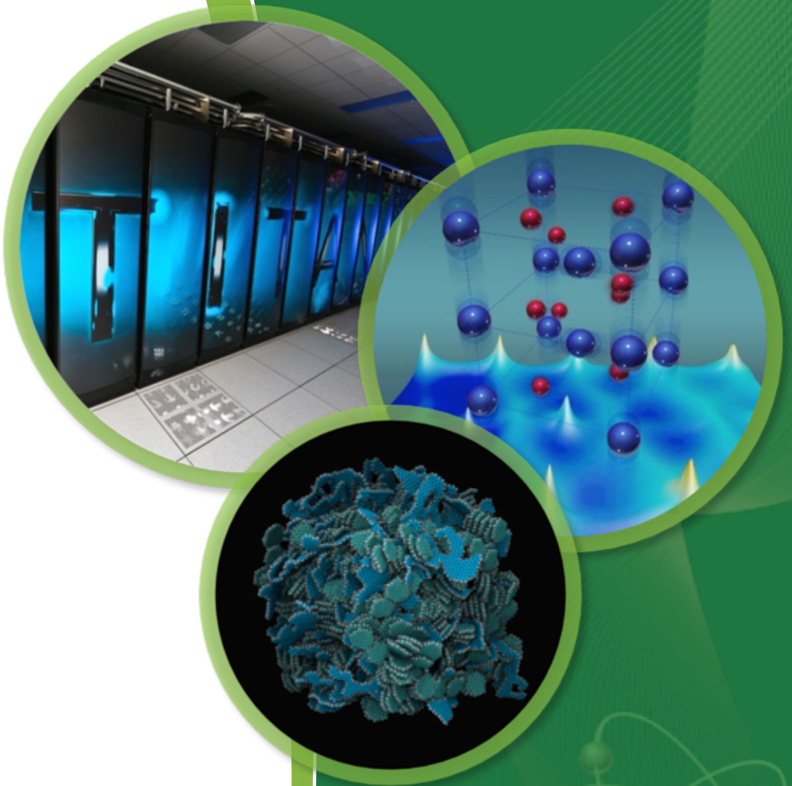
Introduction to GPU Computing

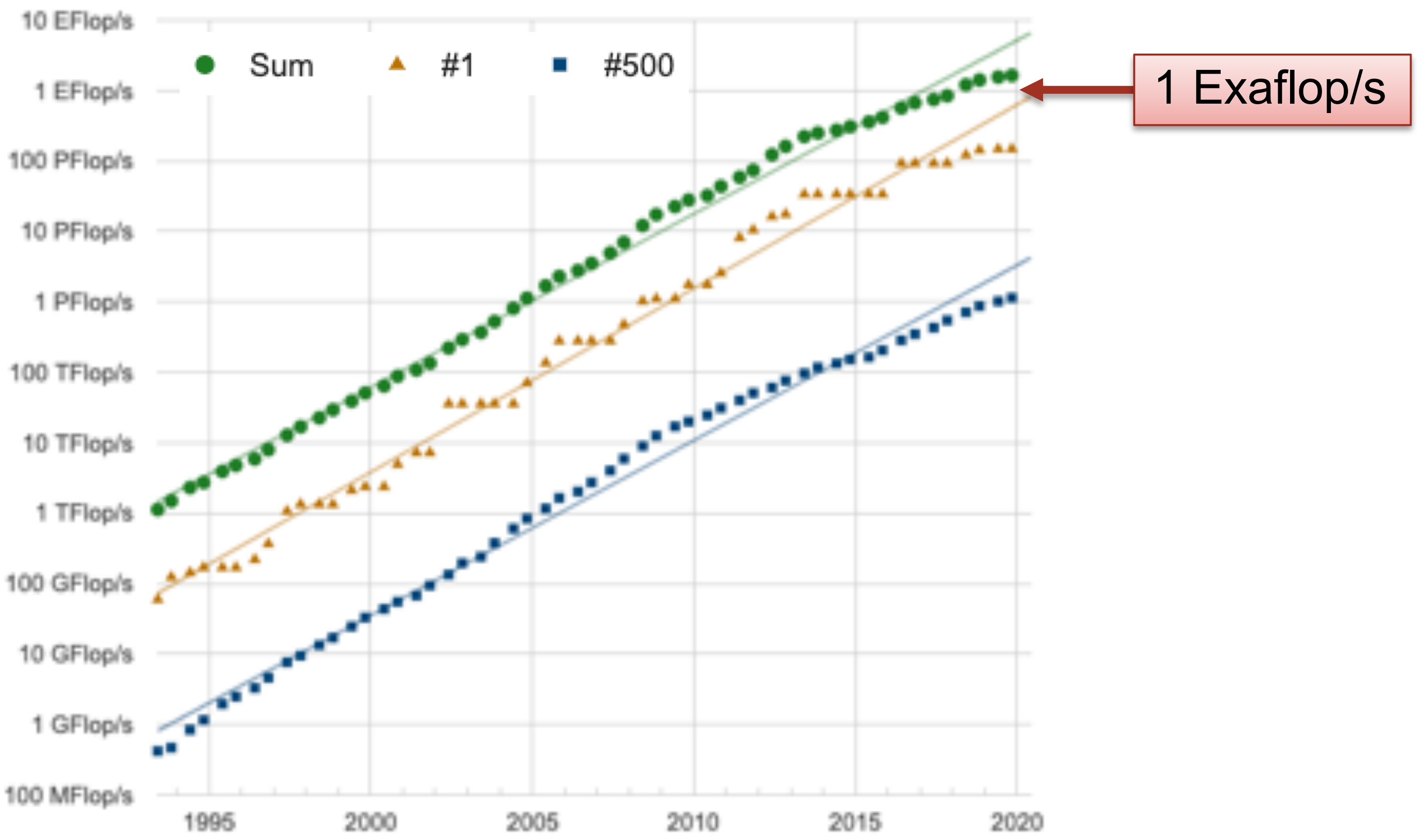
J. Austin Harris

Scientific Computing Group
Oak Ridge National Laboratory



ORNL is managed by UT-Battelle
for the US Department of Energy

Performance Development in Top500



https://www.top500.org/statistics/perfdevel

- Yardstick for measuring performance in HPC
  - Solve Ax = b
  - Measure floating-point operations per second (Flop/s)
