Best Practices for Using Proxy Apps as Benchmarks

Approved for public release

EXASCALE COMPUTING PROJECT

IDEAS Webinar

April 15, 2020

David Richards, LLNL

Joe Glenski, HPE

National Nuclear Security Administration

LLNL-PRES-808399

U.S. DEPARTMENT OF ENERGY | Office of Science

The presentation contains the work of many co-authors

- Jeanine Cook (SNL)
- Jeffery Kuehn (LANL)
- Omar Aaziz (SNL)
- Courtenay Vaughan (SNL)
- Hal Finkel (ANL)
- The ECP Proxy App Team
- Oscar Hernandez (ORNL)
- Verónica Vergara (ORNL)
- Reuben D. Budiardja (ORNL)
- Bronson Messer (ORNL)
- Jack Wells (ORNL)
- Wayne Joubert (ORNL)
- Swen Boehm (ORNL)

This talk is an abridged version of a breakout session originally presented at the 2020 ECP Annual Meeting

https://proxyapps.exascaleproject.org/reports/

Today's talk is full of proxy app and benchmark goodness

For Developers of Proxy Apps & Benchmarks	For Users of Proxy Apps & Benchmarks	For Computing Facilities
• What is needed to make a proxy app into a benchmark? • How do I make my benchmark attractive for users? • How can I reduce the chances that my proxy app will be misused? • How can I quantify the fidelity of my proxy app or benchmark?	• How do I know if a proxy app is a good benchmark? • What do I need to know about using proxy apps? • How can I choose a good set of benchmarks? • Where can I find good benchmarks?	• How can I use benchmarks to get good information from vendors? • How can I tell if benchmarks really represent my workload? • How can I decide which benchmarks to include in my suite?

Benchmarks are sample workloads intended to quantify and compare different aspects of system performance

- Frequently used to guide system design and/or purchasing decisions
- Workloads range from a few lines of code to entire production applications
- Benchmark collections cover a huge array of workloads
  - SPEC: CPU, GPU, Cloud, MPI, OpenMP, etc.
  - NAS parallel benchmarks
  - DOE procurement benchmarks
- Effective benchmarks need a way to quantify the results and ensure results are comparable between users

Proxy applications are models for one or more features of a parent application

- Proxy apps omit many features of parent apps
- Proxy apps come in various sizes
  - Kernels, skeleton apps, mini apps
- Proxies can be models for
  - Performance critical algorithms
  - Communication patterns
  - Programming models and styles
- Like any model, proxies can be misused beyond their regime of validity

Many benchmarks are proxy apps
Proxy apps are not automatically good benchmarks

Beware: models are easy to mis-use

Some blame lies with developers

- Proxies are often widely published even when they are originally intended for internal use
- Developers need to be more clear which proxies make good benchmarks (and what inputs to use)
- Better documentation that is easier to digest is usually needed to help guide researchers
- Verification and reproducibility are frequently not considered as part of proxy design
- Writing code is fun
  Writing documentation is not

But proxy app users aren't innocent

- Proxies are relatively easy to build and run without devoting much thought to the process
- Proxy users aren't always familiar with caveats and limitations of proxies
- Many papers and reports present proxy app performance information without describing input parameters
- Sensitivity analysis is rare
- Did you verify performance expectations or correctness?

An understanding of what you are using and why it's important are essential when using proxy apps

A proxy app becomes a benchmark when it is matched with:

A Figure of Merit (FOM)

- An FOM is a measure of application throughput performance
- Good FOMs usually scale with performance
  - 2X problem run 2X faster (than 1X problem on old platform) = 4X FOM
  - 1X problem run 4X faster = 4X FOM
  - FOM may need to consider application algorithm scaling with system size

A Set of Run Rules

- Run rules may include:
  - Problem specification
  - Code version
  - Weak or strong scaling constraints
  - Allowable code modifications
  - Wall time constraints
  - Misc limits such as memory per MPI rank, node count(s) to run jobs on, etc.

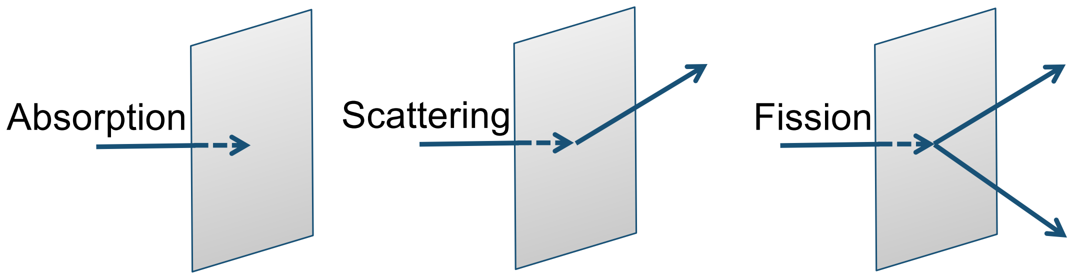
Unless the FOM and run rules are chosen carefully the benchmark may be meaningless

Best practices for effective proxy apps and benchmarks

For Developers of Proxy Apps & Benchmarks	For Users of Proxy Apps & Benchmarks	For Computing Facilities
• Write documentation • Ensure benchmark run rules address issues of scalability, fidelity, ease of use, etc. • Make it easy to identify the figure of merit • Provide a method to verify correct results	• Read documentation • Don't assume every proxy app is useful as a benchmark • Remember that benchmarks have well-defined run rules and a figure of merit • DOE system procurement suites can be a good place to look for benchmark problems	• Avoid large input or output files and complex library dependencies • Make benchmark suites easy to build and automate • Cover all aspects of the ecosystem: Programming models, compilers, debuggers, performance tools

Quicksilver is a proxy for Mercury (Monte Carlo transport)

- Particles interact with matter by a variety of “reactions”



- The probability of each reaction and its outcomes are captured in experimentally measured “cross sections” (Latency bound table lookups)
- Follows many particles (millions or more) and uses random numbers to sample the probability distributions (Very branchy, divergent code)
- Particles contribute to diagnostic “tallies” (Potential data races)

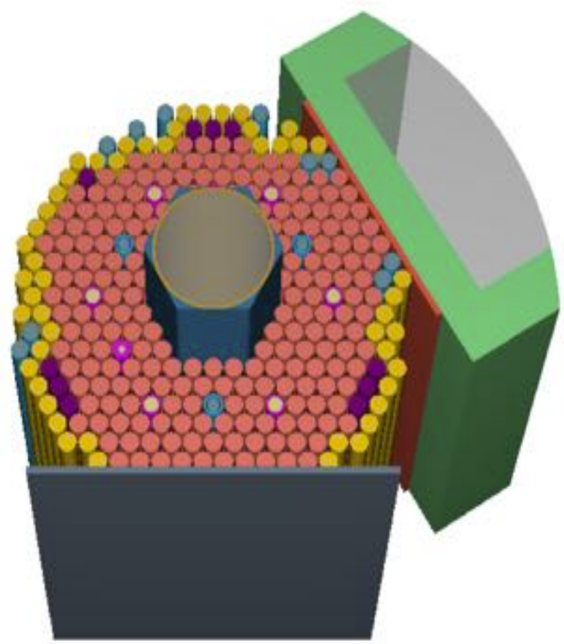
Quicksilver attempts to capture these key traits of Mercury

Defining a good Quicksilver benchmark problem is very challenging

Challenges

- **Huge variation in scale:**
  Benchmark must be equally valid on 1 node or 10,000 nodes
- **Simulation geometry:**
  Any geometry that resembles production use will be difficult to scale

Imagine trying to scale this model!



