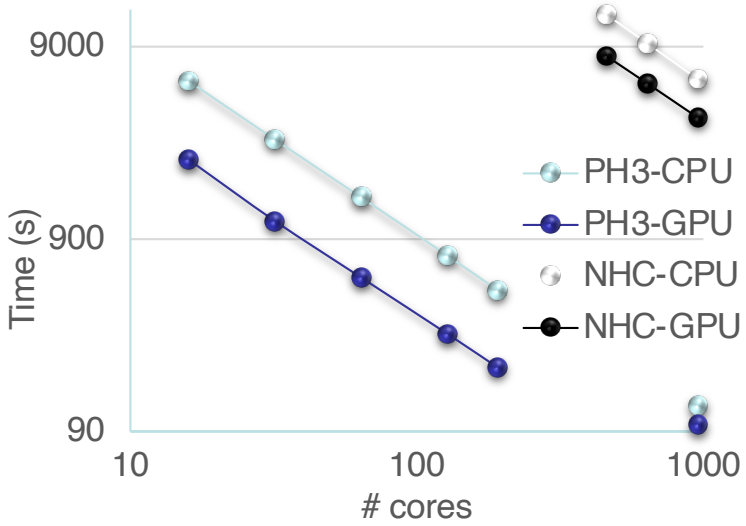
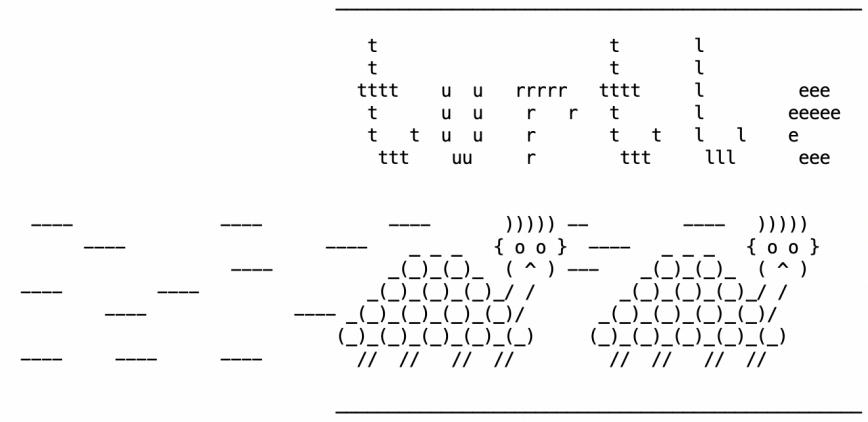
GPU programming with openMP





Remco W. A. Havenith

Stratingh Institute for Chemistry,
Zernike Institute for Advanced Materials,
University of Groningen,
The Netherlands

Ghent Quantum Chemistry Group, Ghent University, Belgium

university of groningen / faculty of science and engineering / stratingh institute for chemistry

