# Agenda

- Mike Heroux, Sandia, HPCG Performance Tuning Overview
- Yutong Lu, NUDT, Tianhe–2 Efforts.
- Kiyoshi Kumahata, RIKEN, K Machine Efforts.
- Massimiliano Fatica, Nvidia, Nvidia Efforts.
- Jongsoo Park, Intel, Intel Efforts
- Audience Discussion
- Jack Dongarra, Piotr Luszczek, Mike Heroux, Announcement of Results, Awards.

hpcg-benchmark.org

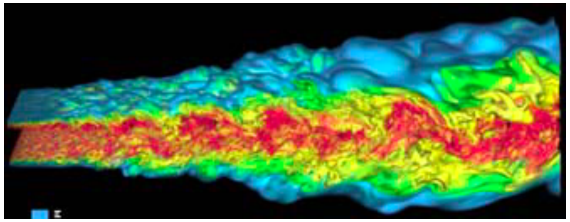
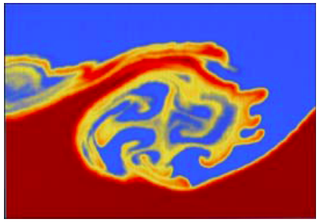
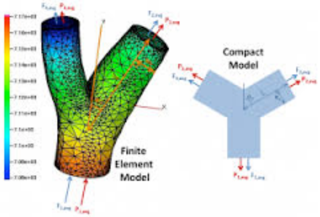
# HPCG: TOWARD A NEW (OR ANOTHER) METRIC FOR RANKING HIGH PERFORMANCE COMPUTING SYSTEMS

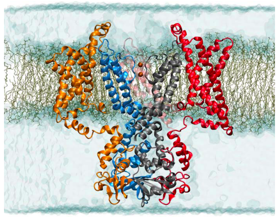

Jack Dongarra & Piotr Luszczek
University of Tennessee/ORNL

Michael Heroux
Sandia National Labs

# Goals for New Benchmark

- Augment the TOP500 listing with a benchmark that correlates with important scientific and technical apps not well represented by HPL



- Encourage vendors to focus on architecture features needed for high performance on those important scientific and technical apps.
  - Stress a balance of floating point and communication bandwidth and latency
  - Reward investment in high performance collective ops
  - Reward investment in high performance point-to-point messages of various sizes
  - Reward investment in local memory system performance
  - Reward investment in parallel runtimes that facilitate intra-node parallelism
- Provide an outreach/communication tool
  - Easy to understand
  - Easy to optimize
  - Easy to implement, run, and check results
- Provide a historical database of performance information
  - The new benchmark should have longevity

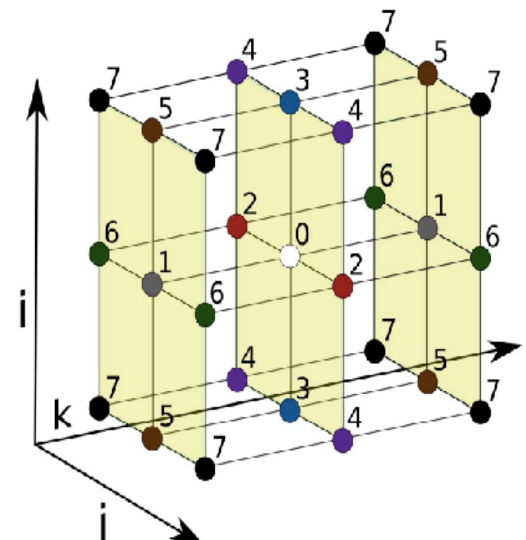
# Proposal: HPCG

- High Performance Conjugate Gradient (HPCG).
- Solves $Ax=b$, $A$ large, sparse, $b$ known, $x$ computed.
- An optimized implementation of PCG contains essential computational and communication patterns that are prevalent in a variety of methods for discretization and numerical solution of PDEs

- Patterns:
  - Dense and sparse computations.
  - Dense and sparse collective.
  - Multi-scale execution of kernels via MG (truncated) V cycle.
  - Data-driven parallelism (unstructured sparse triangular solves).
- Strong verification and validation properties (via spectral properties of PCG).

