Refactoring Climate Applications for Many-core Xeon Processors: Recent Developments

Dr. Richard Loft
Director, Technology Development
Computational and Information Systems Laboratory
National Center for Atmospheric Research

ECMWF 16th HPC Met Workshop
Reading, UK
October 30, 2014

Computational & Information Systems Laboratory

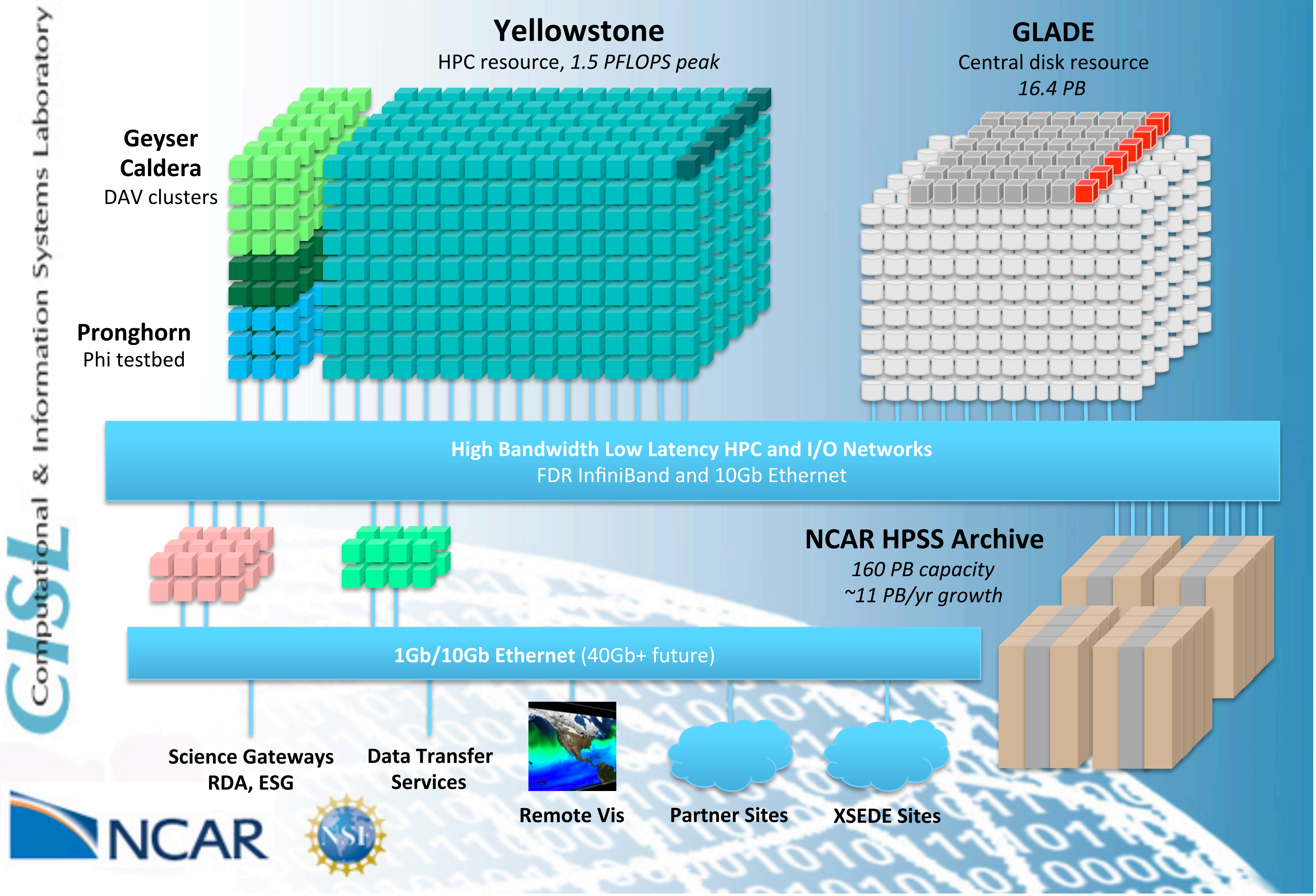
Outline

- How is NCAR's Yellowstone supercomputer being used ?
- Organization of Climate Optimization Efforts
- Early Results
- Optimization Methodology
- Conclusions

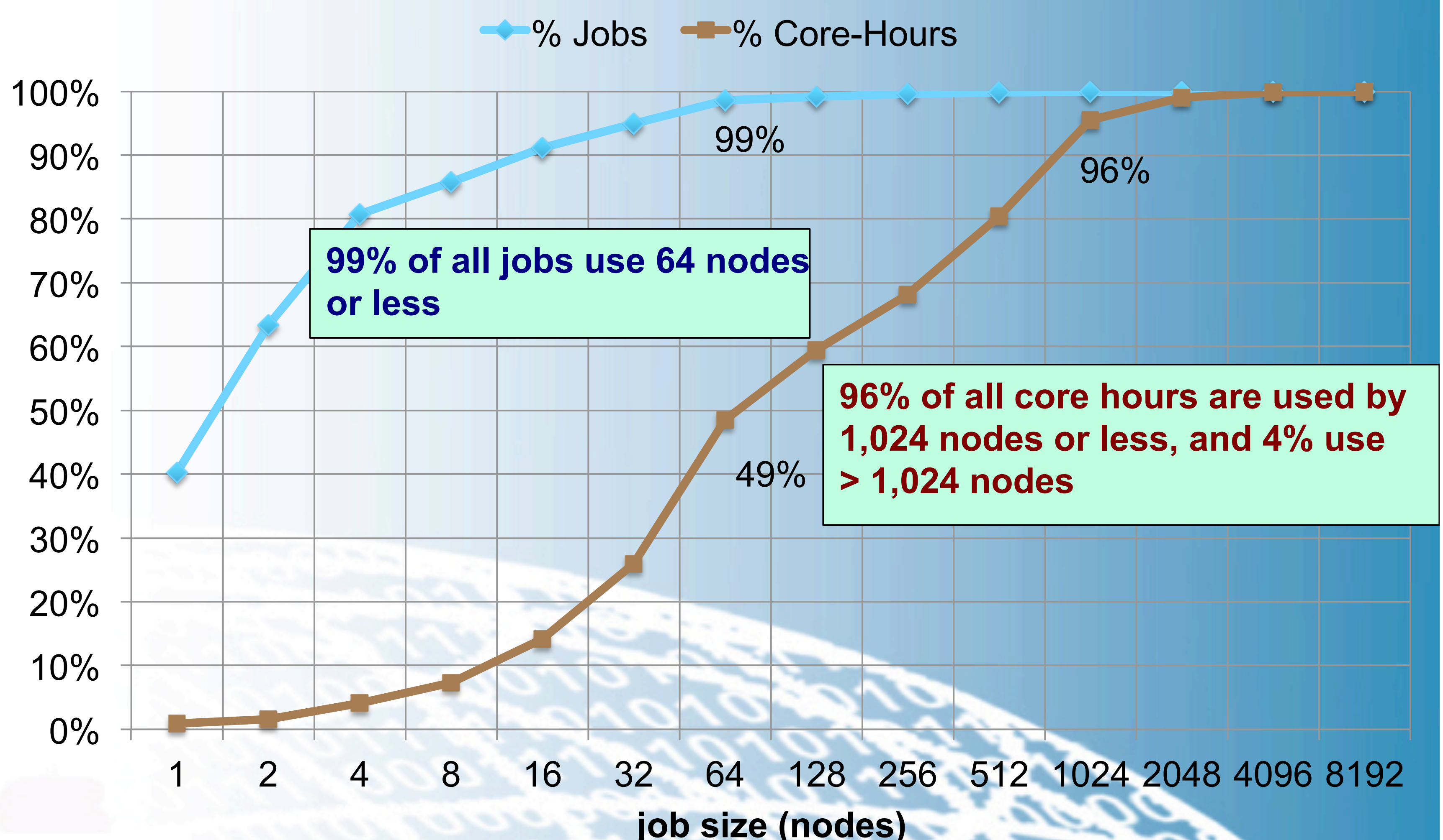
Computational & Information Systems Laboratory

