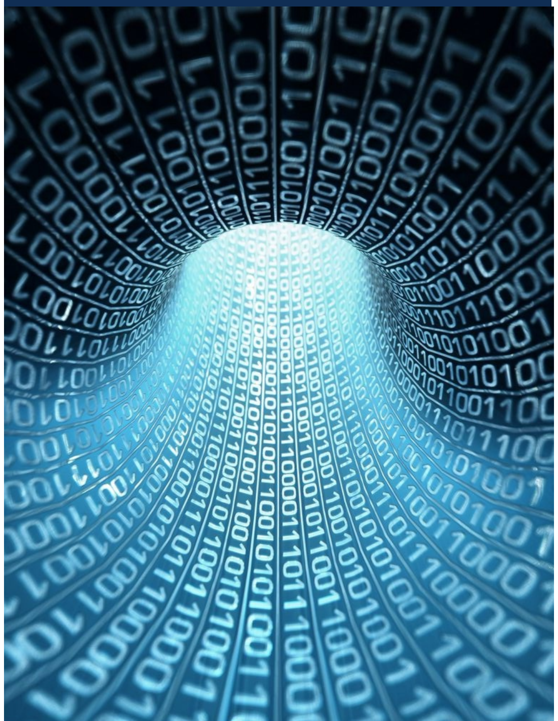


Sandia National Laboratories

Exceptional service in the national interest

Perspectives form US Department of Energy work on parallel programming models for performance portability

Jeremiah J. Wilke

Sandia National Labs

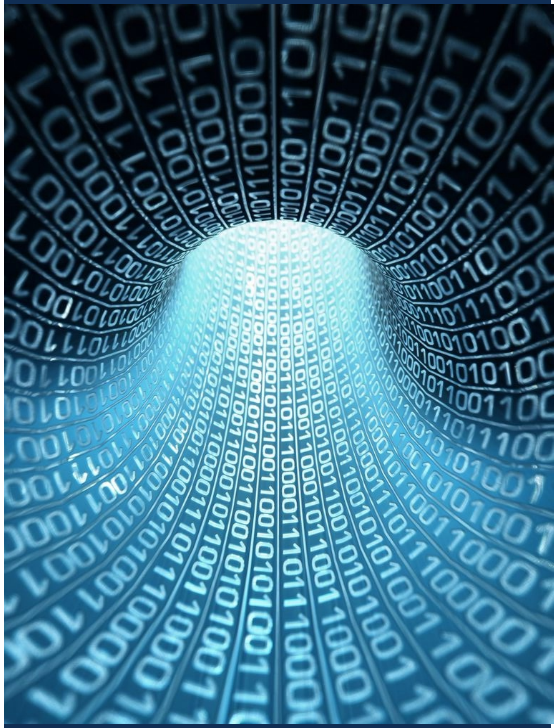
Livermore, CA

IMPACT workshop at HiPEAC

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

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. SAND NO. 2011-XXXXP



Sandia National Laboratories

Exceptional service in the national interest

Art of the deal:

How to sell your tools to DOE app developers

Jeremiah J. Wilke

Sandia National Labs

Livermore, CA

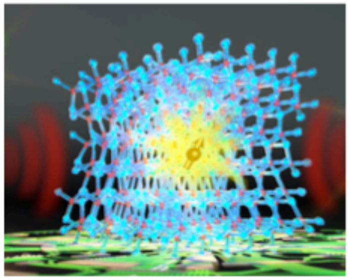
IMPACT workshop at HiPEAC

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. SAND NO. 2011-XXXXP

Very diverse set of application on very diverse set of parallel architectures poses major challenges

Sandia National Laboratories

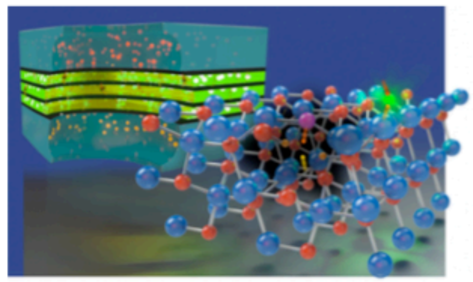
Highlights from NERSC



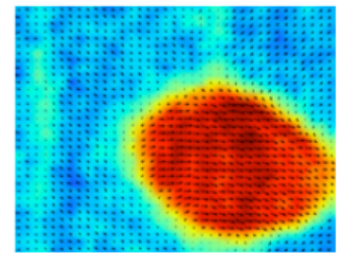
Materials Science
Calculations confirm a less expensive material for making quantum bits. **(Govoni/Galli, Univ. of Chicago, *Nature Scientific Reports*)**