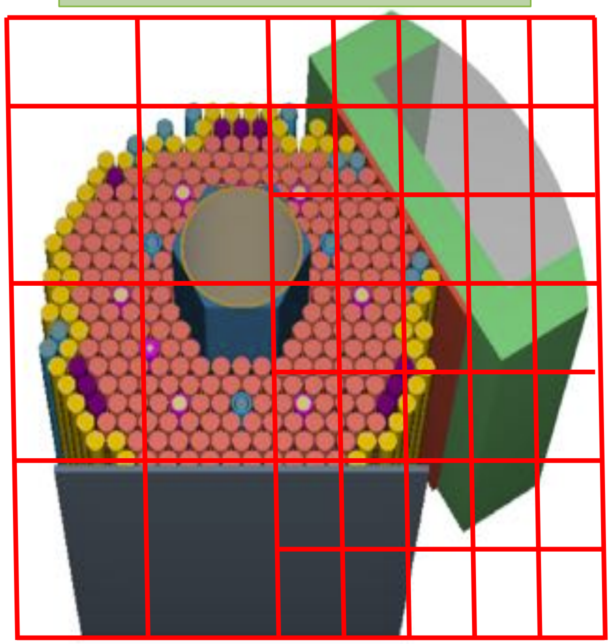
Annular Core Research Reactor

Defining a good Quicksilver benchmark problem is very challenging

Challenges

- **Huge variation in scale:**
  Benchmark must be equally valid on 1 node or 10,000 nodes
- **Simulation geometry:**
  Any geometry that resembles production use will be difficult to scale
- **Load Balance:**
  Imbalanced load distorts performance
- **Realistic behavior:**
  Production behavior arises from complex geometry and multiple materials

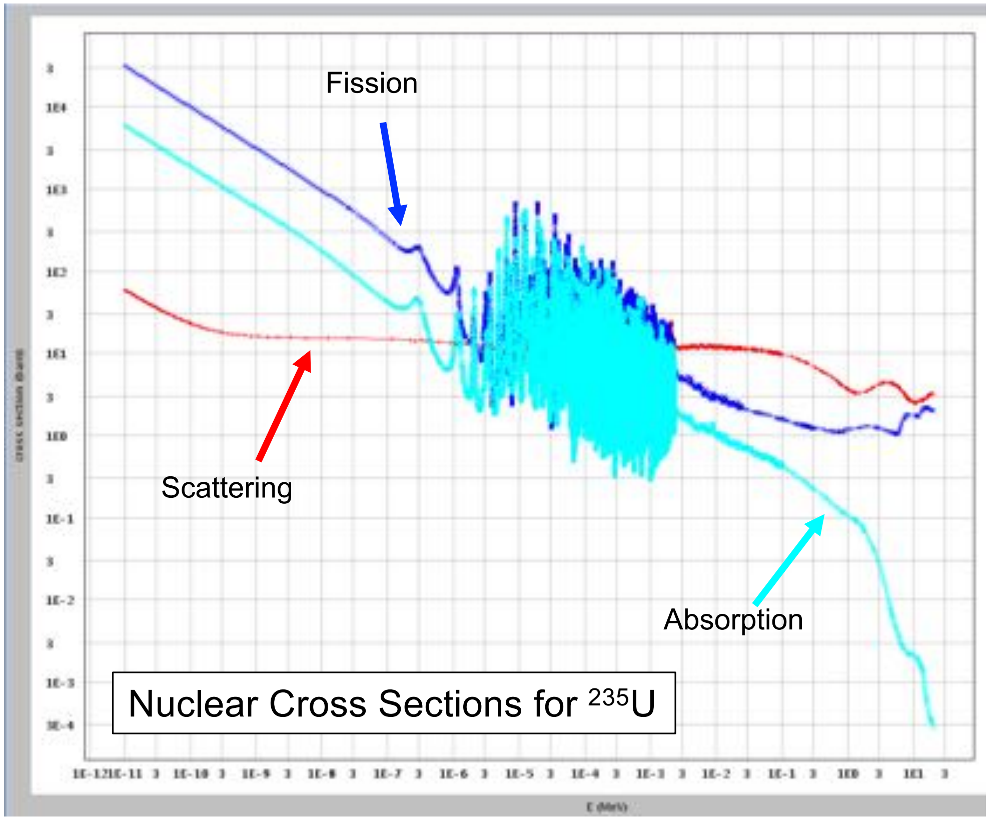
Spatial decomposition is imbalanced at every scale

