Performance Optimization Overview

Boyana Norris

NUCLEI Annual Meeting 2019
May 22, 2019

http://bit.ly/NUCLEI-Perf

Structure

- Summary of performance-related activities over the past year
  - DFTNESS = HFBTHO + HFODD kernels
  - MFDn
  - MPI
  - NuCCOR
- For each code: Performance goals and hopes
  - Please add your input to the letter to the Performance Santa (during or after the talk is not too late!) **http://bit.ly/NUCLEI-Perf**
- Analysis and optimization summary

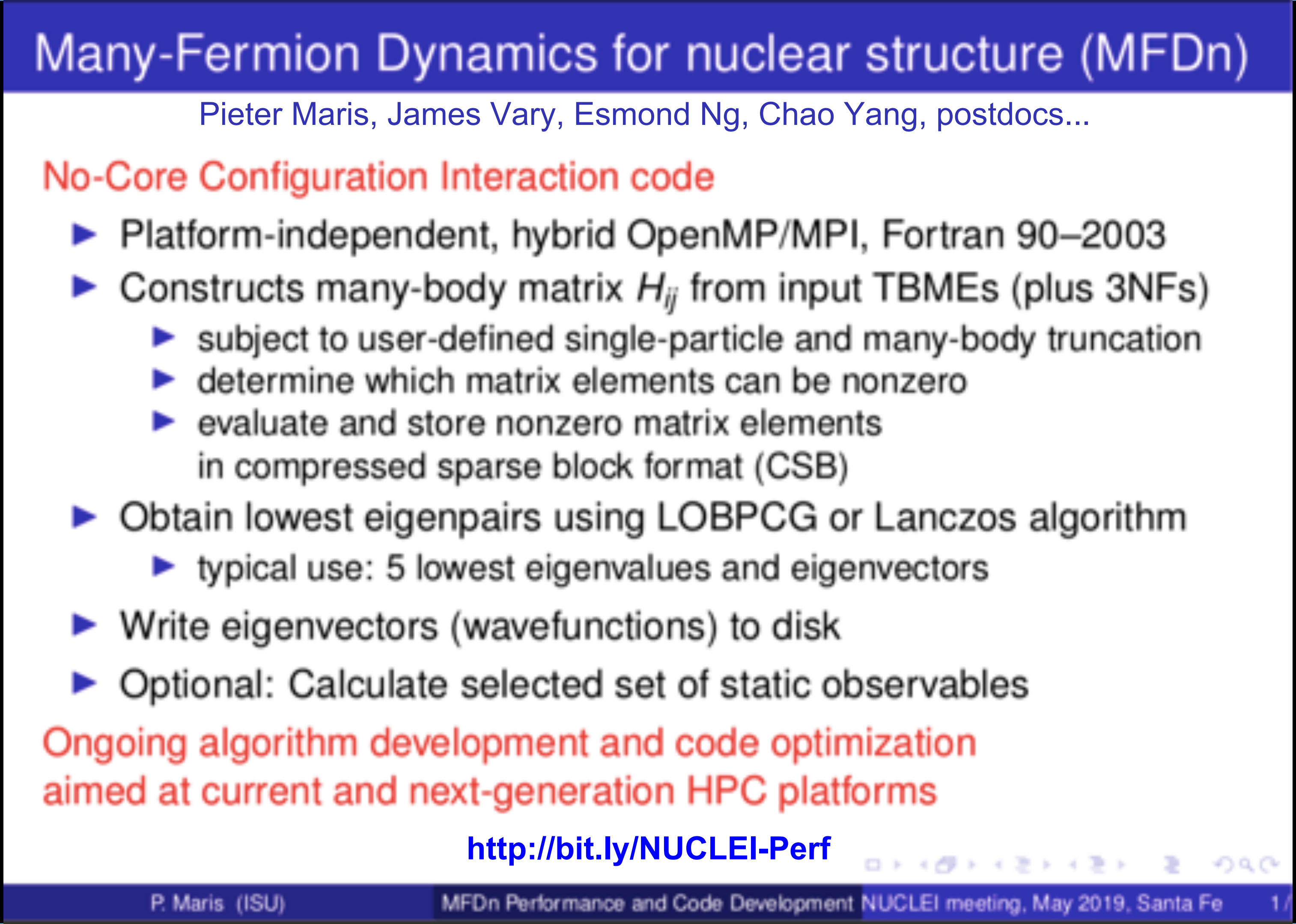
Architectures:

- Heterogeneous
  - multicore CPUs (10s of threads per socket),
  - manycore (100s of threads per device)
  - several GPUs per node (1000s of threads per device)
  - more networking layers (on and off node)
  - in memory computing, FPGAs, ...
- Ubiquitous vectors
- I/O prohibitively expensive

Optimization goals:

- Parallelism: inter-node (multiple networks), intra-node (multicore, manycore, GPUs), vectors
- Memory: optimizing data structures for specific computations or architectures
- I/O

http://bit.ly/NUCLEI-Perf



Many-Fermion Dynamics for nuclear structure (MFDn)

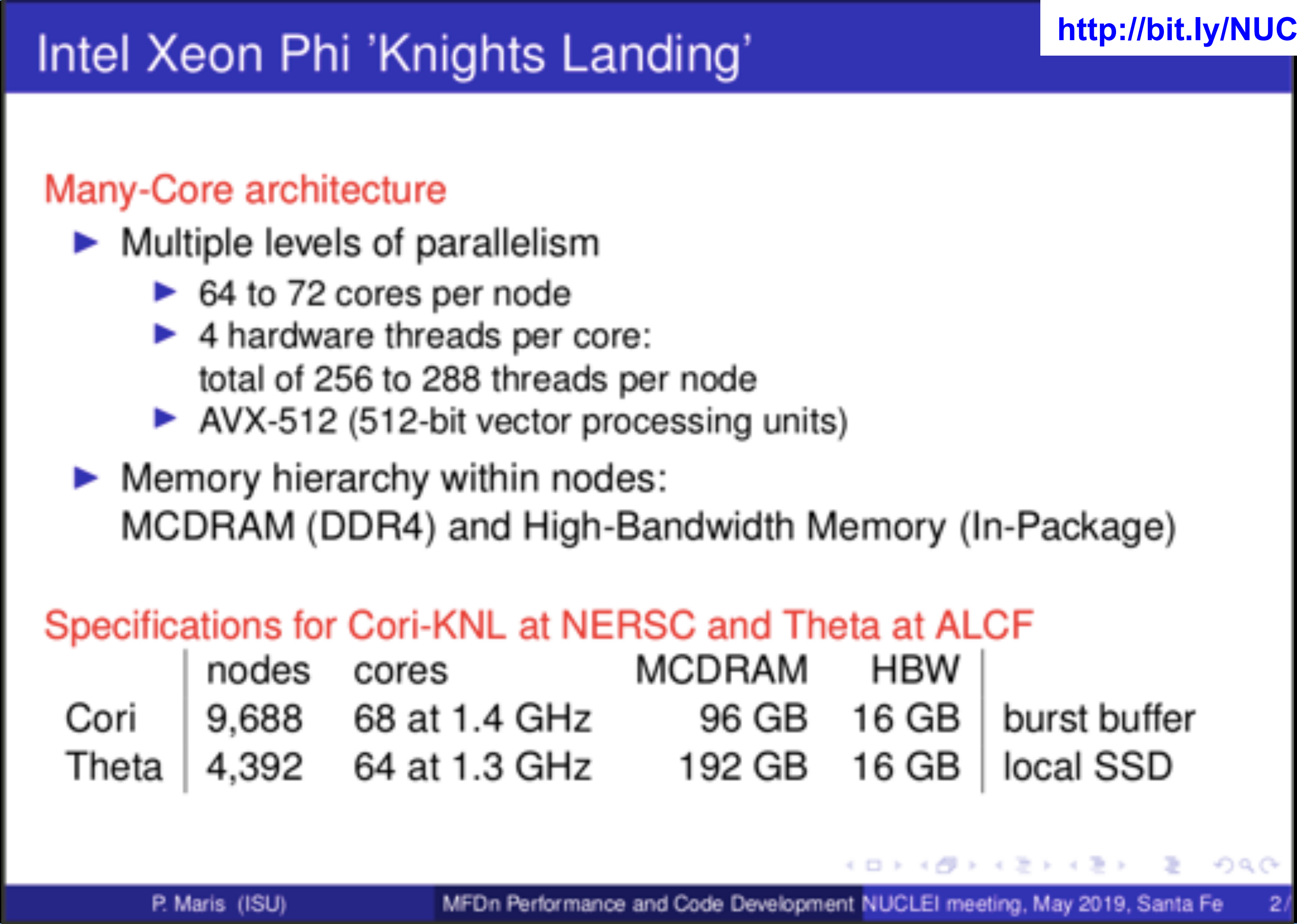
Pieter Maris, James Vary, Esmond Ng, Chao Yang, postdocs...

