OpenACC and CRESTA

Alistair Hart
Cray Exascale Research Initiative Europe

Exascale, but not exawatts

- **Power is a big consideration in an exascale architecture**
  - Jaguar XT (ORNL) used 6MW to deliver 1Pflops
  - The US DoE wants 1Eflops, but using only 20MW...
    - That's 50 Gflops/W

- **A hybrid system is one way to reach this, e.g.**
  - 10^5 nodes (up from 10^4 for Jaguar)
  - 10^4 FPUs/node (up from 10 for Jaguar)
    - some full-featured cores for serial work
    - but a *lot* more cutdown cores for parallel work
  - Instruction level parallelism will be needed
    - continues the long-vector SIMD trend SSE → AVX → ...

- **This looks a lot like current CPU/GPU accelerator model**
  - manycore architecture, split into SIMT threadblocks with vector warps
  - Complicated memory space/hierarchy (internal and PCIe)

Piz Daint

- **No. 1 in Europe Top 500**
  - No. 6 in the world
- **Also No. 4 in Green500**
  - but Gflops/W still needs 15x

Top500 List - November 2013

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
6	Swiss National Supercomputing Centre (CSCS) Switzerland	Piz Daint - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x Cray Inc.	115984	6271.0	7788.9	2325

The Green500 List

Listed below are the November 2013 The Green500's energy-efficient supercomputers ranked from 1 to 10.

Green500 Rank	MFLOPS/W	Site*	Computer*	Total Power (kW)
1	4,503.17	GSIC Center, Tokyo Institute of Technology	TSUBAME-KFC - LX 1U-4GPU/104Re-1G Cluster, Intel Xeon E5-2620v2 6C 2.100GHz, Infiniband FDR, NVIDIA K20x	27.78
2	3,631.86	Cambridge University	Wilkes - Dell T620 Cluster, Intel Xeon E5-2630v2 6C 2.600GHz, Infiniband FDR, NVIDIA K20	52.62
3	3,517.84	Center for Computational Sciences, University of Tsukuba	HA-PACS TCA - Cray 3623G4-SM Cluster, Intel Xeon E5-2680v2 10C 2.800GHz, Infiniband QDR, NVIDIA K20x	78.77
4	3,185.91	Swiss National Supercomputing Centre (CSCS)	Piz Daint - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x Level 3 measurement data available	1,753.66

OpenACC
Directives for Accelerators

CRAY

- **A common directive programming model for accelerators**
  - Announced at SC11 conference
  - Offers portability between compilers
    - Drawn up by: NVIDIA, Cray, PGI, CAPS
    - Multiple compilers offer:
      - portability, debugging, permanence
  - Supports Fortran, C, C++
    - Standard available at openacc.org
    - Initially implementations targeted at NVIDIA GPUs

- **Compiler support: all now complete**
  - Cray CCE
  - PGI Accelerator
  - CAPS
  - gcc: work started in late 2013
  - Various other compilers in development

CAPS CRAY THE SUPERCOMPUTER COMPANY NVIDIA The Portland Group

Accelerator programming

- **Why do we need a new accelerator programming model?**
- **Aren't there enough ways to drive a GPU already?**
  - CUDA (incl. NVIDIA CUDA-C & PGI CUDA-Fortran)
  - OpenCL
  - ...
- **All are quite low-level and closely coupled to the GPU**
  - User needs to rewrite kernels in specialist language:
    - Hard to write and debug
    - Hard to optimise for specific GPU
    - Hard to port to new accelerator
  - Multiple versions of kernels in codebase
    - Hard to ensure all versions continue to function correctly
    - Hard to add new functionality

Directive-based programming

Directives provide a high-level alternative

+ **Based on original source code (Fortran, C, C++)**
  + Easier to maintain/port/extend code
  + Users with OpenMP experience find it a familiar programming model
  + Compiler handles repetitive coding (cudaMalloc, cudaMemcpy...)
  + Compiler handles default scheduling; user tunes only where needed

– **Possible performance sacrifice**
  – Important to quantify this
    – often this can be reduced by compiler improvements
  – Small performance sacrifice is an acceptable
    – trading-off portability and productivity against this
    – after all, who handcodes in assembler for CPUs these days?