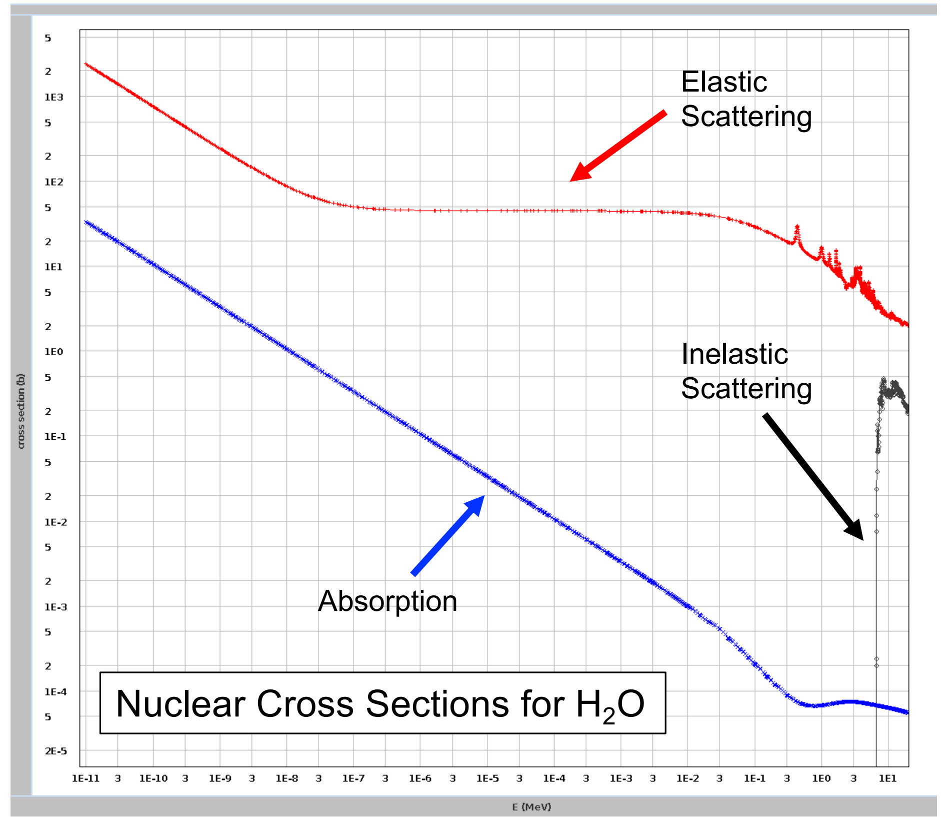


Annular Core Research Reactor

Simplified physics can drastically alter program behavior

Quicksilver's synthetic cross sections struggle to match this complexity





Need scattering from H_20 and fission from ^{235}U

Defining a good Quicksilver benchmark problem is very challenging

Challenges

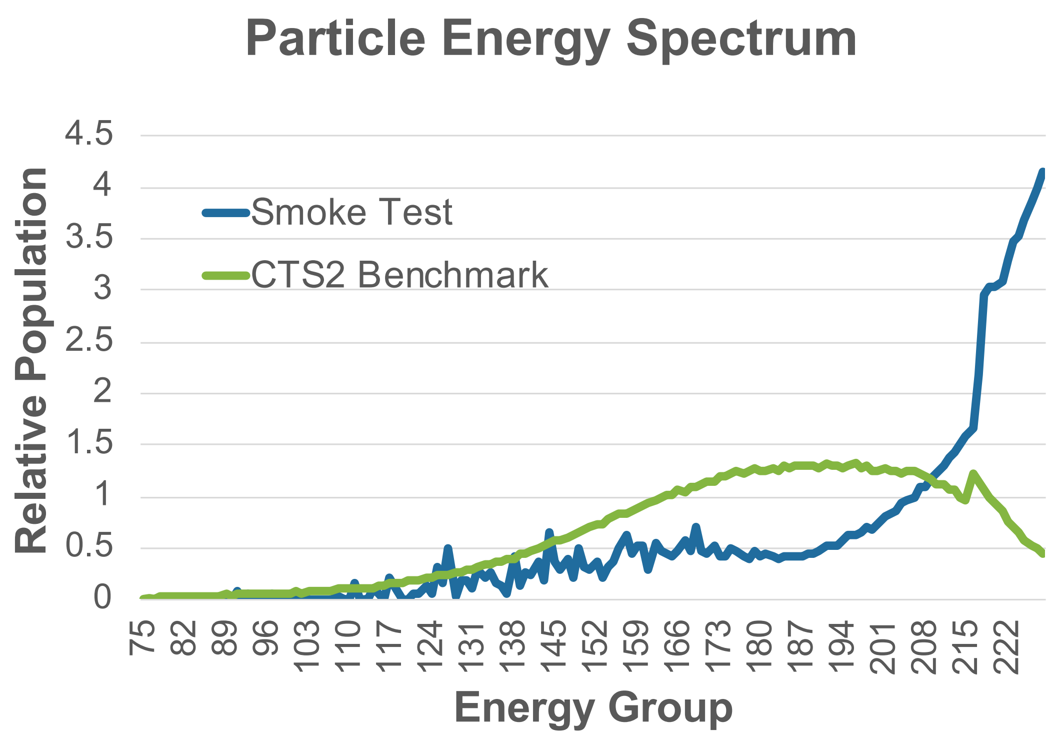
- **Huge variation in scale:**
  Benchmark must be equally valid on 1 node or 10,000 nodes
- **Simulation geometry:**
  Any geometry that resembles production use will be difficult to scale
- **Load Balance:**
  Imbalanced load distorts performance
- **Realistic behavior:**
  Production behavior arises from complex geometry and multiple materials

Solutions

- **Homogeneous single material geometry:**
  Trivially scalable and load balanced
- **Run rules to constrain problem:**
  Fixed mesh size and elements per node.
  Also set target range for wall time per step
- **Made-up Materials:**
  Material properties tailored to interact with simplified physics to produce desired behavior. Blend of real materials

The Quicksilver CTS2 benchmark problem represents memory access patterns more accurately than the default problem

- The default Quicksilver problem is only a “smoke test” intended for developers
- Energy spectrum determines memory access pattern for cross section lookups
- Smoke test overpopulates high energies compared to intended benchmark
- **Moral:** Beware default problems unless you know they are intended to be representative



Best practices for benchmark problems

For Developers of Proxy Apps & Benchmarks

- Ensure benchmark run rules address issues of scalability, fidelity, ease of use, etc.
- Focus on representing program behavior, not “realistic” inputs
- Provide sample inputs and FOM data for common hardware
- Choose reasonable wall time

For Users of Proxy Apps & Benchmarks

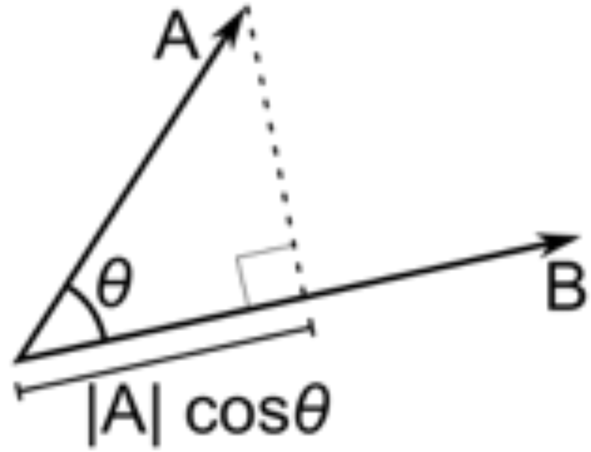
- Don’t assume default problems are good benchmark problems
- Understand and obey run rules
- Verify benchmark performance on standard hardware

For Computing Facilities

- Ensure benchmark problems cover the desired range of system use cases
- Avoid temptation to ask for every benchmark you can think of

Cosine similarity quantifies the fidelity of benchmark suites using a "workload fingerprint"

- Cosine similarity quantifies the relative alignment of vectors in an arbitrary vector space
  - Think: "Projection of **A** in the direction of **B**"
  - cosθ is an angular distance metric independent of vector magnitude
- Similar workload fingerprints mean similar response to a particular design constraint
  - Codes with similar memory B/W fingerprint derive similar benefit from memory B/W improvement
- Allows data-driven selection of codes
  - Alternative to SME debates of perceived relevance, familiarity, ease, etc.
  - Labs and vendors have limited time & staff to construct and respond to RFPs



$$\boldsymbol{A} \cdot \boldsymbol{B} \equiv \sum_{i=1}^{n} a_i b_i = \| \boldsymbol{A} \| \| \boldsymbol{B} \| \cos\theta$$

$$\therefore \cos\theta = \frac{\left(\sum_{i=1}^{n} a_i b_i\right)}{\left(\| \boldsymbol{A} \| \| \boldsymbol{B} \|\right)}$$

Modern Processors can track hundreds of performance events

But they can't all be counted at once

Cache	Pipeline
MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_HIT	FP_ASSIST.ANY
MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_HITM	FP_ASSIST.X87_INPUT
MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_MISS	MEM_UOPS_RETIRED.STLB_MISS_LOADS
L2_LINES_IN.I	MEM_UOPS_RETIRED.STLB_MISS_STORES
MEM_LOAD_UOPS_RETIRED.L3_MISS	LD_BLOCKS.STORE_FORWARD
L2_RQSTS.RFO_HIT	UOPS_ISSUED.SINGLE_MUL
L2_RQSTS.CODE_RD_MISS	LD_BLOCKS.NO_SR
MEM_LOAD_UOPS_RETIRED.L2_MISS	UOPS_ISSUED.FLAGS_MERGE
MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_NONE	ILD_STALL.LCP
MEM_LOAD_UOPS_RETIRED.L3_HIT	DSB2MITE_SWITCHES.PENALTY_CYCLES
L2_LINES_IN.S	DSB2MITE_SWITCHES
ICACHE.MISSES	MISALIGN_MEM_REF.STORES
L2_RQSTS.ALL_CODE_RD	LSD.CYCLES_4_UOPS
L2_TRANS.CODE_RD	LSD.UOPS
MEM_LOAD_UOPS_L3_MISS_RETIRED.LOCAL_DRAM	LSD.ACTIVE
ICACHE.HIT	ARITH.FPU_DIV_ACTIVE
L2_RQSTS.DEMAND_DATA_RD_HIT	UOPS_DISPATCHES_CANCELLED.SIMD_PRF
L2_RQSTS.DEMAND_DATA_RD_MISS	BACLEARS.ANY