How is Yellowstone being used?

Yellowstone Environment



Most jobs are small jobs; **but** ~50% of core-hours are consumed by jobs > 64 nodes

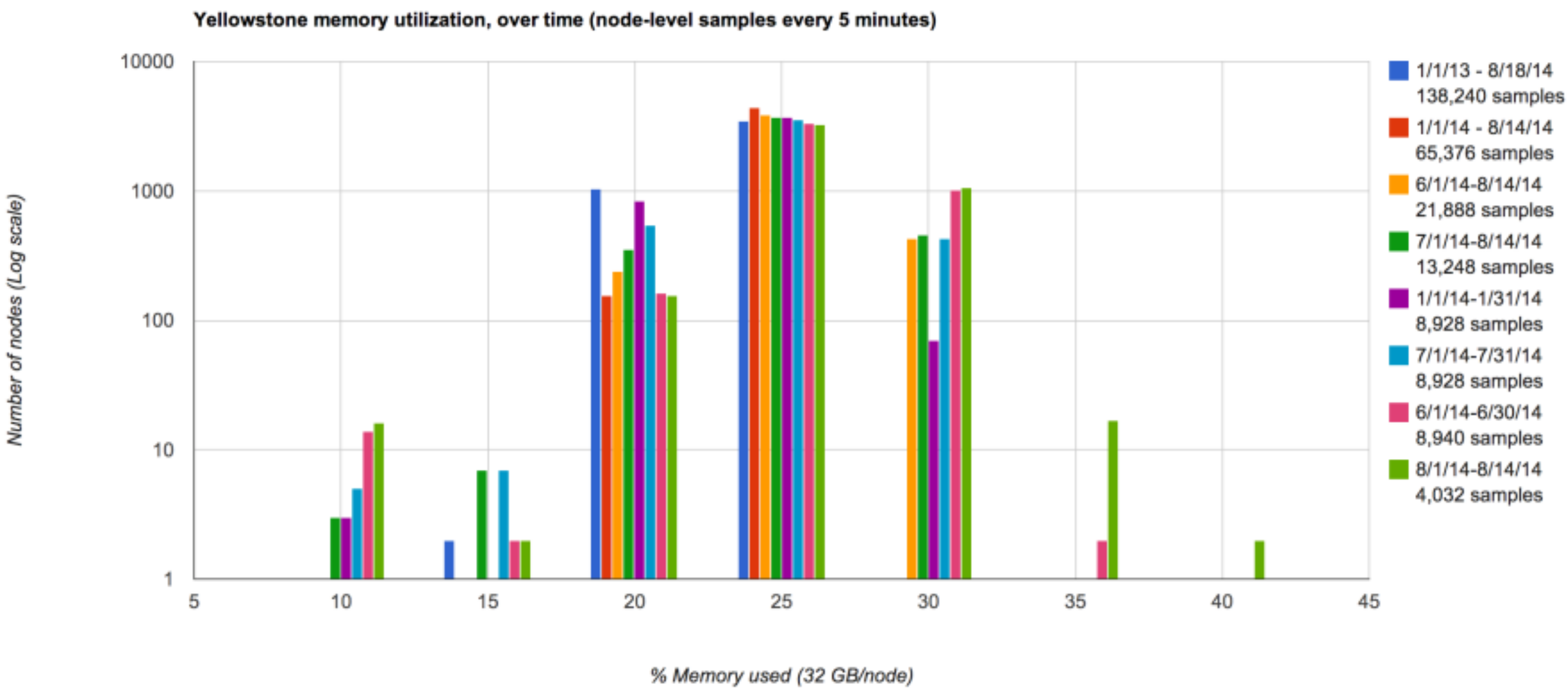


Computational & Information Systems Laboratory

CISL

NCAR

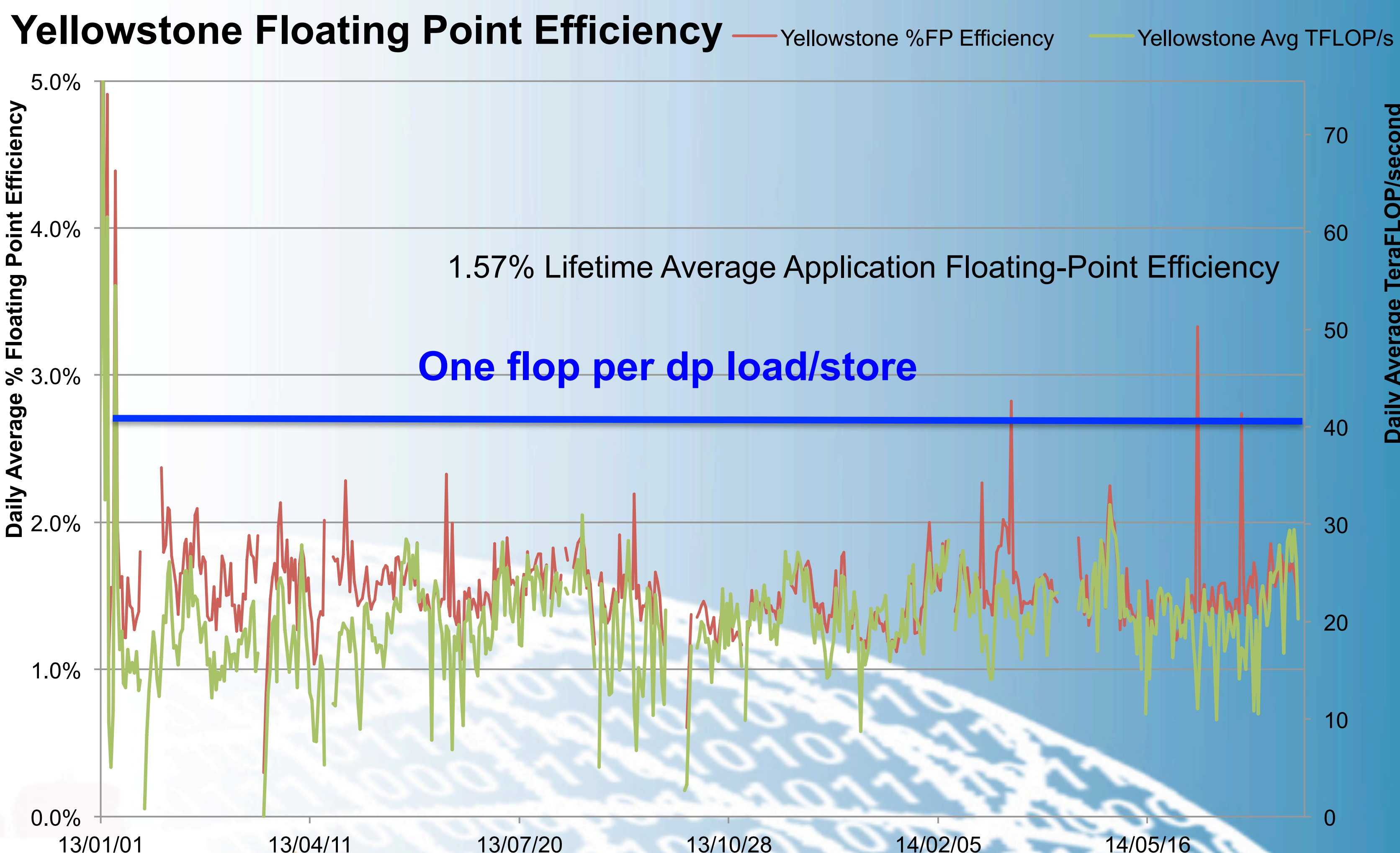
On average, applications use about ~25% of Yellowstone's available memory



- Yellowstone has 32 GB of memory per node (2 GB/core)
- Data collection for various intervals
- Collected from each node every 5 minutes, then averaged

NCAR

Daily Average Floating Point Yellowstone Fraction of Peak



Computational & Information Systems Laboratory

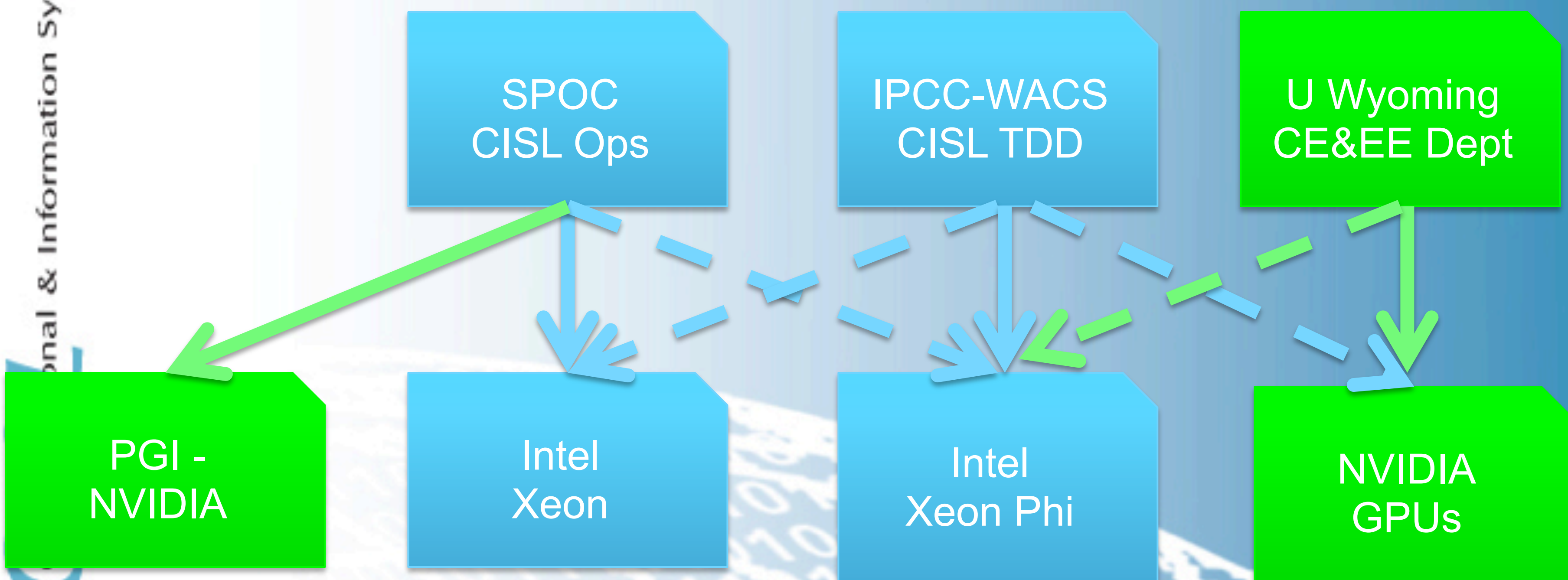
CISL

NCAR

Outline

- How is NCAR's Yellowstone supercomputer being used ?
- Organization of Application Optimization Efforts
- Early Results
- Optimization Methodology
- Conclusions

Organization of CISL Application Optimization Efforts



The Intel Parallel Computing Center for Weather and Climate Simulation (IPCC-WACS)

- IPCC-WACS is an Intel funded effort focused on future technologies involving computer scientists, graduate students, Intel, and international partners.
- A partnership of NCAR, University of Colorado, Boulder and IIS in Bangalore, India.
- Its aim is the development of tools and knowledge to help with the development and performance improvement of CESM, WRF, and MPAS on Intel Xeon and Xeon Phi architectures.