- U.S. targeting Exaflop system as early as 2022
  - Building on recent trend of using GPUs

Hardware Trends

- Scaling number of cores/chip instead of clock speed
- Power is the root cause
  - Power density limits clock speed
- Goal has shifted to performance through parallelism
- Performance is now a software concern

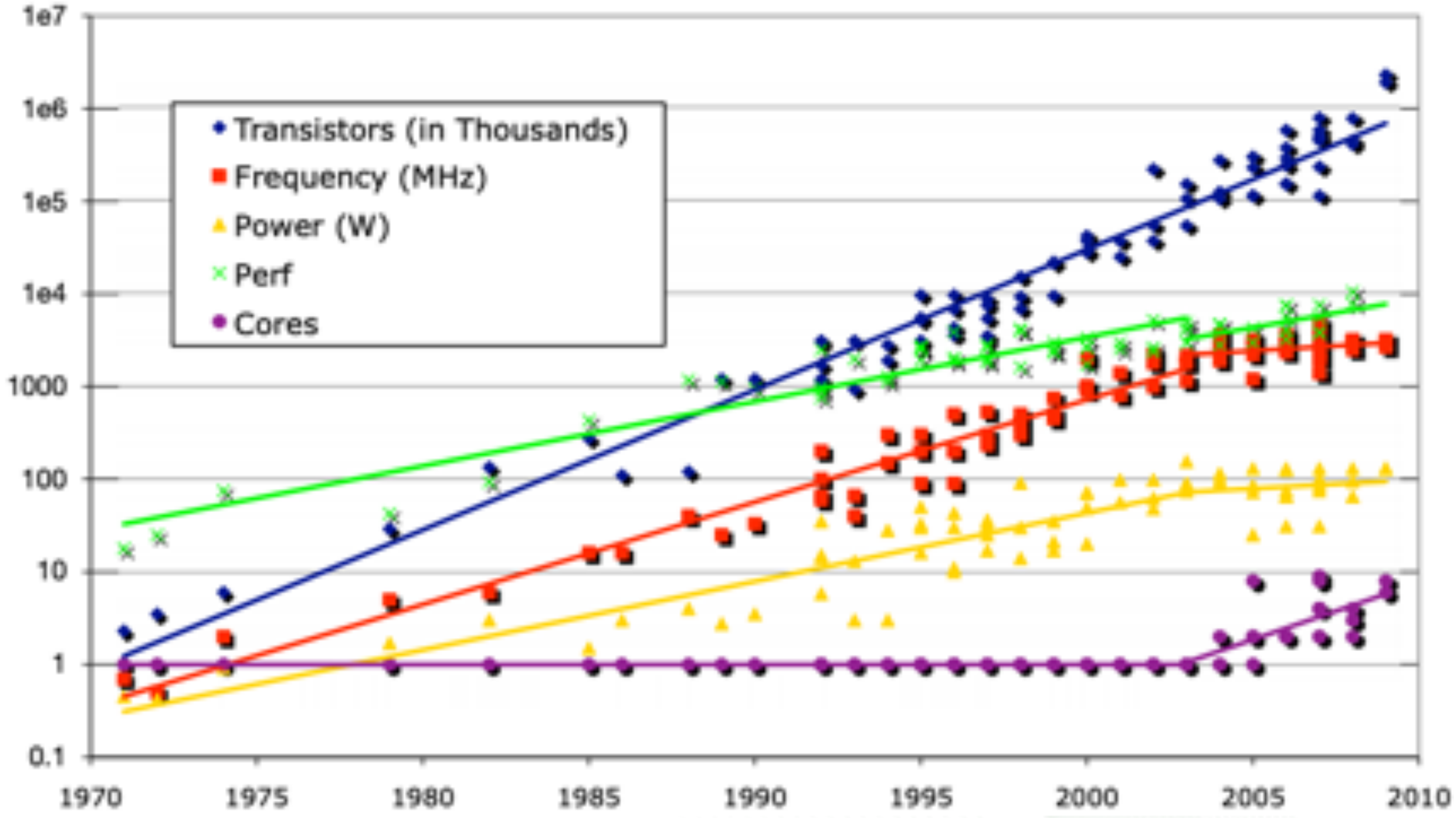


Figure from Kathy Yelick, “Ten Ways to Waste a Parallel Computer.”
Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanoviç.