Exascale architectures will be complicated

the compiler/systemware should help as much as possible

A first example

- **Execute loop nest on GPU**

- **Compiler does the work:**
  - Data movement
    - determines data use in loopnest
    - at start and end of loopnest:
      - allocates/frees GPU memory
      - moves data to/from GPU
  - Synchronisation

```
#pragma acc parallel loop
for (j=0; j<N; j++) {
  for (i=1; i<N-1; i++) {
    c[j][i] = a[j][i] + b[j][i]
  }
}
```

write-only

read-only

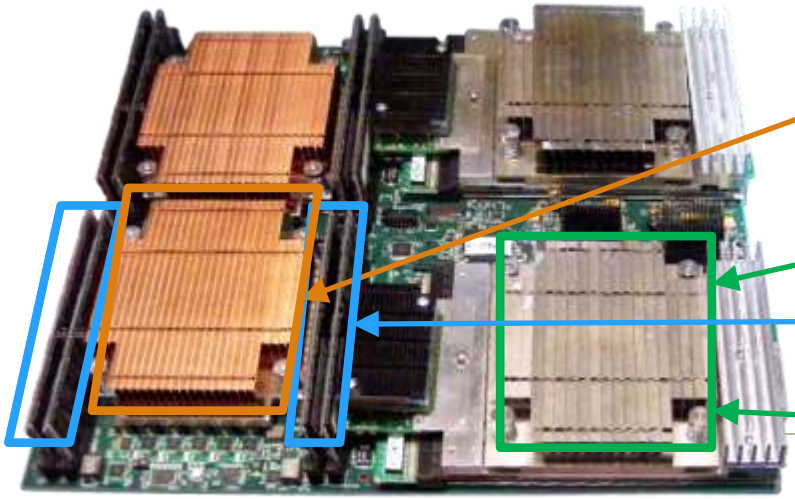
  - Loop schedule: spreading loop iterations over threads of GPU
    - OpenACC will "partition" (workshare) more than one loop in a loopnest
      - compare this to OpenMP, which only partitions the outer loop

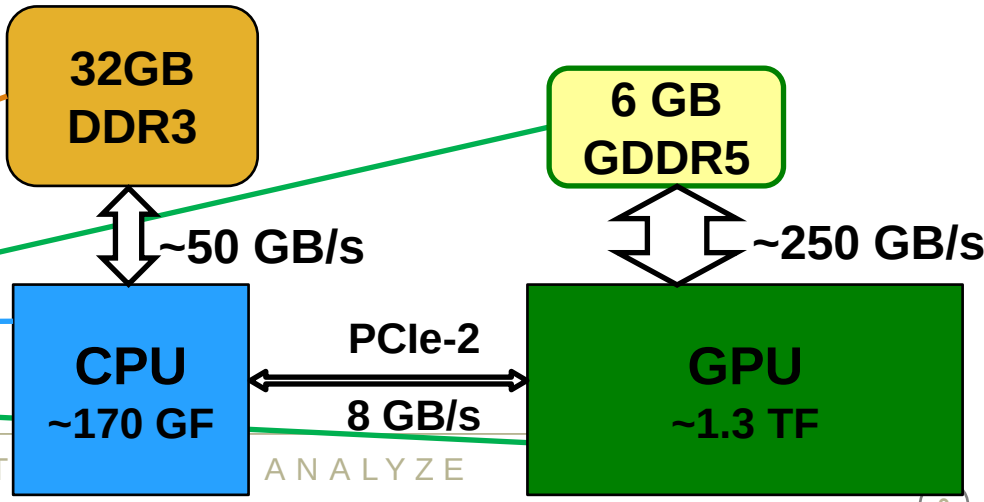
  - Caching (e.g. explicit use GPU shared memory for reused data)
    - automatic caching can be important

- User can tune all default behavior with optional clauses on directives

The main constraint in GPU programming

- **On a typical hybrid system (e.g. Cray XC30):**
  - GPU:CPU ratios: ~8x the flops, ~5x the memory bandwidth
- **The bottleneck is the CPU-GPU link (all architectures)**
  - CPU and GPU have separate memories
  - The real challenge is data locality
    - Moving data to the GPU and keeping it there for as long as possible
    - Only transferring the minimum amount of data needed
    - Overlapping data transfers with computation
- **Most time is spent thinking about data movement**
  - OpenACC allows more thinking time