# Model Problem Description

- Synthetic discretized 3D PDE (FEM, FVM, FDM).
- Single DOF heat diffusion model.
- Zero Dirichlet BCs, Synthetic RHS s.t. solution = 1.
- Local domain: $(n_x \times n_y \times n_z)$
- Process layout: $(np_x \times np_y \times np_z)$
- Global domain: $(n_x * np_x) \times (n_y * np_y) \times (n_z * np_z)$
- Sparse matrix:
    - 27 nonzeros/row interior.
    - 8 – 18 on boundary.
    - Symmetric positive definite.




27-point stencil operator

# HPCG Design Philosophy

- Relevance to broad collection of important apps.
- Simple, single number.
- Few user-tunable parameters and algorithms:
  - The system, not benchmarker skill, should be primary factor in result.
  - Algorithmic tricks don't give us relevant information.
- Algorithm (PCG) is vehicle for organizing:
  - Known set of kernels.
  - Core compute and data patterns.
  - Tunable over time (as was HPL).
- Easy-to-modify:
  - _ref kernels called by benchmark kernels.
  - User can easily replace with custom versions.
  - Clear policy: Only kernels with _ref versions can be modified.

# PCG ALGORITHM

- $p_0 := x_0,\ r_0 := b\text{-}Ap_0$
- Loop $i = 1, 2, \ldots$
  - $z_i := M^{-1} r_{i-1}$
  - if $i = 1$
    - $p_i := z_i$
    - $\alpha_i :=$ `dot_product`$(r_{i-1}, z)$
  - else
    - $\alpha_i :=$ `dot_product`$(r_{i-1}, z)$
    - $\beta_i := \alpha_i / \alpha_{i-1}$
    - $p_i := \beta_i * p_{i-1} + z_i$
  - end if
  - $\alpha_i :=$ `dot_product`$(r_{i-1}, z_i)$ /`dot_product`$(p_i, A * p_i)$
  - $x_{i+1} := x_i + \alpha_i * p_i$
  - $r_i := r_{i-1} - \alpha_i * A * p_i$
  - if $\|r_i\|_2 <$ tolerance then Stop
- end Loop

# Execution: 7 Phases

## Problem Setup

- Construct Geometry.
- Generate Problem.
- Setup Halo Exchange.
- Initialize Sparse Meta-data.
- Call user-defined OptimizeProblem function. This function permits the user to change data structures and perform permutation that can improve execution.

## Validation Testing

- Perform spectral properties PCG Tests:
  - Convergence for 10 distinct eigenvalues:
    - No preconditioning.
    - With Preconditioning
- Symmetry tests:
  - Sparse MV kernel.
  - MG kernel.

## Reference Sparse MV and Gauss-Seidel kernel timing.

- Time calls to the reference versions of sparse MV and MG for inclusion in output report.

## Reference CG timing and residual reduction.

- Time the execution of 50 iterations of the reference PCG implementation.
- Record reduction of residual using the reference implementation. The optimized code must attain the same residual reduction, even if more iterations are required.

## Optimized CG Setup.

- Run one set of Optimized PCG solver to determine number of iterations required to reach residual reduction of reference PCG.
- Record iteration count as **numberOfOptCgIters.**
- Detect failure to converge.
- Compute how many sets of Optimized PCG Solver are required to fill benchmark timespan. Record as **numberOfCgSets**

## Optimized CG timing and analysis.

- Run **numberOfCgSets** calls to optimized PCG solver with **numberOfOptCgIters** iterations.
- For each set, record residual norm.
- Record total time.
- Compute mean and variance of residual values.

## Report results

- Write a log file for diagnostics and debugging.
- Write a benchmark results file for reporting official information.

# Problem Setup

- Construct Geometry.
- Generate Problem.
- Setup Halo Exchange.
  - Use symmetry to eliminate communication in this phase.
  - C++ STL containers/algorithms: Simple code, force use of C++.
- Initialize Sparse Meta-data.

# Validation Testing

- Temporarily modify matrix diagonals:
  - (2.0e6, 3.0e6, … 9.0e6, 1.0e6, …1.0e6).
  - Offdiagonal still -1.0.
  - Matrix looks diagonal with 10 distinct eigenvalues.
- Perform spectral properties PCG Tests:
  - Convergence for 10 distinct eigenvalues:
    - No preconditioning: About 10 iters.
    - With Preconditioning: About 1 iter.
- Symmetry tests:
  - Matrix, preconditioner are symmetric.
  - Sparse MV kernel.
  - MG kernel.

$$x^T A y = y^T A x$$

$$x^T M^{-1} y = y^T M^{-1} x$$

# Reference Sparse MV and Gauss-Seidel kernel timing.

- Time calls to the reference versions of sparse MV and MG for inclusion in output report.