It takes dozens of runs to measure all events

Some counters are highly correlated to performance differences

Selectivity is similar to principal component analysis

Cache	Selectivity
MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_HIT	2.721
MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_HITM	2.213
MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_MISS	2.178
L2_LINES_IN.I	1.531
MEM_LOAD_UOPS_RETIRED.L3_MISS	1.482
L2_RQSTS.RFO_HIT	1.410
L2_RQSTS.CODE_RD_MISS	1.406
MEM_LOAD_UOPS_RETIRED.L2_MISS	1.383
MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_NONE	1.305
MEM_LOAD_UOPS_RETIRED.L3_HIT	1.305
L2_LINES_IN.S	1.267
ICACHE.MISSES	1.131
L2_RQSTS.ALL_CODE_RD	1.073
L2_TRANS.CODE_RD	1.070
MEM_LOAD_UOPS_L3_MISS_RETIRED.LOCAL_DRAM	1.067
ICACHE.HIT	1.023
L2_RQSTS.DEMAND_DATA_RD_HIT	1.018
L2_RQSTS.DEMAND_DATA_RD_MISS	0.999

Pipeline	Selectivity
FP_ASSIST.ANY	3.162
FP_ASSIST.X87_INPUT	3.162
MEM_UOPS_RETIRED.STLB_MISS_LOADS	2.839
MEM_UOPS_RETIRED.STLB_MISS_STORES	2.577
LD_BLOCKS.STORE_FORWARD	2.212
UOPS_ISSUED.SINGLE_MUL	2.114
LD_BLOCKS.NO_SR	2.039
UOPS_ISSUED.FLAGS_MERGE	1.977
ILD_STALL.LCP	1.796
DSB2MITE_SWITCHES.PENALTY_CYCLES	1.777
DSB2MITE_SWITCHES	1.777
MISALIGN_MEM_REF.STORES	1.656
LSD.CYCLES_4_UOPS	1.650
LSD.UOPS	1.608
LSD.ACTIVE	1.580
ARITH.FPU_DIV_ACTIVE	1.551
UOPS_DISPATCHES_CANCELLED.SIMD_PRF	1.434
BACLEARS.ANY	1.358

Reduce effort by collecting only selective events

We computed cosine similarity for several proxies and parents

	ExaMiniMD	LAMMPS	MiniQMC	QMCPack	sw4lite	sw4	SWFFT	HACC	pennant	snap
ExaMiniMD	0.00	10.24	84.61	83.55	61.94	64.17	86.71	85.58	75.88	44.50
LAMMPS	10.24	0.00	75.12	73.95	53.63	56.50	79.66	78.51	70.97	34.97
MiniQMC	84.61	75.12	0.00	5.97	42.91	47.75	51.57	51.28	66.16	43.41
QMCPack	83.55	73.95	5.97	0.00	37.71	42.28	45.85	45.52	60.31	40.89
sw4lite	61.94	53.63	42.91	37.71	0.00	6.47	27.99	26.86	30.17	24.55
sw4	64.17	56.50	47.75	42.28	6.47	0.00	23.59	22.42	23.83	29.89
SWFFT	86.71	79.66	51.57	45.85	27.99	23.59	0.00	1.22	18.65	51.79
HACC	85.58	78.51	51.28	45.52	26.86	22.42	1.22	0.00	18.14	50.70
pennant	75.88	70.97	66.16	60.31	30.17	23.83	18.65	18.14	0.00	51.63
snap	44.50	34.97	43.41	40.89	24.55	29.89	51.79	50.70	51.63	0.00

BROADWELL

Proxies are similar to parents
Unrelated applications are clearly different

BROADWELL

	ExaMiniMD	LAMMPS	MiniQMC	QMCPack	sw4lite	sw4	SWFFT	HACC	pennant	snap
ExaMiniMD	0.00	10.24	84.61	83.55	61.94	64.17	86.71	85.58	75.88	44.50
LAMMPS	10.24	0.00	75.12	73.95	53.63	56.50	79.66	78.51	70.97	34.97
MiniQMC	84.61	75.12	0.00	5.97	42.91	47.75	51.57	51.28	66.16	43.41
QMCPack	83.55	73.95	5.97	0.00	37.71	42.28	45.85	45.52	60.31	40.89
sw4lite	61.94	53.63	42.91	37.71	0.00	6.47	27.99	26.86	30.17	24.55
sw4	64.17	56.50	47.75	42.28	6.47	0.00	23.59	22.42	23.83	29.89
SWFFT	86.71	79.66	51.57	45.85	27.99	23.59	0.00	1.22	18.65	51.79
HACC	85.58	78.51	51.28	45.52	26.86	22.42	1.22	0.00	18.14	50.70
pennant	75.88	70.97	66.16	60.31	30.17	23.83	18.65	18.14	0.00	51.63
snap	44.50	34.97	43.41	40.89	24.55	29.89	51.79	50.70	51.63	0.00

SKYLAKE

	ExaMiniMD	LAMMPS	MiniQMC	QMCPack	sw4lite	sw4	SWFFT	HACC	pennant	snap
ExaMiniMD	0.00	8.97	81.96	68.83	38.66	39.55	28.51	37.76	43.58	22.20
LAMMPS	8.97	0.00	81.38	68.47	38.60	39.33	29.50	38.49	42.40	20.45
MiniQMC	81.96	81.38	0.00	16.35	47.28	47.63	58.78	49.85	46.58	65.55
QMCPack	68.83	68.47	16.35	0.00	36.05	36.40	46.19	37.82	36.33	53.30
sw4lite	38.66	38.60	47.28	36.05	0.00	4.05	20.56	17.09	12.89	21.69
sw4	39.55	39.33	47.63	36.40	4.05	0.00	19.82	15.87	11.91	22.79
SWFFT	28.51	29.50	58.78	46.19	20.56	19.82	0.00	10.33	24.49	21.44
HACC	37.76	38.49	49.85	37.82	17.09	15.87	10.33	0.00	19.92	26.67
pennant	43.58	42.40	46.58	36.33	12.89	11.91	24.49	19.92	0.00	25.00
snap	22.20	20.45	65.55	53.30	21.69	22.79	21.44	26.67	25.00	0.00