Sharing GPU data between subprograms

```
PROGRAM main

!$acc data copyout(a)

!$acc parallel loop
  DO i = 1,N
   a(i) = i
  ENDDO
!$acc end parallel loop

  CALL double_array(a)
!$acc end data

END PROGRAM main
```

```
SUBROUTINE double_array(b)

!$acc parallel loop present(b)
  DO i = 1,N
   b(i) = 2*b(i)
  ENDDO
!$acc end parallel loop

END SUBROUTINE double_array
```

- **Data regions prevent "data-sloshing"**
  - present clause uses GPU-resident data without further copies

- **Planning data regions takes most of developer's time**
  - Managing the memory hierarchy is difficult (EPiGRAM)
  - This is true for CUDA as well; OpenACC makes it easier

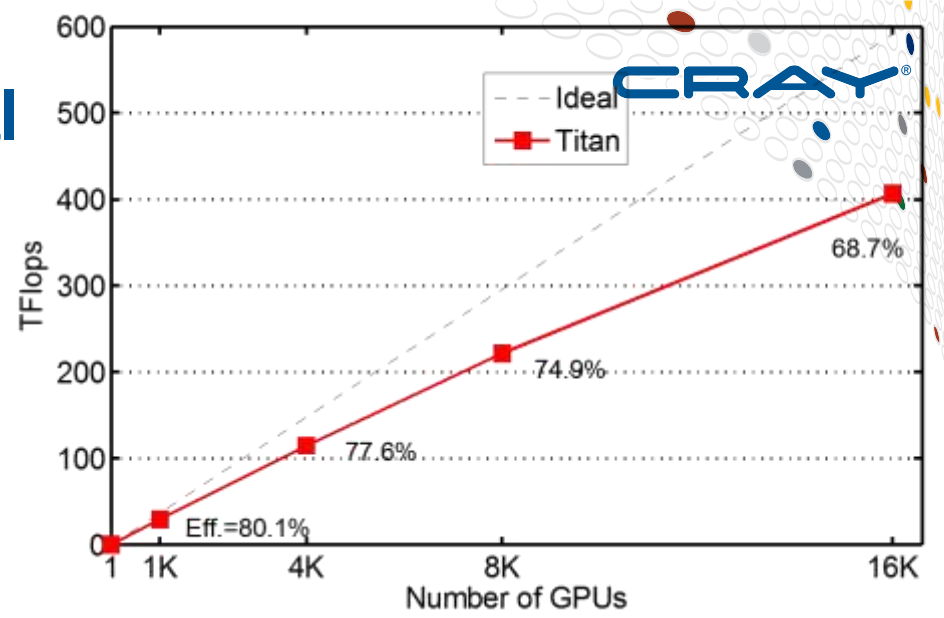
Strategic risk factors of OpenACC

- **Will there be machines to run my OpenACC code on?**
  - Now? Lots of Nvidia GPU accelerated systems
    - Cray XC30s and XK7s, plus other vendors (OpenACC is multi-vendor)
  - Future? OpenACC can be targeted at other accelerators
    - PGI and CAPS already target Intel Xeon Phi, AMD GPUs
  - Plus you can always run on CPUs using same codebase

- **Will OpenACC continue?**
  - Support? Cray, PGI, CAPS committed to support. Now gcc as well.
    - Lots of big customer pressure to continue to run OpenACC
  - Develop? OpenACC committee now 18 partners
    - v2.0 finalised in 2013, now working on next version (2.1 or 3.0)

- **Will OpenACC be superseded by something else?**
  - Auto-accelerating compilers? Yes, please! But never managed before
    - Data locality adds to the challenge
  - OpenMP accelerator directives? Immature at the moment
    - OpenACC work not wasted: thinking takes more time than coding
    - Very similar programming model; can transition when these release if wish

CRESTA: OpenACC is central



- **Co-design:**
  - Nek5000 ported with OpenACC
    - 1 directive per 1000 lines of code
    - Run on 16,000+ GPUs
      - ORNL titan system (INCITE program)
  - GROMACS:
    - Ported main kernel to OpenACC
    - interaction with R&D compiler team drove major improvements in Cray compiler
- **Driver technology:**
  - Exascale compiler studies
    - Two deliverables used OpenACC kernels as the main test codes
  - Parallel autotuning framework
    - Nek5000 OpenACC benchmark was main test-bed
  - Power monitoring investigation
    - OpenACC codes used to compare GPU energy efficiency
- **Associated activities:**
  - CRESTA tools partners support OpenACC (Allinea DDT, TUD Vampir)
  - CRESTA representation in SC12, SC13, CUG2014 tutorial presenters

Do you have any questions?