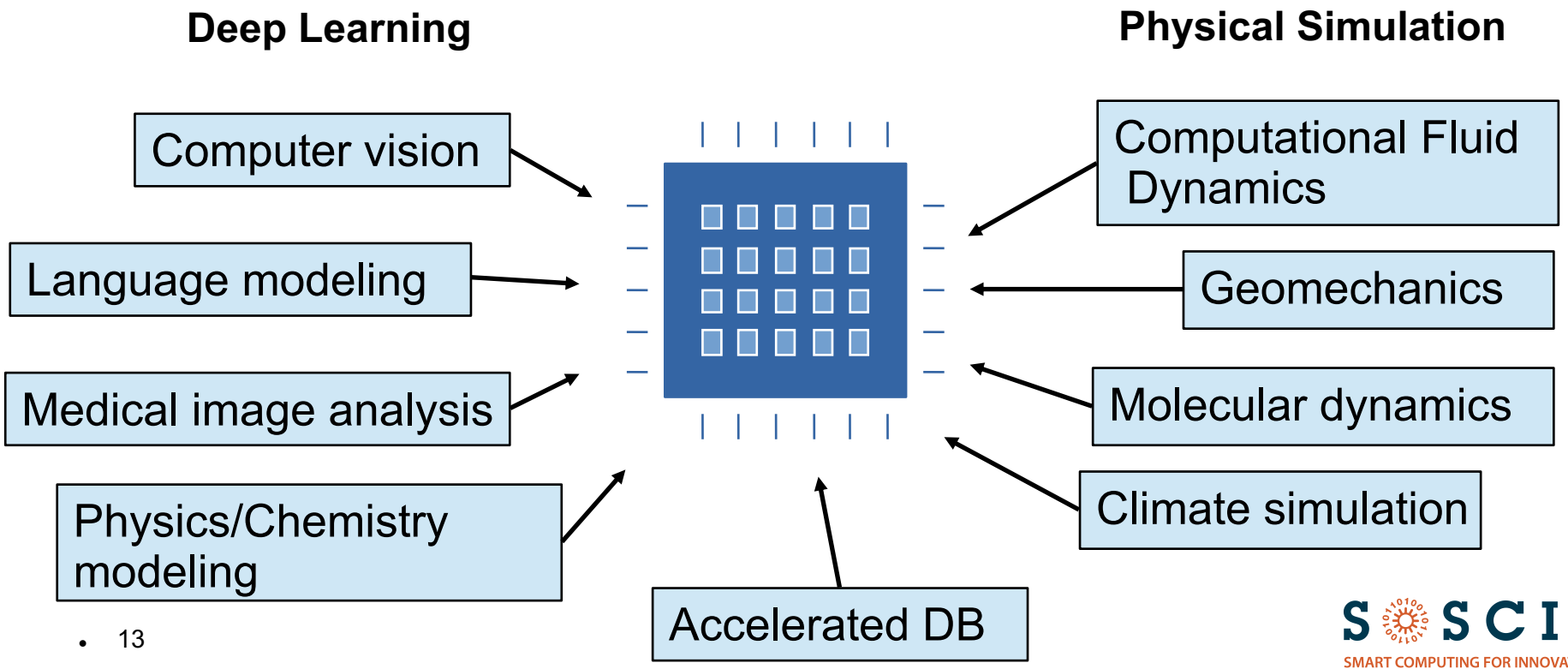


Filesystem

- Shared with Blue Gene/Q which has dedicated 500TB filesystem based on GPFS

file system	Purpose	user quota	backed up	purged	environment variable
/home	development	50 GB	Yes	Never	$HOME
/scratch	computation	20TB/ 1M files	No	not currently	$SCRATCH

Typical Applications



Available Software

Physics/Chemistry:

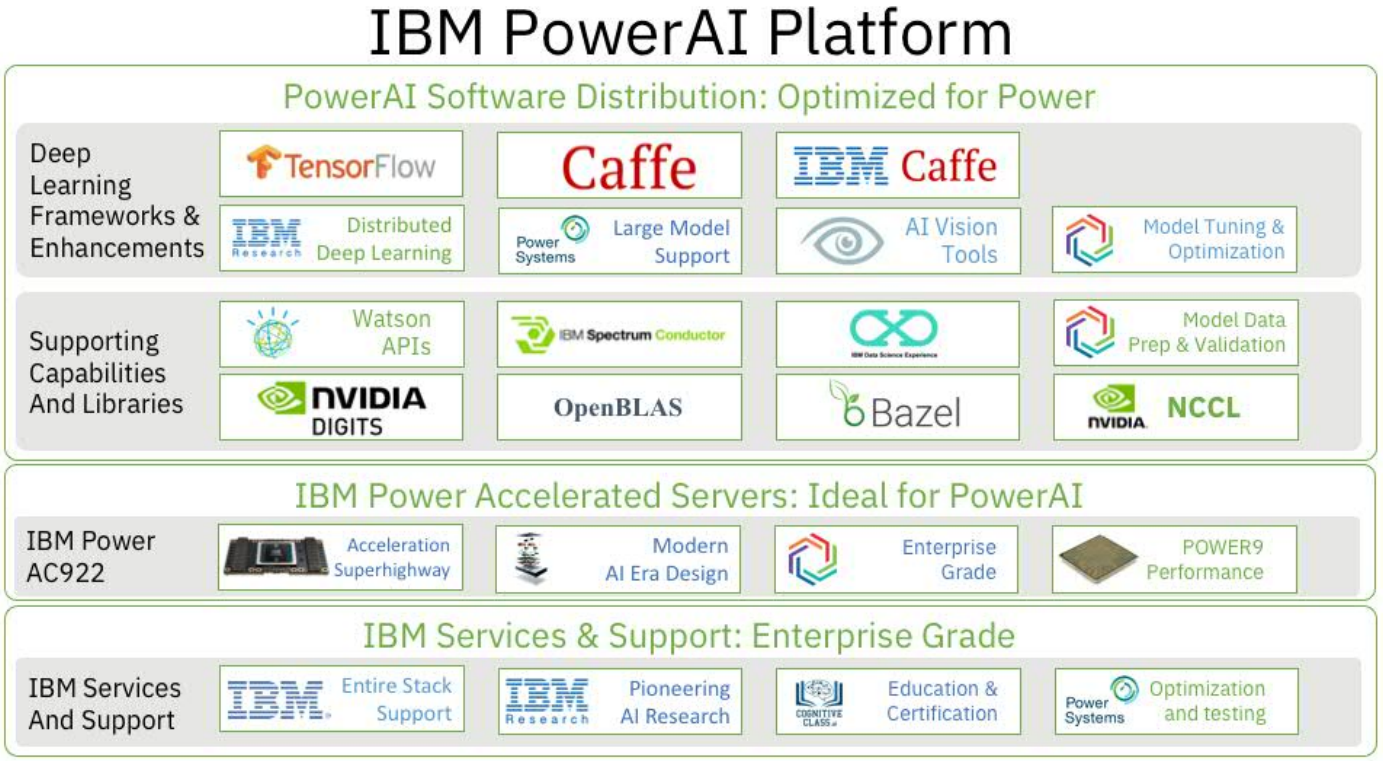
Deep Learning:

Dev Tools: IBM XL C/C++/Fortran IBM Advance Toolchain

Libraries: IBM MASS IBM ESSL/PESSL

SCIP
SMART COMPUTING FOR INNOVATION

PowerAI



Getting Help

Contact support team:

Email ***soscip-support@scinet.utoronto.ca***

Help Wiki page:

https://docs.scinet.utoronto.ca/index.php/SOSCIP_GPU

Access and Login

User account => SciNet Account

Login process:

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

Submitting Jobs to SLURM

Dev/test jobs

- Run interactively at login node (sgc01) for a short time
- Shared with all users
- Use CUDA_VISIBLE_DEVICES to choose empty GPU(s)

Batch jobs

- sbatch <myjob.script>
- Add “#SBATCH -p long” for jobs > 12h

Interactive jobs

- salloc --gres=gpu:4
  (Long dev/test job)

Job script (myjob.script)

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


hostname
nvidia-smi

module load cuda/9.2            #software installed as module
./vectorAdd
```

Monitoring/cancelling Jobs

squeue -u <username>

- Show all the user jobs the scheduler is managing at the moment
- squeue -u <username> -t RUNNING
- squeue -u <username> -t PENDING

scancel <jobid>

- scancel -u <username>
- scancel -t PENDING -u <username>

Scheduling policies

Limits:

- Max 8 jobs/8 nodes each project

By-node job only

- For each job, you have 4 GPUs available to use for each node
- Pack 4 (or more) single-GPU sub jobs into 1 SLURM job (check wiki page for instructions)

Multifactor Priority

- Equal Fair-share for each project
- Fair-share is the main factor to calculate priority

Check queue priority

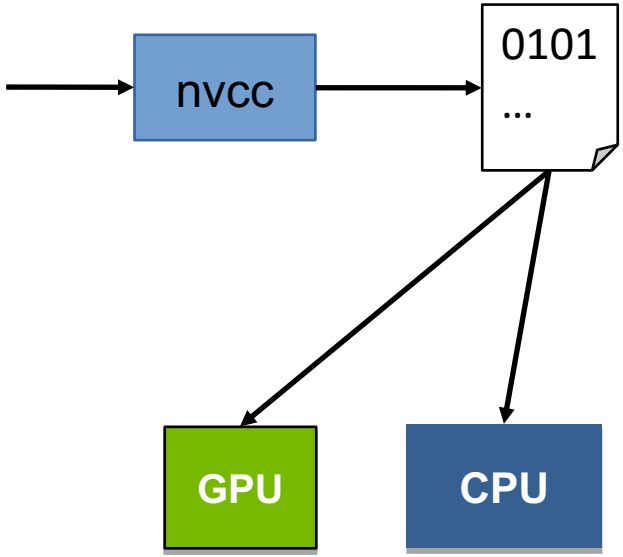
- squeue -S p (highest priority job at the bottom)

Programming GPUs

CUDA

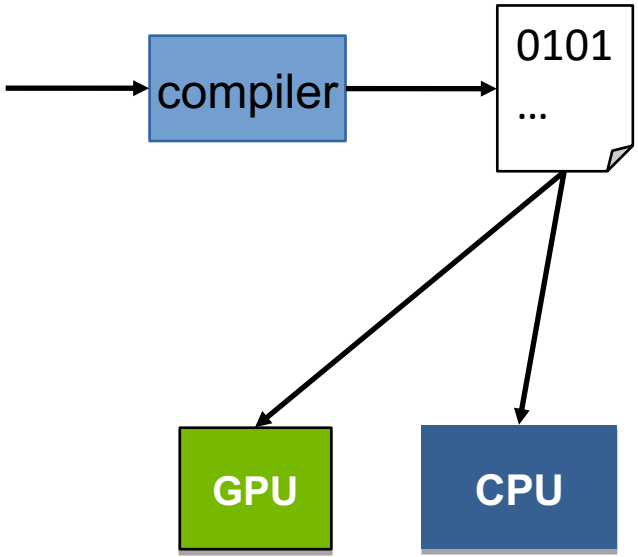
```
// Device code
__global__ void VecAdd_kernel(const float *A,
                              const float *B, float *C, int N)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < N)
        C[i] = A[i] + B[i];
}


// 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
    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)); //this is to use unified memory
                        ...
                        cudaSetDevice(gpu);
                        cudaStreamCreate(&stream);
                        VecAdd_kernel<<<1,N/NUMDEV,0,stream>>>(a[gpu], b[gpu], c[gpu], N/NUMDEV);
                        ...
                        cudaFree(a[gpu]);
                        ...
}
```

Using Multiple GPUs: OpenMP offload

```
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
      for (int i=0; i<stride; i++)
      {
        z[i] = x[i] + y[i];
      }
    }
  }
}
```