Similarity exposes architectural constraints

Similarity can reveal unusual stressors on select event groups

Cosine similarities calculated using only cache events

	ExaMiniMD	LAMMPS	MiniQMC	QMCPack	sw4lite	sw4	SWFFT	HACC	pennant	snap
ExaMiniMD	0.00	5.02	54.54	38.73	11.70	12.49	6.58	6.38	13.21	7.13
LAMMPS	5.02	0.00	54.69	38.62	15.66	16.27	4.87	6.38	13.60	10.88
MiniQMC	54.54	54.69	0.00	17.15	47.12	46.08	50.02	48.98	42.16	49.15
QMCPack	38.73	38.62	17.15	0.00	32.64	31.67	33.92	32.94	26.29	33.78
sw4lite	11.70	15.66	47.12	32.64	0.00	1.15	13.41	11.40	11.15	5.07
sw4	12.49	16.27	46.08	31.67	1.15	0.00	13.74	11.70	10.69	5.69
SWFFT	6.58	4.87	50.02	33.92	13.41	13.74	0.00	2.24	9.09	8.80
HACC	6.38	6.38	48.98	32.94	11.40	11.70	2.24	0.00	7.86	6.87
pennant	13.21	13.60	42.16	26.29	11.15	10.69	9.09	7.86	0.00	9.37
snap	7.13	10.88	49.15	33.78	5.07	5.69	8.80	6.87	9.37	0.00

QMC use cache differently from other apps

Similarity can help find gaps & redundancies in suites

	Average App1& App2	App1	App2	Proxy 10	Proxy 04	Proxy 05	Proxy 08	Proxy 11	Proxy 01	Proxy 02	Proxy 07	Proxy 09	Proxy 03	Proxy 06	Proxy 12	Exclusive Sum across Proxies	min	Avg	Exclusive Average across Proxies
App1		0.00	20.54	12.06	18.50	23.40	25.00	24.85	26.74	26.02	19.80	[illegible]	[illegible]	[illegible]	[illegible]	351.24	12.06	25.09	27.02
App2	14.49	20.54	0.00	28.06	27.71	31.51	31.34	32.49	33.15	28.21	34.85	[illegible]	[illegible]	[illegible]	[illegible]	461.34	20.54	32.95	35.49
Proxy10	21.54	12.06	28.06	0.00	[illegible]	17.06	19.18	18.00	[illegible]	24.83	22.13	25.28	42.80	43.61	47.40	333.64	12.06	23.83	25.66
Proxy04	23.53	18.50	27.71	[illegible]	0.00	6.28	7.24	7.16	9.12	[illegible]	33.39	36.72	40.62	54.79	59.35	329.38	6.28	23.53	25.34
Proxy05	27.73	23.40	31.51	[illegible]	6.28	0.00	3.28	2.18	3.90	[illegible]	37.11	40.17	43.66	56.80	62.00	344.33	2.18	24.60	26.49
Proxy08	28.33	25.00	31.34	19.18	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	39.45	42.55	43.01	59.69	64.80	356.77	2.55	25.48	27.44
Proxy11	28.90	24.85	32.49	[illegible]	7.16	2.18	2.91	0.00	2.96	[illegible]	38.39	41.33	44.29	57.83	62.79	352.58	2.18	25.18	27.12
Proxy01	30.10	26.74	33.15	[illegible]	9.12	3.90	2.55	2.96	0.00	[illegible]	40.88	43.87	44.31	60.38	65.49	370.79	2.55	26.48	28.52
Proxy02	27.13	26.02	28.21	24.83	[illegible]	16.99	15.77	17.39	[illegible]	0.00	43.08	46.40	27.94	64.92	69.36	413.64	15.77	29.55	31.82
Proxy07	28.26	19.80	34.85	22.13	33.39	[illegible]	[illegible]	[illegible]	40.88	43.08	0.00	6.38	51.38	25.77	30.93	423.53	6.38	30.25	32.58
Proxy09	31.90	23.91	38.40	25.28	36.72	40.17	42.55	41.33	43.87	46.40	6.38	0.00	54.11	23.21	27.30	449.65	6.38	32.12	34.59
Proxy03	37.17	38.49	35.81	42.80	40.62	43.66	43.01	44.29	44.31	27.94	51.38	54.11	0.00	70.77	73.73	610.93	27.94	43.64	46.99
Proxy06	51.15	43.65	57.92	43.61	54.79	56.80	59.69	57.83	60.38	64.92	25.77	23.21	70.77	0.00	16.33	635.67	16.33	45.41	48.90
Proxy12	55.08	48.28	61.36	47.40	59.35	62.00	64.80	62.79	65.49	69.36	30.93	27.30	73.73	16.33	0.00	689.13	16.33	49.22	53.01

Gap

Redundancy

Application behavior can vary with input choice

Angular difference in signatures for clamr_cpuonly -n_1024_-i_200_-t_1000						
	regular-grid	cell-in-place	regular-grid-by-faces	face-in-place	cell	face
regular-grid	0.00	6.27	22.37	8.34	16.07	12.43
cell-in-place	6.27	0.00	18.29	4.88	10.69	6.97
regular-grid-by-faces	22.37	18.29	0.00	15.70	11.05	13.09
face-in-place	8.34	4.88	15.70	0.00	8.76	5.67
cell	16.07	10.69	11.05	8.76	0.00	4.91
face	12.43	6.97	13.09	5.67	4.91	0.00
sum	65.49	47.10	80.50	43.35	51.47	43.06

Don’t assume a single run represents all behavior

Best practices for benchmark selection and fidelity

For Developers of Proxy Apps & Benchmarks	For Users of Proxy Apps & Benchmarks	For Computing Facilities
• Quantify the fidelity of your proxy relative to the actual workload • Provide multiple inputs	• Choose proxies and benchmarks according to the hardware they stress • Understand input sensitivities	• Consider gaps and redundancies in benchmark suites

Facilities use benchmarks for a wide variety of purposes

Marketing and Program Development

Application Development and Readiness

- The Center for Accelerated Application Readiness (CAAR) is the vanguard for the broader application readiness ecosystem and for future science

OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

CAAR

U.S. DEPARTMENT OF ENERGY INCITE LEADERSHIP COMPUTING

ALCC ASCR Leadership Computing Challenge

Programming Model Development

- SPEC Accel compares performance of
  - Accelerators (GPUs, Co-processors, etc.)
  - Supporting software tool chains (Compilers, Drivers, etc.)
  - Interface (CPU, PCIe, etc.)
- Three distinct benchmarks for OpenCL, OpenACC, and OpenMP, updated in 2017
- These (and other benchmarks) are used by DOE labs to drive compiler development

An Example: DOE Proxy Apps in LLVM's Test Suite

MultiSource/Benchmarks/DOE-ProxyApps-C[++]

- Pathfinder
- RSBench
- SimpleMOC
- XSBench
- MiniAMR
- miniGMG
- CLAMR
- HACCKernels
- HPCCG
- PENNANT
- miniFE

Name	Current	%
MultiSource/Benchmarks/DOE-ProxyApps-C++/CLAMR/CLAMR	1.5409	-
MultiSource/Benchmarks/DOE-ProxyApps-C++/HACCKernels/HACCKernels	0.4073	-
MultiSource/Benchmarks/DOE-ProxyApps-C++/HPCCG/HPCCG	0.6004	-
MultiSource/Benchmarks/DOE-ProxyApps-C++/miniFE/miniFE	3.1404	-
MultiSource/Benchmarks/DOE-ProxyApps-C++/PENNANT/PENNANT	0.5841	-
MultiSource/Benchmarks/DOE-ProxyApps-C/miniAMR/miniAMR	0.6306	-
MultiSource/Benchmarks/DOE-ProxyApps-C/miniGMG/miniGMG	0.8441	-
MultiSource/Benchmarks/DOE-ProxyApps-C/Pathfinder/PathFinder	2.7840	1.81%
MultiSource/Benchmarks/DOE-ProxyApps-C/RSBench/rsbench	0.5323	-
MultiSource/Benchmarks/DOE-ProxyApps-C/SimpleMOC/SimpleMOC	1.8238	-
MultiSource/Benchmarks/DOE-ProxyApps-C/XSBench/XSBench	2.3148	-4.18%
Geometric Mean	0.0396	-

LLVM is an open-source compiler infrastructure used by many parts of our exascale ecosystem...