OAK RIDGE National Laboratory LEADERSHIP COMPUTING FACILITY

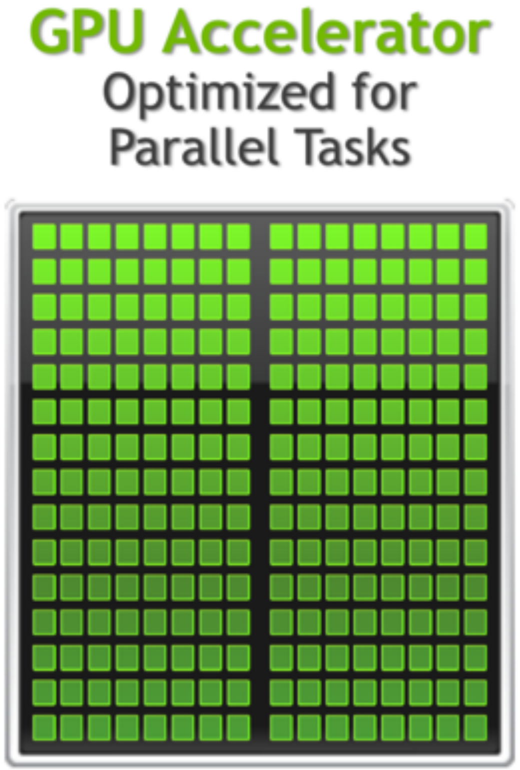
GPUs for Computation

- Excellent at graphics rendering
  - Fast computation (e.g., TV refresh rate)
  - High degree of parallelism (millions of independent pixels)
  - Needs high memory bandwidth
    - Often sacrifices latency, but this can be ameliorated

- This computation pattern common in many scientific applications

GPUs for Computation

- CPU Strengths
  - Large memory
  - Fast clock speeds
  - Large cache for latency optimization
  - Small number of threads that can run very quickly
- CPU Weaknesses
  - Low mem. bandwidth
  - Costly cache misses
  - Low perf./watt

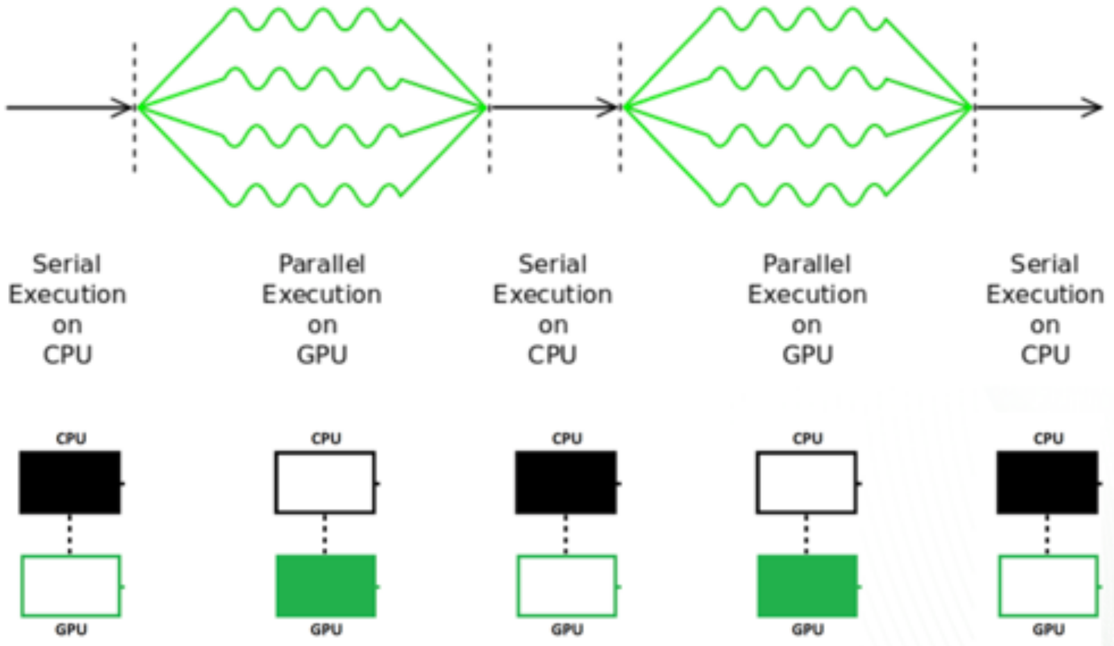


Slide from Jeff Larkin, "Fundamentals of GPU Computing"

- GPU Strengths
  - High mem. BW
  - Latency tolerant via parallelism
  - More compute resources (cores)
  - High perf./watt
- GPU Weaknesses
  - Low mem. Capacity
  - Low per-thread perf.