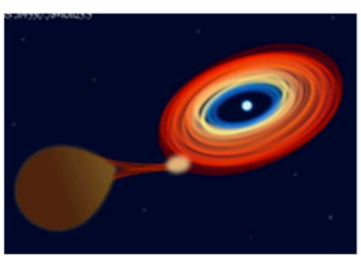
Applied Math
A new mathematical framework allows researchers to capture fluid dynamics at unprecedented detail. **(Saye, LBNL, *Science Advances*)**

Materials Science
Simulations point to more efficient light emitting diodes. **(Dreyer et al.,UC Santa Brabara, *Applied Physics Letters***

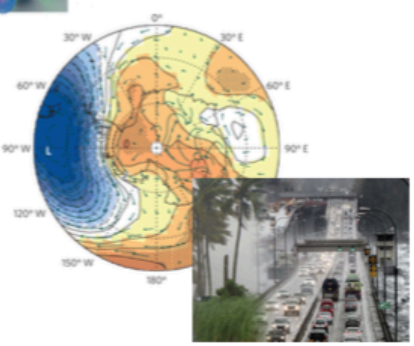


Energy/Materials
Multiscale simulations run on Edison explain behavior of ferroelectric materials. **(Rappe, Univ. of Pennsylvania, *Nature)***





High Energy Physics
Model confirms existence of an irradiated brown-dwarf companion to an accreting white dwarf. **(Baron, U. Oklahoma, *Nature*)**



Climate Research
Simulations show why Antarctic sea ice is expanding. **(Meehl, NCAR, *Nature Geoscience)***

Understanding interactions between climate and regional weather. (**Hagos, PNNL, *Journal of Climate)***

U.S. DEPARTMENT OF ENERGY | Office of Science

My worldview: find *general purpose* programming models that enable performance portability

- Funding goal: Solution to replace MPI-only that gets to exascale
- Single piece of code should run *well*, not just correctly, across many different platforms
- Many hardware trends pushing new programming models
  - GPU, CPU, KNL – loop traversal order, optimal data layouts
  - Multi-level memories – tiling, caching optimizations to utilize high-bandwidth memory, unified memory models with hardware support
  - Communication avoiding algorithms, asynchronous models to hide communication latency
- Growing number of production apps use OpenMP, Kokkos, Raja for multithreading
- Next-generation experimental codes are exploring asynchronous many-task models including Legion, Charm++, many more

DOE and collaborator efforts on performance portability

- Modern C++ (metaprogramming)
- ``Regular'' C++
- Directives
- New languages

DOE and collaborator efforts on performance portability

- Modern C++ (metaprogramming)
  - Kokkos (Sandia): Carter Edwards and Christian Trott (array views and executors, loop traversal order, data layouts, lambdas and functors)
  - Raja (Livermore): Jeff Keasler (executors, loop traversal order, data staging)
  - DARMA (Sandia): Janine Bennett (coarse-grain tasks, runtime dep analysis)
  - HPX (LSU): Hartmut Kaiser (futures, tasks, templated sync primitives)
- ``Regular'' C++
- Directives
- New languages

DOE and collaborator efforts on performance portability

- Modern C++ (metaprogramming)
  - Kokkos (Sandia): Carter Edwards and Christian Trott (array views and executors, loop traversal order, data layouts, lambdas and functors)
  - Raja (Livermore): Jeff Keasler (executors, loop traversal order, data staging)
  - DARMA (Sandia): Janine Bennett (coarse-grain tasks, runtime dep analysis)
  - HPX (LSU): Hartmut Kaiser (futures, tasks, templated sync primitives)
- ``Regular'' C++ (not so much metaprogramming)
  - UPC++ (Berkeley): Kathy Yelick (global address space)
  - Legion (Standord, Los Alamos, NVidia): McCormick, Aiken, Bauer (coarse-grain tasks, runtime dep analysis, strong data model)
  - Charm++ (UIUC): Kale (actor model)
- Directives
- New languages

DOE and collaborator efforts on performance portability

- Modern C++ (metaprogramming)
  - Kokkos (Sandia, Edwards), Raja (Livermore, Keasler): executors, loop traversal order, data layouts, lambdas and functors)
  - DARMA (Sandia): Janine Bennett (coarse-grain tasks, runtime dep analysis)
  - HPX (LSU): Hartmut Kaiser (futures, tasks, templated sync primitives)
- ``Regular'' C++
  - UPC++ (Berkeley): Kathy Yelick (global address space)
  - Legion (Standord, Los Alamos, NVidia): McCormick, Aiken, Bauer (coarse-grain tasks, runtime dep analysis, strong data model)
  - Charm++ (UIUC): Kale (actor model)
- Directives:
  - OpenMP
  - OmpSs (Barcelona)
- New languages