SPOC's Motivation

- Strategic Parallel Optimization Computing
- ***Application performance improvements achieved through optimization are likely to surpass in importance those obtained from mere architectural improvements.***
- This will increase the amount of productive science that can be performed on Yellowstone ***today.***
- Improvements made now will accrue to ***future systems.***

Resource Overview

- **Investments**
  - 6 new staff over last 2 years.
- **Partners**
  - **DoE** - ACME (new)
  - **G8 ECS project** (3 yrs - concluded)
  - **Intel Parallel Computing Center – CU & IIS** (< 1 year)
  - **University of Wyoming** GPU Collaboration (very new)
- **Workshops/Meetings**
  - NCAR "Heterogeneous Multi-core Platforms Workshop" (4 yrs)
- **Platforms**
  - U.S.: Yellowstone, Titan, Stampede, Babbage, etc…

Outline

- How is NCAR's Yellowstone supercomputer being used ?
- Organization of Application Optimization Efforts
- Early Results
- Optimization Methodology
- Conclusions

Computational & Information Systems Laboratory

CISL

NCAR

NSF

SPOC single node MPAS dynamics optimizations on Yellowstone

Bob Sinkovits, SDSC

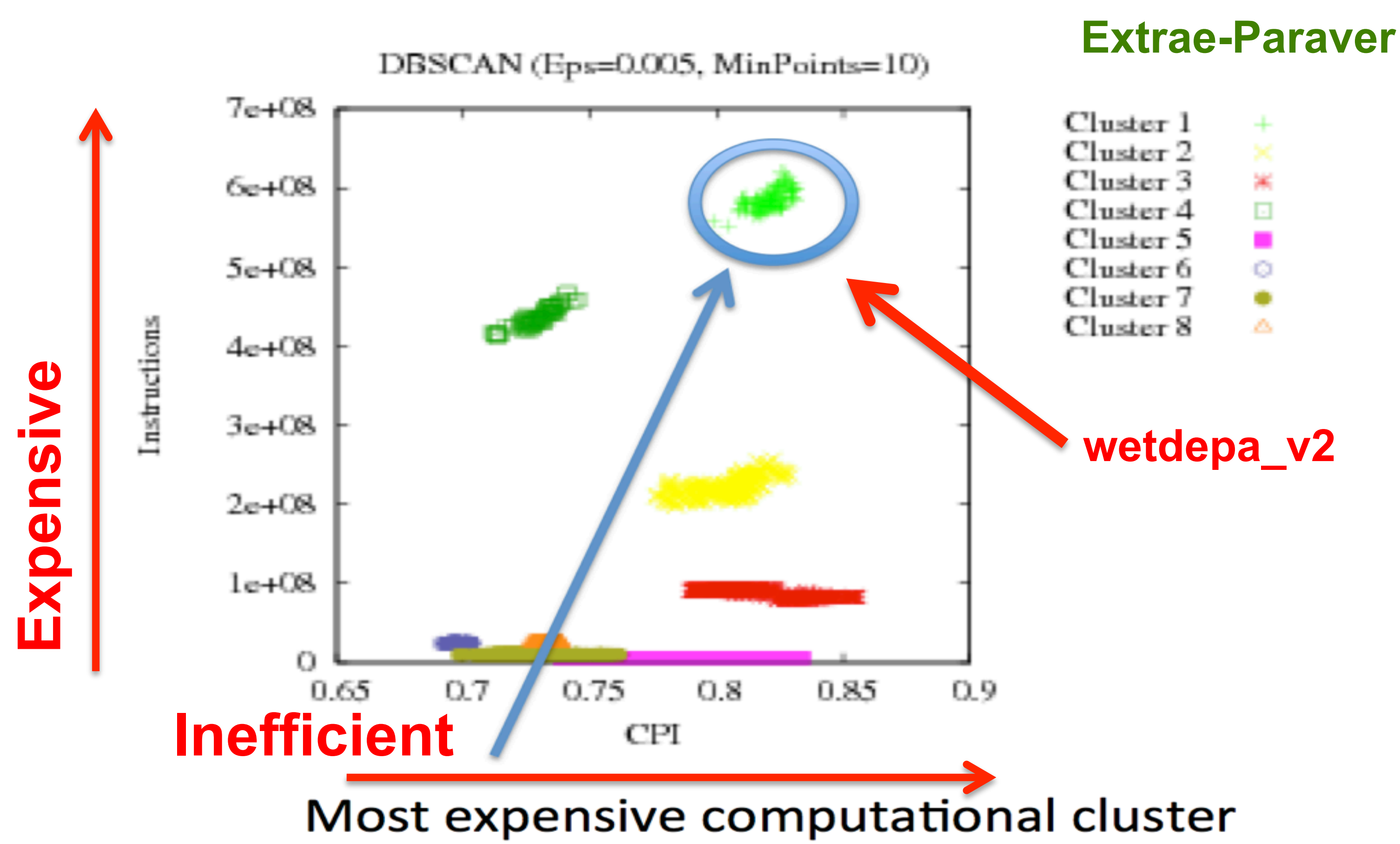
Basic optimizations to speed up atm_advance_scalars in MPAS dynamics by 2x for 106-scalar "scalar" advection case:

- **Use Intel 13.1.2 compiler with -O3 -xHost -no-prec-div**
- **Precomputation of scalar weights.**
- **Special treatment of the case of an edge between two hexagonal cells.**
- **Taking advantage of fact that coef_3rd_order is non-negative to avoid unnecessary multiplication.**
- **Flattening nested loops over vertical levels and scalars into a single loop.**
- **Replacement of divisions by cell areas with multiplications by inverse.**
- **Avoiding initialization of entire flux array to zero. Not needed for special case of edge between hexagonal cells. For default case, distinguish first iteration of loop over nAdvCellsForEdge to avoid need for initialization.**

IPCC-WACS Results

Computational & Information Systems Laboratory

BSC tools helping to find high priority sections that are expensive *and* inefficient.



- Result of an Extrae trace of CESM on Yellowstone.
- Vertical ~ time; horizontal ~ 1/flops

Using profiler results to manually speed up wetdepa

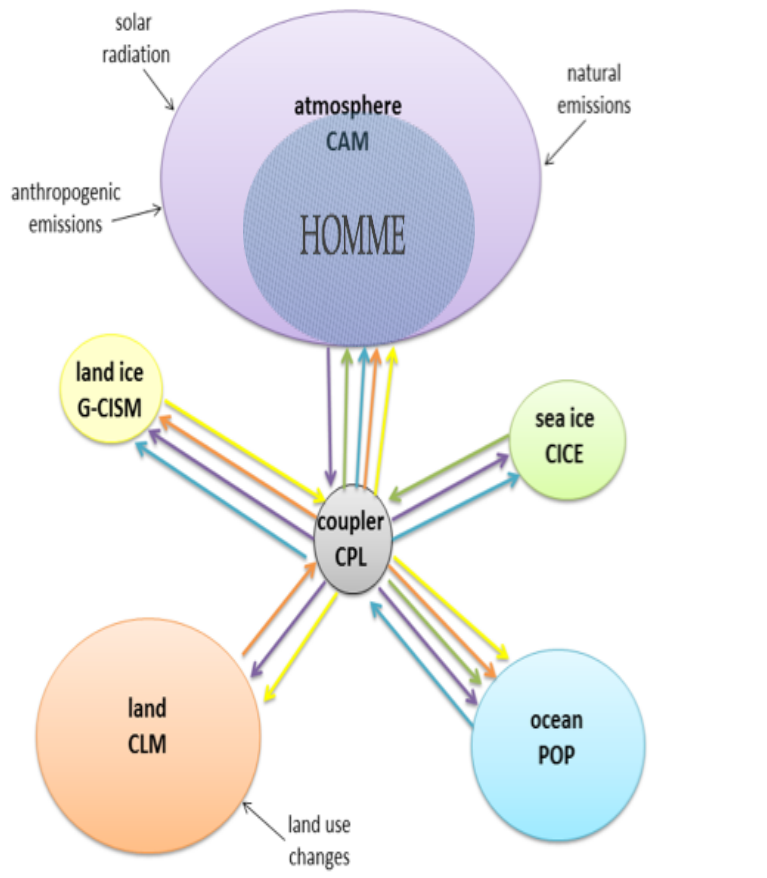
Single core

	Intel Phi (Intel 13.1.1)			Intel Sandybridge (Intel 13.1.2)		
	-O2	-O3	-O3 -fast	-O2	-O3	-O3 -fast
orig	42.85	41.24	3.74	3.43	3.32	0.97
mod	6.50	6.61	4.58	1.09	1.12	1.04

- wetdepa is small only ~600 lines
- Restructured branched loops + promoted scalars to vectors.
- -O3 -fast for original code gave incorrect results
- 2.5% to 0.7% of code execution time = $222K savings

Significant gains possible from code refactoring!

Review of CESM Nomenclature

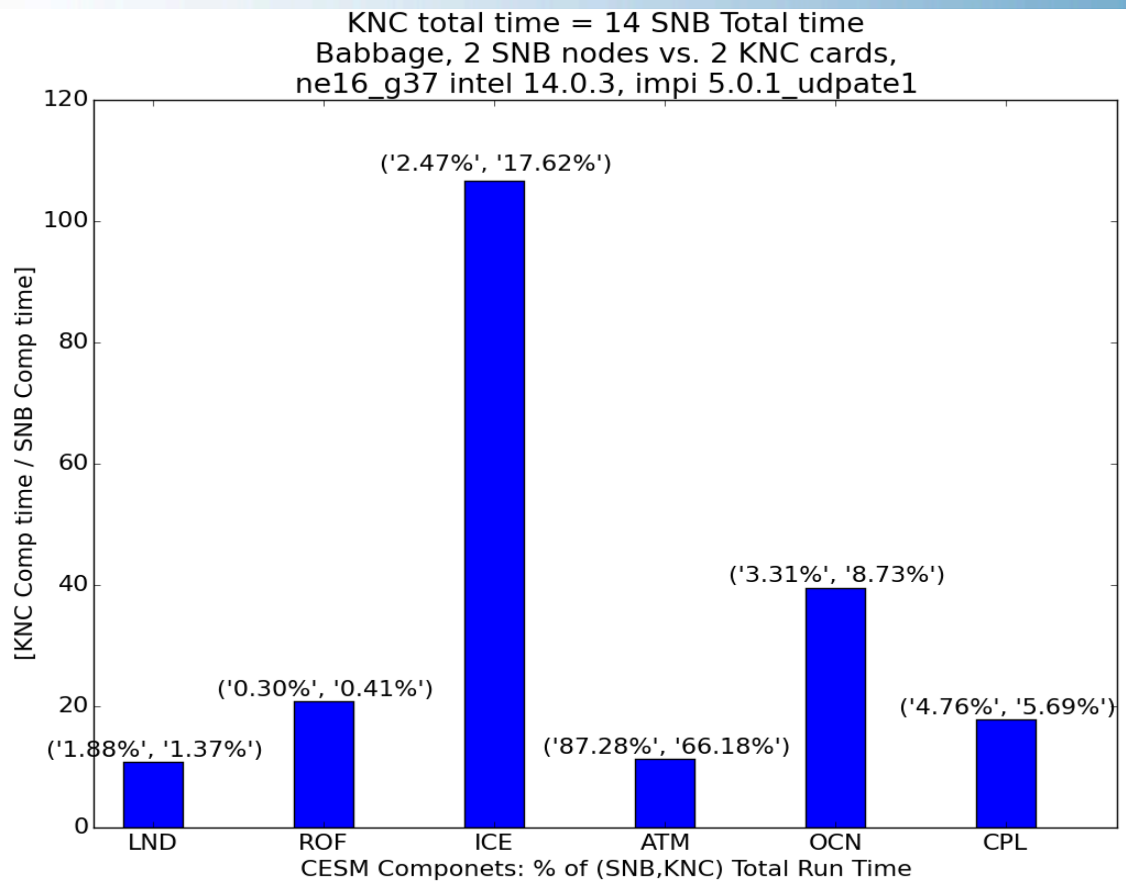


K, Alexander, S. Easterbrook, 2011: h>p://climatesight.files.wordpress.com/2011/12/agu.pdf

- **FIDEAL = dynamical core only with all fluxes from data files [HOMME + coupler].**
- **FC5 ~ CAM-SE =HOMME (dynamical core) + atmospheric physics and chemistry + active land (CLM)**
- **BC5 -fully coupled version.**

Functionality Port of CESM to KNC

*not optimal PE-component layout



Neither FC5 nor BC5 configurations throughput have been optimized for KNC .

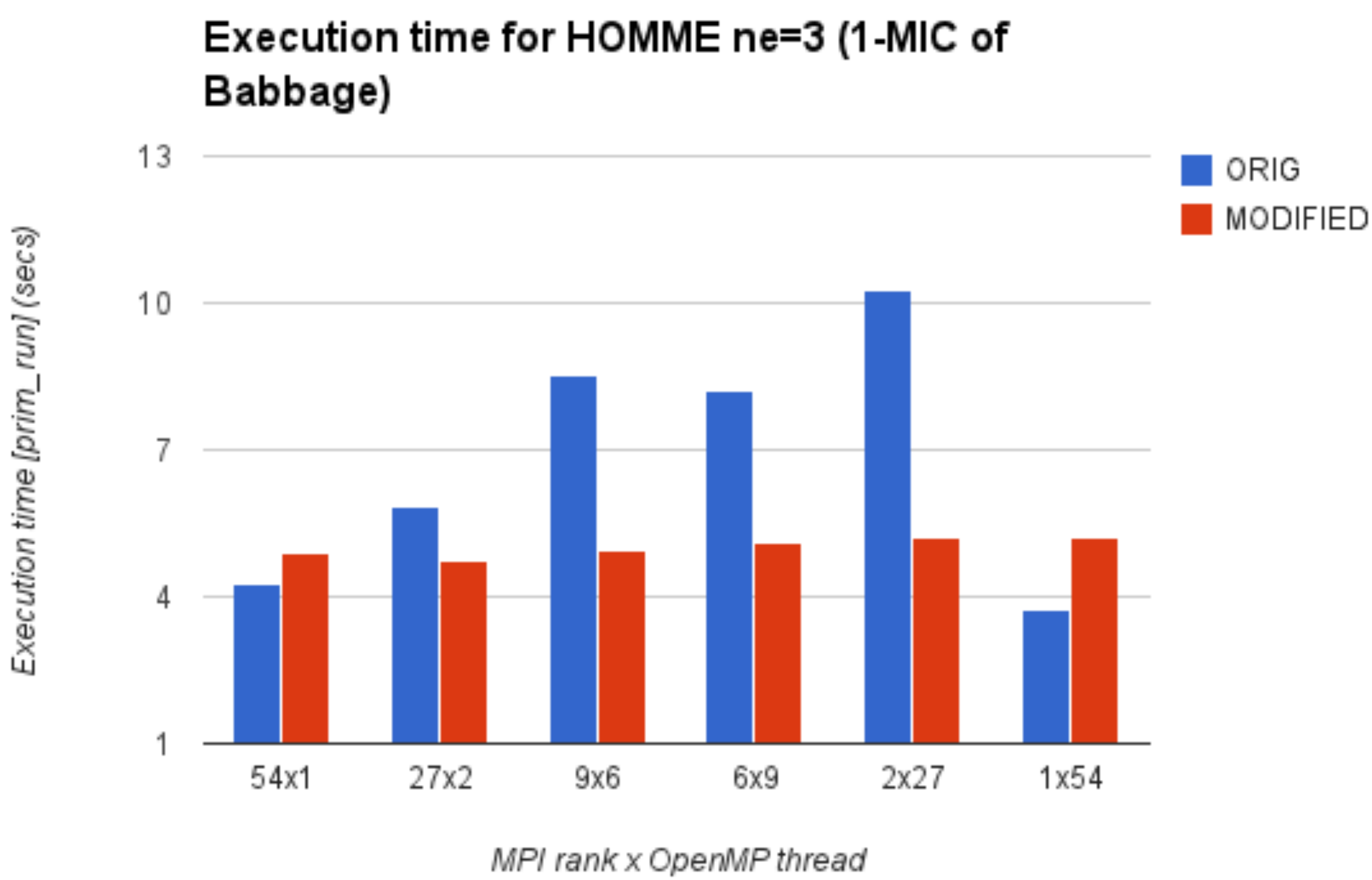
Computational & Information Systems Laboratory

CISL

NCAR

NSF

Spectral Element Thread Scaling Improvements



Computational & Information Systems Laboratory

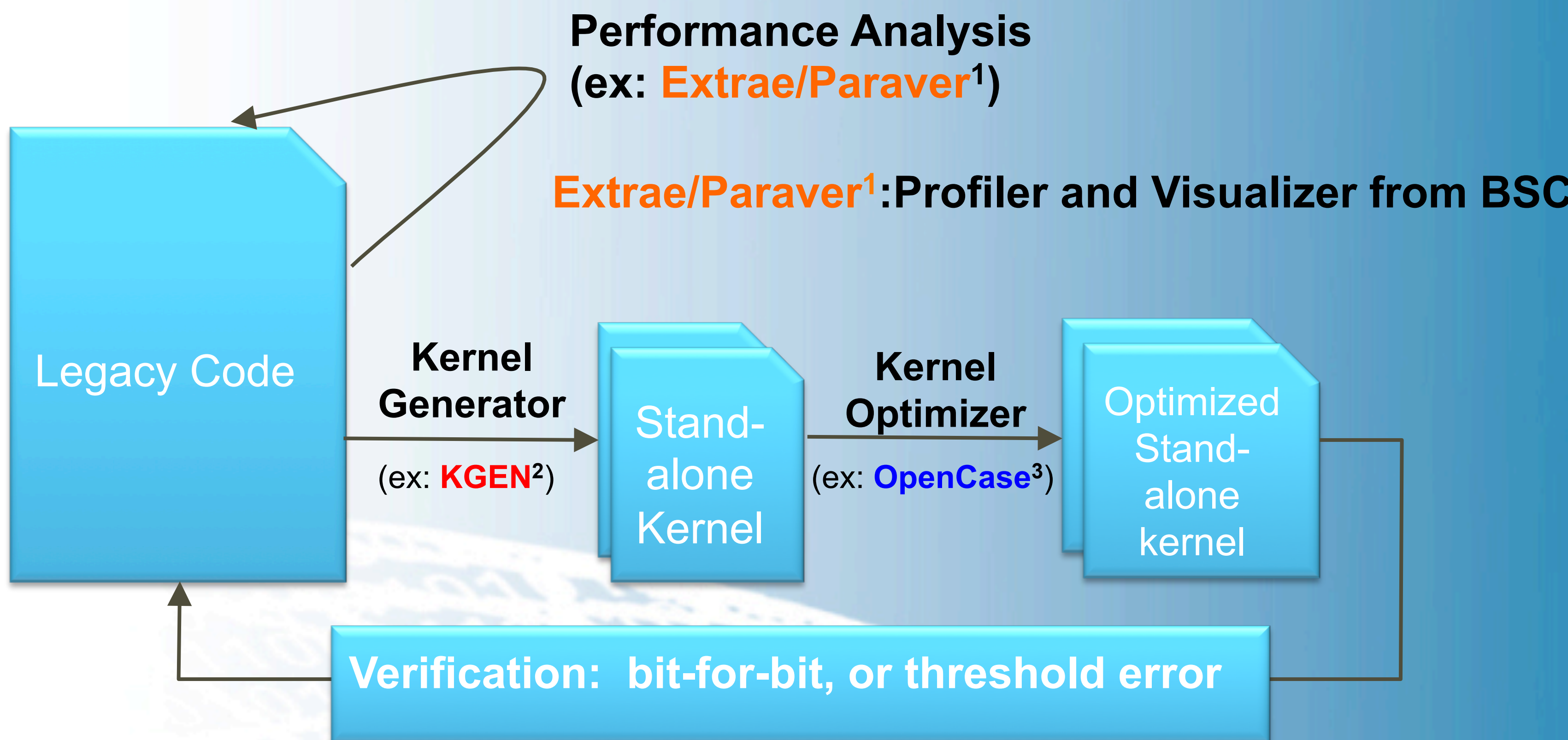
Outline

- How is NCAR's Yellowstone supercomputer being used ?
- Organization of Application Optimization Efforts
- Early Results
- Optimization Methodology
- Conclusions

How do you refactor entire codes?

- **Lots of code (1.5M lines in CESM)**
  - Single source multiple architectures
  - portability & performance resilience
- **Code is rapidly changing**
  - Optimization cycle of 1 month – OK
  - Optimization cycle of 6 months – start over
- **Verification/validation requirements**
  - Most restrictive in climate but some sort of software verification/science validation is always required.

Refactoring science code fast enough to keep up



KGEN[2]: Semi-automated Kernel Generator from NCAR
OpenCase[3]: User-directed Performance Optimizer
(YS Kim's Ph.D Research)

Computational & Information Systems Laboratory

Kernel Generator

Computational & Information Systems Laboratory

CISL

NCAR

NSF

KGEN: Kernel Generator
Current status

Computational & Information Systems Laboratory

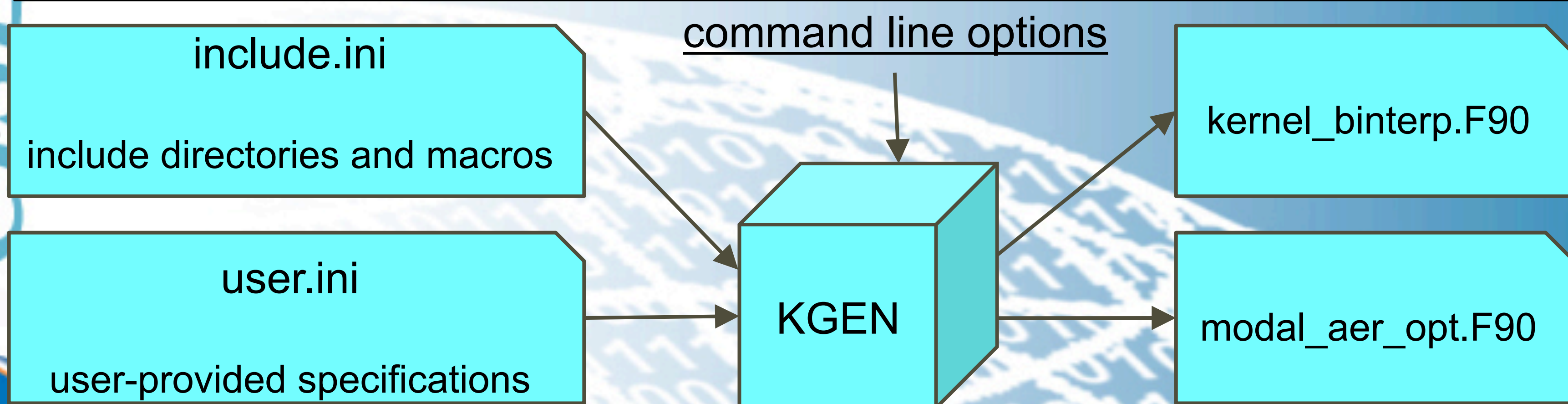
- **Overview**
  - Written in Python
  - Extension of Fortran parser distributed in F2PY python package
- **Semi-automated approach**
  - KGEN tries to generate kernel fully automated way
  - If it fails, user provide required information through familiar INI configuration file
    - Fortran specification, files to search first, names to ignore, etc.
- **Current limitations**
  - Derived type having allocatable or pointer component is not supported
  - Array of derived types is not supported

KGEN Example: PORT
(a stand-alone code of CESM/RRTMG)

"binterp" subroutine identified as one of hotspots in PORT using Extrae/Paraver

```
KERNEL_FILE := modal_aer_opt.F90
KERNEL_FUNCTION := binterp

kgen -p include.ini -i user.ini -n 1 -m 0,1,2 ${KERNEL_FILE}:${KERNEL_FUNCTION} ${KERNEL_FILE}:639
```



KGEN Example: PORT - Continue

modal_aer_opt.F90

```
subroutine modal_aero_sw(list_idx, state, pbuf, nnite,
idxnite, tauxar, wa, ga, fa)

  ! save input state
  WRITE(UNIT = kgen_unit) itab
  WRITE(UNIT = kgen_unit) lbound(refitabsw, 1)
  WRITE(UNIT = kgen_unit) ubound(refitabsw, 1)
  WRITE(UNIT = kgen_unit) lbound(refitabsw, 2)
  WRITE(UNIT = kgen_unit) ubound(refitabsw, 2)
  WRITE(UNIT = kgen_unit) refitabsw
          …
  ! call kernel
  CALL binterp(extpsw(:,:,:,isw), ncol, … )
          …
  ! save output state
  WRITE(UNIT = kgen_unit) utab
  WRITE(UNIT = kgen_unit) lbound(refitabsw, 1)
  WRITE(UNIT = kgen_unit) ubound(refitabsw, 1)
  WRITE(UNIT = kgen_unit) lbound(refitabsw, 2)
  WRITE(UNIT = kgen_unit) ubound(refitabsw, 2)
  WRITE(UNIT = kgen_unit) refitabsw
          …
end subroutine
```

kernel_binterp.F90

```
program kernel_binterp
  …
  ! type declaration statements
  INTEGER :: itab(pcols)
  REAL(KIND = r8) :: utab(pcols)
  REAL(KIND = r8), POINTER :: refitabsw(:, :)
  …
  ! read state
  READ(UNIT = kgen_unit) utab
  READ(UNIT = kgen_unit) kgen_bound(1, 1)
  READ(UNIT = kgen_unit) kgen_bound(2, 1)
  READ(UNIT = kgen_unit) kgen_bound(1, 2)
  READ(UNIT = kgen_unit) kgen_bound(2, 2)
  ALLOCATE(refitabsw(kgen_bound(2, 1) - kgen_bound(1, 1) + 1, kgen_bound(2, 2) -
kgen_bound(1, 2) + 1))
  READ(UNIT = kgen_unit) refitabsw
  …
  ! call kernel
  CALL binterp(extpsw(:,:,:,isw), ncol, … )
  …
  ! verify output from a generated kernel
  IF ( ALL( outstate_itab == itab ) ) THEN
     WRITE(*,*) "itab is IDENTICAL."
  ELSE
     WRITE(*,*) "itab is NOT IDENTICAL."
     WRITE(*,*) "RMS of difference is ", &
     sqrt(sum((outstate_itab - itab)**2)/real(size(outstate_itab)))
  ENDIF
  …
end program
```

OpenCase

OpenCase:
User-directed performance optimizer

- **Programming techniques for manual optimization**
  - Source modification
    - loop unrolling, loop splitting, loop merge, data structure refactoring…
  - Compiler options
    - alignment options, binary generation options, vectorization options, prefetching options, …
  - Intrinsic or assembly modification
- **Can we mimic what programmer manually does in more automated way?**
  - <u>Try every combination of the techniques</u>

OpenCase - Features

- **Source code transformation**
  - loop transformation: unrolling, split, merge, interchange
  - primitive: line insertion/deletion
  - array: index swap
- **Compiler flags**
  - any arrangement of compiler flags
- **Performance measurement and verification**
  - Measure performance result and sort
  - Measure output and verify against tolerance or other factors
- **Case generation algorithm**
  - current research topic

OpenCase: Case generation

- **Environmental variables**
  - KMP_AFFINITY=(balanced; compact)1
- **Compiler Options**
  - (-O2; -O3)
  - {-align array64byte, -opt-prefetch=(0;3)1, '-no-prec-sqrt', '-no-prec-div'}*
- **Source transformation**
  - Loop unrolling, merge, split, interchange
  - Line insert/delete
  - ...

=> 356,483,072 cases

OpenCase example: divergence_sphere() from HOMME

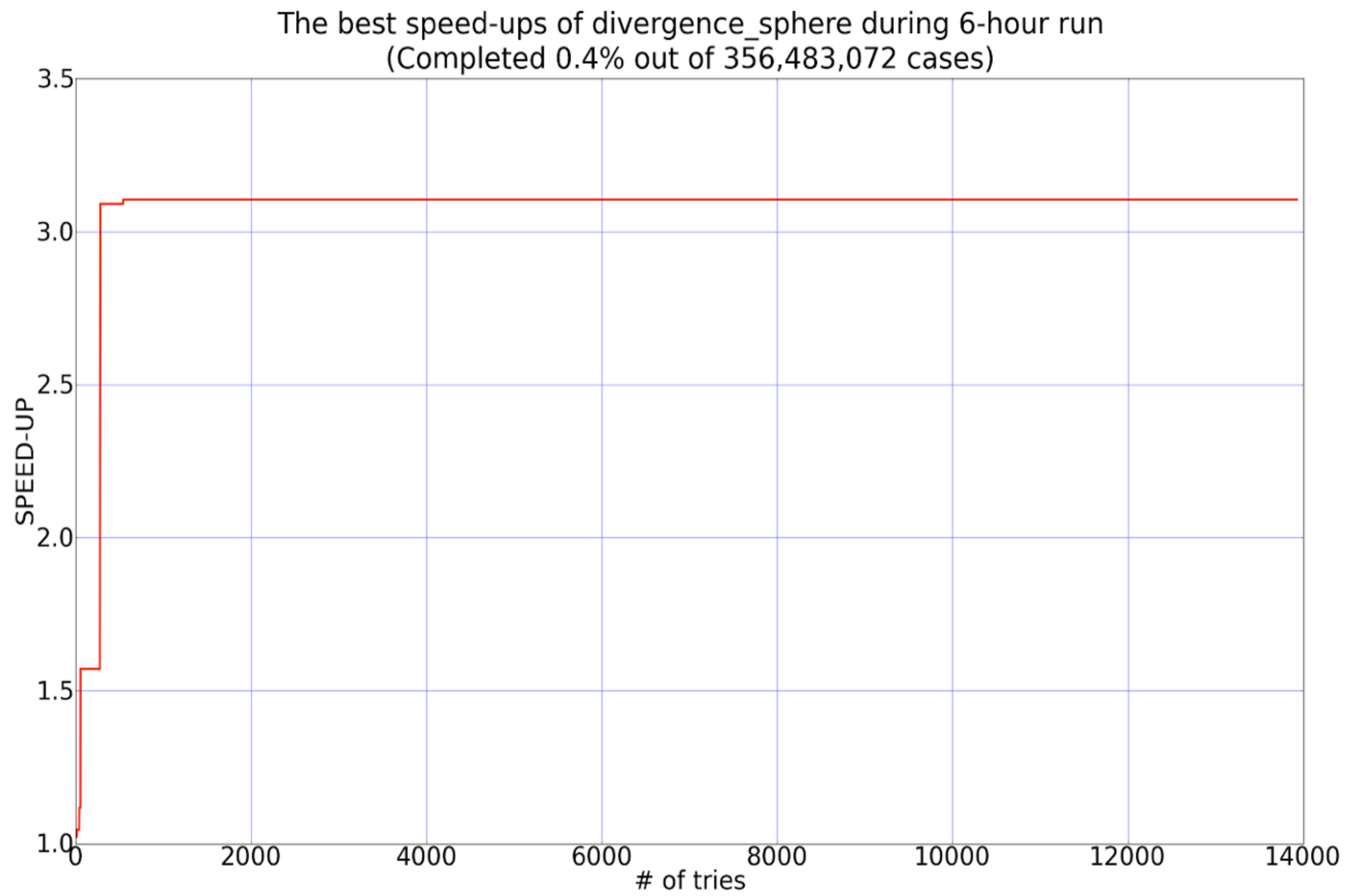
subroutine divergence_sphere()

```
do j=1,np
  do i=1,np
    gv(i,j,1)=elem_metdet(i,j)*(elem_DinvC(i,j,1,1)*v(i,j,1) + elem_DinvC(i,j,1,2)*v(i,j,2))
    gv(i,j,2)=elem_metdet(i,j)*(elem_DinvC(i,j,2,1)*v(i,j,1) + elem_DinvC(i,j,2,2)*v(i,j,2))
  enddo
enddo

! compute d/dx and d/dy
do j=1,np
  do l=1,np
    dudx00=0.0d0
    dvdy00=0.0d0
    do i=1,np
      dudx00 = dudx00 + deriv_Dvv(i,l  )*gv(i,j  ,1)
      dvdy00 = dvdy00 + deriv_Dvv(i,l  )*gv(j  ,i,2)
    end do
    kgen_div(l  ,j  ) = dudx00
    vvtemp(j  ,l  ) = dvdy00
  end do
end do

do j=1,np
  do i=1,np
    kgen_div(i,j)=(kgen_div(i,j)+vvtemp(i,j))*(elem_rmetdet(i,j)*rrearth)
  end do
end do
```

OpenCase: speed-up during 6-hour run



OpenCase:
Source code generated by case # 252,607,293

subroutine divergence_sphere()

```
100    do j=1,np
!DEC$ VECTOR ALWAYS ALIGNED
200    DO i=1,np,1
300        gv(i,j,1) = elem_metdet(i,j)*(elem_dinvc(i,j,1,1)*v(i,j,1) + elem_dinvc(i,j,1,2)*v(i,j,2))
       END DO
!DEC$ VECTOR ALWAYS ALIGNED
       DO i=1,np,1
400        gv(i,j,2) = elem_metdet(i,j)*(elem_dinvc(i,j,2,1)*v(i,j,1) + elem_dinvc(i,j,2,2)*v(i,j,2))
600    enddo

! compute d/dx and d/dy
!DEC$ VECTOR ALWAYS ALIGNED
!DEC$ UNROLL (np)
800     do l=1,np
700         do j=1,np
900             dudx00=0.0d0
1000            dvdy00=0.0d0
                i = 1
                dudx00 = dudx00 + deriv_dvv(i, l) * gv(i, j, 1)
                dvdy00 = dvdy00 + deriv_dvv(i, l) * gv(j, i, 2)
                i = i + 1
                dudx00 = dudx00 + deriv_dvv(i, l) * gv(i, j, 1)
                dvdy00 = dvdy00 + deriv_dvv(i, l) * gv(j, i, 2)
                i = i + 1

                ...
```

Outline

- How is NCAR's Yellowstone supercomputer being used ?
- Organization of Application Optimization Efforts
- Early Results
- Optimization Methodology
- Conclusions

Computational & Information Systems Laboratory

CISL

NCAR

NSF

Conclusions

- ***Application performance improvements achieved through optimization are likely to surpass in importance those obtained from mere architectural improvements.***
- ***We need robust, semi-optimized profiling tools (Extrae/ Paraver), unit test generators (KGEN), and source code generators (OpenCase) to produce optimized code fast enough to keep up with size and rate of change of climate code base.***
- ***This requires broad collaboration between vendors, computer scientists and domain scientists to achieve.***