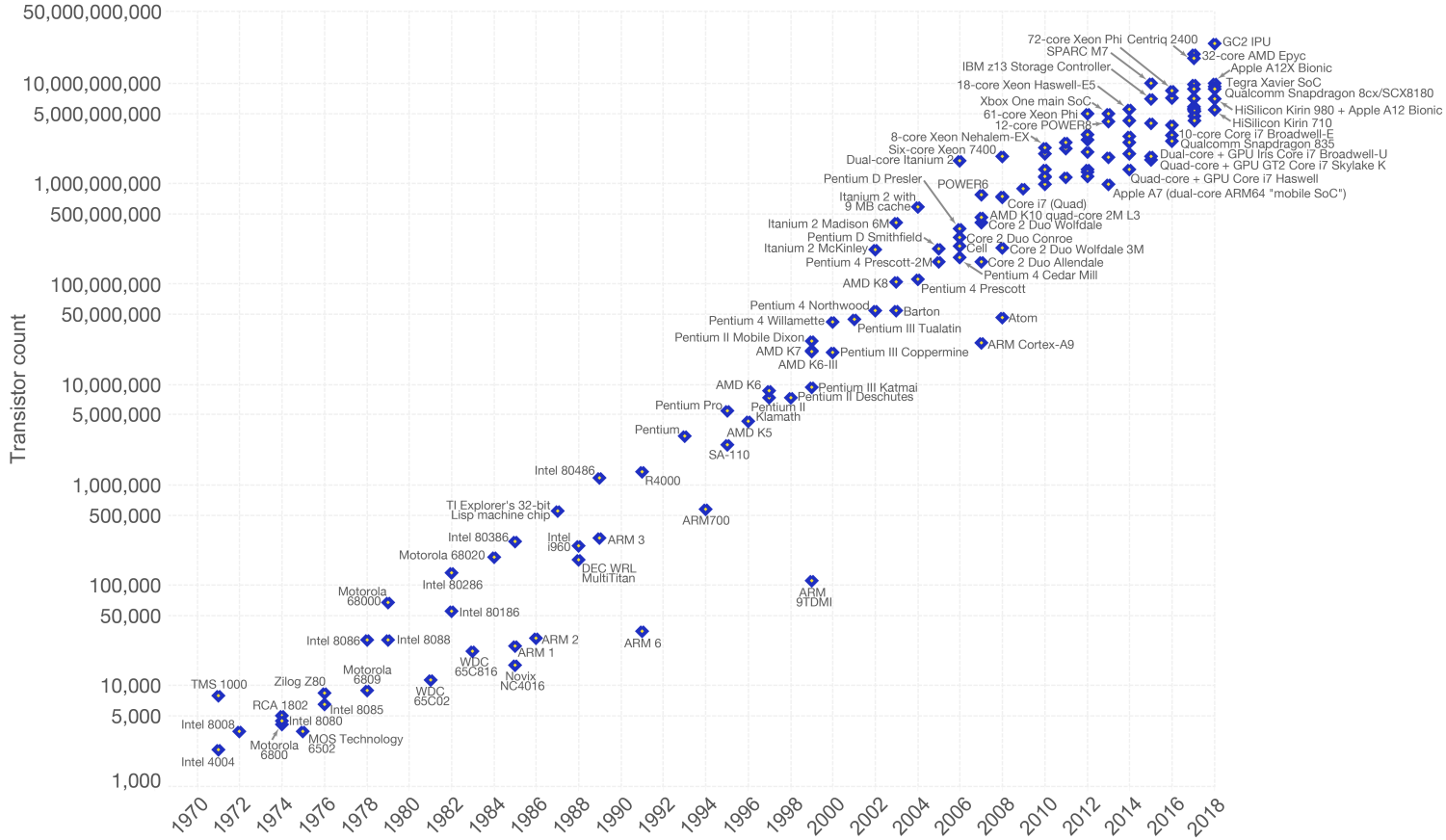
Overview

- Parallel programming with openMP and GPU offloading
  - Introduction
  - Basic steps
  - Examples
  - Hands-on session

Moore’s law

Moore’s Law – The number of transistors on integrated circuit chips (1971-2018)

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

Current supercomputers

- High node and core counts
- Equipped with accelerators

Summit at OLCF

Introduction to GPU with openMP

- CPU:
  - Fast clock
  - Multiple cores (8-64)
  - Complex cores

- GPU:
  - Slow clock
  - Thousands of cores
  - Lightweight cores

Why use openMP

- Methods to program heterogeneous systems:
  - Vendor specific library routines
    - cuBlas
  - Vendor specific frameworks
    - Cuda
  - Public domain frameworks
    - OpenCL
  - Compiler directive based API
    - OpenMP

Low programming effort, high performance, non portable

High programming effort, high performance, non portable

High programming effort, high performance, portable

Medium programming effort, high performance, portable

Advantages of openMP

- Compiler directives
- Runtime library routines
- Environment variables
- Heterogeneous programming since OpenMP 4.0
  - Integrates into existing openMP code
  - Supported by many compilers

openMP

- Compiler directive
  - Programmer inserted hint/command for the compiler
- Directive syntax
  - Fortran
    - Paired with matching *end* surrounding a code block

```
!$omp directive
    code
!$omp end directive
```

  - C
    - No *end* directive needed ({})

```
#pragma omp directive
{
code
}
```

openMP worksharing

- Parallel directive
  - Spawns a team of threads
  - Execution continues on all threads of team
  - All threads join at the end and the master thread continues

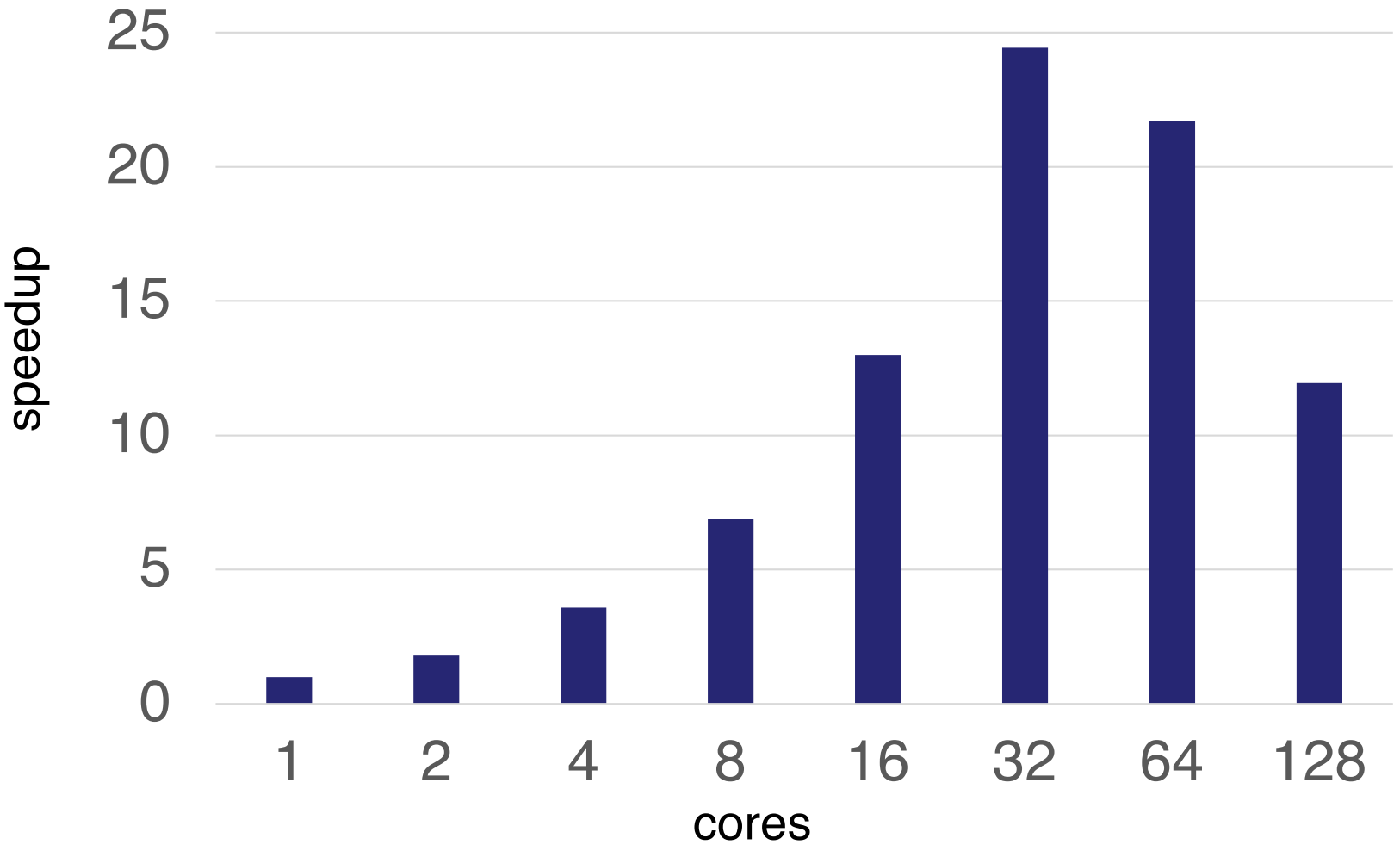
- FOR/DO (loop) directive
  - Divides the iterations of the next loop across the threads in the team

Matrix-matrix multiplication with openMP

```
      tstart=omp_get_wtime()
!$omp parallel do
      do i=1,ndim
        do j=1,ndim
          do k=1,ndim
            c(i,j)=c(i,j)+a(i,k)*b(k,j)
          enddo
        enddo
      enddo
      tend=omp_get_wtime()
```

Create a team of threads and workshare this loop across the threads

CPU version



GPU offloading

- gfortran v. 10
- AMD CPUs/GPUs and IBM Power 9 CPUs/GPUs
- Options:
  - -fopenmp [-foffload=amdgcn-amdhsa="-march=gfx906”]

```
      tstart=omp_get_wtime()
!$omp target
!$omp parallel do
      do i=1,ndim
        do j=1,ndim
          do k=1,ndim
            c(i,j)=c(i,j)+a(i,k)*b(k,j)
          enddo
        enddo
      enddo
      tend=omp_get_wtime()
```

Relocates execution to the Target (GPU) device

university of groningen / faculty of science and engineering / stratingh institute for chemistry

GPU timing

Cores	Time (s)	Speedup
1	4698	1
2	2619	1.8
4	1313	3.6
8	681	6.9
16	362	13.0
GPU	27225	0.2

- No speedup! Only slow-down! Why?
- Parallel directive creates a single level of parallelism, and threads must be able to synchronize ⇒ GPU uses 1 thread block
- The *teams* directive to introduce a second level of parallelism

openMP teams

```
      tstart=omp_get_wtime()
!$omp target teams distribute
!$omp parallel do
      do i=1,ndim
        do j=1,ndim
          do k=1,ndim
            c(i,j)=c(i,j)+a(i,k)*b(k,j)
          enddo
        enddo
      enddo
      tend=omp_get_wtime()
```

Relocate execution to the target device, generate teams, distribute loop to teams and workshare

- Timing: 557 s; speedup: 8.4 x faster than serial version
- No synchronization between teams

Collapse clause

```
        tstart=omp_get_wtime()
!$omp target teams distribute
!$omp& parallel do collapse(2)
        do i=1,ndim
          do j=1,ndim
            do k=1,ndim
              c(i,j)=c(i,j)+a(i,k)*b(k,j)
            enddo
          enddo
        enddo
        tend=omp_get_wtime()
```

Collapse loops

ndim=4096

Machine	Time (s) No collapse	Time (s) Collapse
AMD - CPU	39.20	24.05
AMD - GPU	72.13	198.92
IBM - CPU	302.6	303.75
IBM - GPU	13.10	13.20

- Advantage not clear

Collapse clause (2)

```
      do while(error.gt.tol)
      error=0.0d0
      iter=iter+1
!$omp target teams distribute
!$omp& parallel do reduction(max:error)
!$omp& collapse(2) map(error)
      do i=2,ndim-1
        do j=2,ndim-1
          anew(j,i)=0.25*(a(j,i+1)+a(j,i-1)
     &     +a(j-1,i)+a(j+1,i))
          error=max(error,dabs(anew(j,i)-a(j,i)))
        enddo
      enddo
      …
      end do
```

Collapse loops

ndim=20000

Machine	Time (s) No collapse	Time (s) Collapse
AMD - CPU	558.22	519.98
AMD - GPU	7009.52	4228.25
IBM - CPU	2000.49	3174.13
IBM - GPU	250.57	251.58

- Collapse improves performance on AMD

Data movement – Power method

```
do while (error.gt.tol)
!$omp target teams …
  v2=a*v
!$omp target teams …
  dd=ddot(v2)
  val=dsqrt(dd)
!$omp target teams …
  v2=v2/val
!$omp target teams …
  error=ddot(v2-v)
  v=v2
end do
```

- Size of matrix: 10000

Machine	Time (s)
AMD - CPU	45.94
AMD - GPU	124.22
IBM - CPU	88.72

- Bad timings due to data movement

Data movement – Power method

```
!$omp target data map(v,v2,a)
do while (error.gt.tol)
!$omp target teams …
  v2=a*v
!$omp target teams …
  dd=ddot(v2)
  val=dsqrt(dd)
!$omp target teams …
  v2=v2/val
!$omp target teams …
  error=(v2-v)**2
  v=v2
end do
!$omp end target data
```

- Size of matrix: 10000

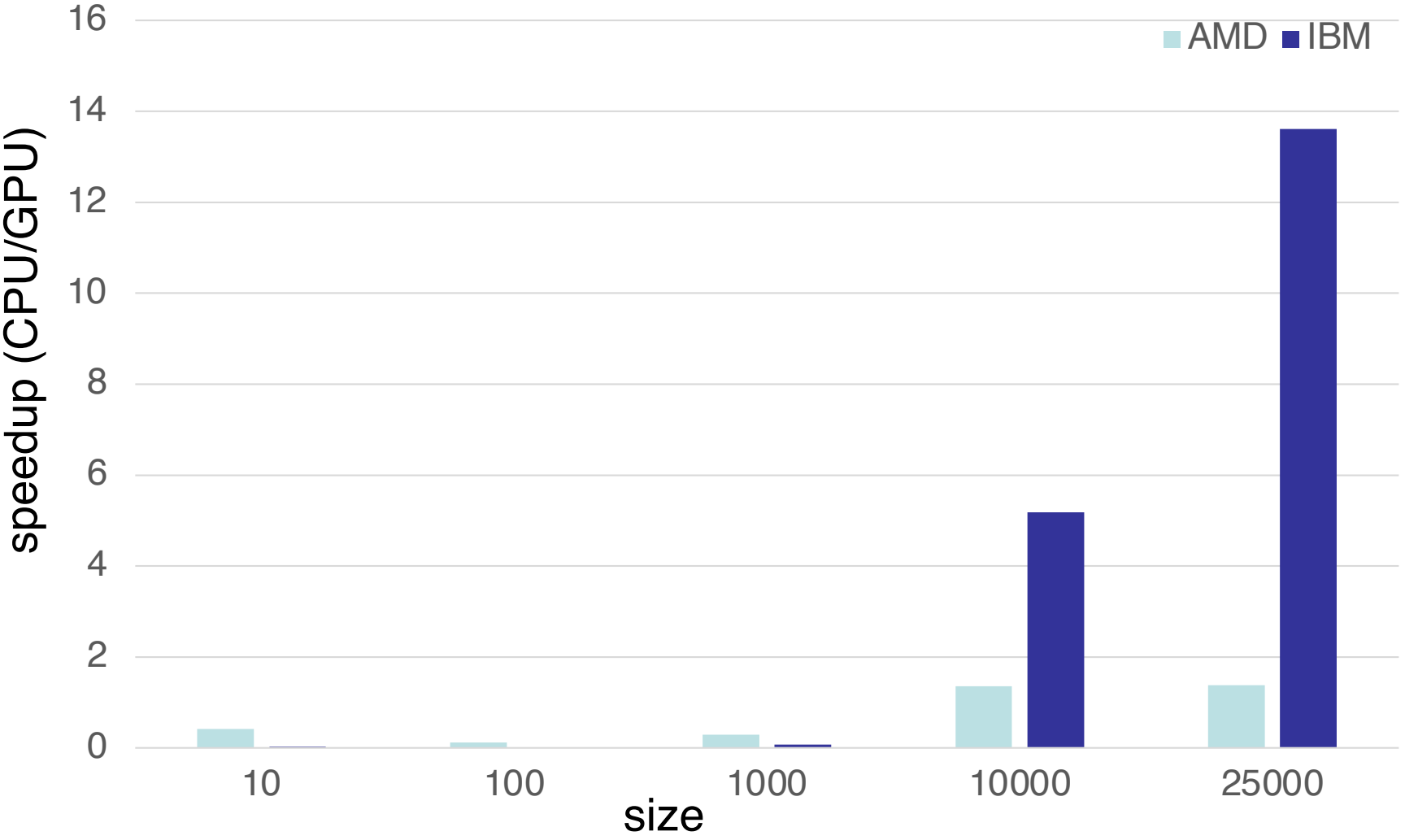
Machine	Time (s)
AMD - CPU	45.94
AMD - GPU	124.22
AMD - GPU2	33.69
IBM - CPU	88.72
IBM - GPU	--
IBM - GPU2	16.86

- Considerable improvement in performance

target data map

- target data map(<options>)
  - to – create space on the GPU and copy input data
  - from – create space on the GPU and copy output data
  - tofrom – create space on the GPU and copy input and output data
  - alloc – create space on the GPU, do not copy data

Power method, size dependence?



- GPU is not always faster! Only when the workload is high enough

openMP Host fallback

```
      tstart=omp_get_wtime()
!$omp target parallel do if(ndim>5000)
      do i=1,ndim
        do j=1,ndim
          do k=1,ndim
            c(i,j)=c(i,j)+a(i,k)*b(k,j)
          enddo
        enddo
      enddo
      tend=omp_get_wtime()
```

Only relocate execution to the device if the condition is met. Both a host and device version is built.

Device pointers

- use_device_ptr allows OpenMP device arrays to be passed to CUDA or accelerated libraries
- is_device_ptr clause allows CUDA arrays to be used within OpenMP target regions

```
!$omp target data map(x,y)
!$omp target teams distribute parallel do
        do i=1,ndim
          x(i)=1.0
          y(i)=2.0
        enddo
!$omp target data use_device_ptr(x,y)
        call cublasSaxpy(ndim,2.0,x,y,1)
```

Calling a function/subroutine

```
      program funct
      parameter (ndim=100)
      allocatable ival(:)
!$omp declare target(iff)
      allocate(ival(ndim))
!$omp target teams distribute parallel do map(from:ival)
      do i=1,ndim
         ival(i)=iff(i)
      enddo
!$omp end target teams distribute parallel do
      write(6,'(10I8)')(ival(i),i=1,ndim)
      end
      function iff(n)
!$omp declare target
      iff=n*n
      return
      end
```

Generate device code

Some openMP functions and environments

- OMP_DEFAULT_DEVICE
- omp_set_default_device/omp_get_default_device
- omp_get_num_devices
- omp_target_alloc/omp_target_free

Other options

- Launch kernel asynchronously (nowait)
- Depend clause
- OpenMP tasks
  - Define independent blocks of work
- Target enter/exit data construct
  - Map variables to the device data environment
    - Can span more than one function
- Target update: synchronize data with variable in device data environment

- Many more options:
  - See www.openmp.org

Hybrid MPI/openMP

- To fully exploit a machine, combine MPI with openMP

- Summit
  - 6 GPUs, 44 CPU cores
- AMD machine
  - 8 GPUs, 128 CPU cores
- Cartesius
  - 2 GPUs, 16 CPU cores

- Distribute work over GPUs and CPUs

- …

Important lessons

- Always use teams and distribute directive to expose all available parallelism
- Use the target data directive and map clauses to minimize data movement
- Aggressively collapse loops to increase available parallelism?
  - Test and tune!
- Use accelerated libraries
- Use host fallback to generate host and device code

Resources

- Jeff Larkin (NVIDIA) – OpenMP on GPUs, First Experiences and Best Practices
- James C. Beyer (NVIDIA) et al. – Advanced OpenMP Tutorial
- www.openmp.org
  - Openmp-examples
  - Openmp-4.5

- Many examples on the web

Acknowledgments

Hands-on session

- Write a program with some loop and possibility for data movement (multiple loops with same vectors)
  - For example (or one of the openACC examples):
    - ddot=ddot+v(i)*v(i)
    - a(i)=c*b(i)+d
    - …
- Parallelize it with openMP
- Introduce target directives
- Do some timings on the CPU (use different number of threads (OMP_NUM_THREADS environment))
- Compare with different GPU timings (openACC vs openMP)

Compiling on Cartesius

- module use /sw/arch/RedHatEnterpriseServer7/EB_testing/modulefiles
- module load compiler/GCCcore/10.2.0
- gfortran/gcc –fopenmp –o name.x name.f/c (or use make with makefile)

- Interactive usage:
  - ssh gcn1
  - ./name.x
  - (check in another terminal with nvidia-smi the GPU usage)

- Batch usage
  - Make a batch script name
  - sbatch name
  - (Check with squeue on which node it runs
    - ssh to the node, and check with nvidia-smi GPU usage)

Batch script

```
#!/bin/bash
#SBATCH --job-name=name
#SBATCH --output=name.o%J
#SBATCH --time=0:30:00
#SBATCH --nodes=1
#SBATCH --reservation=prace_gpu_day_4
#SBATCH --ntasks-per-node=1

./program.x
```

- Use ‘sbatch *script*’ to submit job
- Example: /scratch/shared/ccur0054/example.tar