GPU Programming Approach

- Heterogeneous Programming
  - Small, non-parallelizable tasks on CPU
  - Large, parallel tasks on GPU
- Challenges
  - Increase in parallelism
    - New algorithms?
  - Increase in communication cost
    - PCIe bandwidth between devices much slower than that on GPU or CPU alone



GPU Programming Models

3 ways to program GPUs

Libraries
(cuBLAS, cuFFT, MAGMA, …)

High performance for limited code change

Limited by availability of libraries

Compiler Directives
(OpenACC, OpenMP, …)

High-level extensions to existing languages

Less fine-grain control over performance

Programming Languages
(CUDA, OpenCL, HIP, …)

Expose low-level details to maximize performance

More difficult and time consuming to implement

Error prone

GPU Accelerated Libraries
(not an exhaustive list)

Linear Algebra (dense)	cuBLAS, cuSPARSE, MAGMA, SLATE
FFT	cuFFT, FFT-X, heFFTe
Random Number Generation	cuRAND
Linear Solvers	cuSOLVER, PETSc, SuperLU
ODE	SUNDIALS
Algebraic Multigrid	AmgX, hypre
Tensor Algebra	cuTENSOR, TAL-SH
Data Structures (e.g., sort, scan, …)	Thrust, rocPRIM
ML & AI	cuDNN, CUTLASS, Rapids

* Most NVIDIA libraries (e.g., cuBLAS) have AMD counter-parts (rocBLAS)

GPU Libraries Example

```fortran
real(8) :: x(n), y(n), a
integer :: n, i

! Initialize data
allocate(x(n),y(n))
call initData(x,y)

do i=1,n
  y(i) = a*x(i)+y(i)
enddo
```

```fortran
use iso_c_binding
use cublas_module ! Fortran interfaces
use cuda_module ! Fortran interfaces

real(8) :: x(n), y(n), a
integer :: n, i, ierr
type(c_ptr) :: dx, dy

! Initialize data on CPU
allocate(x(n),y(n))
call initData(x,y)

! Copy data from CPU->GPU
ierr = cudaMalloc(dx,n*sizeof(x))
ierr = cudaMalloc(dy,n*sizeof(y))
ierr = cublasSetVector(n,sizeof(x),x,1,dx,1)
ierr = cublasSetVector(n,sizeof(y),y,1,dy,1)

ierr = cublasDaxpy(n, a, dx, 1, dy, 1)

! Bring the result back to the CPU
ierr = cublasGetVector(n,sizeof(y),dy,1,y,1)
```

Compiler Directives

- OpenMP device offload
  - Extension of OpenMP for multi-core threading to accelerators
  - Part of standard since 4.5
  - Supports Fortran, C, C++
  - Features lag OpenACC by 1+ year
  - Limited compiler support, but this is changing
    - Available on Summit with IBM XL compilers

- OpenACC
  - Designed specifically for accelerators
  - More advanced features than OpenMP
  - Only available with GNU and PGI compilers

OpenACC Parallel Directive

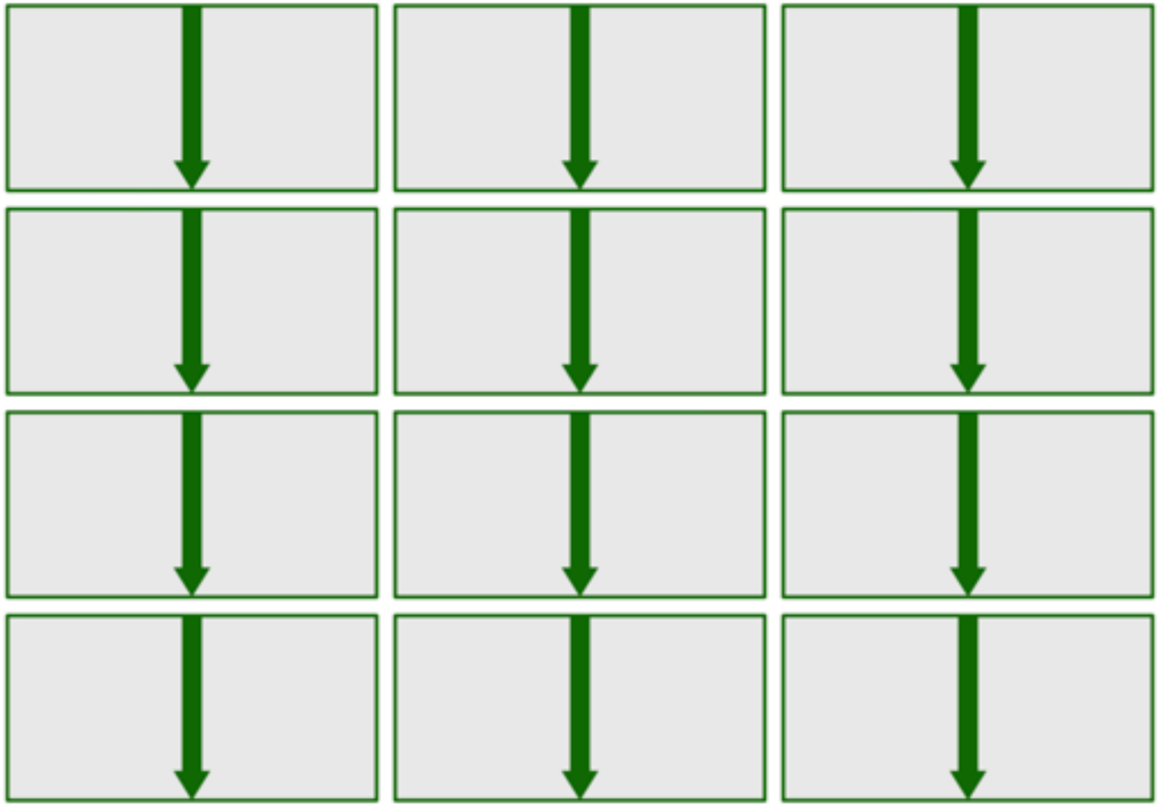
C/C++ : #pragma acc parallel
Fortran: !$acc parallel