No-Core Configuration Interaction code

- Platform-independent, hybrid OpenMP/MPI, Fortran 90–2003
- Constructs many-body matrix H_{ij} from input TBMEs (plus 3NFs)
  - subject to user-defined single-particle and many-body truncation
  - determine which matrix elements can be nonzero
  - evaluate and store nonzero matrix elements
    in compressed sparse block format (CSB)
- Obtain lowest eigenpairs using LOBPCG or Lanczos algorithm
  - typical use: 5 lowest eigenvalues and eigenvectors
- Write eigenvectors (wavefunctions) to disk
- Optional: Calculate selected set of static observables

Ongoing algorithm development and code optimization
aimed at current and next-generation HPC platforms

http://bit.ly/NUCLEI-Perf



http://bit.ly/NUCLEI-Perf

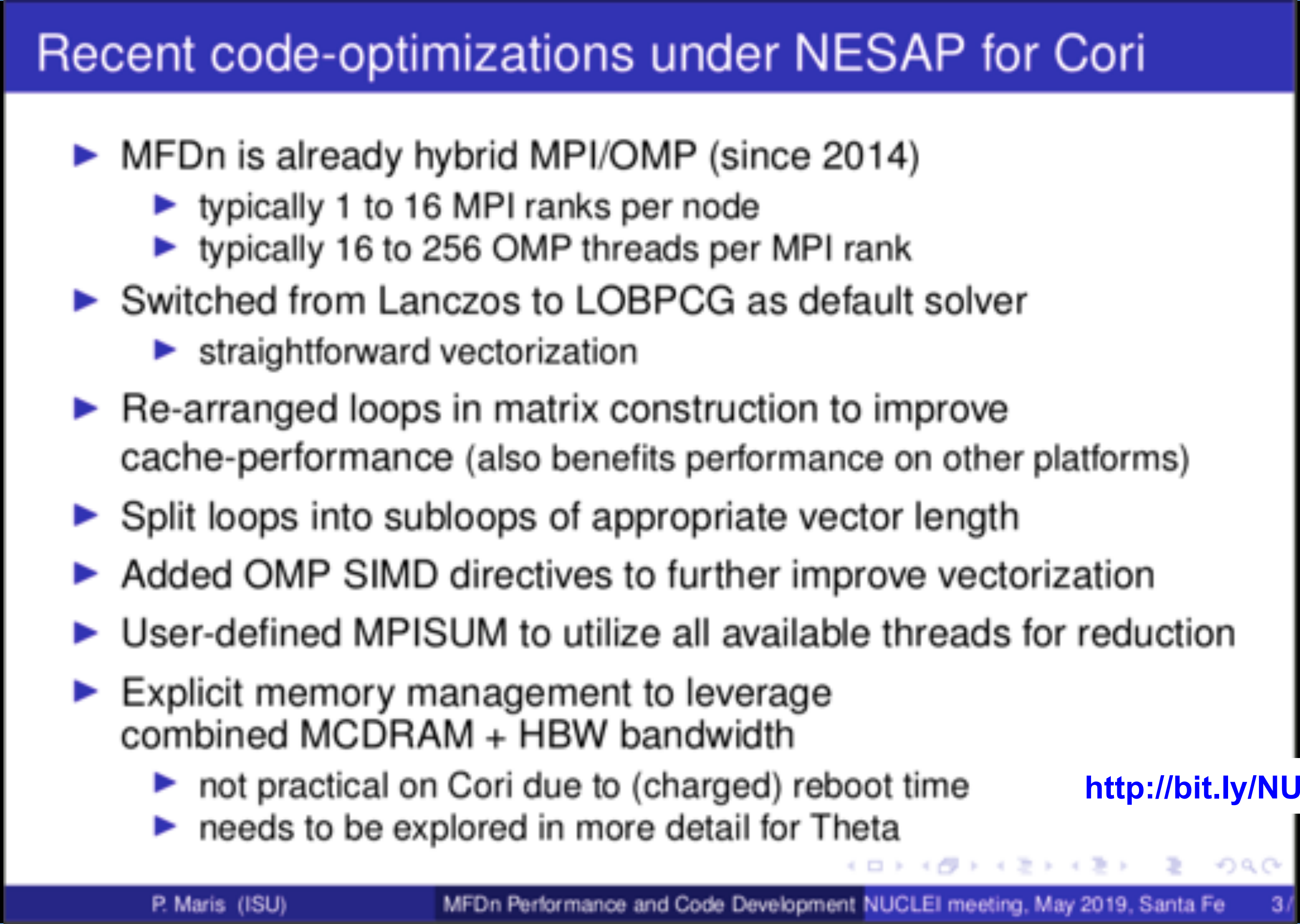
Intel Xeon Phi 'Knights Landing'

Many-Core architecture

- Multiple levels of parallelism
  - 64 to 72 cores per node
  - 4 hardware threads per core:
    total of 256 to 288 threads per node
  - AVX-512 (512-bit vector processing units)
- Memory hierarchy within nodes:
  MCDRAM (DDR4) and High-Bandwidth Memory (In-Package)

Specifications for Cori-KNL at NERSC and Theta at ALCF

	nodes	cores	MCDRAM	HBW	
Cori	9,688	68 at 1.4 GHz	96 GB	16 GB	burst buffer
Theta	4,392	64 at 1.3 GHz	192 GB	16 GB	local SSD



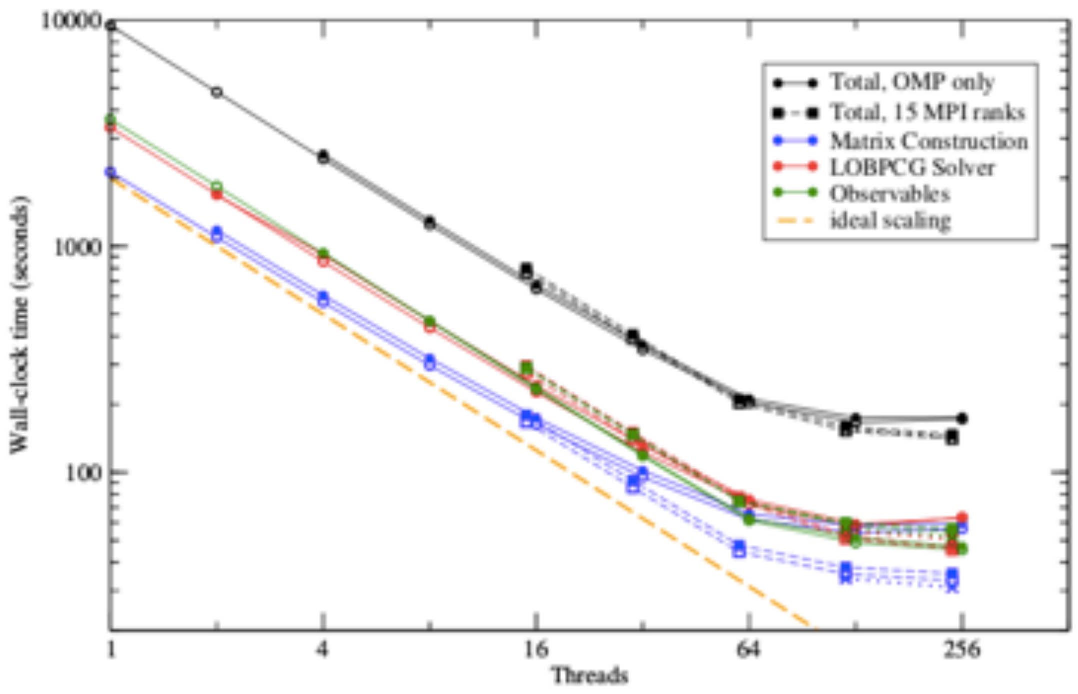
Recent code-optimizations under NESAP for Cori

- MFDn is already hybrid MPI/OMP (since 2014)
  - typically 1 to 16 MPI ranks per node
  - typically 16 to 256 OMP threads per MPI rank
- Switched from Lanczos to LOBPCG as default solver
  - straightforward vectorization
- Re-arranged loops in matrix construction to improve cache-performance (also benefits performance on other platforms)
- Split loops into subloops of appropriate vector length
- Added OMP SIMD directives to further improve vectorization
- User-defined MPISUM to utilize all available threads for reduction
- Explicit memory management to leverage combined MCDRAM + HBW bandwidth
  - not practical on Cori due to (charged) reboot time
  - needs to be explored in more detail for Theta

http://bit.ly/NUCLEI-Perf

http://bit.ly/NUCLEI-Perf

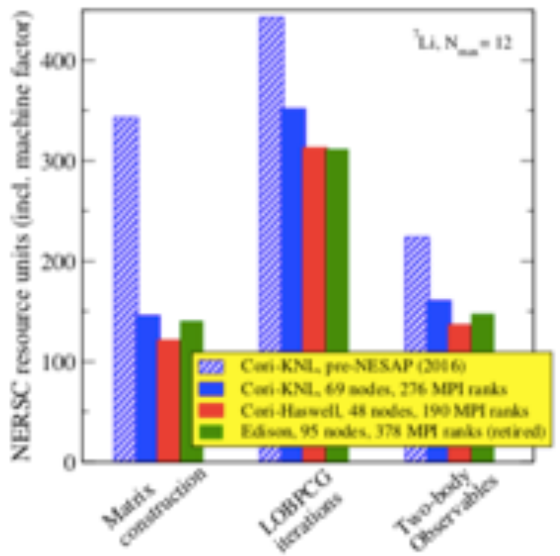
Single-node scaling on KNL (Cori, Theta)

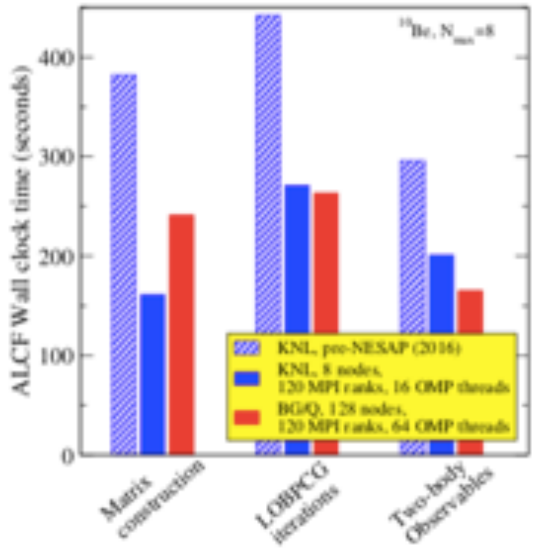


- Good scaling up to number of cores available on both Cori (open symbols) and Theta (closed symbols)

