Directives Beyond Shared Memory

Dr. Michael K. Bane

HIGH END COMPUTE
http://highendcompute.co.uk

OMP Versions 1, 2, 3

- OpenMP formed as *the* standard for shared memory programming
  - Directives to set-up parallel regions (v1)
  - Directives to share the work (v1)
  - Directives for task-based (v3)

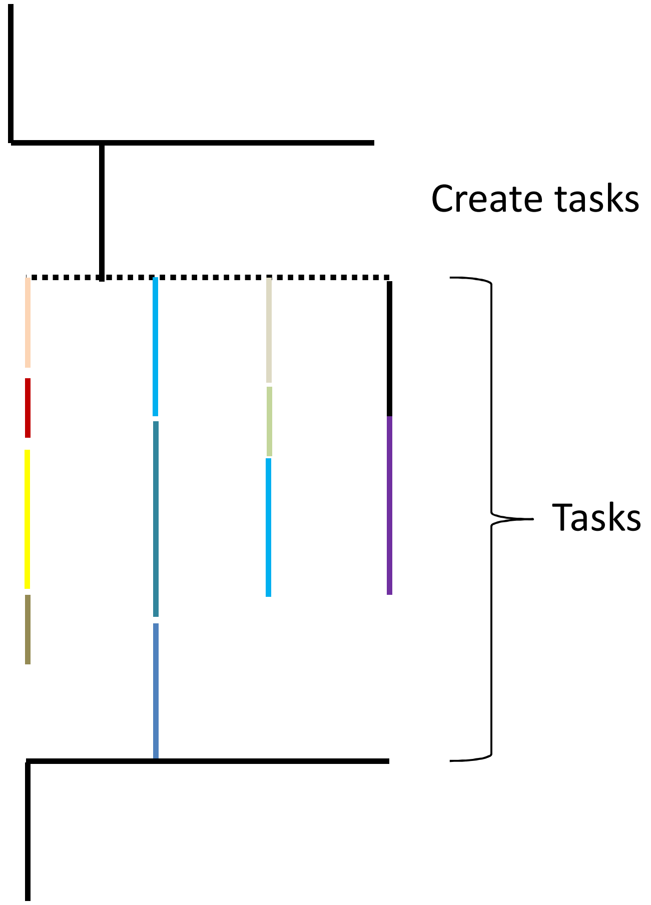
Tasks

- Quantum of independent work
  - "independent" as in *internal* work can proceed without any need for further input
- Then define the simulation as
  - Set of tasks
  - Dependency between tasks (eg DAG)
  - More of a *dataflow* approach

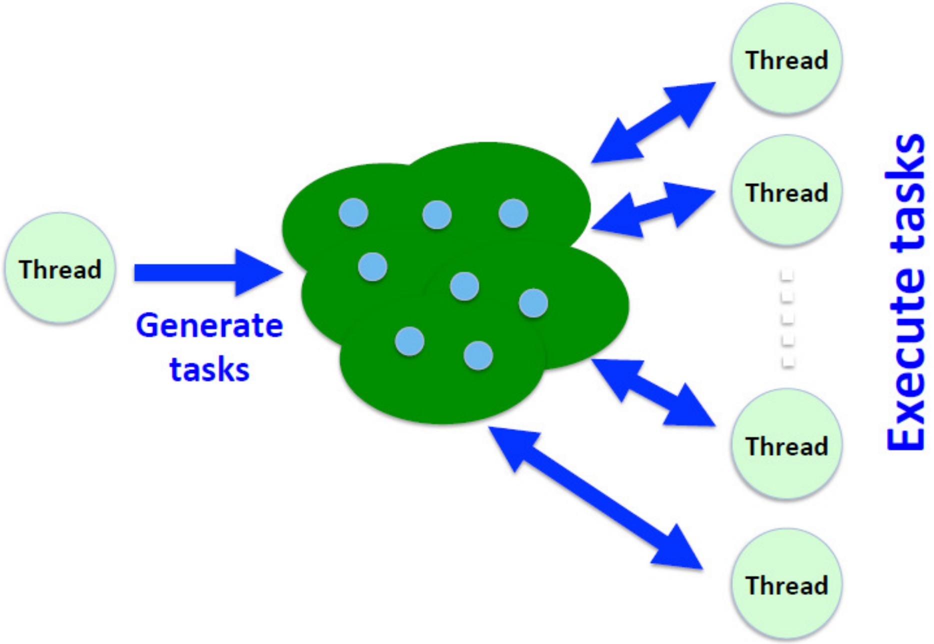
- Presuming an excellent task manager, then should get good throughput and speedup