Generates parallelism

#pragma acc parallel
{

The *parallel* directive will generate 1 or more parallel *gangs* which execute redundantly

}

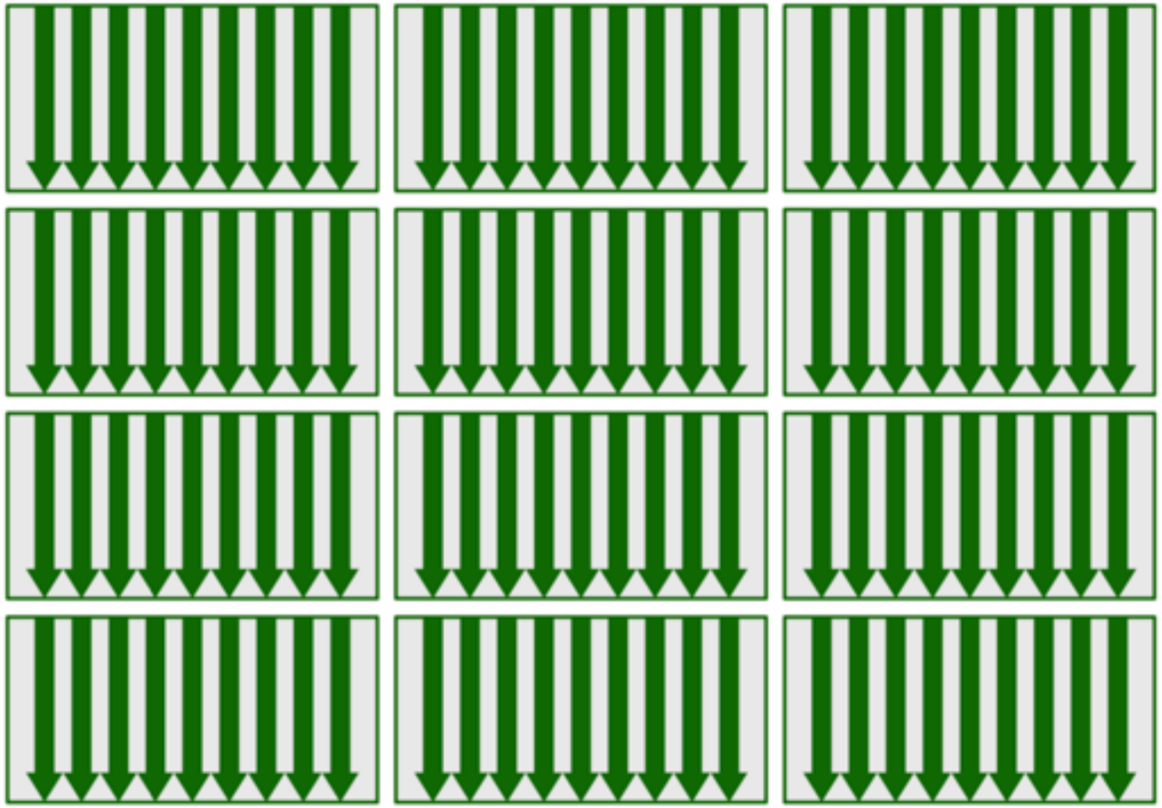


OpenACC Loop Directive

C/C++ : #pragma acc loop
Fortran: !$acc loop

Identifies loops to run in parallel

```
#pragma acc parallel
{

   #pragma acc loop
   for (i=0; i<n; ++i)
   {
   }

}
```