Benchmarks are an essential element of system procurements

- CORAL benchmarks should
  - Span the breadth of the NNSA (LLNL, LANL, SNL) workload
  - Span the time-dependent(!) and much broader space of LCF (ORNL, ANL) workloads
  - Span co-spaces of algorithms, implementations, and use cases
  - Provide adequate drivers for system SW and library development
- CORAL benchmarks must
  - not be so numerous that vendors cannot provide analyses on O(weeks) time scale
    - Significant challenge to cover/span the breadth of concerns, while not being onerous on vendors
  - not encumber application developers with 24-7 support responsibilities during those weeks
  - use proxies for NNSA apps

The benchmark suite for CORAL-2 had to satisfy multiple wants and constraints

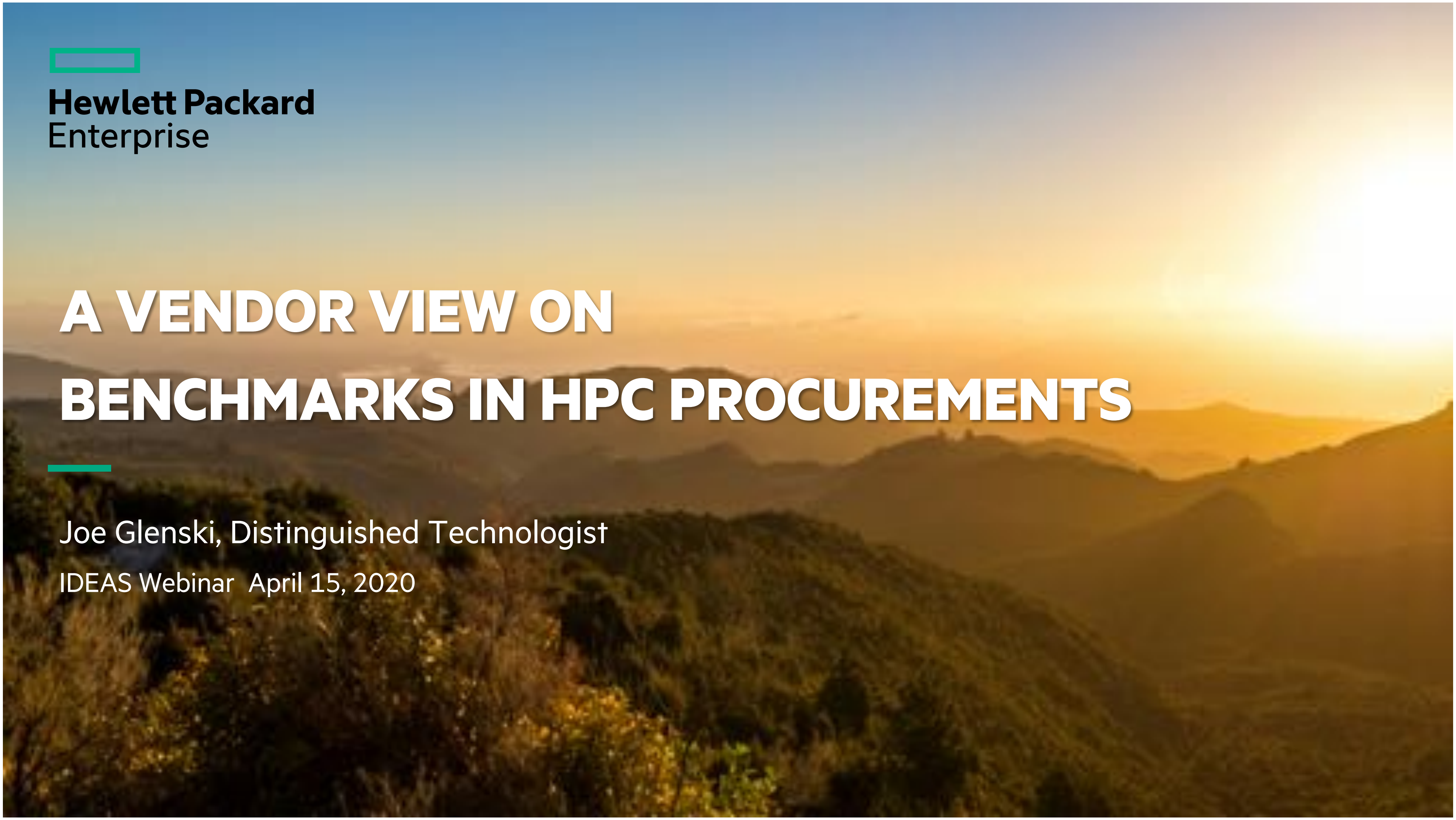
The CORAL-2 benchmark suite is a mixture of production codes and proxies

- Scalable Science Benchmarks: HACC, Nekbone, QMCPACK, LAMMPS
- Throughput Benchmarks: AMG, Kripke, Quicksilver, PENNANT
- Data Science and Deep Learning Benchmarks:
  - Big Data Analytic Suite
    - [Schmidt, et al., "Defining Big Data Analytics Benchmarks for Next Generation Supercomputers," https://arxiv.org/abs/1811.02287]
  - Deep Learning Suite
- Skeleton Benchmarks
- Microkernel Benchmarks

https://asc.llnl.gov/coral-2-benchmarks/

Best practices for benchmarks at facilities

For Developers of Proxy Apps & Benchmarks	For Users of Proxy Apps & Benchmarks	For Computing Facilities
• Make it easy to run your benchmark in an automated framework • Carefully consider whether to use proxies or full applications	• Benchmark suites are usually good indications of facility interests and concerms	• Build suites that can cover a variety of use cases • Avoid overly large benchmark collections • Automate as much as possible • Look for lessons learned that can be transferred to production codes



