SAND2014-19833C

Designing the Future: How Successful Codesign Helps Shape Hardware and Software Development

Christian Trott

Unclassified, Unlimited release

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

CoDesign at Sandia

Programming model for hardware abstraction
- Memory abstraction: spaces, access traits, layouts
- Execution abstraction: spaces, policies

Design influenced by information about future architectures
- interaction with all vendors allows for future-safe general applicable abstractions
- concepts in place to handle platforms in 2020

Influence hardware design for better programmability
- what concepts work well for app developers
- which capabilities are missing in architectures

Influencing C++ standard to adopt successful concepts

Post CM

New technologi

Testbed

Early Access Hardware

wide range of fidelity
- cores struction level
- memory subsystem
- full system network

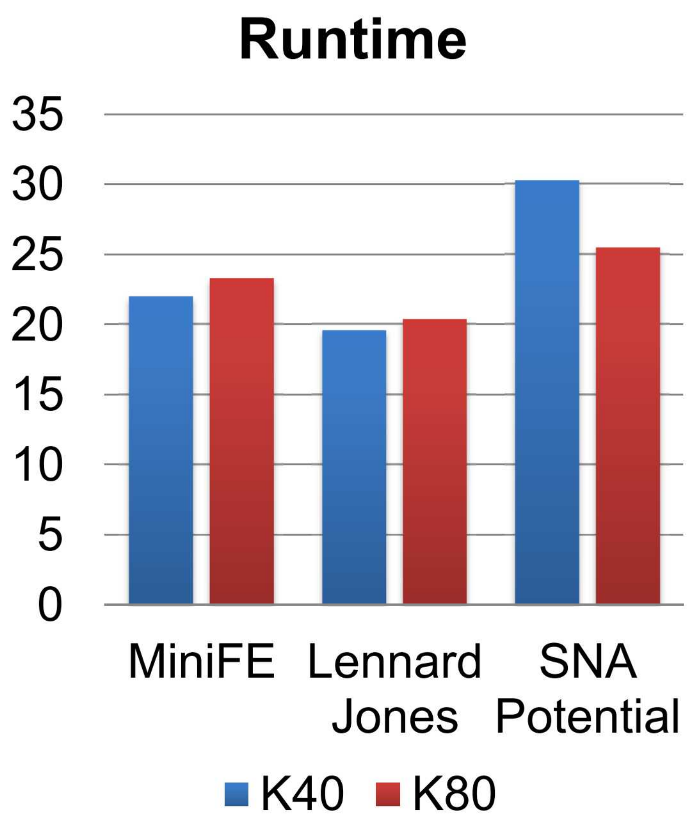
Modular design
- add new capabilities

s

Testbeds: Shannon

- Primary GPU Testbed
- 32 Dual Sandy-Bridge nodes
- QDR Infiniband
- 128 GB Ram: experiment with RAMDisk
- November 2012: 64 K20x
- November 2013: K40s
- November 2014: 8 nodes with 2xK80s

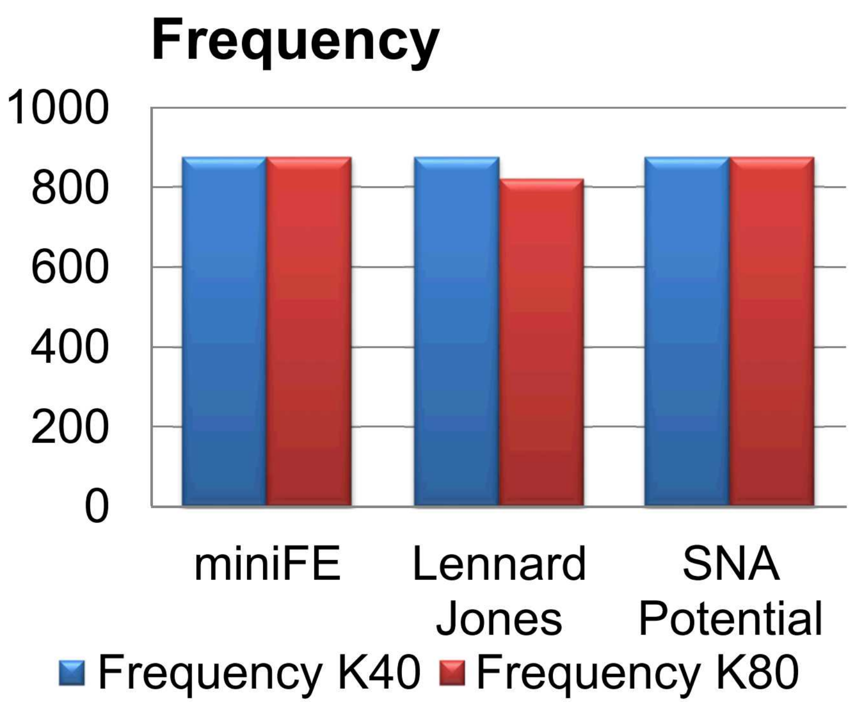
- K80 properties:
- mostly two K40s on a single board
- increased register count 2x
- increased L1/shared memory 2x
- power limit 150W per GPU