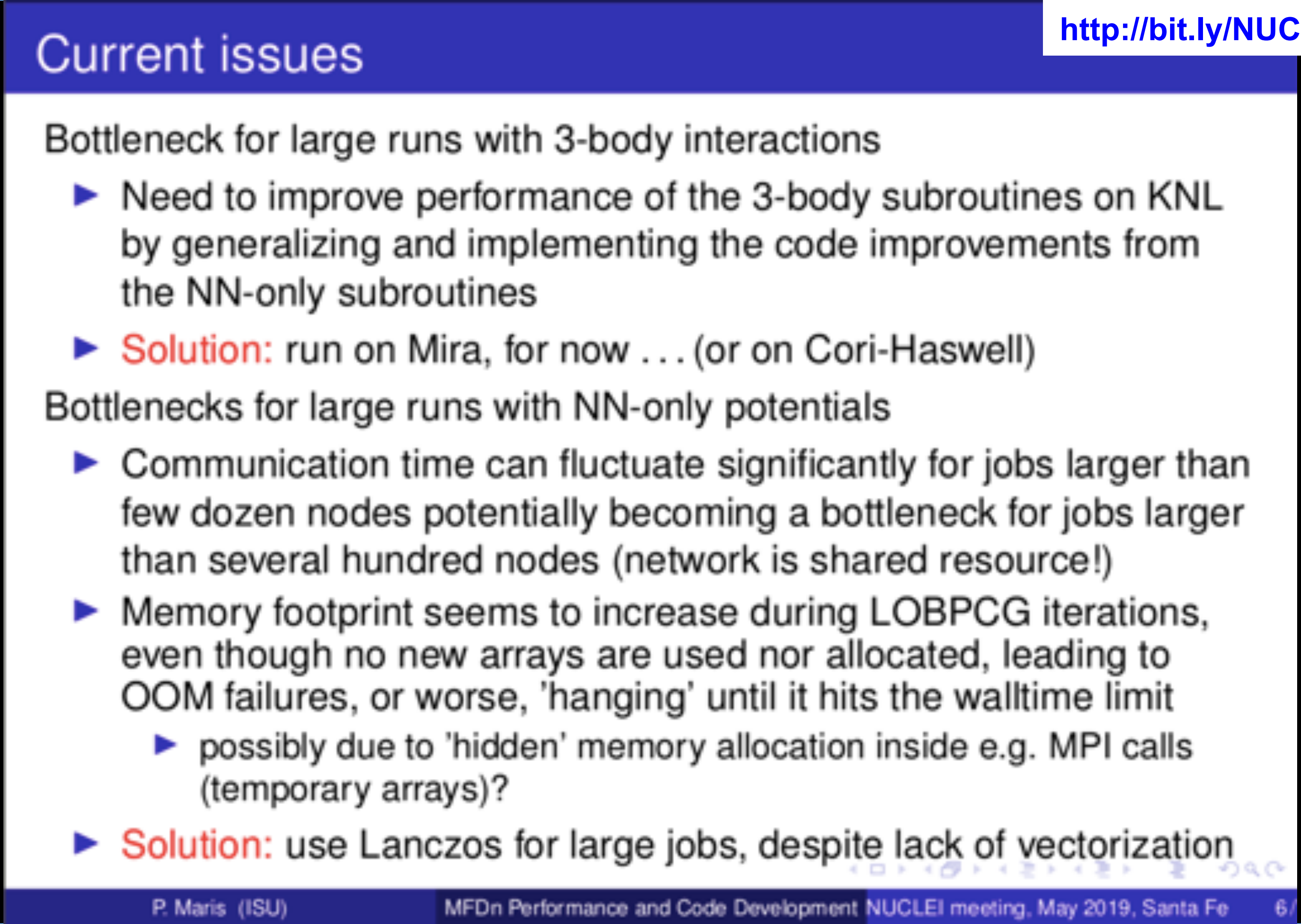
Comparison Cori-Haswell vs. KNL vs. Mira

http://bit.ly/NUCLEI-Perf





- Pre-NESAP version performs poorly on KNL
- Post-NESAP-Cori version performs
  - similar in terms of 'resource units' on Edison, C-Haswell and C-KNL
  - better on Theta than Pre-NESAP version on Mira
    (Theta node has 13× flops and 12× memory of Mira node)



http://bit.ly/NUCLEI-Perf

Current issues

Bottleneck for large runs with 3-body interactions

- Need to improve performance of the 3-body subroutines on KNL by generalizing and implementing the code improvements from the NN-only subroutines
- Solution: run on Mira, for now . . . (or on Cori-Haswell)

Bottlenecks for large runs with NN-only potentials

- Communication time can fluctuate significantly for jobs larger than few dozen nodes potentially becoming a bottleneck for jobs larger than several hundred nodes (network is shared resource!)
- Memory footprint seems to increase during LOBPCG iterations, even though no new arrays are used nor allocated, leading to OOM failures, or worse, 'hanging' until it hits the walltime limit
  - possibly due to 'hidden' memory allocation inside e.g. MPI calls (temporary arrays)?
- Solution: use Lanczos for large jobs, despite lack of vectorization

http://bit.ly/NUCLEI-Perf

Code developments and plans

- Fault tolerance (PhD project Nathan Weeks, AMCS student)
  - various technical issues encountered and addressed
    - ULFM not suitable for Fortran (e.g. no long-jump in Fortran standard)
    - Fortran 2018 standardized syntax and semantics for recovery from failed images may be more useful, but currently lacks implementation
- Post-processor to evaluate expectation values of arbitrary two-body operators for wavefunctions w. same number nucleons
  - functional, and being tested for correctness
  - performance needs to be optimized (straightforward, analous to two-body matrix construction)
- Calculation of electroweak operators consistent with Hamiltonian
  - main focus: M1 moments and transitions; (double) β decay
  - status: in progress, several PhD students
- Use parallel HDF5 for IO of interaction files and wavefunctions

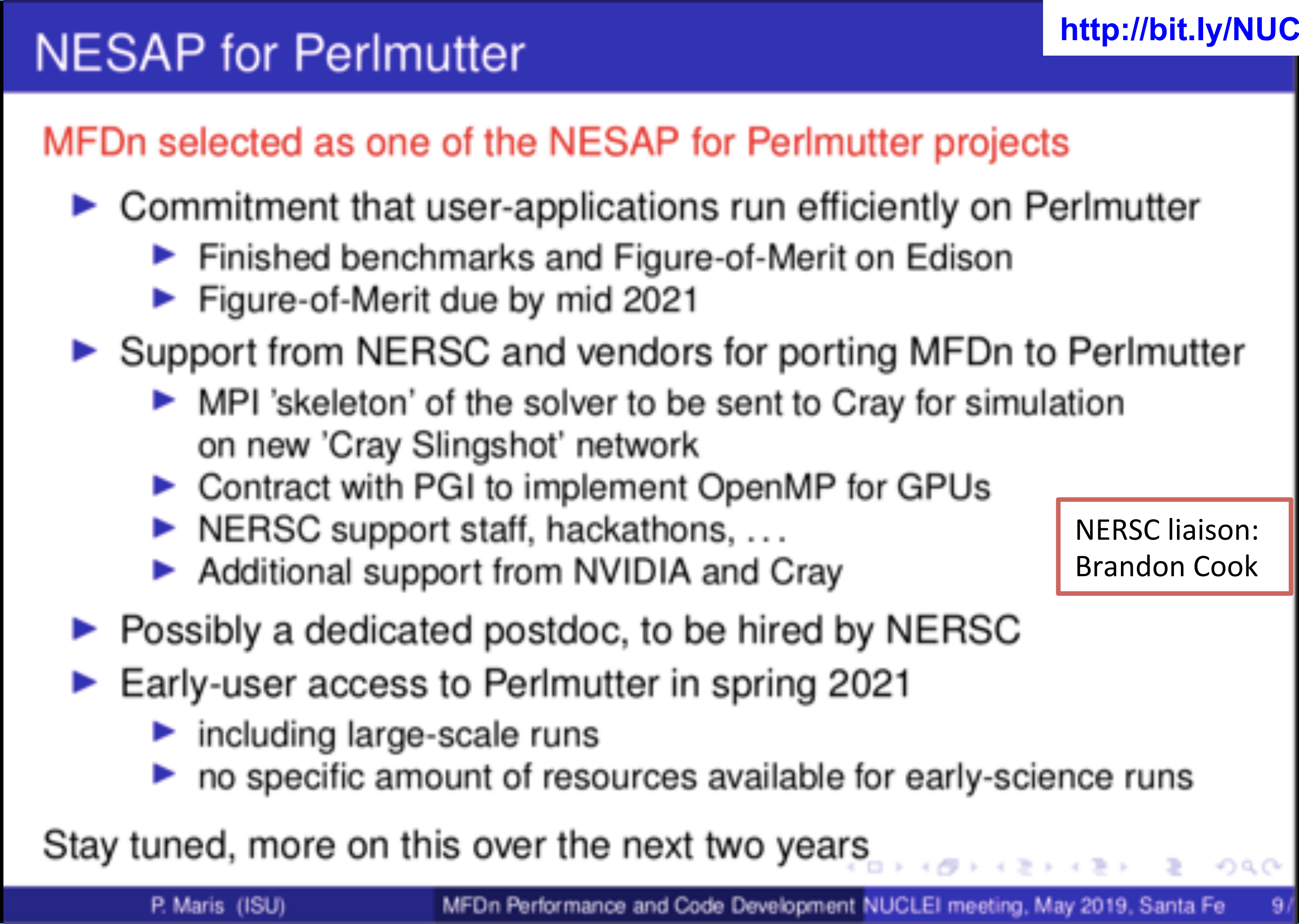
Looking forward: Perlmutter

http://bit.ly/NUCLEI-Perf

NERSC-9: A System Optimized for Science

- Cray Shasta System providing 3-4x capability of Cori system
- First NERSC system designed to meet needs of both large scale simulation and data analysis from experimental facilities
  - Includes both NVIDIA GPU-accelerated and AMD CPU-only nodes
  - Cray Slingshot high-performance network will support Terabit rate connections to system
  - Optimized data software stack enabling analytics and ML at scale
  - All-Flash filesystem for I/O acceleration
- Robust readiness program for simulation, data and learning applications and complex workflows
- Delivery in late 2020

http://bit.ly/NUCLEI-Perf



NESAP for Perlmutter

MFDn selected as one of the NESAP for Perlmutter projects

- Commitment that user-applications run efficiently on Perlmutter
  - Finished benchmarks and Figure-of-Merit on Edison
  - Figure-of-Merit due by mid 2021
- Support from NERSC and vendors for porting MFDn to Perlmutter
  - MPI 'skeleton' of the solver to be sent to Cray for simulation on new 'Cray Slingshot' network
  - Contract with PGI to implement OpenMP for GPUs
  - NERSC support staff, hackathons, . . .
  - Additional support from NVIDIA and Cray
- Possibly a dedicated postdoc, to be hired by NERSC
- Early-user access to Perlmutter in spring 2021
  - including large-scale runs
  - no specific amount of resources available for early-science runs

Stay tuned, more on this over the next two years

NERSC liaison:
Brandon Cook

USING SHARED MEMORY – A MOTIVATING EXAMPLE

- GFMC is a hybrid code using both MPI and OpenMP parallelism.
- Best performance on current many-core machines (like Argonne's Theta) requires multiple (~6) MPI ranks per node, since the OpenMP parallelism within an MPI rank is limited (~10 threads per rank).
- Each thread requires access (read-only) to a large (7.5 GB) table, which is read in at the start of a run. (Actually part doubles and part integers)
- As the size of this table has grown, multiple copies per node (one per rank) have become a limitation.
- Solution: share a single copy of this table among multiple processes (MPI ranks) on the same node.
- Challenge: how to do this portably and conveniently.
- MPI to the rescue.

http://bit.ly/NUCLEI-Perf

MPI Update

Rusty Lusk, ANL

1. General MPI consulting for NUCLEI, in particular the use of new MPI-3 features for using shared memory, the documentation for which is inadequate, particularly for Fortran programs. Short tutorial at last year’s meeting.

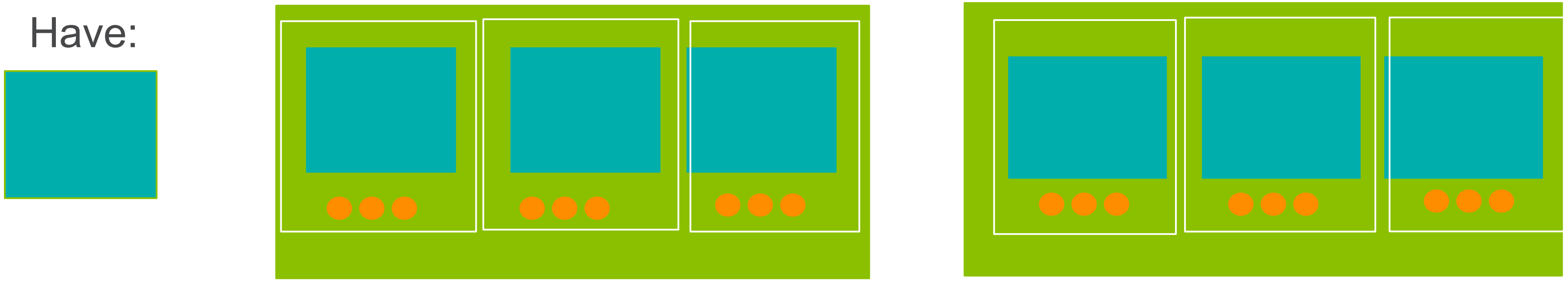
2. Development of a long-message MPI library, useful since the signatures of basic standard MPI communication routines have integer arguments, where these days, in GFMC in particular, one would like these to be long integers (integer*8 in Fortran). The MPIL library implements this feature, so that long messages can be used in applications with minimal changes to them.

3. Improving the build and run system of the GFMC code, which is notoriously complex – new Python scripts being develped, which hopefully help to extend the life of the GFMC code and serve as a model for the next generation. This is in an early stage.

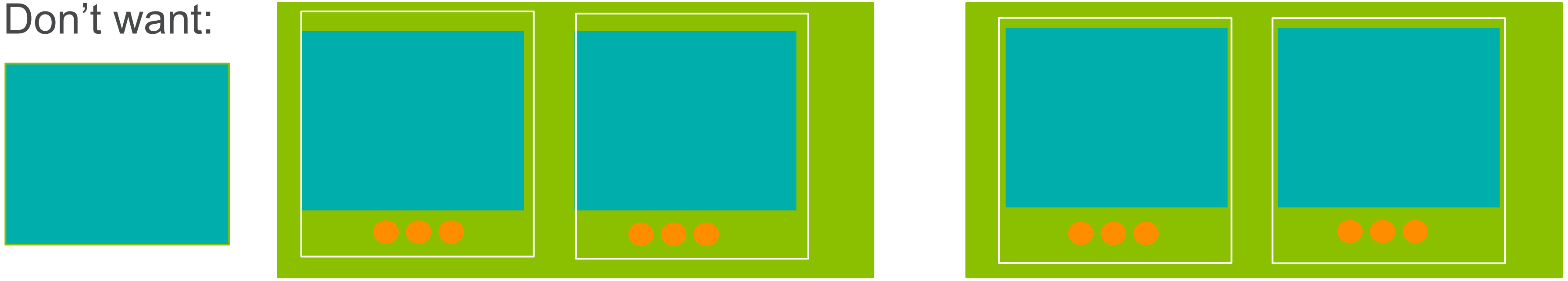
http://bit.ly/NUCLEI-Perf

THE CURRENT SITUATION

Have:



Don't want:

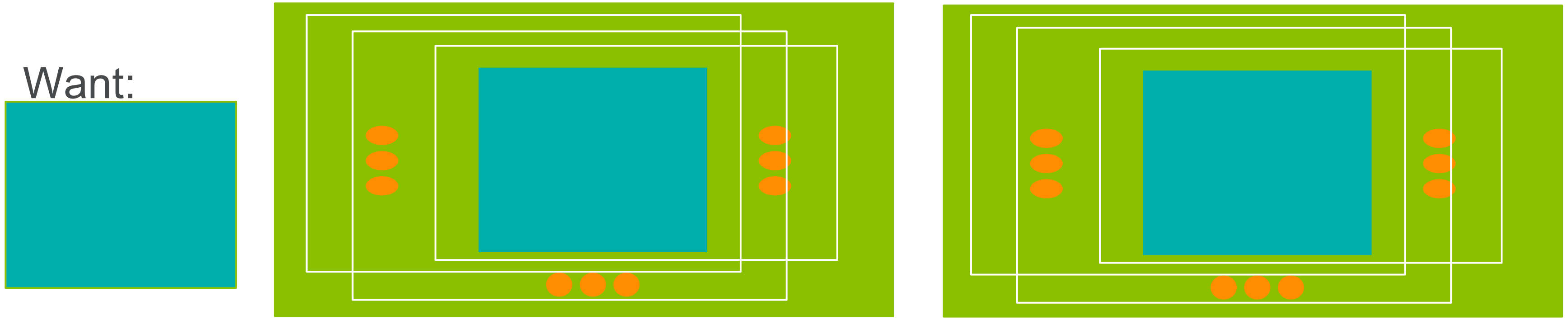


http://bit.ly/NUCLEI-Perf

Argonne NATIONAL LABORATORY

DESIRED SITUATION

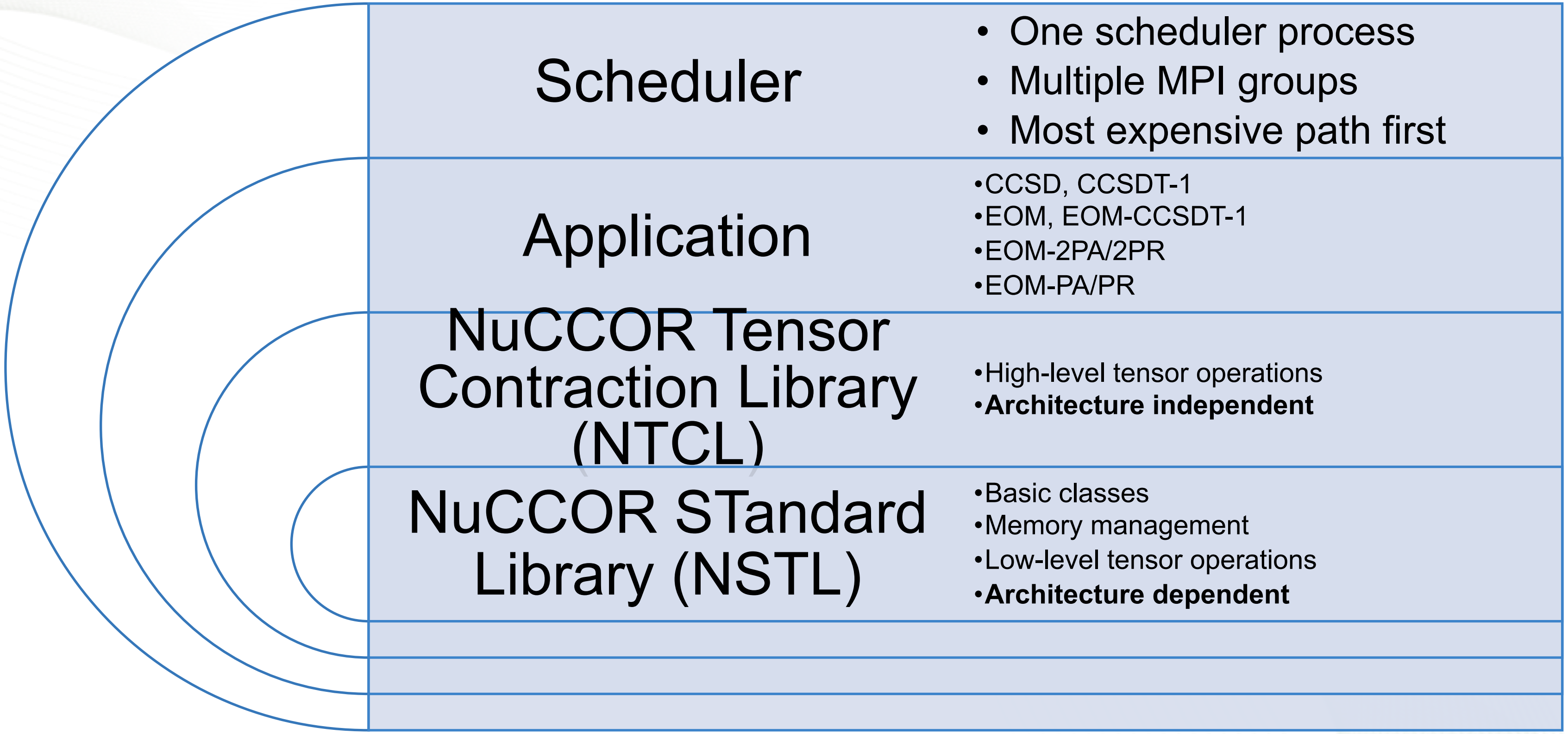
Want:



http://bit.ly/NUCLEI-Perf

NuCCOR: At a glance

Gustav R. Jansen, ORNL



Scheduler	• One scheduler process • Multiple MPI groups • Most expensive path first
Application	•CCSD, CCSDT-1 •EOM, EOM-CCSDT-1 •EOM-2PA/2PR •EOM-PA/PR
NuCCOR Tensor Contraction Library (NTCL)	•High-level tensor operations •**Architecture independent**
NuCCOR STandard Library (NSTL)	•Basic classes •Memory management •Low-level tensor operations •**Architecture dependent**

Applications: Gustav Jansen, Gaute Hagen, Thomas Papenbrock, Titus Morris

http://bit.ly/NUCLEI-Perf

NuCCOR: High-level structure

- Multiple levels of abstraction
  - Architecture dependent low-level library (NSTL)
    - Can easily be extended to new architectures
  - Architecture independent high-level library (NTCL)
    - New types of tensor contractions can be written using NSTL primitives
  - Applications interface with NTCL and runs on any supported architecture
  - Multiple instances of an application can run asynchronously using a scheduler.
    - The scheduler manages multiple groups of MPI ranks, where each group is responsible for a single calculation

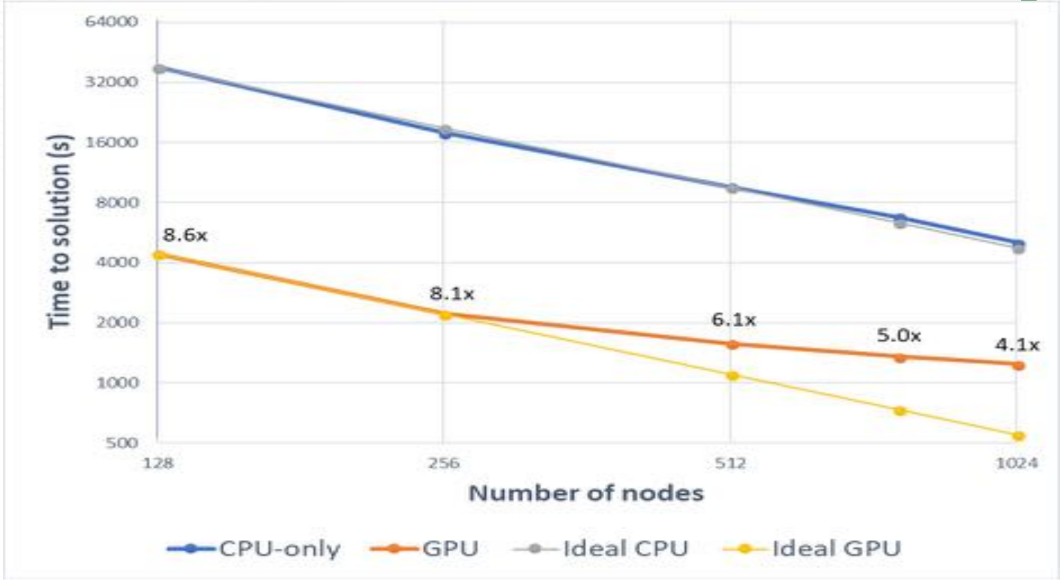
http://bit.ly/NUCLEI-Perf

NuCCOR: Programming environment

- Languages and libraries
  - Written in Fortran 2018 and C
  - Programming model
    - Supports Nvidia GPU’s using CUDA C and HIP
    - Supports AMD GPU’s using HIP (in development)
    - Uses OpenMP 3.1 to support multithreading on CPU’s
    - Uses MPI 3.1 for message passing
  - Linear algebra interfaces
    - BLAS/LAPACK
    - cuBLAS/cuSOLVER
    - rocBLAS (in development)
    - MAGMA (in development)
  - Distributed data structures
    - One-sided MPI implementation (MPI 3.1)
    - OpenSHMEM implementation (in development)
  - IO
    - Parallel HDF5
    - ADIOS (in development)

http://bit.ly/NUCLEI-Perf

NuCCOR: Strong scaling on Summit



- Ground states of nuclei from 4He up to 132Sn with CCSDT-1.

1.32 trillion non-zero triples amplitudes for 132Sn at Nmax=14

Sub-optimal scaling at high node counts mainly due to load imbalance within an MPI group.

http://bit.ly/NUCLEI-Perf

NuCCOR: Outlook

- First open-source release of NuCCOR libraries this fall
- Continue porting legacy code to use the tensor contraction library (NTCL)
- New adaptive load balancing scheme to improve scaling at high node counts.
- Extend NTCL to support cartesian tensors to target deformed nuclei
- Port the NuCCOR standard library to AMD GPUs to run on Frontier

http://bit.ly/NUCLEI-Perf

DFT Update

Nicolas Schunck

- DFTNESS (Density Functional Theory for Nuclei at Extreme ScaleS) codebase; with Nicolas Schunck
- DFTNESS = HFBTHO + HFODD kernels
- Scope: run a large number (10^6) of HFB calculations in parallel
- Both HFBTHO and HFODD are HFB solvers

HFBTHO

- Axial HFB solver: 1 min <= walltime <= 20 min on 6-12 cores depending on problem
- Fortran 90, OpenMP kernel
- Task management: Python with mpi4py
- Target architecture: LLNL Sierra/Lassen (IMP P9 + NVIDIA V100)
- Wishlist: reduce cost of calculation to less than 1 min for all problem sizes (= 1 order of magnitude speed-up)
  - Port entire code to GPU?
  - Fast load-balance for on-node task management

HFODD

http://bit.ly/NUCLEI-Perf

- Symmetry-unrestricted HFB solver: 1 h <= walltime <= 24 h depending on problem
- Fortran 77/90, MPI-OpenMP kernel
- Task management: 2-layer MPI (1 communicator for single HFB solve, 1 communicator to handle all HFB solves)
- Target architecture: LLNL Sierra/Lassen (IMP P9 + NVIDIA V100)
- Wishlist: reduce cost of calculation to less than 1 hour for all problem sizes (= 1-2 orders of magnitude speed-up)
    - Port only a few subroutines to GPU
    - Current code is already MPI-OpenMP for single execution: where/how to put GPU in the mix?
    - Change I/O?

Performance tools, approaches   Boyana Norris, UO

HPCL

DFTNESS (Density Functional Theory for Nuclei at Extreme ScaleS) codebase; with Nicolas Schunck

- ➔ Over 130,000 lines of Fortran (SLOC)
- ➔ Parallelized with OpenMP and MPI
- ➔ Many calls to BLAS routines

- ➔ Currently investigating limitations to:
  - ➔ OpenMP scaling
  - ➔ Vectorization

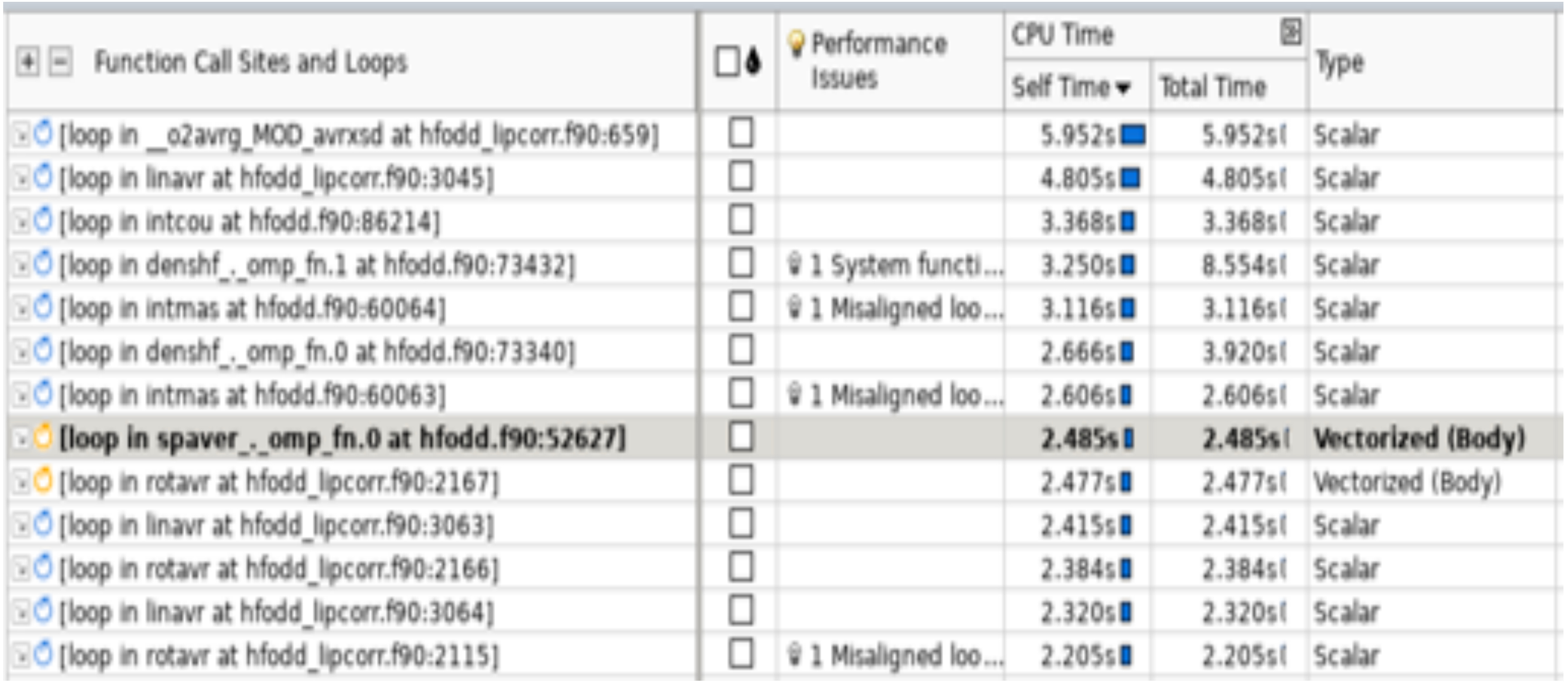
UNIVERSITY OF OREGON

http://bit.ly/NUCLEI-Perf

OpenMP scaling

- In progress – developing techniques that combine performance measurement with compiler analysis
    - Currently considering compiler reports (gcc 9, Intel 19) – many optimization details available, but hard to interpret; no tool connects this detailed information directly with performance measurement results (vendor tools present only vague high-level advice at best)
    - Creating “fake” prototype scenarios to explore scaling limits, e.g., marking all variables private

Vectorization (HFODD)

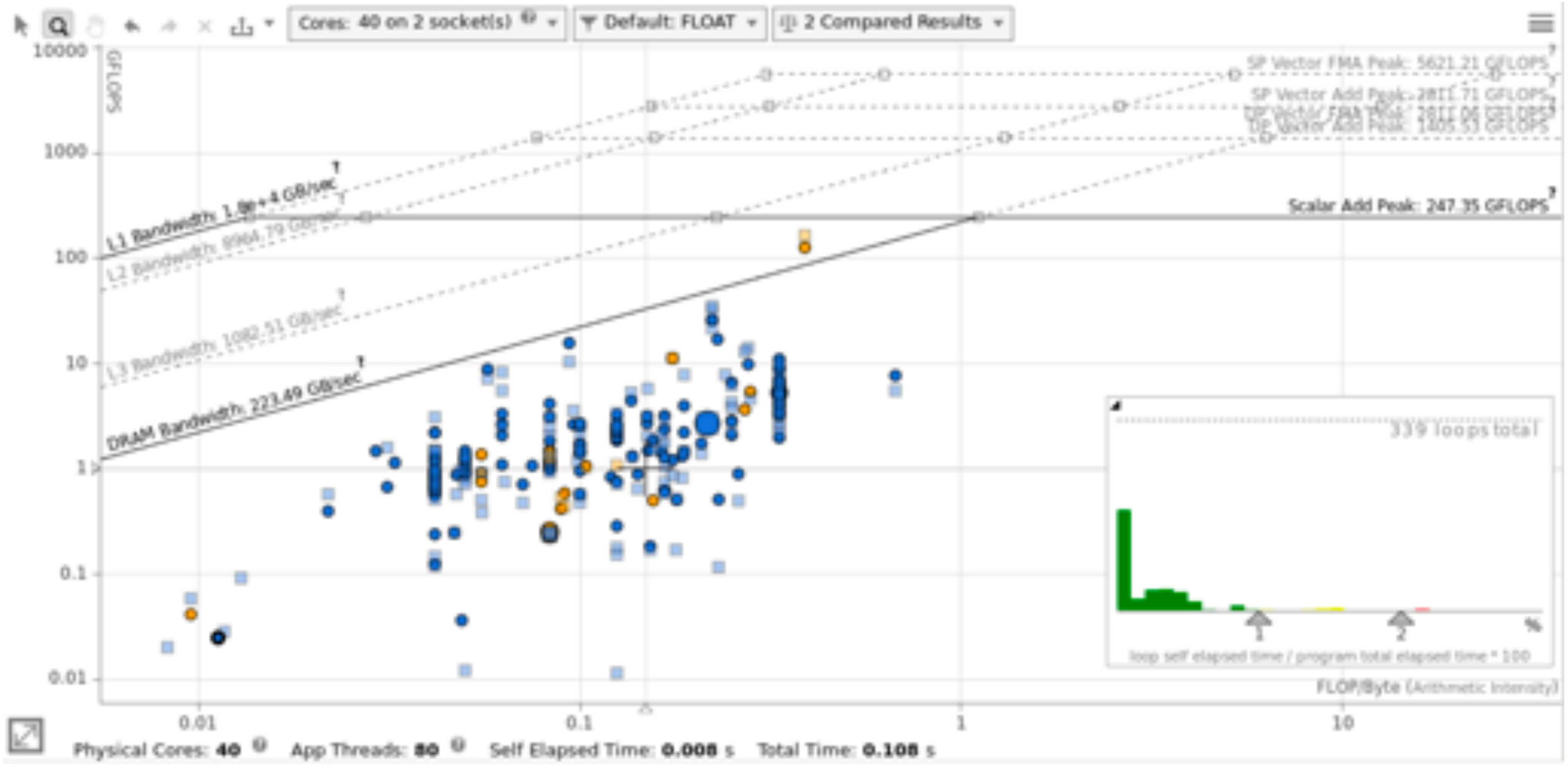


Function Call Sites and Loops	☐	Performance Issues	CPU Time		Type
			Self Time	Total Time	
[loop in __o2avrg_MOD_avrxsd at hfodd_lipcorr.f90:659]	☐		5.952s	5.952s	Scalar
[loop in linavr at hfodd_lipcorr.f90:3045]	☐		4.805s	4.805s	Scalar
[loop in intcou at hfodd.f90:86214]	☐		3.368s	3.368s	Scalar
[loop in denshf_._omp_fn.1 at hfodd.f90:73432]	☐	1 System functi...	3.250s	8.554s	Scalar
[loop in intmas at hfodd.f90:60064]	☐	1 Misaligned loo...	3.116s	3.116s	Scalar
[loop in denshf_._omp_fn.0 at hfodd.f90:73340]	☐		2.666s	3.920s	Scalar
[loop in intmas at hfodd.f90:60063]	☐	1 Misaligned loo...	2.606s	2.606s	Scalar
[loop in spaver_._omp_fn.0 at hfodd.f90:52627]	☐		**2.485s**	**2.485s**	**Vectorized (Body)**
[loop in rotavr at hfodd_lipcorr.f90:2167]	☐		2.477s	2.477s	Vectorized (Body)
[loop in linavr at hfodd_lipcorr.f90:3063]	☐		2.415s	2.415s	Scalar
[loop in rotavr at hfodd_lipcorr.f90:2166]	☐		2.384s	2.384s	Scalar
[loop in linavr at hfodd_lipcorr.f90:3064]	☐		2.320s	2.320s	Scalar
[loop in rotavr at hfodd_lipcorr.f90:2115]	☐	1 Misaligned loo...	2.205s	2.205s	Scalar

UNIVERSITY OF OREGON

http://bit.ly/NUCLEI-Perf

Overall performance



Cores: 40 on 2 socket(s)
Default: FLOAT
2 Compared Results
GFLOPS
10000
1000
100
10
1
0.1
0.01
SP Vector FMA Peak: 5621.21 GFLOPS
SP Vector Add Peak: 2811.71 GFLOPS
DP Vector FMA Peak: 2811.06 GFLOPS
DP Vector Add Peak: 1405.53 GFLOPS
Scalar Add Peak: 247.35 GFLOPS
L1 Bandwidth: 1.8e+4 GB/sec
L2 Bandwidth: 8964.79 GB/sec
L3 Bandwidth: 1082.51 GB/sec
DRAM Bandwidth: 223.49 GB/sec
339 loops total
loop self elapsed time / program total elapsed time * 100
FLOP/Byte (Arithmetic Intensity)
0.01
0.1
1
10

Physical Cores: 40 App Threads: 80 Self Elapsed Time: 0.008 s Total Time: 0.108 s

http://bit.ly/NUCLEI-Perf

HPCL

Tools

UNIVERSITY OF OREGON

http://bit.ly/NUCLEI-Perf

Current state

- Performance analysis
  - A number of vendor tools provide extensive measurement and some analysis capabilities
    - Vendor: Intel Advisor, NVIDIA nsight, PGI Profiler
    - Open source: PAPI, TAU (+ TAU Commander), Vampir, Scalasca, ... (too much of a good thing?)
  - Automation still challenging
  - Some codes have built-in measurement

- Performance optimization
  - Largely manual
  - A few research autotuners

My wish list

- In the next couple of months:
  - Build knowledge on the current capabilities and needs
  - Create a low-overhead forum for sharing performance analysis and optimization findings and approaches

- Longer term
  - Automation of repetitive analyses that have been shown to be useful to at least one code team
  - More methodical autotuner development based on actual application needs

HPCL

Summary

Architectures:

- Heterogeneous
  - multicore CPUs (10s of threads per socket),
  - manycore (100s of threads per device)
  - several GPUs per node (1000s of threads per device)
  - more networking layers (on and off node)
  - in memory computing, FPGAs, ...
- Ubiquitous vectors
- I/O prohibitively expensive

Given: No single programming language/model/library clearly best.