Hewlett Packard Enterprise

A VENDOR VIEW ON BENCHMARKS IN HPC PROCUREMENTS

Joe Glenski, Distinguished Technologist

IDEAS Webinar April 15, 2020

FORWARD LOOKING STATEMENTS

This presentation may contain forward-looking statements that involve risks, uncertainties and assumptions. If the risks or uncertainties ever materialize or the assumptions prove incorrect, the results of Hewlett Packard Enterprise Company and its consolidated subsidiaries ("Hewlett Packard Enterprise") may differ materially from those expressed or implied by such forward-looking statements and assumptions. All statements other than statements of historical fact are statements that could be deemed forward-looking statements, including but not limited to any statements regarding the expected benefits and costs of the transaction contemplated by this presentation; the expected timing of the completion of the transaction; the ability of HPE, its subsidiaries and Cray to complete the transaction considering the various conditions to the transaction, some of which are outside the parties' control, including those conditions related to regulatory approvals; projections of revenue, margins, expenses, net earnings, net earnings per share, cash flows, or other financial items; any statements concerning the expected development, performance, market share or competitive performance relating to products or services; any statements regarding current or future macroeconomic trends or events and the impact of those trends and events on Hewlett Packard Enterprise and its financial performance; any statements of expectation or belief; and any statements of assumptions underlying any of the foregoing. Risks, uncertainties and assumptions include the possibility that expected benefits of the transaction described in this presentation may not materialize as expected; that the transaction may not be timely completed, if at all; that, prior to the completion of the transaction, Cray's business may not perform as expected due to transaction-related uncertainty or other factors; that the parties are unable to successfully implement integration strategies; the need to address the many challenges facing Hewlett Packard Enterprise's businesses; the competitive pressures faced by Hewlett Packard Enterprise's businesses; risks associated with executing Hewlett Packard Enterprise's strategy; the impact of macroeconomic and geopolitical trends and events; the development and transition of new products and services and the enhancement of existing products and services to meet customer needs and respond to emerging technological trends; and other risks that are described in our Fiscal Year 2018 Annual Report on Form 10-K, and that are otherwise described or updated from time to time in Hewlett Packard Enterprise's other filings with the Securities and Exchange Commission, including but not limited to our subsequent Quarterly Reports on Form 10-Q. Hewlett Packard Enterprise assumes no obligation and does not intend to update these forward-looking statements.

TOPICS

- The ~~big headache~~ Challenge of Writing Requests For Proposals (RFPs)
- How Benchmarks are Used in Typical RFPs
- Evaluation Metrics
- Projections and Estimates
- Optimization and Code Modification
- Suggestions from Benchmarkers – Do's and Don'ts

Special thanks to Tricia Balle, who provide ideas and material for this presentation

THE CHALLENGE OF WRITING RFPS

- **Identify desired system characteristics and ensure the RFP requirements reflect them**
  - How to eliminate what you do not want and ensure what you do want is scored appropriately?
  - How to easily compare vendor offerings?
- **Ensure the document is clear and unambiguous**
  - Lack of clarity -> questions
  - Questions -> time wasted -> delays in procurement schedule -> installation delays / risk of loss of funding
- **Allow vendors time to ask questions and share most questions and responses**
  - Clarification questions can identify issues that will affect all vendors
  - Releasing benchmarks early can shake out problems before official RFP release
  - Do allow for vendor-specific queries to be kept confidential if at all possible!
- **Beware of the law of unintended consequences**
  - A requirement for more HPL performance than budget supports can lead to trouble if vendors bid what you did not actually want

WHY USE BENCHMARKS IN RFPS?

Basic Aim: **To measure the vendors' proposed machine capabilities in comparison to the customer's workload requirements.**

Basic Requirement: **Understand what you value and how you will score proposals, then provide the smallest set of benchmarks necessary to compare performance.**

- Keep expectations of the vendors in proportion to value of the deal

Common Scenarios for Benchmark Use in RFPs:

- To **enable evaluation** of offered systems and their capability to handle expected workloads.
  - Sometimes just a simple evaluation of performance of proposed hardware
  - If optimizations are allowed, can also evaluate vendors' support capabilities with eye to support post-delivery
- To **design and size the system** required to run the workload
- As **a hurdle** to limit responses from non-HPC savvy vendors

EVALUATION METHODS AND METRICS

- **A clearly defined evaluation metric is important so we understand where to target performance and what you value**

- **Also important to understand how highly benchmarks are weighted in overall scoring**
  - Are benchmarks a very small proportion of the total final score?
  - Will HPL Rmax determine system size, regardless of benchmark performance?

- *Beware of benchmark requirements that have nothing to do with the purpose of the machine (e.g., if you need a lot of network, don't just use low node count benchmarks).*
- *If the workload is known to be memory bandwidth limited, maybe include codes similar to STREAM (or weight them highly) and exclude things like SPEC (mostly clock bound).*
- *Consider a benchmark such as GPCNet to get a measure of ability of system to handle congestion on the network*

EVALUATION METRICS – COMMON SCENARIOS

- Simply **run and report performance** (often used as a barrier to entry)
- Run each benchmark test in **under a specified target time** (makes most sense in cases such as operational weather with predefined constraints)
- **Evaluate applications individually** (relative to each vendor) Often includes an evaluation of scaling performance up to system size or scaling limit
- **Throughputs**
  - A well thought out throughput mix can be a useful tool and help evaluate I/O performance
  - Throughput metrics are tough for vendors to model and require additional work, so should ideally displace other benchmarks
- **Weighted metrics** (for example: “SSP” – Sustained System Performance)
  - Bundle mix of applications and kernels (don’t just use small kernels)
  - Weight each one appropriately for workload priorities
  - Create single metric for easier evaluation (often done with Geomeans)
  - Can allow variation within mix at acceptance - especially good for future hardware

PROJECTIONS AND ESTIMATES

- **Projections are essential for any system with hardware not yet available or for system sizes beyond what is available**
- **How to ensure vendors know what they are doing?**
  - Prior record
  - Good explanation of methodology (but don't expect full details)
  - Good relationships
  - Full commitments to proposed performance
- **Decide if you will allow processor or interconnect vendors to supply benchmark results**
  - This can lead to identical results submitted by multiple system vendors
  - Requiring the system vendor to run benchmarks can demonstrate potential for support in the future
  - Who will estimate future system performance and commit to it?
- **Be careful of applications that have Random Number Generators or Iterative solvers**
  - Need iteration counts to be consistent from run to run
  - If have to scale out to higher core counts, must know number of iterations for reliable projection

OPTIMIZATION AND CODE MODIFICATION

- **Best to allow optimization with guidelines,** such as:
  - Specify types of optimizations allowed (I/O, communications., OpenMP, etc.)
  - Specify that scientific validity of results should not change
  - Don't allow optimizations that are specific to the benchmark problem itself
  - Require vendor to supply full details of all optimizations made
  - Retain ability to reject optimizations if they are too complicated, etc.
- **Legacy apps often just don't scale up efficiently without being adapted to current or future hardware** (processor types, node counts, and networks)
- **Optimizations allows ability to evaluate full potential of system hardware, compiler, libraries, etc.**
- **Also allows ability to evaluate vendor skill** (important if collaboration is considered)

THE "DO'S" AND "DON'T"S

What benchmarkers like (and don't like) to see…

DO….