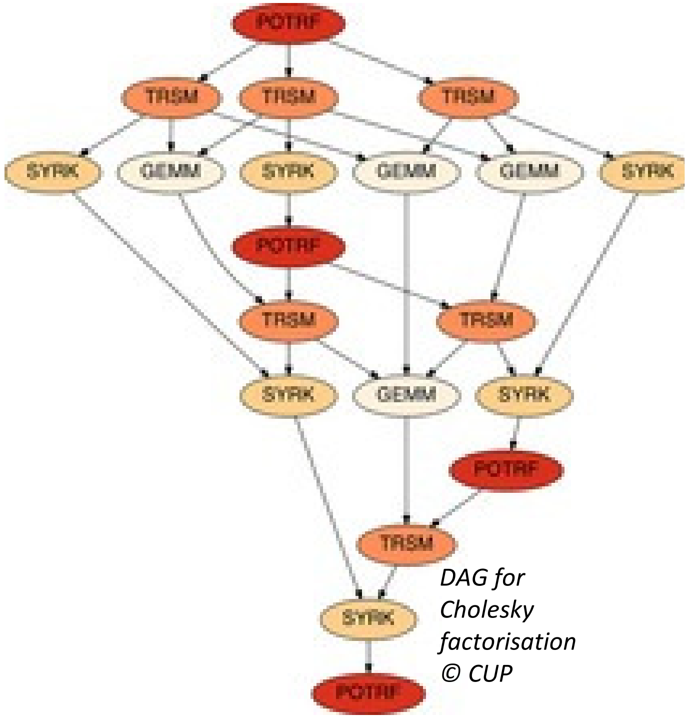
Tasks: The OpenMP Way

- Create parallel region
- Have a single thread create the tasks
- Then the tasks launch (one per thread over all threads of parallel region)



The Tasking Concept In OpenMP





DAG for Cholesky factorisation © CUP

```
#pragma omp parallel
{
#pragma omp single
{
printf(“A “);
#pragma omp task
{printf("car ");}
#pragma omp task
{printf("race ");}
#pragma omp taskwait
printf(“is fun to watch “);
}
} // End of parallel region
```

2 tasks

Synchronisation (barrier) for tasks

OpenMP Tasking Explained

Ruud van der Pas

Senior Principal Software Engineer

SPARC Microelectronics

Santa Clara, CA, USA

SC'13 Talk at OpenMP Booth

Wednesday, November 20, 2013

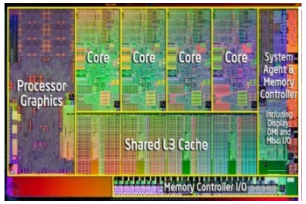
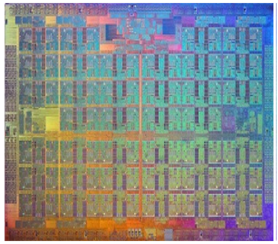
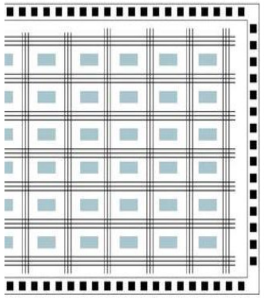
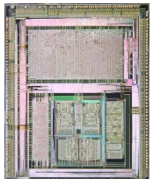
1 OpenMP Tasking Explained
Ruud van der Pas

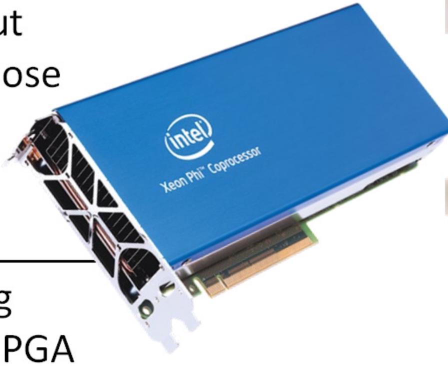
```
$ cc -xopenmp -fast hello.c
$ export OMP_NUM_THREADS=2
$ ./a.out
A car race is fun to watch
$ ./a.out
A car race is fun to watch
$ ./a.out
A race car is fun to watch
```

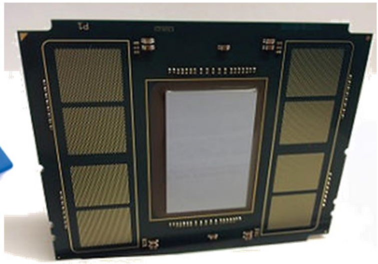
What about Accelerators?

- OpenMP 4 introduced directives to offload work to a co-processor (GPU, KNC at end of PCI-e)
- OpenMP 4.5 refined & improved

- OpenACC
  - Directives based
  - Somewhere similar to OpenMP (liked by Intel)
  - Moves more quickly, but less vendors (loved by NVIDIA)

 Core Core Core Core Processor Graphics System Agent & Memory Controller Including Display, DMI and Misc I/O Shared L3 Cache Memory Controller I/O	**CPU**	1 to maybe 64 cores, running at 2 to 3 GHz	High clock speed but general purpose
	GPU	15 to 56 "streaming multiprocessors" (SMs), each with 64-128 "CUDA Cores". Base freq about 1 GHz	Very high throughput of vector arithmetic (particularly integer)
	**Xeon Phi**	60-70 cores	Low grunt but general purpose cores
	**FPGA**	Research paper now out showing OpenMP code being pushed to FPGA (without user doing intermediate steps)	
	**ASIC**	(out of the reach of us mere mortals!)	





OpenMP Example

```
!$OMP PARALLEL DO
DO I=1, N
  Y(I) = A*X(I)*X(I) + B*X(I) + C
END DO
!$OMP END PARALLEL DO
```

CPU

```
!$OMP PARALLEL TARGET DEVICE(0) DO
DO I=1, N
  Y(I) = A*X(I)*X(I) + B*X(I) + C
END DO
!$OMP END PARALLEL DO
```

ACCELERATOR

TARGET is referring to a device (GPU or XPhi) for pushing the iterations of the DO loop. Impl dep how defined hw to DEVICE(n)

For GPU, most likely also want to use TEAMS DISTRIBUTE to make effective use of their Streaming Multiprocessors

OpenACC Example

```
!$ACC PARALLEL LOOP
DO I=1, N
  Y(I) = A*X(I)*X(I) + B*X(I) + C
END DO
!$ACC END PARALLEL LOOP
```

!$ACC Parallel directive: put it on the accelerator

!$ACC Loop directive: spread iterations over threads

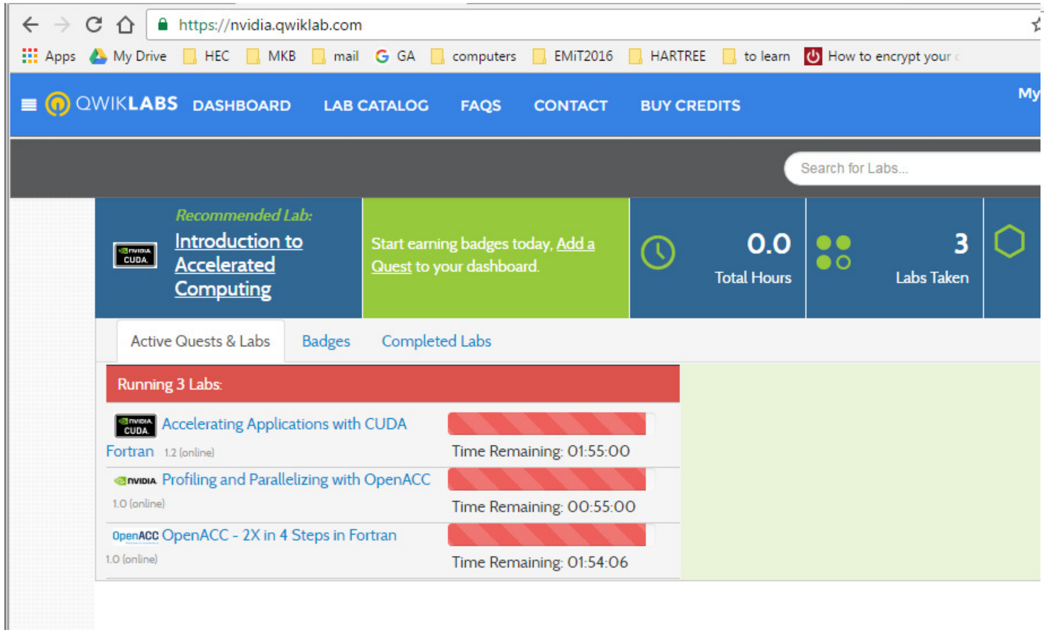
Improved efficiency by use of !$ACC Data directive (determine which IO (and can do async IO) to accelerator)

```
!$OMP PARALLEL TARGET DO
DO I=1, N
  Y(I) = A*X(I)*X(I) + B*X(I) + C
END DO
!$OMP END PARALLEL DO
```

OpenMP 4

Want to know/try more…

- "Supplementary Materials directory
- HEC qwikLabs offer
  - Choice of labs that use GPUs in AWS cloud
  - Tokens for *you*
    indicate on feedback form



Recommended Lab:
Introduction to Accelerated Computing

Start earning badges today, Add a Quest to your dashboard.

0.0 Total Hours

3 Labs Taken

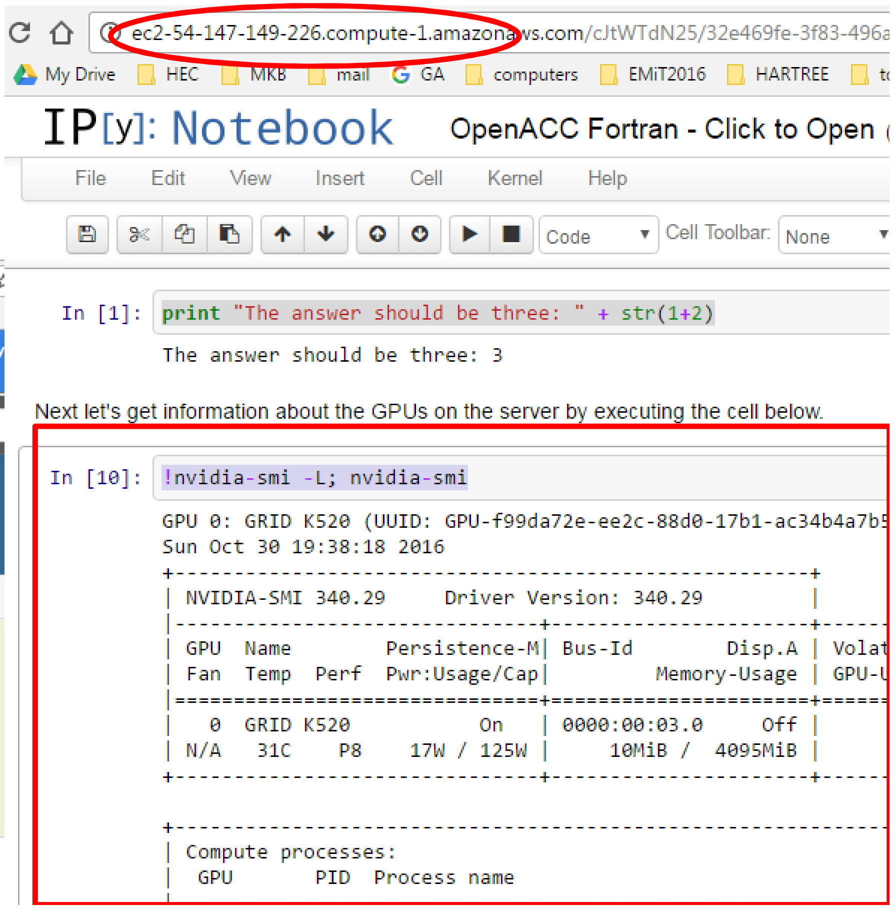
Active Quests & Labs | Badges | Completed Labs

Running 3 Labs:

Accelerating Applications with CUDA Fortran 1.2 (online) — Time Remaining: 01:55:00

Profiling and Parallelizing with OpenACC 1.0 (online) — Time Remaining: 00:55:00

OpenACC - 2X in 4 Steps in Fortran 1.0 (online) — Time Remaining: 01:54:06



```
In [1]:  print "The answer should be three: " + str(1+2)

         The answer should be three: 3
```

Next let's get information about the GPUs on the server by executing the cell below.

```
In [10]: !nvidia-smi -L; nvidia-smi

         GPU 0: GRID K520 (UUID: GPU-f99da72e-ee2c-88d0-17b1-ac34b4a7b5
         Sun Oct 30 19:38:18 2016
         +------------------------------------------------------+
         | NVIDIA-SMI 340.29     Driver Version: 340.29         |
         |-------------------------------+----------------------+------
         | GPU  Name        Persistence-M| Bus-Id        Disp.A | Volat
         | Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-U
         |===============================+======================+======
         |   0  GRID K520           On   | 0000:00:03.0     Off |
         | N/A   31C    P8    17W / 125W |     10MiB /  4095MiB |
         +-------------------------------+----------------------+------


         +-------------------------------------------------------------
         | Compute processes:
         |  GPU       PID  Process name
```