➔ What can we do to make optimization less effort-intensive without sacrificing code portability and maintainability?

Our General Approach to Empirical Analysis/Modeling

- Define reusable, extensible workflows to collect and perform analysis
- Currently
  - Use TAU C
  - Use Pythor
  - Jupyter notebooks for initial developmen ar ring results with application scientists

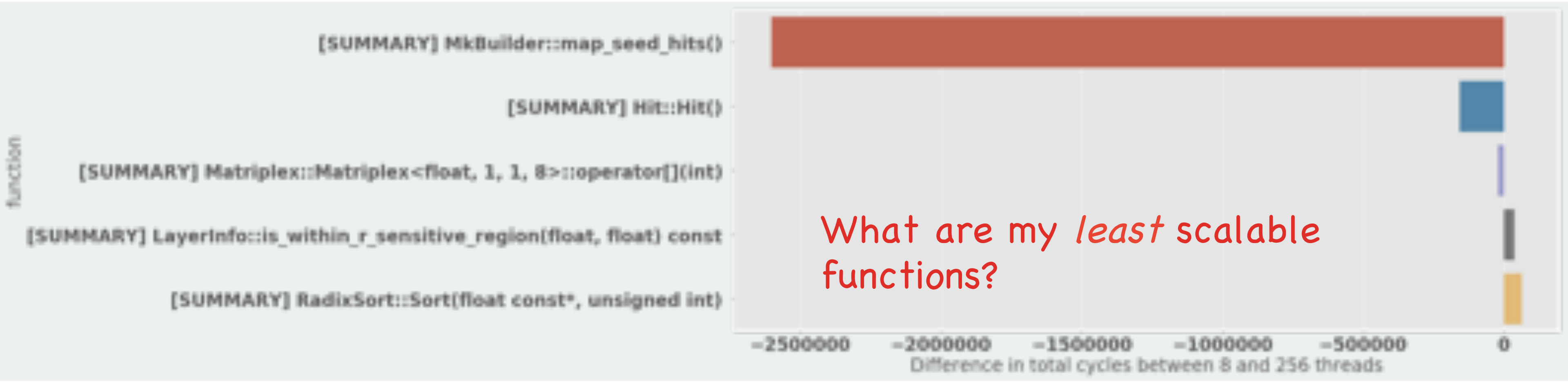
You may be tempted to google "TAU Commander."
It's not that one.
See http://taucommander.paratools.com

- Long-term goals -- eliminate the nee rr nce expert" help, enable more thorough and freque pe or e te g & analysis, reproducibility!

UNIVERSITY OF OREGON

http://bit.ly/NUCLEI-Perf

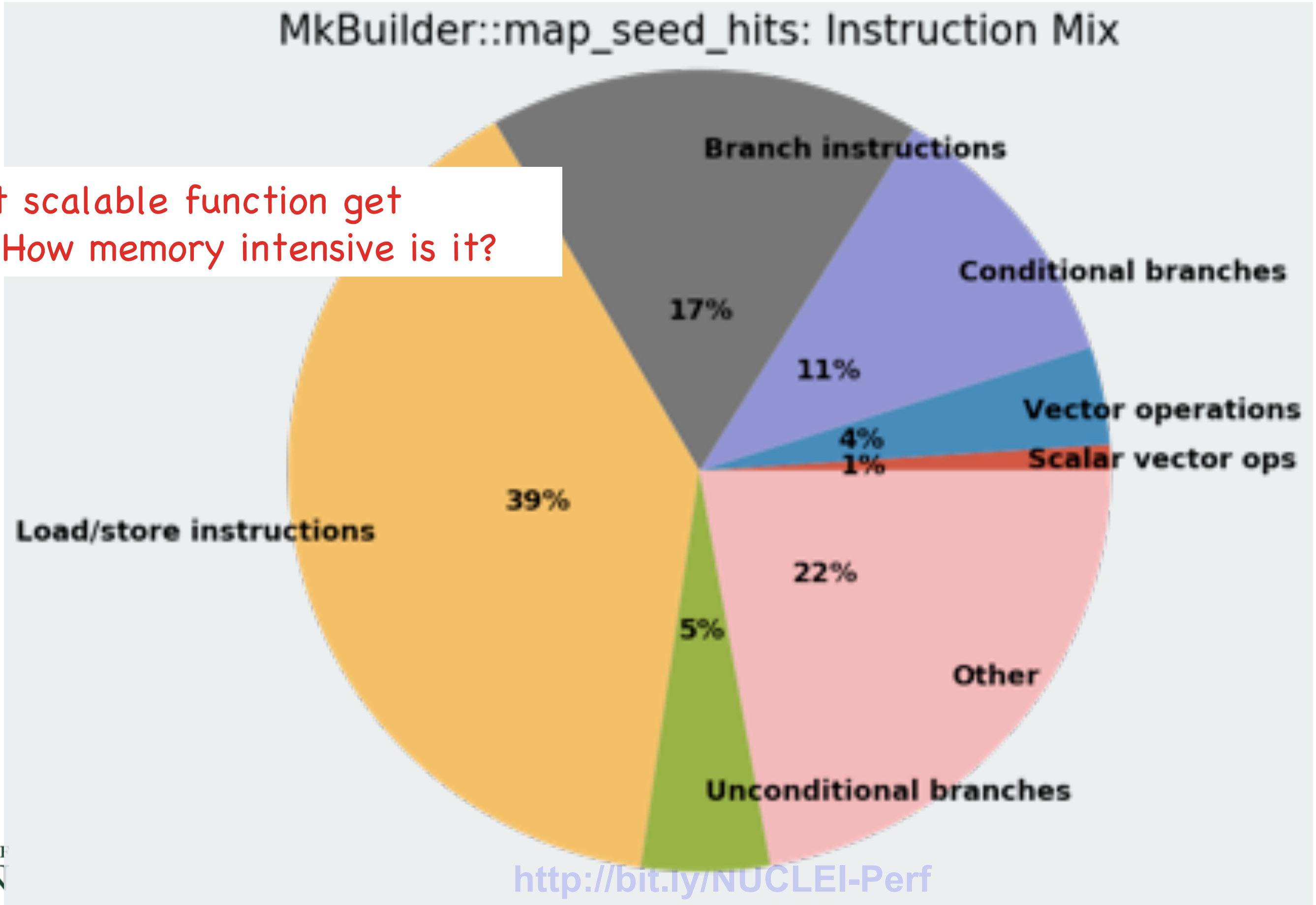
Example Empirical Analysis Q&A

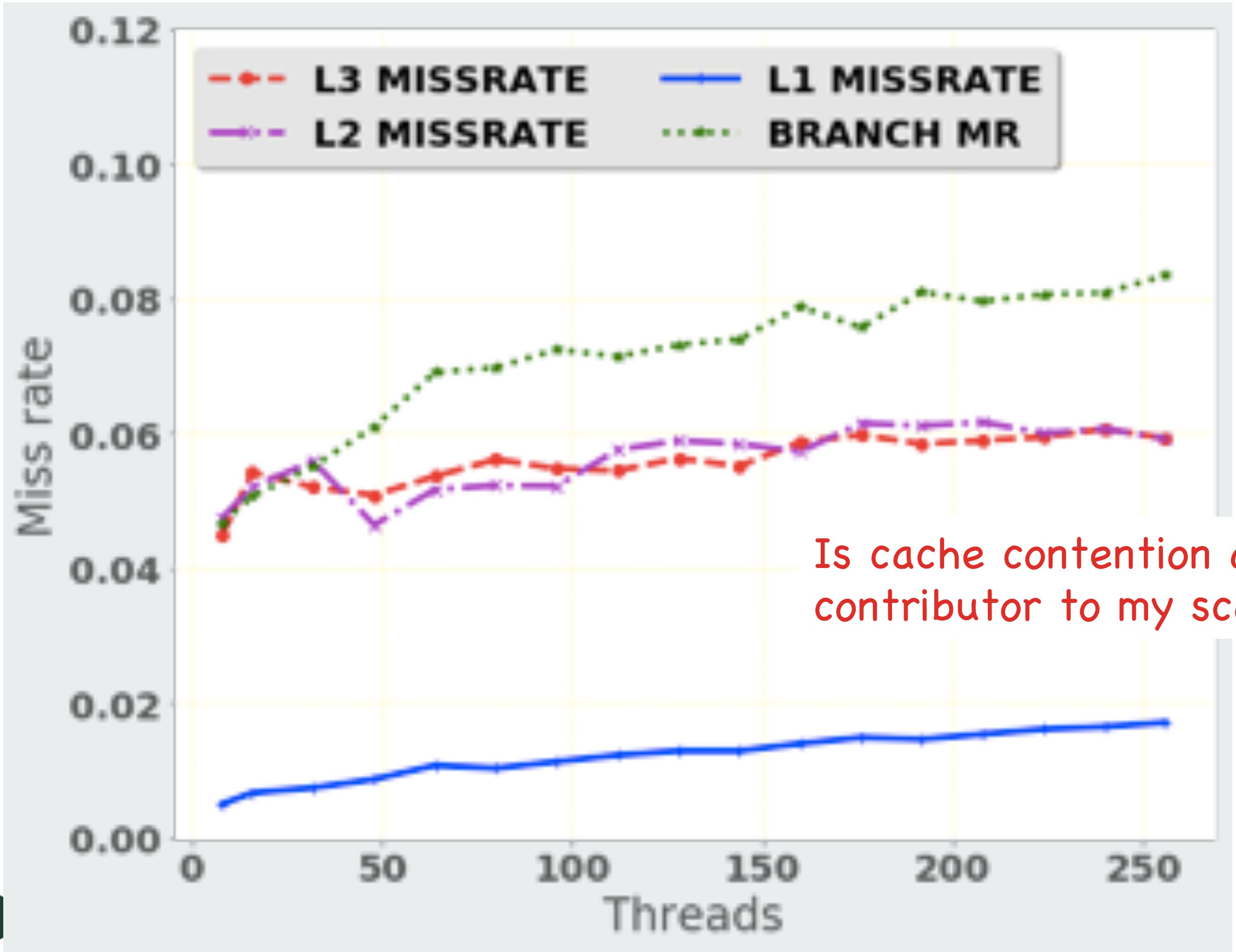


Cerati et. al, “Parallelized and Vectorized Tracking Using Kalman Filters with CMS Detector Geometry and Events”, 2018. https://arxiv.org/abs/1811.04141

http://bit.ly/NUCLEI-Perf

Did my least scalable function get vectorized? How memory intensive is it?





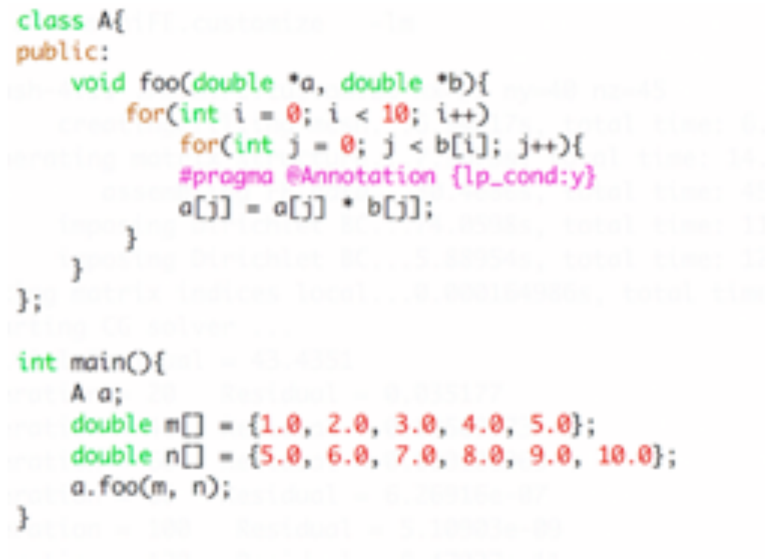
Is cache contention a significant contributor to my scaling problems?

Building Models

- Empirical – linear regression most common
- Static
  - Use source or binary code analysis
  - Parameterize by (some) architectural features

Example static modeling result

Example source code:

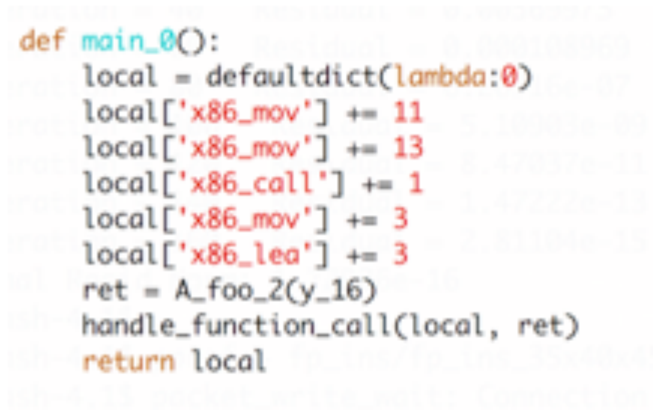


```
class A{
public:
    void foo(double *a, double *b){
        for(int i = 0; i < 10; i++)
            for(int j = 0; j < b[i]; j++){
            #pragma @Annotation {lp_cond:y}
            a[j] = a[j] * b[j];
        }
    }
};

int main(){
    A a;
    double m[] = {1.0, 2.0, 3.0, 4.0, 5.0};
    double n[] = {5.0, 6.0, 7.0, 8.0, 9.0, 10.0};
    a.foo(m, n);
}
```

ROSE-based code analysis

The Python "model" with instruction counts:



```
def main_0():
    local = defaultdict(lambda:0)
    local['x86_mov'] += 11
    local['x86_mov'] += 13
    local['x86_call'] += 1
    local['x86_mov'] += 3
    local['x86_lea'] += 3
    ret = A_foo_2(y_16)
    handle_function_call(local, ret)
    return local
```

K. Meng and B. Norris. "Mira: A framework for static performance analysis." *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 103–113, Sept 2017.

http://bit.ly/NUCLEI-Perf

Another (Newer) Example output

Matrix representation

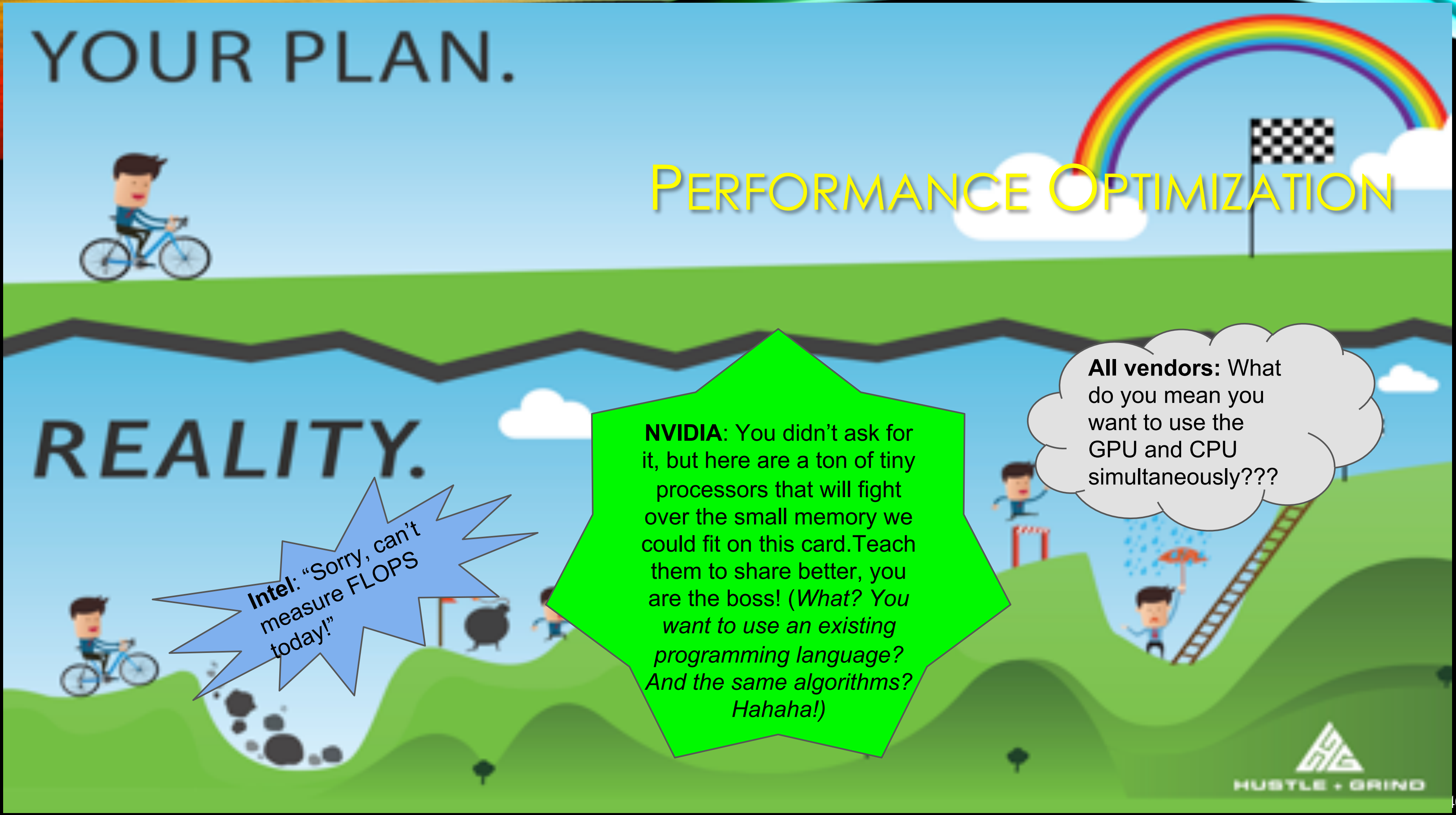
```
5
0:0:1:0 0.000000e+00    1.000000e+00    0.000000e+00    0.000000e+00    0.000000e+00
0:2:1:0 0.000000e+00    0.000000e+00    1.000000e+00    0.000000e+00    0.000000e+00
2:6:1:1 0.000000e+00    0.000000e+00    8.000000e-01    2.000000e-01    0.000000e+00
0:0:1:1 0.000000e+00    7.900000e-01    0.000000e+00    0.000000e+00    2.100000e-01
0:0:1:1 7.500000e-01    2.500000e-01    0.000000e+00    0.000000e+00    0.000000e+00
```

Total number of nodes (basic block)

floating-point : memory ops : control ops : integer ops   transition probabilities to other nodes

http://bit.ly/NUCLEI-Perf

Performance Optimization



Intel: "Sorry, can't measure FLOPS today!"

NVIDIA: You didn't ask for it, but here are a ton of tiny processors that will fight over the small memory we could fit on this card.Teach them to share better, you are the boss! (*What? You want to use an existing programming language? And the same algorithms? Hahaha!*)

All vendors: What do you mean you want to use the GPU and CPU simultaneously???

Current optimization approaches

- Eliminate unnecessary computation
- Use optimized methods (libraries) – and use them well!
- Create new algorithms when old approaches don’t map well to current architectures
- Rethink data structures
- Low-level optimizations
  - Manual: rewrite code so loops can be better optimized by compilers, or as a last resort, write low-level optimized code
  - Automatic: use a compiler-like tool to generate optimized code (autotuners)