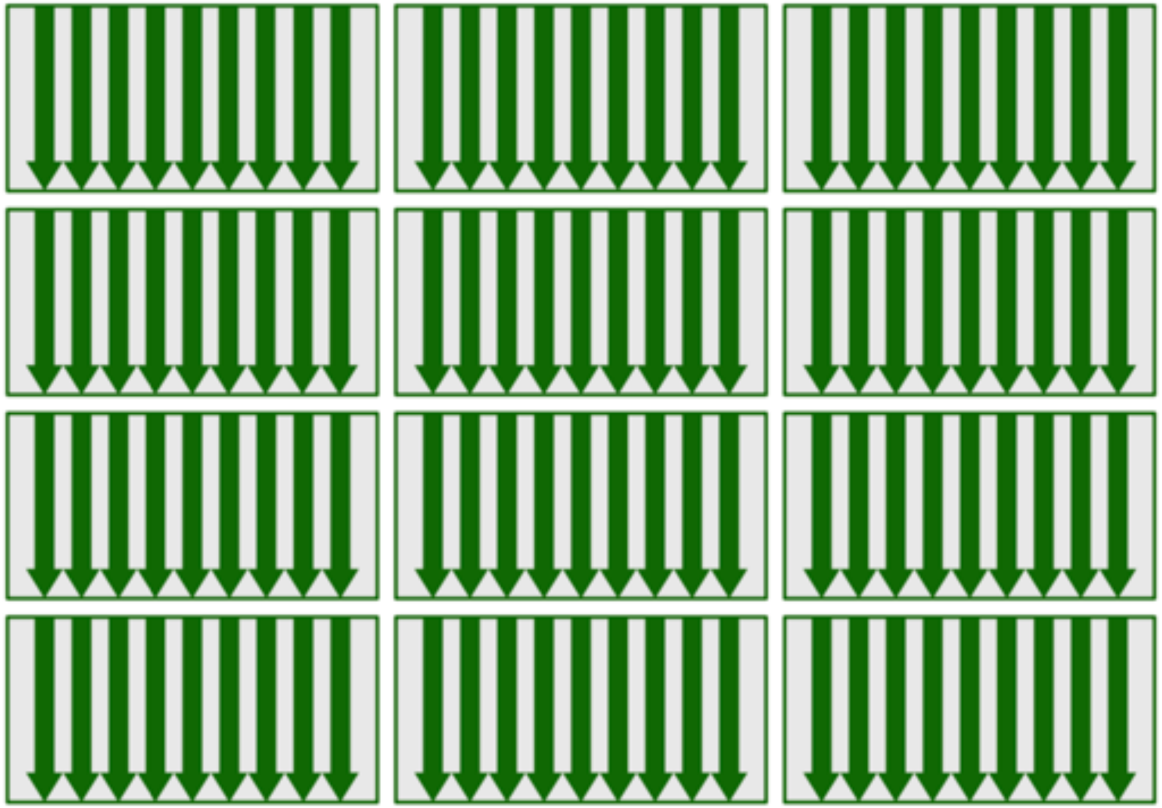


OpenACC Parallel Loop Directive

C/C++ : #pragma acc parallel loop
Fortran: !$acc parallel loop

Generates parallelism AND identifies loop in one directive

```
#pragma acc parallel loop
for (i=0; i<n; ++i)
{
}
```



Directives Example

```fortran
real(8) :: x(n), y(n), a
integer :: n, i

! Initialize data
allocate(x(n),y(n))
call initData(x,y)

do i=1,n
  y(i) = a*x(i)+y(i)
enddo
```

```fortran
real(8) :: x(n), y(n), a
integer :: n, i, ierr


! Initialize data on CPU
allocate(x(n),y(n))
call initData(x,y)

#ifdef USE_OPENACC
!$acc enter data copyin(x,y)

!$acc parallel loop gang vector
#else
!$omp target enter data map(to:x,y)

!$omp target teams distribute &
!$omp parallel do simd
#endif

do i=1,n
  y(i) = a*x(i)+y(i)
enddo

! Bring the result back to the CPU
#ifdef USE_OPENACC
!$acc exit data copyout(y)
#else
!$acc target exit data map(from:y)
#endif
```

Preprocessor directives for portability

GPU Programming Languages

- CUDA C
  - NVIDIA extension to C programming language
    - *"At its core are three key abstractions – a hierarchy of thread groups, shared memories, and barrier synchronization – that are simply exposed to the programmer as a minimal set of language extensions (to C programming language)" -- CUDA Programming Guide*
  - Compiled with nvcc compiler

- Other GPU languages are around, but won't discuss these today
  - HIP, SYCL, OpenCL, DPC++

CUDA C

```
__global__ void daxpy(int n,double a,double *x,double *y)
```

- __global__ keyword
  - ind… …ice

Number of th… within each b… | Which block the thread belongs to | Local thread ID within thread block

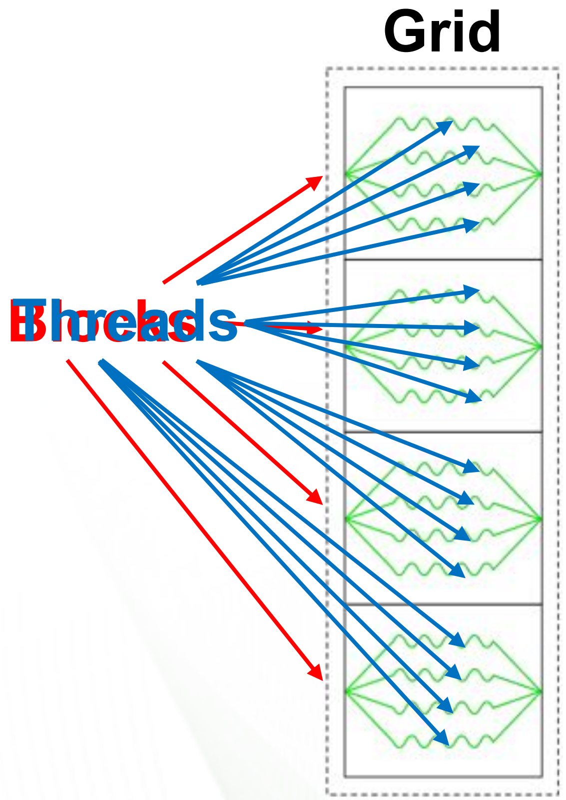
```
int i = blockIdx.x*blockDim.x + threadIdx.x;
```

- Thread indexing
  - This defines a unique thread ID among all threads in a grid

```
if (i < n) y[i] = a*x[i] + y[i];
```