# Reference CG timing and residual reduction.

- Time the execution of 50 iterations of the reference CG implementation.
- Record reduction of residual using the reference implementation.
- The optimized code must attain the same residual reduction, *even if more iterations are required.*
  - Most graph coloring algorithms improve parallel execution at the expense of increasing iteration counts.

# Optimized CG Setup.

- Call user-defined OptimizeProblem function.
  - Permits the user to change data structures and perform permutation that can improve execution.
- Run one set of Optimized PCG solver to determine number of iterations required to reach residual reduction of reference PCG.
- Record iteration count as **numberOfOptCgIters.**
- Detect failure to converge.
- Compute how many sets of Optimized PCG Solver are required to fill benchmark timespan. Record as **numberOfCgSets**

# Optimized CG timing and analysis.

- Run **numberOfCgSets** calls to optimized PCG solver with **numberOfOptCgIters** iterations.
- For each set, record residual norm.
- Record total time.
- Compute mean and variance of residual values.

# Report results

- Write a log file for diagnostics and debugging.
- Write a benchmark results file for reporting official information.

# What can be optimized? _ref

CG_ref.hpp
ComputeDotProduct_ref.hpp
ComputeMG_ref.hpp
ComputeProlongation_ref.hpp
ComputeRestriction_ref.hpp
ComputeSPMV_ref.hpp
ComputeSYMGS_ref.hpp
ComputeWAXPBY_ref.hpp
GenerateLevelMatrix_ref.hpp

# Key Computation Data Patterns

- Domain decomposition:
  - SPMD (MPI): Across domains.
  - Thread/vector (OpenMP, compiler): Within domains.
- Vector ops:
  - AXPY: Simple streaming memory ops.
  - DOT/NRM2 : Blocking Collectives.
- Matrix ops:
  - SpMV: Classic sparse kernel (option to reformat).
  - Symmetric Gauss-Seidel: sparse triangular sweep.
    - Exposes real application tradeoffs:
      - threading & convergence vs. SPMD and scaling.
    - Enables leverage of new parallel patterns, e.g., futures.

# Merits of HPCG

- Includes major communication/computational patterns.
  - Represents a minimal collection of the major patterns.
- Rewards investment in:
  - High-performance collective ops.
  - Local memory system performance.
  - Low latency cooperative threading.
- Detects/measures variances from bitwise reproducibility.
- Executes kernels at several (tunable) granularities:
  - nx = ny = nz = 104 gives
  - nlocal = 1,124,864; 140,608; 17,576; 2,197
  - ComputeSymGS with multicoloring adds one more level:
    - 8 colors.
    - Average size of color = 275.
    - Size ratio (largest:smallest): 4096
  - Provide a “natural” incentive to run a big problem.

# Impact of HPCG design points

- Global collective:
  - Large variation in runtimes on some networks.
  - Limits performance on several systems.
- Neighborhood collective:
  - Significant impact on one system's results.
  - Does not impact HPL performance.
- Gauss-Seidel kernel:
  - High throughput variants do more iterations: 58 vs. 50.
- Significant variation vs. HPL (come tomorrow).

- **Note: One additional design consideration:**
  - True finite volume/element construction.
    - Higher flop, int instruction rates.
    - Some interesting computational, data access patterns.
    - May make HPCG more representative for some apps.

# HPCG and HPL

- We are NOT proposing to eliminate HPL as a metric.
- The historical importance and community outreach value is too important to abandon.
- HPCG will serve as an alternate ranking of the Top500.
  - Similar perhaps to the Green500 listing.

# HPL vs. HPCG: Bookends

- Some see HPL and HPCG as “bookends” of a spectrum.
  - Applications teams know where their codes lie on the spectrum.
  - Can gauge performance on a system using both HPL and HPCG numbers.

# Signs of Uptake

- Discussions with and results from every vendor.
- Major, deep technical discussions with several.
- Same with most LCFs.
- SC’14 BOF on Optimizing HPCG.
- One ISC’14 and two SC’14 papers submitted.
  - Nvidia and Intel.
- Optimized results for MIC-based, Nvidia GPU-based systems.
- Increase from 15 to 25 systems on the list.
- Improved numbers for many previous systems.