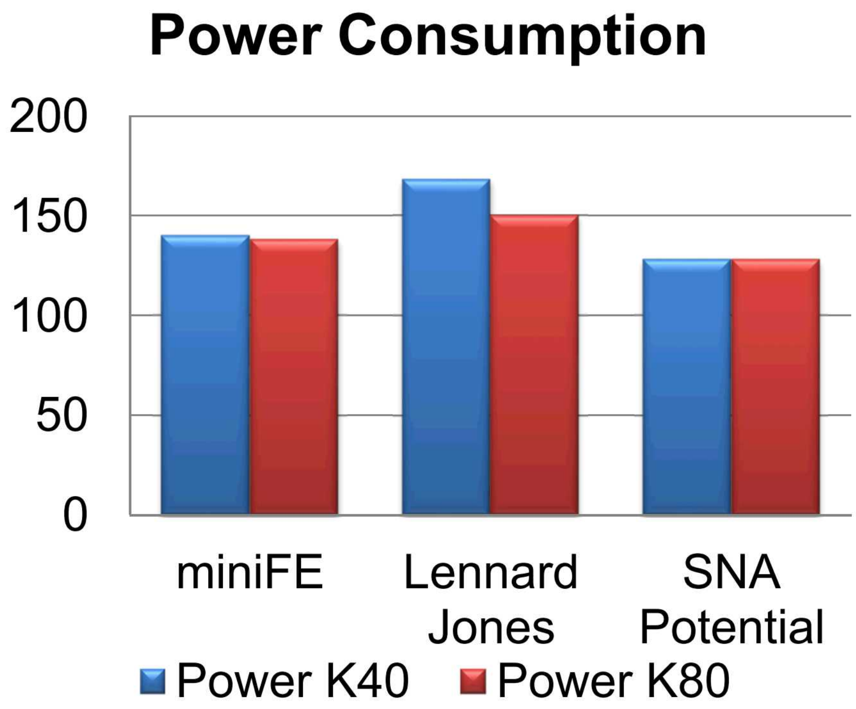


A closer look at NVIDIAs K80

Power consumption:

- on previous GPUs most applications pull significantly less than TDP
- use that knowledge to design dual GPU with no performance penalty





IBM Power 8 & NVIDIA K20x

8 nodes of dual socket Power 8
2x K20 per node
Cluster is running
CUDA 5.5 + GCC Toolchain works
A lot of other software expected on HPC platforms in early stages
-> e.g. no CUDA aware MPI
Getting CUDA applications to run relatively painless
Performance as expected (i.e. the same as on X86 based systems with K20x)
-> this is for apps running exclusively on GPUs

Goal: shake out problems with software stack now
-> ready for Power based system with NVLink in 2016

OpenACC and C++

C++ Situation 2013:

- no support for class member access
- not able to call class member functions inside kernels
- replace all members with temporaries / explicit inlining
- can't copy up class instances

```
class SomeClass {
  int a;
  int *array;
  int n;
  void compute() {
    const int n_tmp = n;
    const int a_tmp = a;
    const int array_tmp = array
    #pragma acc parallel loop pcopy(array_tmp[0:n_tmp])
    for(int i = 0; i< n_tmp ; i++) {
      array_tmp[i] = a_tmp + i;
    }
  }
```

Temporaries needed since "this" pointer not valid in kernel.

OpenACC and C++

C++ Situation now:

- worked with PGI to address issues
- possibility to “attach” arrays to classes
- class member access and inline functions work
- nested classes still problematic

```
class SomeClass {
  int a;
  int *array;
  int n;
  void compute() {
    #pragma acc parallel loop pcopy(array[0:n])
    for(int i = 0; i< n ; i++) {
      array[i] = a + i;
    }
  }
```

CUDA and C++11

Experimental, undocumented support in CUDA 6.5

- LAMBDA inside of Kernels
- auto, decltype
- variadic templates
- other misc stuff

Official support in CUDA 7.0

Enables simpler code, faster porting

Kokkos: hierarchical parallelism

```
parallel_for(TeamVectorPolicy<16>(n_bins,8), Functor());

struct Fur

  KOKKO
  void ope
    …
    paralle
      auto item_i = load_item(
      double sum

      parallel_for
        sum += C
      },sum_i);

      VectorSingle([&] ()
        accumulate(item_
      });
    });
  }
}
```

Launch 3-level parallel kernel
- teams, threads, vector (n_bins x 16 x 8)
- on GPU: teams = blocks; threads = blockDim.y; vector = blockDim.x

Loop with threads in the team over a range
- chunk on CPUs; give consecutive indicies on GPUs
- on GPU threads with same threadIdx.x get same i

Do a vector loop
- normal loop with auto vectorization form compiler on CPUs
- Split range over threads in a warp with same threadIdx.y

Questions and further discussion: crtrott@sandia.gov