DOE and collaborator efforts on performance portability

- Modern C++ (metaprogramming)
  - Kokkos (Sandia, Edwards), Raja (Livermore, Keasler): executors, loop traversal order, data layouts, lambdas and functors)
  - DARMA (Sandia): Janine Bennett (coarse-grain tasks, runtime dep analysis)
  - HPX (LSU): Hartmut Kaiser (futures, tasks, templated sync primitives)
- ``Regular'' C++
  - UPC++ (Berkeley): Kathy Yelick (global address space)
  - Legion (Standord, Los Alamos, NVidia): McCormick, Aiken, Bauer (coarse-grain tasks, runtime dep analysis, strong data model)
  - Charm++ (UIUC): Kale (actor model)
- Directives: OpenMP, OmpSs
- New languages:
  - Chapel (Cray): global address space language
  - Regent (Stanford): expressive interface to Legion based on Terra, Lua

DOE and collaborator efforts on performance portability

- Modern C++ (metaprogramming)
  - Kokkos (Sandia, Edwards), Raja (Livermore, Keasler): executors, loop traversal order, data layouts, lambdas and functors)
  - DARMA (Sandia): Janine Bennett (coarse-grain tasks, runtime dep analysis)
  - HPX (LSU): Hartmut Kaiser (futures, tasks, templated sync primitives)
- ``Regular'' C++
  - UPC++ (Berkeley): Kathy Yelick (global address space)
  - Legion (Standord, Los Alamos, NVidia): McCormick, Aiken, Bauer (coarse-grain tasks, runtime dep analysis, strong data model)
  - Charm++ (UIUC): Kale (actor model)
- Directives: OpenMP, OmpSs
- New languages:
  - Chapel (Cray): global address space language
  - Regent (Stanford): expressive interface to Legion based on Terra, Lua
- Some polyhedral work R-stream/OCR, OSU+ Lawrence, Utah

DOE and collaborator efforts on performance portability

- Modern C++ (metaprogramming)
  - Kokkos (Sandia, Edwards), Raja (Livermore, Keasler): executors, loop traversal
  - alysis)
  - imitives)
- `

These are all general purpose programming models!

  -
  - arse-
  -
- Directives: OpenMP, OmpSs
- New languages:
  - Chapel (Cray): global address space language
  - Regent (Stanford): expressive interface to Legion based on Terra, Lua
- Some polyhedral work R-stream + OCR, Ohio State + Lawrence

Programming models are economics and sociology: Maximize programming *productivity* and *happiness*

- Thought experiment #1: If a single computer architecture existed, would apps teams learn and write assembly?

Programming models are economics and sociology: Maximize programming *productivity* and *happiness*

- Thought experiment #1: If a single computer architecture existed, would apps teams learn and write assembly?
  - No, compilers are necessary for productivity even without any concerns for portability of code

Programming models are economics and sociology: Maximize programming *productivity* and *happiness*

- Thought experiment #1: If a single computer architecture existed, would apps teams learn and write assembly?
  - No, compilers are necessary for productivity even without any concerns for portability of code
- Thought experiment #2: If single parallel architecture existed, would apps teams want polyhedral compilers?

Programming models are economics and sociology: Maximize programming *productivity* and *happiness*

- Thought experiment #1: If a single computer architecture existed, would apps teams learn and write assembly?
  - No, compilers are necessary for productivity even without any concerns for portability of code
- Thought experiment #2: If single parallel architecture existed, would apps teams want anything but MPI + OpenMP?
- Parallel app developers are experts in *problem decomposition*
  - MPI not really that burdensome to write – only annoying if different architecture demands different problem decompositions
  - Kokkos,Raja,OpenMP: Not perfect, but not rate-limiting step in development
  - Vectorization? Need broader survey of what developers find difficult
  - Apps developers like controlling details of execution

Programming models are economics and sociology: Maximize programming *productivity* and *happiness*

Sandia National Laboratories

- Thought experiment #1: If a single computer architecture existed, would apps teams learn and write assembly?
  - No, compilers are necessary for productivity even without any concerns for portability of code
- Thought experiment #2: If single parallel architecture existed, would apps teams want anything but MPI + OpenMP?
- Parallel app developers are experts in *problem decomposition*
  - MPI not really that burdensome to write – only annoying if different architecture demands different problem decompositions
  - Kokkos,Raja,OpenMP: Not perfect, but not rate-limiting step in development
  - Vectorization? Need broader survey of what developers find difficult
  - Apps developers like controlling details of execution
- *Performance portability* changes calculus – the ``buzz word'' that will get app developer's attention

Conclusions: What are possible outcomes? And what can you do to influence outcomes?

- Outcome #1: DOE ports all of its codes to new general purpose runtime (might just be MPI + OpenMP)
- Outcome #2: General purpose loses – quicker to just build domain-specific runtimes and models
- Outcome #1 is *programmatically* preferred, DOE doesn’t want a bunch of ad hoc technologies lumped together
- Path forward:
  - Sell the product – current solutions not breeding mass discontent
  - Don’t oversell the product
  - Compare compiler tools to general purpose programming models (particularly Legion, Kokkos, Raja)
  - Use “performance portability” as an ice-breaker