# Versions of HPCG

- Reference version on GitHub:
  - https://github.com/hpcg-benchmark/hpcg
  - Website:  hpcg-benchark.org
  - Mail list hpcg.benchmark@gmail.com
- Intel
  - MKL has packaged CPU version of HPCG
    - See: http://bit.ly/hpcg-intel
  - In the process of packaging Xeon Phi version.  Released?
- Nvidia
- Bull

# HPCG Tech Reports

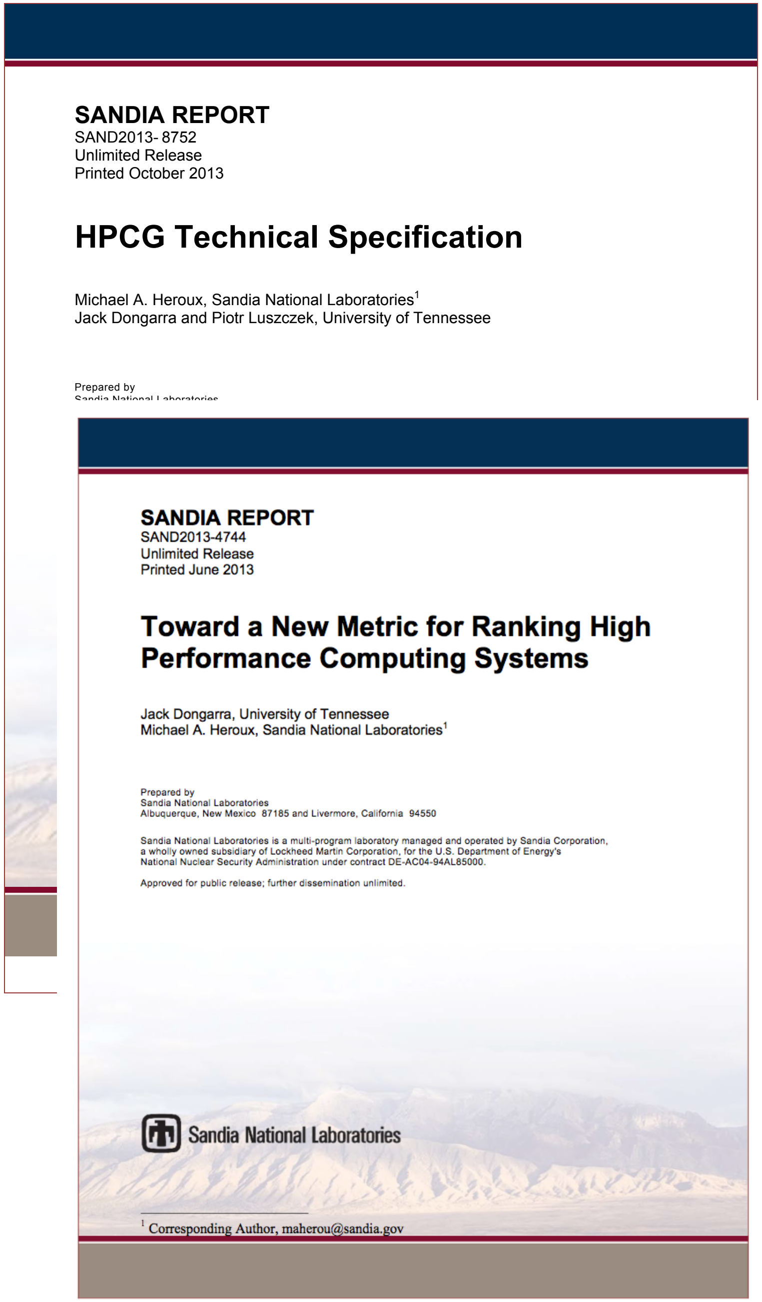
*Toward a New Metric for Ranking High Performance Computing Systems*

- Jack Dongarra and Michael Heroux

*HPCG Technical Specification*

- Jack Dongarra, Michael Heroux, Piotr Luszczek

- hpcg-benchmark.org



**SANDIA REPORT**
SAND2013- 8752
Unlimited Release
Printed October 2013

**HPCG Technical Specification**

Michael A. Heroux, Sandia National Laboratories[1]
Jack Dongarra and Piotr Luszczek, University of Tennessee

Prepared by

**SANDIA REPORT**
SAND2013-4744
Unlimited Release
Printed June 2013

**Toward a New Metric for Ranking High Performance Computing Systems**

Jack Dongarra, University of Tennessee
Michael A. Heroux, Sandia National Laboratories[1]

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

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.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

[1] Corresponding Author, maherou@sandia.gov