- Check that thread ID is not larger than number of elements

A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	A_{14}	



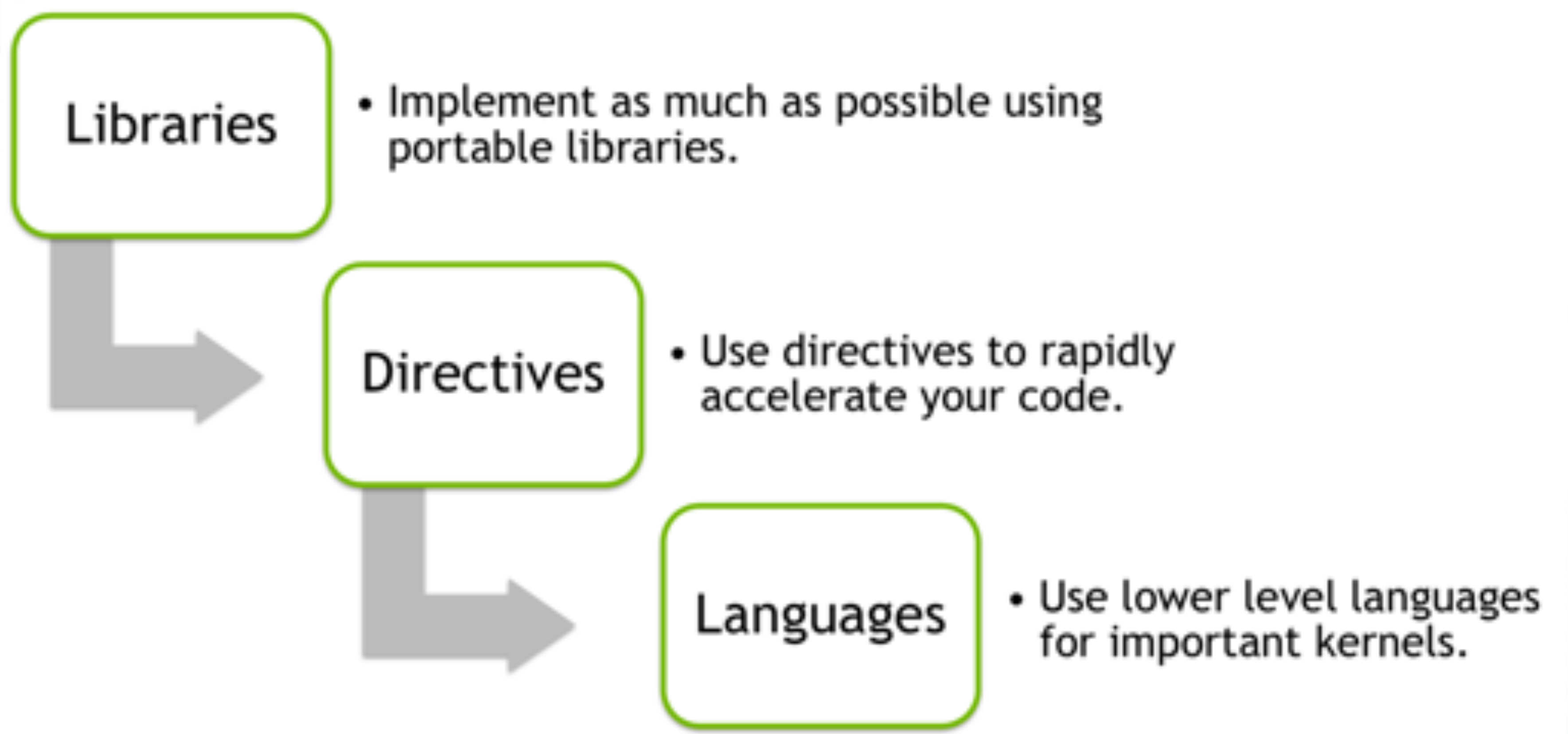
CUDA C Example

```
int n = 1000000;
double a = 2.0;

x = (double *)malloc(N * sizeof(double));
y = (double *)malloc(N * sizeof(double));
initData(x, y);


for (int i = 0; i < n; ++i)
  y[i] = a*x[i] + y[i];
```

<<<blocks in grid, threads per block>>>

```
__global__
void daxpy(int n,double a,double *x,double *y)
{
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < n) y[i] = a*x[i] + y[i];
}

int n = 1000000;
double a = 2.0;

x = (double *)malloc(N * sizeof(double));
y = (double *)malloc(N * sizeof(double));
cudaMalloc(&dx, n * sizeof(double));
cudaMalloc(&dy, n * sizeof(double));
initData(x, y);

cudaMemcpy(dx,x,n,cudaMemcpyHostToDevice);
cudaMemcpy(dy,y,n,cudaMemcpyHostToDevice);

daxpy<<<4096,256>>>(n, a, dx, dy);

cudaMemcpy(y,dy,n,cudaMemcpyDeviceToHost);
```

GPU Programming Models



Slide from Jeff Larkin, “Fundamentals of GPU Computing”