- **First, figure out what you want**, e.g., "the fastest running job, no matter how many nodes it takes", or "maximum number of jobs on the system"?
- **Make benchmark instructions clear**
  - Check that README does not conflict with main document
  - Get directions and files tested by people not involved in the benchmark preparation before you release them to vendors
  - Remember that your working directory is not a benchmark distribution!
- **Supply validation requirements and make sure they are also clear**
  - e.g. “WRF output should match to within 5%” is **not** clear
- **Watch run length!** A good benchmark will run for 5 to 60 minutes.
  - Under 1 hour allows us more time to debug, optimize and find the best way to run your applications. But…. sub-10 second runs are not very useful ☺
  - If you shorten a run, consider evaluating only the post-initialization portion to get a more useful result
  - Decent problem sizes will differentiate vendors better

MORE DO…

- **Set an appropriate deadline for getting results returned**
  - Allow enough time for the vendor to do the work
  - More complicated RFPs take more time
  - If the time is too short, the quality of response goes down
- **Remember the impact of year end holidays**
  - Releasing an RFP in early December and asking for response in early January will not get you good results
- **Make sure any penalties around missing performance targets are clearly defined in the RFP document** (we need to understand risks)
- **At Acceptance, be pragmatic about meeting targets**
  - If the system hardware was not yet in production when estimates were made, must expect some variation in actual performance. Weighted metrics like SSP help with this

DON'T…

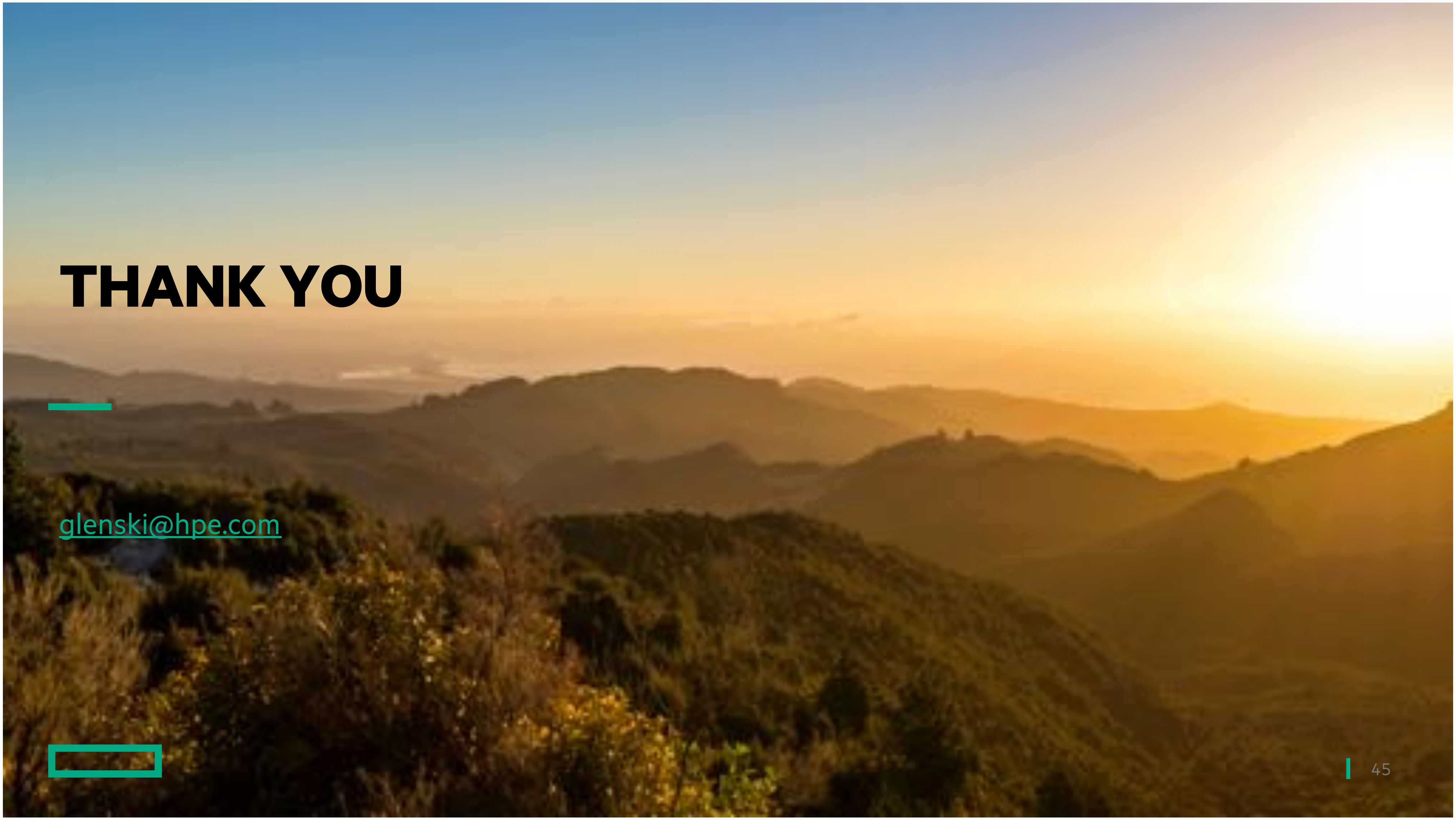
- **Don't add too many requirements that restrict how benchmarks can be run**
  - For example, don't specify number of MPI ranks / OpenMP threads to be used
  - Allow vendor flexibility to demonstrate best way to run app on proposed architecture
  - Don't assume anything about numbers of CPUs, cores, accelerators per node (unless they are mandatory requirements for system). This often occurs when too focused on an existing system
  - Allow the use of multiple compilers, MPIs, etc.
- **Don't ask for large numbers of commitments for no clear purpose**
  - Only ask for numbers that are clear to interpret and are useful
  - For example, it is easy to ask for results for a huge variety of MPI tests, but hard to understand what the results mean for the real workload. And hard for the vendor to provide them

MORE DON'T…

- **Don't expect output to be bit identical to that from another system**
  - How much precision do you really need in your results? If input data are based on measurements with 3 significant digits, don't ask for 14 digits of accuracy in comparison to data from original system. Determine what a scientifically valid result is and ask for that
  - If identical runs must give identical output, say so. If runs must give identical output across all rank and thread counts, say so
    - Code must be written to be bit reproducible in the first place
    - This can limit optimizations possible
- **Don't require huge amounts of output data to be returned**
  - Will you really look at all of it? Can you look at output from just the final step/iteration?
  - Can you provide a tool that can postprocess the data before return?
  - Large return data requirements can add up to a week to write a drive and ship, which leads to requests for extension or less time available to dedicate to actual benchmarking work

IN CONCLUSION

- Define your workload before designing the minimal set of benchmark tests to reflect that workload
- Write the RFP benchmark requirements as clearly as you can, and have them tested before releasing to vendors
- Define a clear evaluation metric to enable valid comparison among vendors and to ensure you end up with the system you want
- Allow vendors to show what their proposed system can do to help your scientific workloads perform as well and as efficiently as possible


THANK YOU
glenski@hpe.com
45

Summary of best practices

For Developers of Proxy Apps & Benchmarks

- Write documentation
- Ensure benchmark run rules address issues of scalability, fidelity, ease of use, etc.
- Make it easy to identify the FOM and verify correct results
- Focus on representing program behavior, not “realistic” inputs
- Choose reasonable wall time
- Quantify the fidelity of your proxy relative to the actual workload
- Provide multiple inputs

For Users of Proxy Apps & Benchmarks

- Read documentation
- Benchmarks have well-defined run rules and a figure of merit
- Understand and obey run rules
- Don’t assume every proxy app is useful as a benchmark
- Understand input sensitivities
- Verify benchmark performance on standard hardware
- DOE system procurement suites can be a good place to look for benchmark problems

For Computing Facilities

- Cover all aspects of the ecosystem: Programming models, compilers, debuggers, performance tools
- Avoid temptation to ask for every benchmark you can think of
- Consider gaps and redundancies in benchmark suites
- Avoid large input or output files and complex library dependencies
- Make benchmark suites easy to build and automate
- Build suites that can cover a variety of use cases

Disclaimer
This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC