

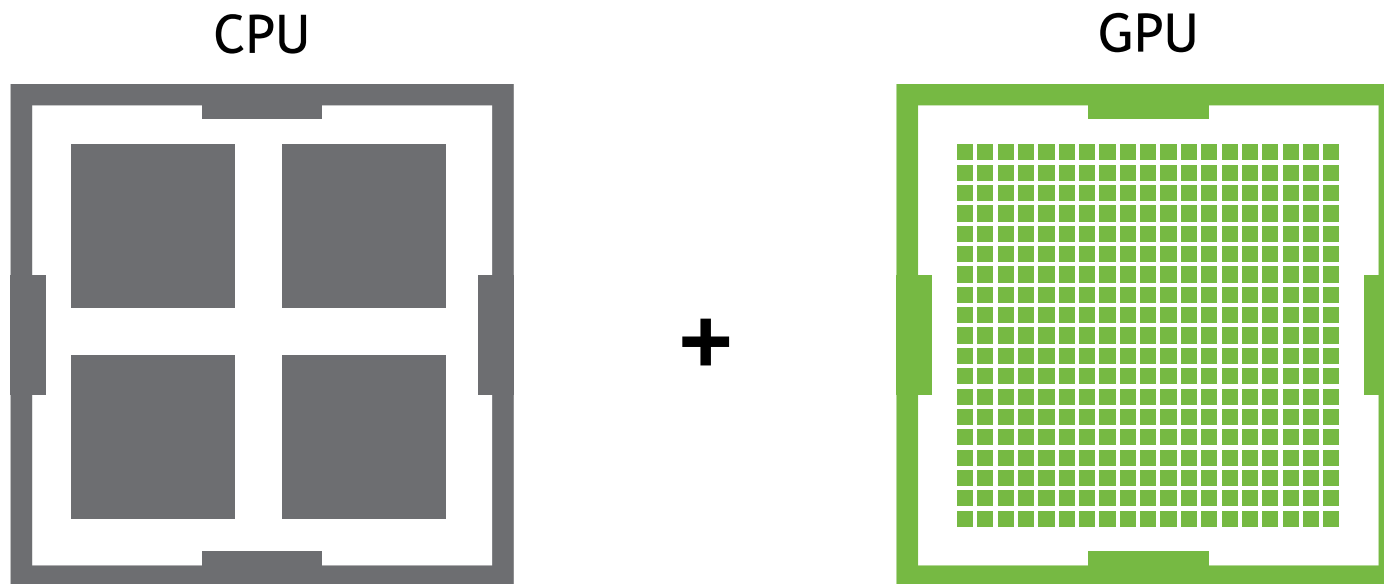


LECTURE ON VOLTA GPU ARCHITECTURE

Mathias Wagner, November 18th 2019

ADD GPUS

Accelerate science applications



ADD GPUS

Accelerate science applications

CPU



CPU Strengths

- Very large main memory
- Very fast clock speeds
- Latency optimized via large caches
- Small number of threads can run very quickly

+

CPU Weaknesses

- Relatively low memory bandwidth
- Cache misses very costly
- Low performance per watt

ADD GPUS

Accelerate science applications

GPU Strengths

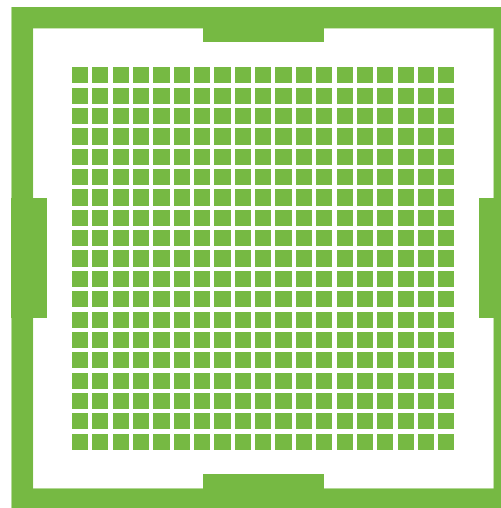
- High bandwidth main memory
- Latency tolerant via parallelism
- Significantly more compute resources
- High throughput
- High performance per watt

+

GPU Weaknesses

- Relatively low memory capacity
- Low per-thread performance

GPU



SPEED VS THROUGHPUT

Speed

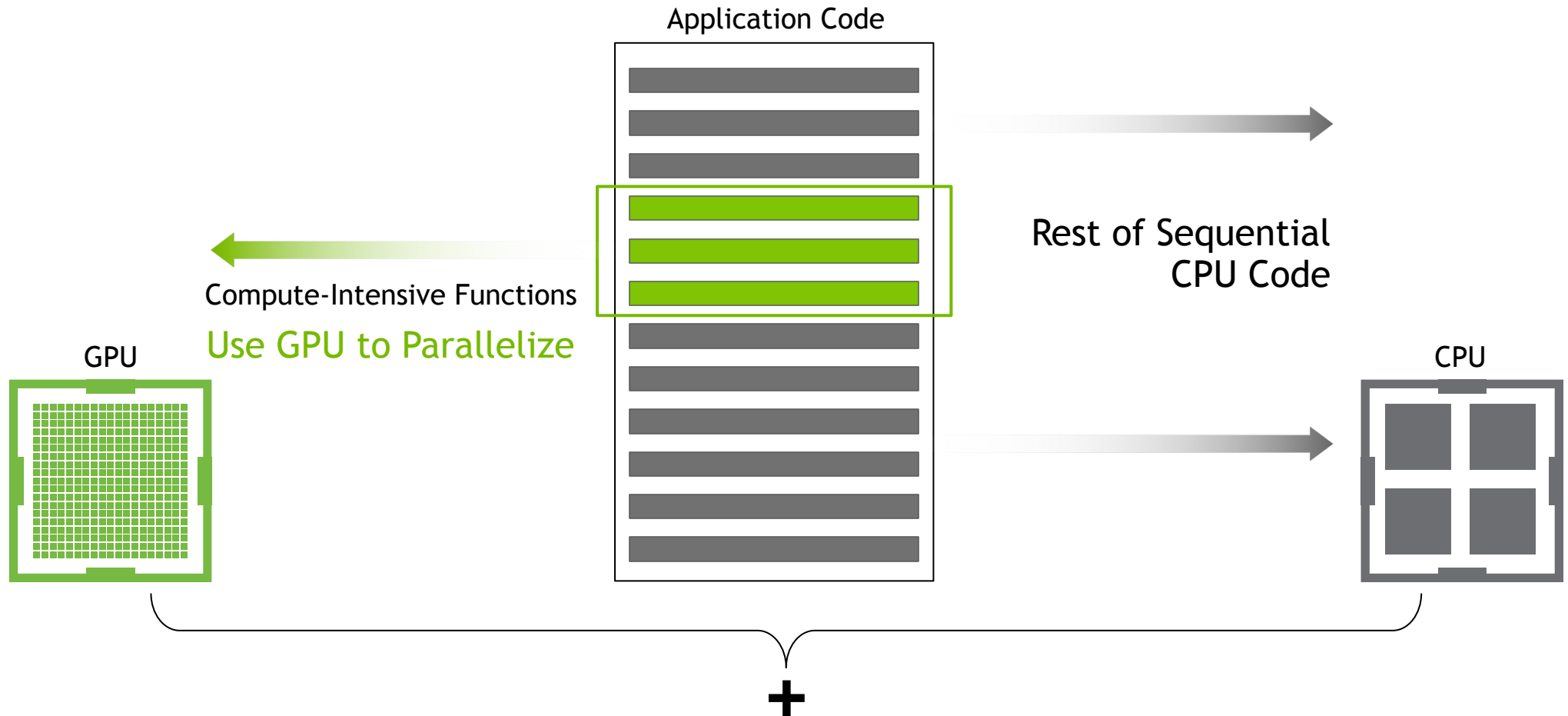


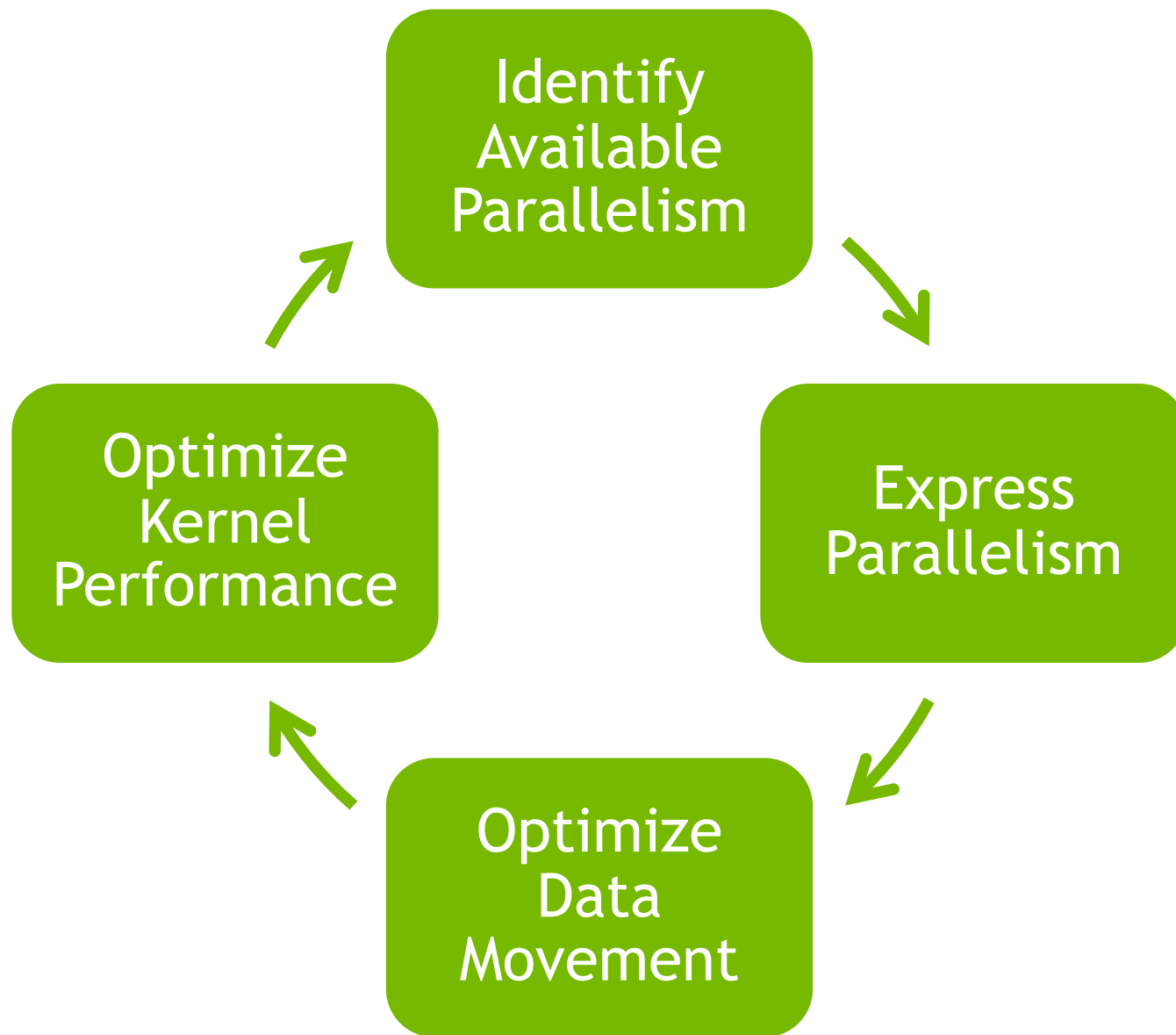
Throughput



Choose the right tool for the job

SMALL CHANGES, BIG SPEED-UP





3 WAYS TO ACCELERATE APPLICATIONS

Applications

Libraries

“Drop-in” Acceleration

OpenACC Directives

Easily Accelerate Applications

Programming
Languages

Maximum Flexibility

OPENACC DIRECTIVES

Manage
Data
Movement

Initiate
Parallel
Execution

Optimize
Loop
Mappings

```
#pragma acc data copyin(x,y) copyout(z)
{
    #pragma acc parallel
    {
        #pragma acc loop gang vector
        for (i = 0; i < n; ++i) {
            z[i] = x[i] + y[i];
            ...
        }
    }
    ...
}
```

OpenACC
Directives for Accelerators

Incremental

Single source

Interoperable

Performance portable

CPU, GPU, MIC

The background is a dark blue field with a complex network of thin, light green lines crisscrossing across it. At various points where these lines intersect or terminate, there are small, bright green circular dots. Some of these dots have a soft, out-of-focus glow around them. The overall effect is one of a dynamic, interconnected system, possibly representing a network or data flow.

EXPRESSING PARALLELISM

TESLA V100

21B transistors
815 mm²

80 SM
5120 CUDA Cores
640 Tensor Cores

16/32 GB HBM2
900 GB/s HBM2
300 GB/s NVLink



*full GV100 chip contains 84 SMs

VOLTA GV100 SM

	GV100
FP32 units	64
FP64 units	32
INT32 units	64
Tensor Cores	8
Register File	256 KB
Unified L1/Shared memory	128 KB
Active Threads	2048



OPENACC DIRECTIVES

Express Parallelism

```
real* restrict const A = (real*) malloc(nx*ny*sizeof(real));  
real* restrict const Aref = (real*) malloc(nx*ny*sizeof(real));  
...  
#pragma acc parallel loop  
for (int iy = iy_start; iy < iy_end; iy++) {  
    for( int ix = ix_start; ix < ix_end; ix++ ) {  
        A[iy*nx+ix] = Anew[iy*nx+ix];  
    }  
}
```

An abstract network diagram with green nodes and lines on a dark background. The nodes are represented by small, glowing green circles of varying sizes, and the lines are thin, green, semi-transparent lines connecting the nodes in a complex, web-like pattern. The background is a dark, almost black, gradient with some subtle blue and green bokeh effects.

OPTIMIZING DATA LOCALITY

NVLINK - GPU CLUSTER

Two fully connected quads,
connected at corners

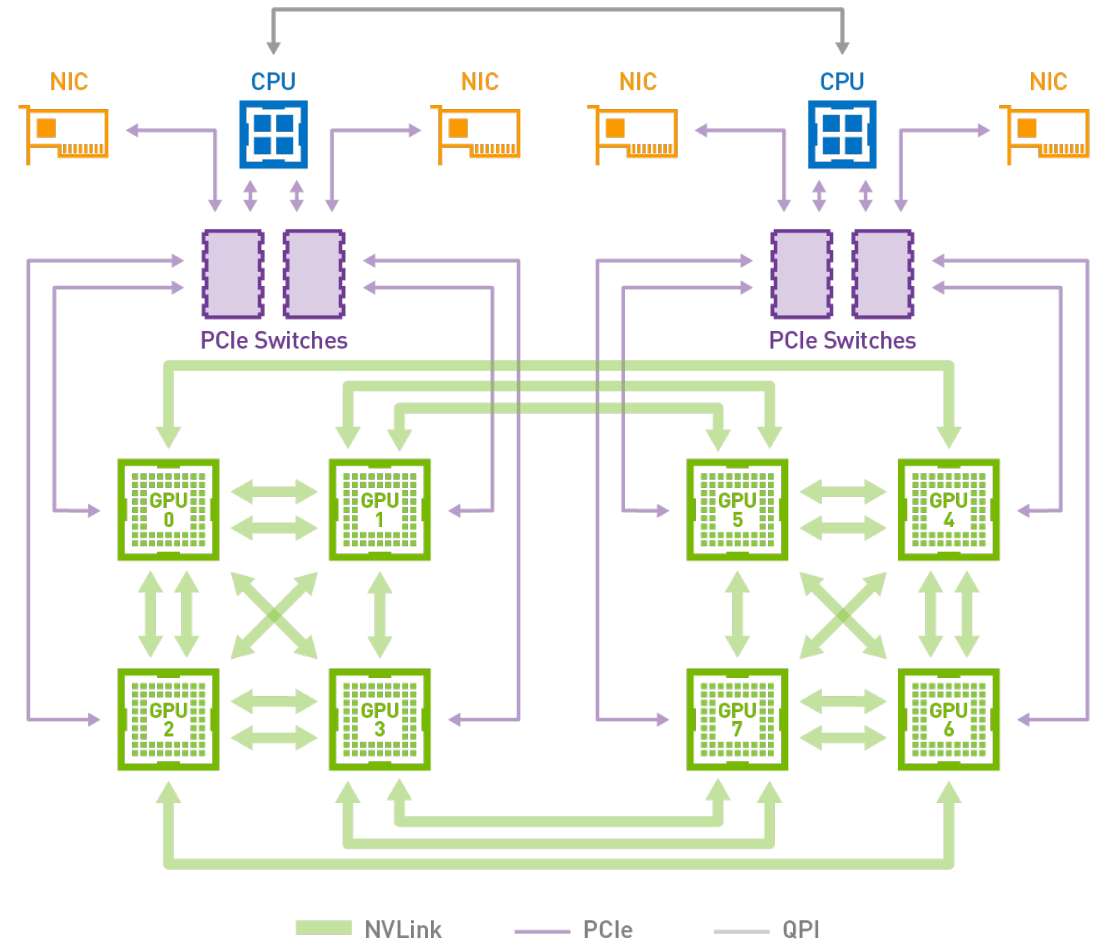
300 GB/s per GPU bidirectional to Peers

Load/store access to Peer Memory

Full atomics to Peer GPUs

High speed copy engines for bulk data copy

PCIe to/from CPU



NVLINK - NVSWITCH GPU CLUSTER

16 GPUs

Fully-connected

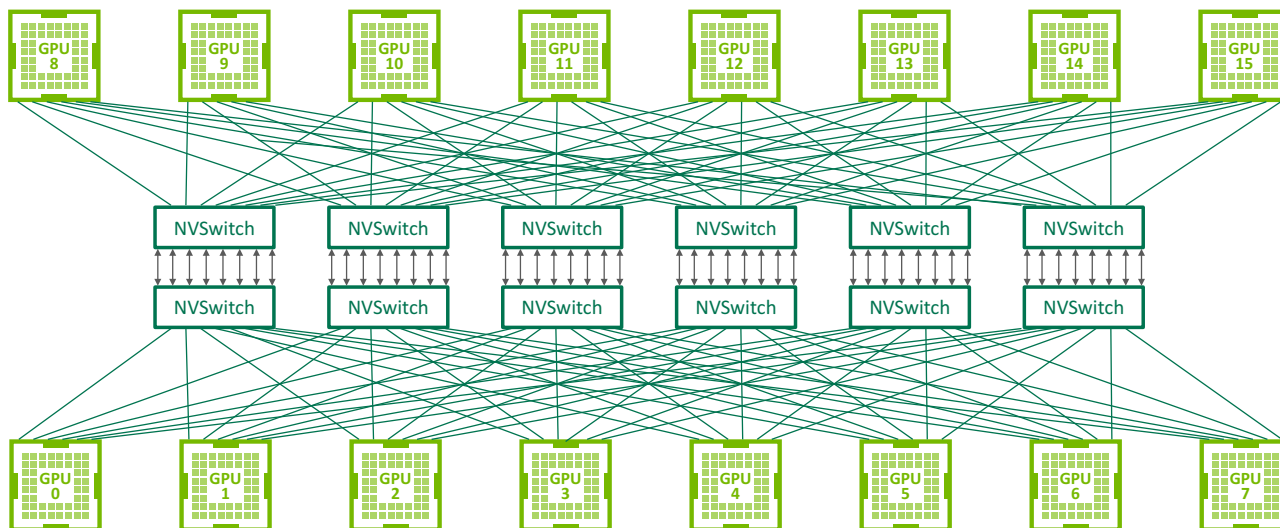
300 GB/s bidirectional bandwidth
between any pair of GPUs

2.4 TB/s bisection bandwidth

Full atomics to Peer GPUs

High speed copy engines for bulk
data copy

PCIe to/from CPU



NVLINK TO CPU



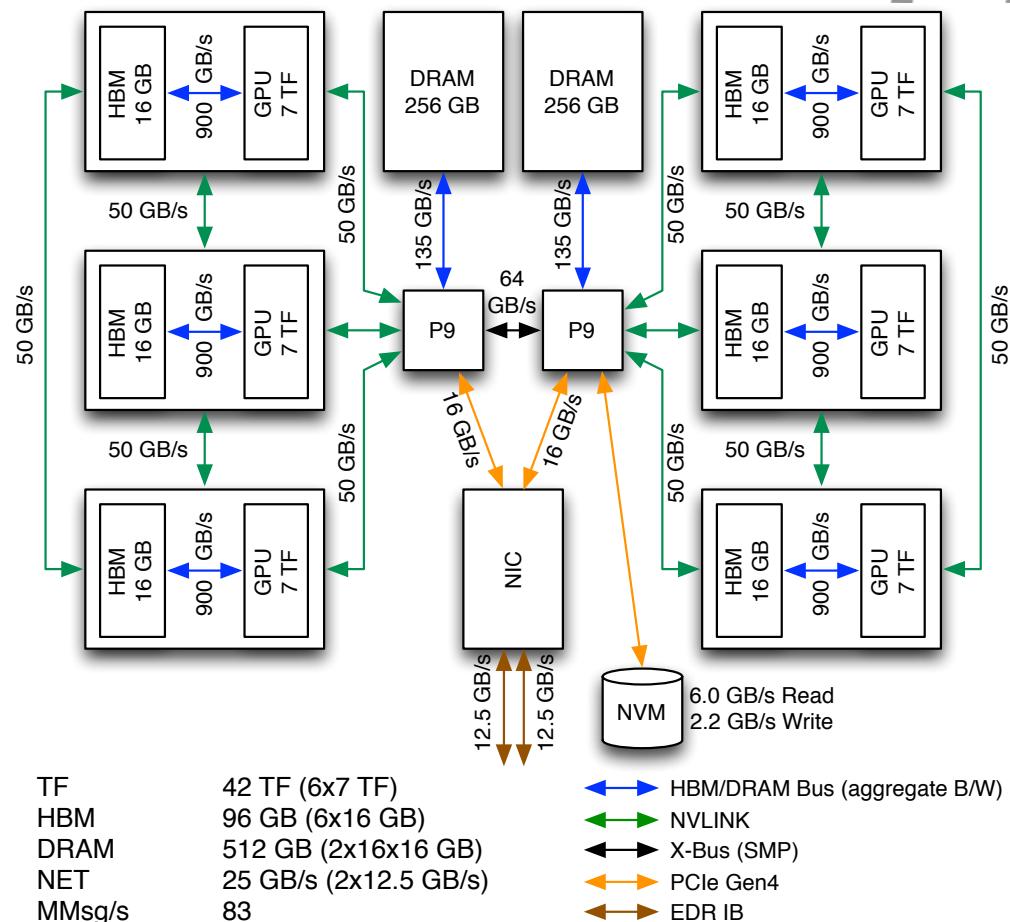
Fully connected quad

200 GB/s per GPU bidirectional for peer traffic

100 GB/s per GPU bidirectional to CPU

Direct Load/store access to CPU Memory

High Speed Copy Engines for bulk data movement



HBM & DRAM speeds are aggregate (Read+Write).
All other speeds (X-Bus, NVLink, PCIe, IB) are bi-directional.

OPENACC DIRECTIVES

Data Management

```
#pragma acc enter data create(A[0:nx*ny],Aref[0:nx*ny],Anew[0:nx*ny],rhs[0:nx*ny])
...
#pragma acc update device(A[(iy_start-1)*nx:((iy_end-iy_start)+2)*nx])
#pragma acc update device(rhs[iy_start*nx:(iy_end-iy_start)*nx])
while ( error > tol && iter < iter_max ) {
    ...
    iter++;
}
#pragma acc update self(A[(iy_start-1)*nx:((iy_end-iy_start)+2)*nx])
...
#pragma acc exit data delete(A,Aref,Anew,rhs)
```

UNIFIED MEMORY FUNDAMENTALS

Single pointer

On-demand migration

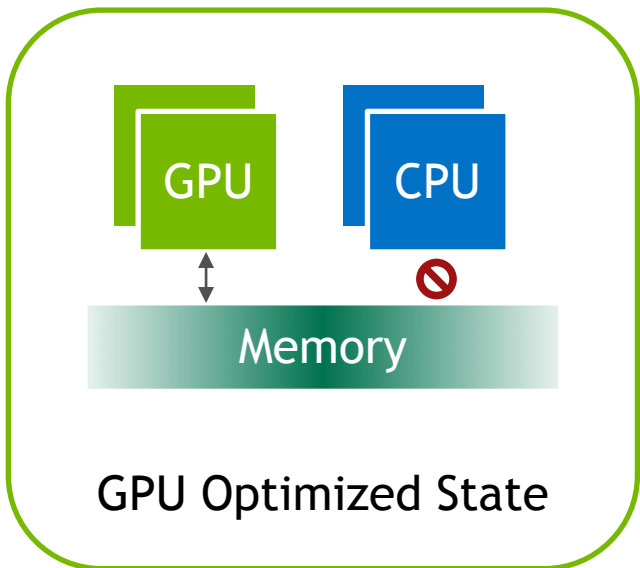
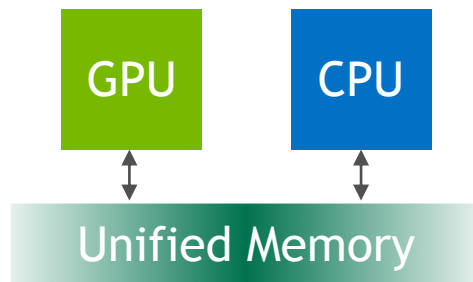
GPU memory oversubscription

Concurrent CPU/GPU access

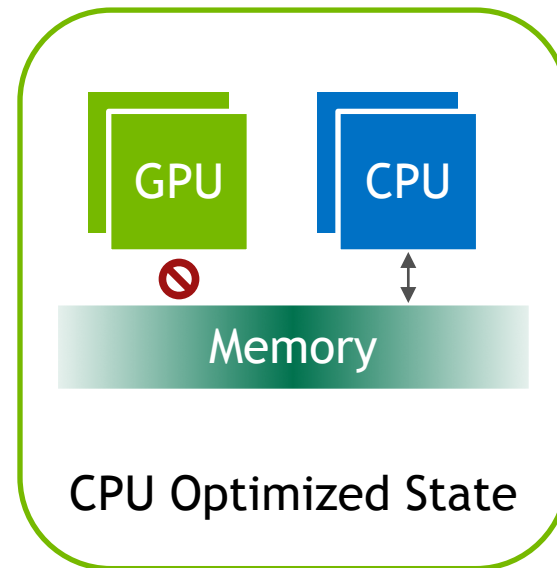
System-wide atomics

```
void *data;  
data = malloc(N);  
  
cpu_func1(data, N);  
  
gpu_func2<<<...>>>(data, N);  
cudaDeviceSynchronize();  
  
cpu_func3(data, N);  
  
free(data);
```

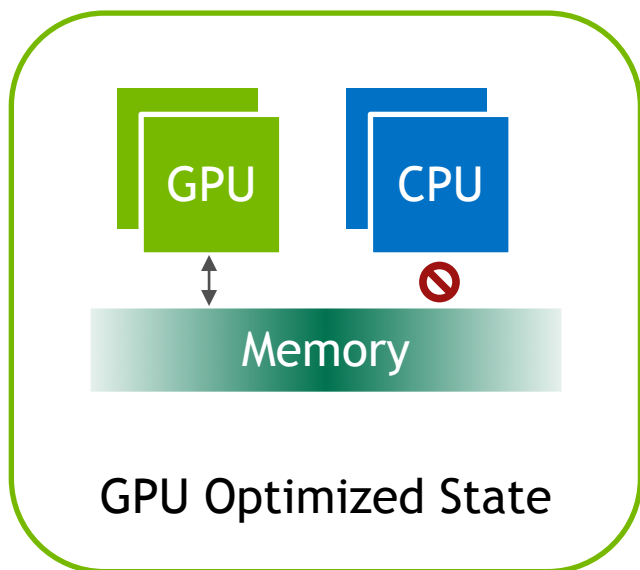
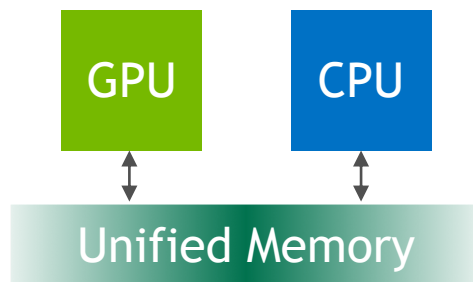

PASCAL UNIFIED MEMORY



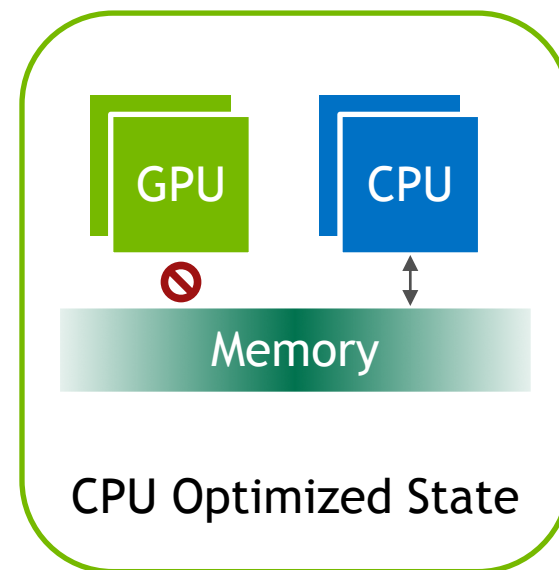
Page Migration Engine



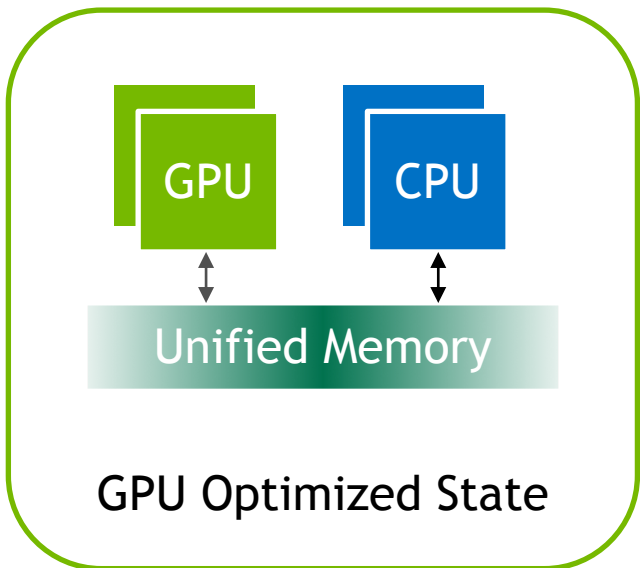
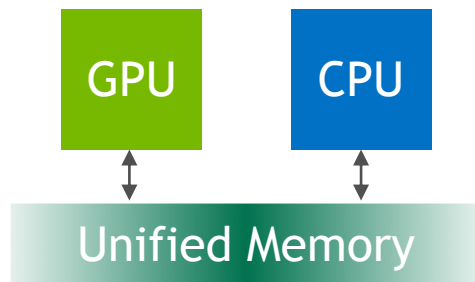
VOLTA + PCIE CPU UNIFIED MEMORY



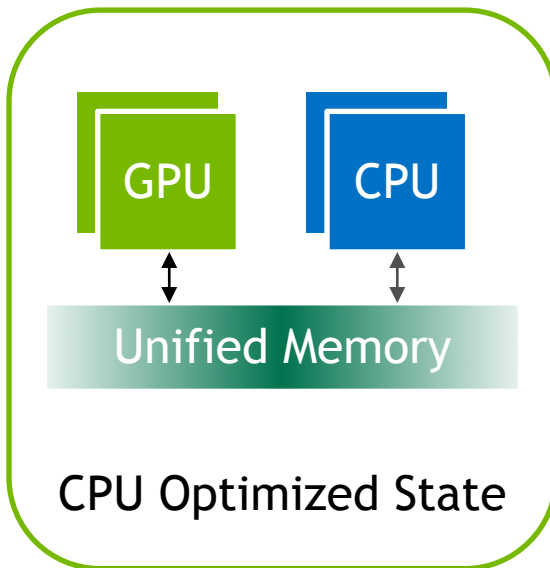
Page Migration Engine
↔
+ *Access counters*



VOLTA + NVLINK CPU UNIFIED MEMORY



Page Migration Engine
↔
+ *Access counters*
+ **New NVLink Features**
(*Coherence, Atomics, ATS*)



P9+V100 NEW FEATURES

NVLINK2: increased migration throughput

NVLINK2: enabling HW coherency (CPU access GPU memory)

Indirect peers: GPU access memory of remote GPUs on a different socket

Native atomics support for all accessible memory

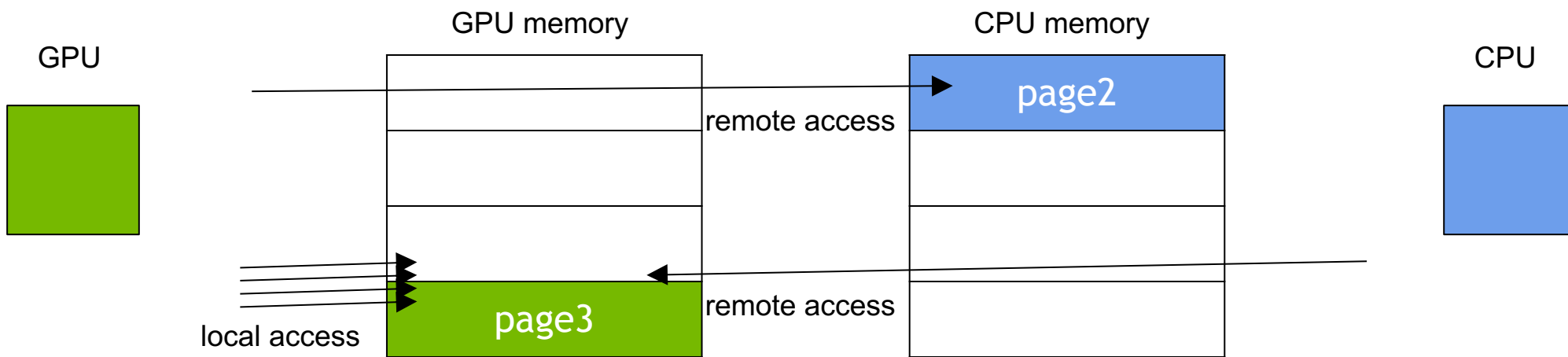
cudaMallocManaged may use access counters to guide migrations (opt-in)

ATS: GPU can access all system memory (malloc, stack, mmap files)

NVLINK2: COHERENCY

Works for `cudaMallocManaged` and `malloc`

CPU can directly access and cache GPU memory; *native* CPU-GPU atomics



page2 maps to **CPU** physical memory, page3 maps to **GPU** physical memory

LINUX AND UNIFIED MEMORY

ANY memory will be available for GPU*

CPU code

```
void sortfile(FILE *fp, int N) {  
    char *data;  
    data = (char *)malloc(N);  
  
    fread(data, 1, N, fp);  
  
    qsort(data, N, 1, compare);  
  
    use_data(data);  
  
    free(data);  
}
```

GPU code with Unified Memory

```
void sortfile(FILE *fp, int N) {  
    char *data;  
    data = (char *)malloc(N);  
  
    fread(data, 1, N, fp);  
  
    qsort<<<...>>>(data,N,1,compare);  
    cudaDeviceSynchronize();  
  
    use_data(data);  
  
    free(data);  
}
```

*on supported operating systems

The background is a dark blue field with a complex network of thin, light green lines. These lines connect various points, some of which are highlighted as bright green dots. The overall effect is a sense of dynamic connectivity and data flow.

**OPTIMIZE KERNEL
PERFORMANCE**

PROFILING OPENACC APPLICATIONS

Using pgprof

Use on the command line to get a flat profile

```
pgprof ./app
```

Collect performance metrics

```
pgprof --metrics gld_efficiency,gst_efficiency ./app
```

or write output to file and import into pgprof GUI:

```
pgprof -o timeline.pgprof ./app
```

```
pgprof --analysis-metrics -o metrics.pgprof ./app
```

PROFILING OPENACC APPLICATIONS

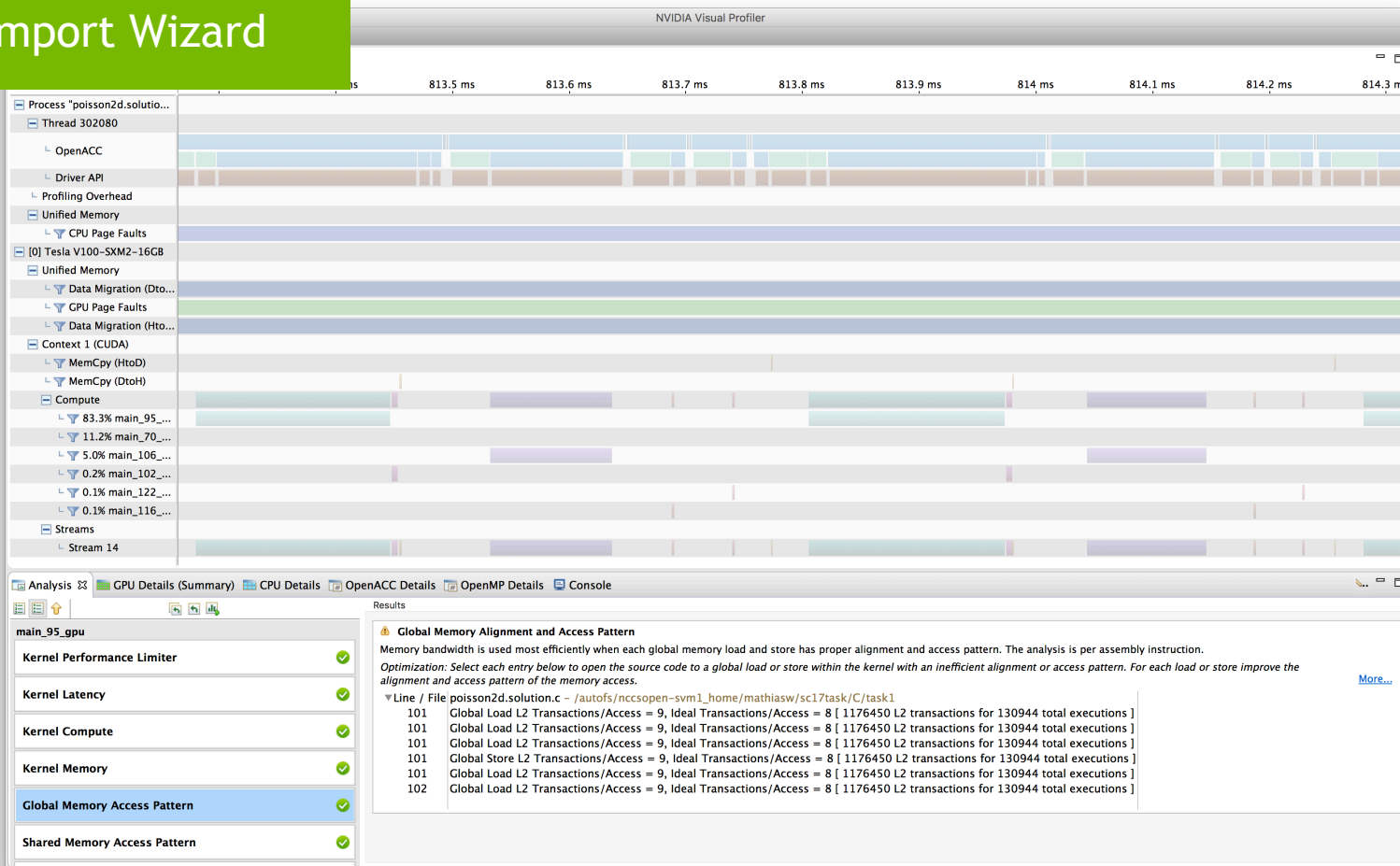
Using pgprof

```
5. ssh
x ssh %1
bash-4.2$ jsrun -n1 -g 1 -c 1 -a 1 pgprof --cpu-profiling off -o /gpfs/wolf/gen110/proj-shared/mathiasw/task1.timeline.nvvp ./poisson2d.solution 10
==156733== PGPROF is profiling process 156733, command: ./poisson2d.solution 10
==156733== Generated result file: /gpfs/wolf/gen110/proj-shared/mathiasw/task1.timeline.nvvp
Jacobi relaxation Calculation: 2048 x 2048 mesh
Calculate reference solution and time serial CPU execution.
0, 0.249999
GPU execution.
0, 0.249999
2048x2048: 1 CPU: 0.1250 s, 1 GPU: 0.0225 s, speedup: 5.55
bash-4.2$ jsrun -n1 -g 1 -c 1 -a 1 pgprof --cpu-profiling off --analysis-metrics -o /gpfs/wolf/gen110/proj-shared/mathiasw/task1.metrics.nvvp ./poisson2d.solution 3
==156755== PGPROF is profiling process 156755, command: ./poisson2d.solution 3
==156755== Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
==156755== Generated result file: /gpfs/wolf/gen110/proj-shared/mathiasw/task1.metrics.nvvp
Jacobi relaxation Calculation: 2048 x 2048 mesh
Calculate reference solution and time serial CPU execution.
0, 0.249999
GPU execution.
0, 0.249999
2048x2048: 1 CPU: 0.0491 s, 1 GPU: 15.4494 s, speedup: 0.00
bash-4.2$
```

PROFILING OPENACC APPLICATIONS

Using pgprof

Use the import Wizard

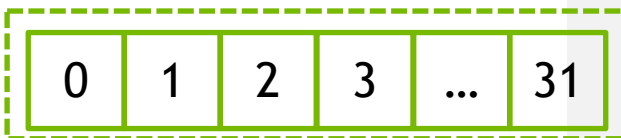


ACCESSING GLOBAL MEMORY

Best Practices

- Threads issue instructions in groups of 32 threads (warp) operating in SIMT mode
- Each issued memory instructions generates multiple 32byte transactions
- 4 consecutive 32byte segments form a cache line
- The memory subsystem works most efficient if all moved data is consumed and as few memory instructions as possible are used

threadIdx.x



```
95, Accelerator kernel generated
```

```
Generating Tesla code
```

```
96, #pragma acc loop gang /* blockIdx.x */
```

```
99, #pragma acc loop vector(128) /* threadIdx.x */
```

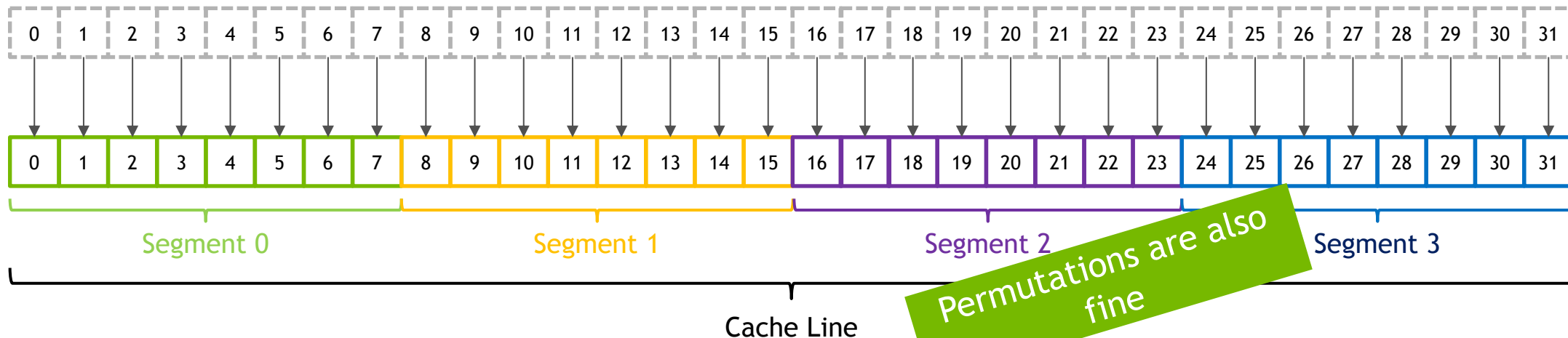
```
104, Generating implicit reduction(max:error)
```

ACCESSING GLOBAL MEMORY

optimal access pattern (4byte words) - fully coalesced

```
#pragma acc loop vector
```

```
for( int ix = 0; ix < nx; ix++ ) A[iy*nx+ix] = 0.0;
```



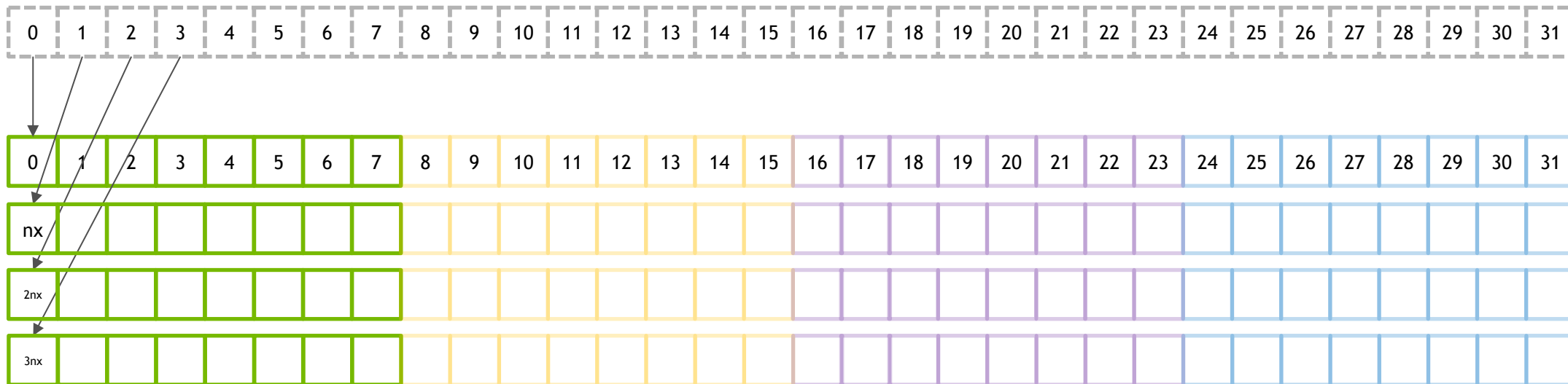
Permutations are also fine

ACCESSING GLOBAL MEMORY

worst case access pattern (4byte words) - fully uncoalesced

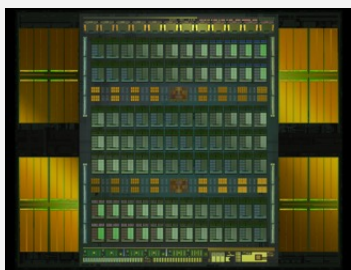
```
#pragma acc loop vector
```

```
for( int iy = 0; iy < ny; iy++ ) A[iy*nx+ix] = 0.0;
```



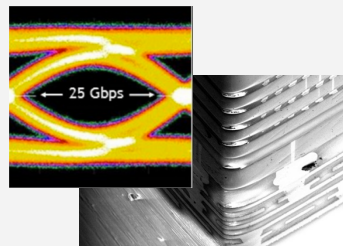
INTRODUCING TESLA V100

Volta Architecture



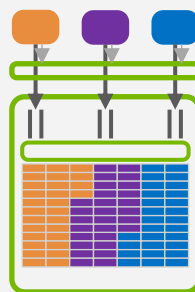
Most Productive GPU

Improved NVLink & HBM2



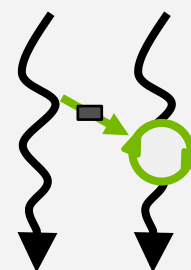
Efficient Bandwidth

Volta MPS



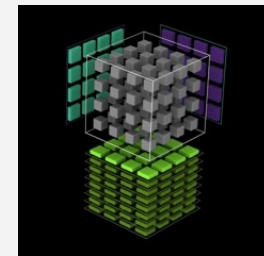
Inference Utilization

Improved SIMT Model



New Algorithms

Tensor Core



120 Programmable
TFLOPS Deep Learning

More V100 Features: 2x L2 atomics, int8, new memory model, copy engine page migration, and more ...

The Fastest and Most Productive GPU for Deep Learning and HPC