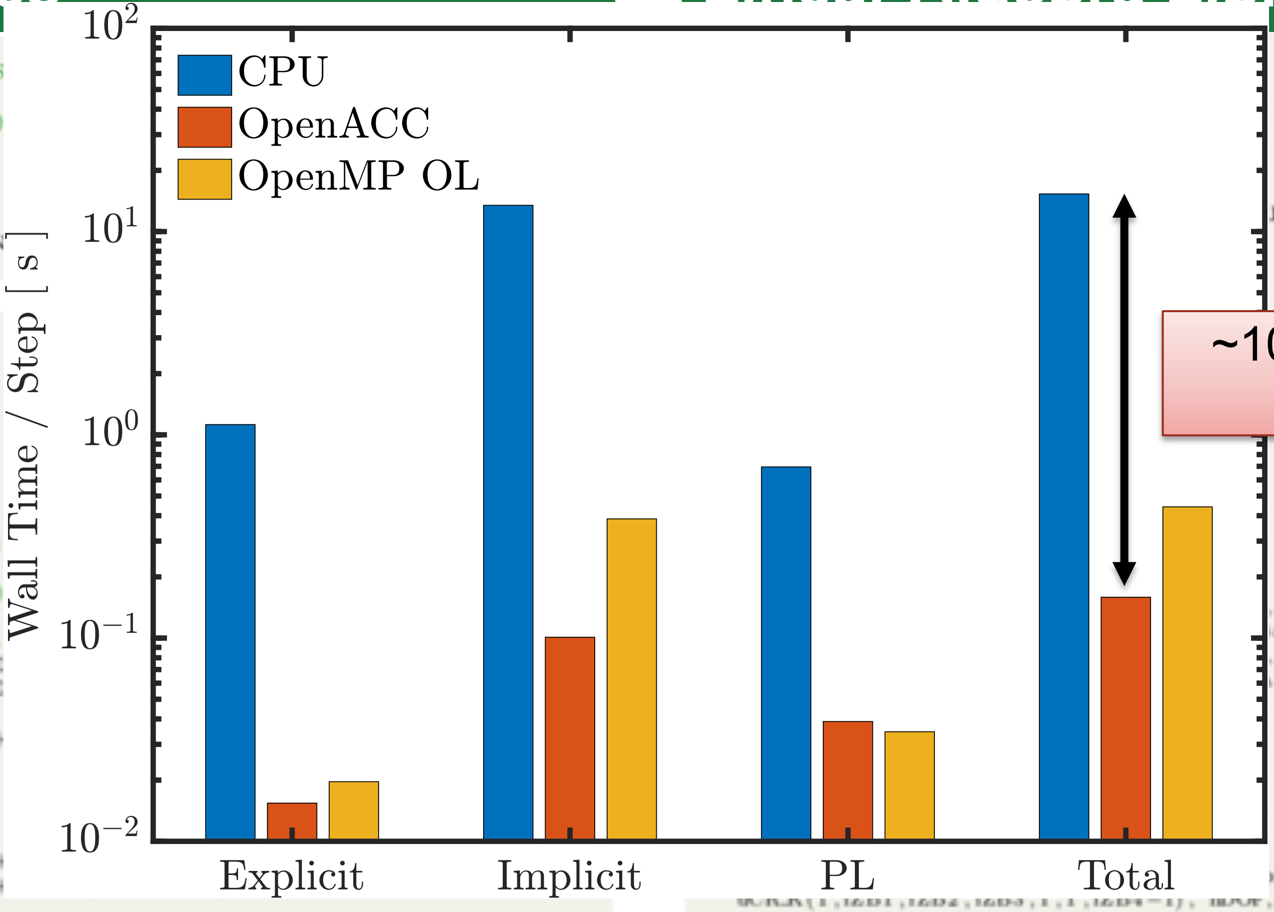
Scientific Application Example

Compiler Directives

```
#if defined(THORNADO_OMP_OL)
!$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO S
#elif defined(THORNADO_OACC)
!$ACC PARALLEL LOOP GANG VECTOR COLLAPSE(7)
#elif defined(THORNADO_OMP)
!$OMP PARALLEL DO SIMD COLLAPSE(7)
#endif
DO iZ4 = iZB4-1, iZE4+1 ; ... ; DO iNode =
  uCR_K(iNode,iZ1,iZ2,iZ3,iM,iS,iZ4) = U(iN
END DO ; ... ; END DO

#if defined(WEAKLIB_OMP_OL)
!$OMP TARGET TEAMS DISTRIBUTE IF( do_gpu )
#elif defined(WEAKLIB_OACC)
!$ACC PARALLEL LOOP GANG ASYNC( async_flag
#endif
DO k = 1, SIZE( LogT )
  iT = ... ; dT = ...
  iX = ... ; dX = ...
#if defined(WEAKLIB_OMP_OL)
  !$OMP PARALLEL DO SIMD PRIVATE( i0, j0, i
#elif defined(WEAKLIB_OACC)
  !$ACC LOOP VECTOR PRIVATE( i0, j0, i, j )
#endif
  DO ij = 1, SizeE*(SizeE+1)/2 ! Fused tria
    j0 = MOD( (ij-1) / ( SizeE + 1 ), SizeE
    i0 = MOD( (ij-1)                , SizeE
    IF ( i0 > j0 ) THEN
      j = SizeE - j0 + 1 ; i = SizeE - i0 +
    ELSE
      j = j0 ; i = i0
    END IF
    Interpolant(i,j,k) = BiLinear &
      ( Table(i,j,iT  ,iX  ), Table(i,j,iT+
        Table(i,j,iT  ,iX+1), Table(i,j,iT+
  END DO
END DO
```

Linear Algebra Libraries

```
)
LOC( pc )
, b, ldb, beta, c, ldc )
a, transb
, k, lda, ldb, ldc
, beta
, nDOF_X3, &
, Zero, &
uCR_L(1,iZB1,iZB2,iZB3,1,1,iZB4 ), nDOF_X3 )
```



~100x speedup relative to serial CPU code

OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY