Migrating CESM to many-core architectures

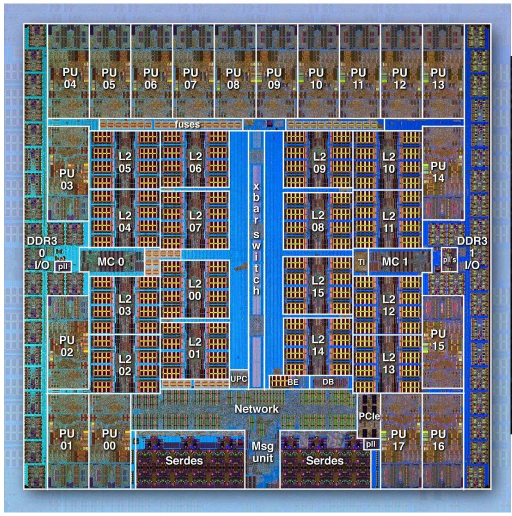
John Dennis (dennis@ucar.edu)
Srinath Vadlamani,Youngsung Kim (NCAR)
Harald Servat, Jesus Labarta, Judit Gimenez (BSC)

Application Scalability and Performance (ASAP) many core effort

- Enable NCAR applications to efficient utilize many-core architectures
- Personnel
  - Srinath Vadlamani (*)
  - Youngsung Kim (*)
  - Michael Arndt
  - Rich Loft
- Active collaboration for HOMME on Intel Phi
  - Mark Greenfield (Intel)
  - Mark Lubin (Intel)
  - Ruchira Sasanka (Intel)
  - Sergey Egorov (Intel)
  - Karthik Raman (Intel)
  - Ilene Carpenter (NREL)

(*) dedicated staff

The current many-core architectures planned for evaluation at NCAR



IBM BG/Q
Cores: 16 + 2
Multithread: 4-way
Coprocessor: no
Boot Linux: yes

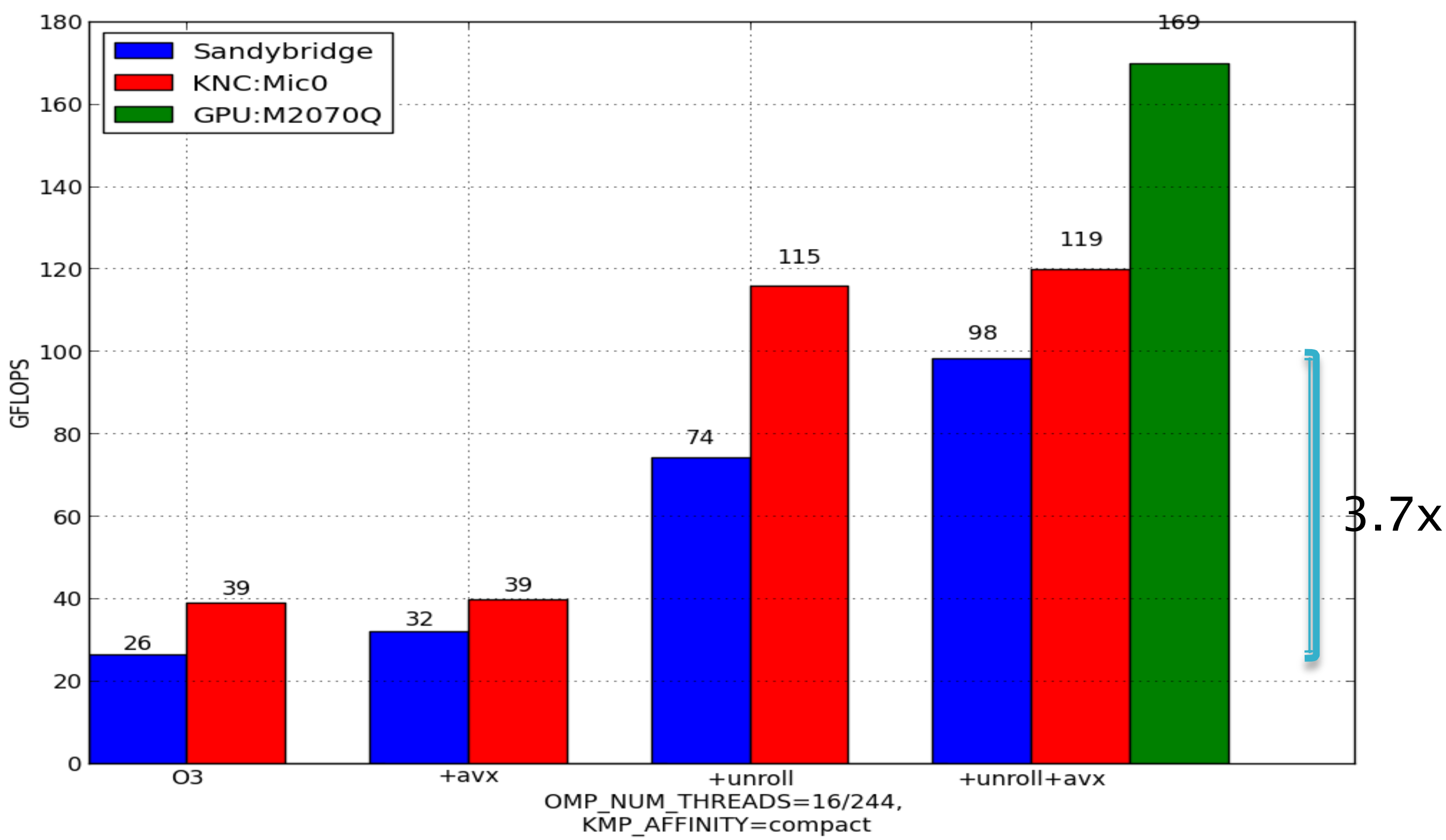
Intel Phi
Cores: 61
Multithread: 4-way
Coprocessor: yes
Boot Linux: yes

NVIDIA Fermi->Kepler
DP Cores: 512->832
Multithread: 32-way
Coprocessor: yes
Boot Linux: no

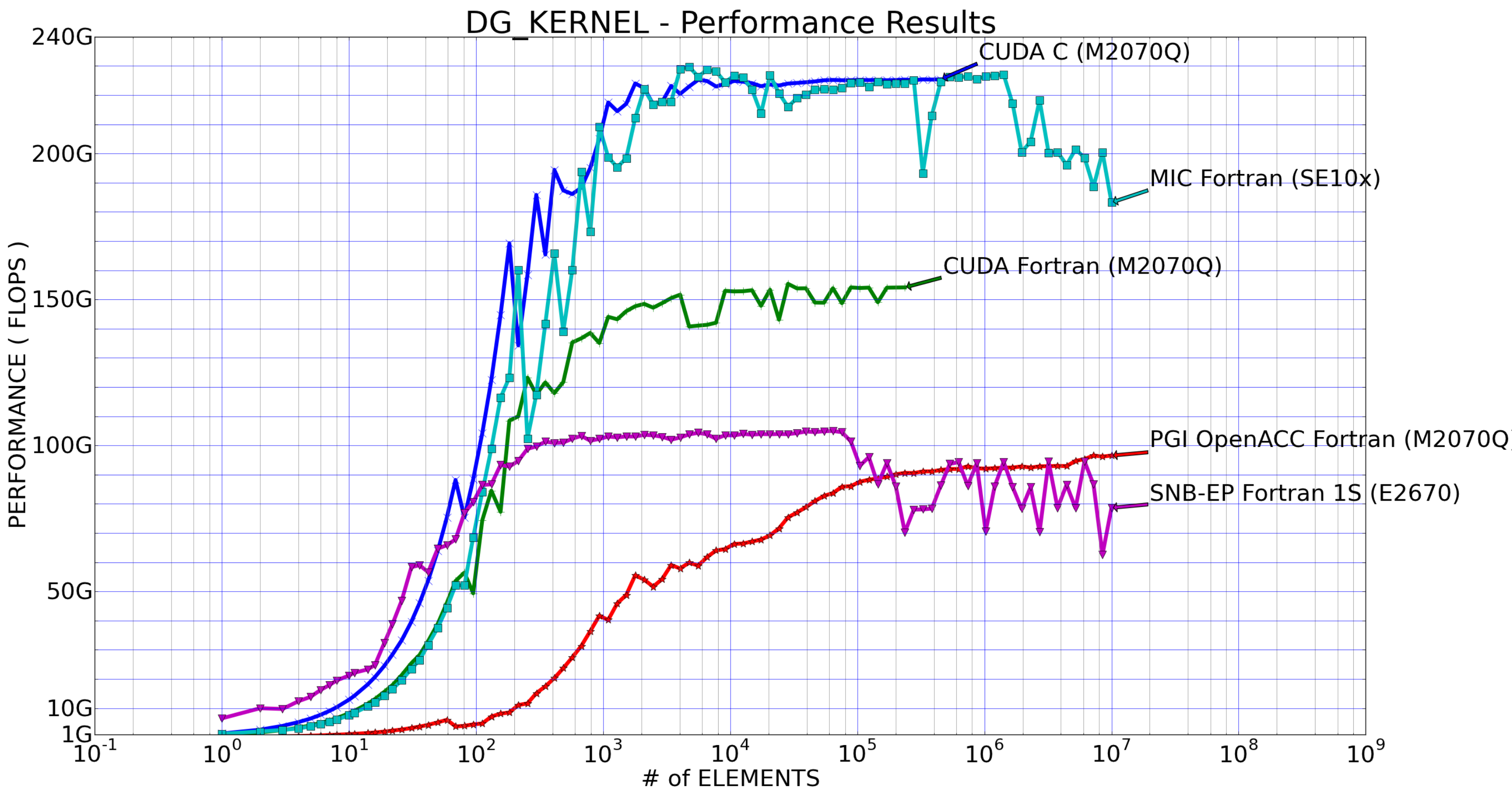
DG-kernel

- Discontinous Galerkin (DG) gradient kernel
  - Similar to derivative kernel in CAM-SE
- Small piece of code ~100 lines
- Written in a variety of languages
  - Fortran
  - CUDA Fortran
  - CUDA
  - OpenACC
- Performance and portability
  - Intel SandyBridge
  - Intel Phi
  - nVidia GPU 2070Q

DG-kernel: Intel SNB, Intel Phi, and Nvidia 2070Q



DG-kernel performance (single socket)



DG-kernel Observations

- Apples-to-apples comparisons are hard
- Our methodology
  - Socket-to-socket performance
  - Like generations of HW (as closely as possible)
  - Best (optimized) implementations
  - Multiple programming models
- 2070q initially 6.5x Intel SNB and 3.25x Intel Phi
- After optimization this drops to 2.1x and parity
- Optimizations for Xeon Phi help SNB and vice versa
- Optimized performance much closer than expected
- OpenACC performance lags due to use of shared memory
- Challenging to get good Phi performance

Accelerator Strategy

- Significant potential to improve many-core performance
- Improvement Cycle
  - Identify poorly performing code
    - i.e. poor vectorization
  - Restructure code
    - vectorize
    - Benefits both traditional and accelerator
  - Repeat

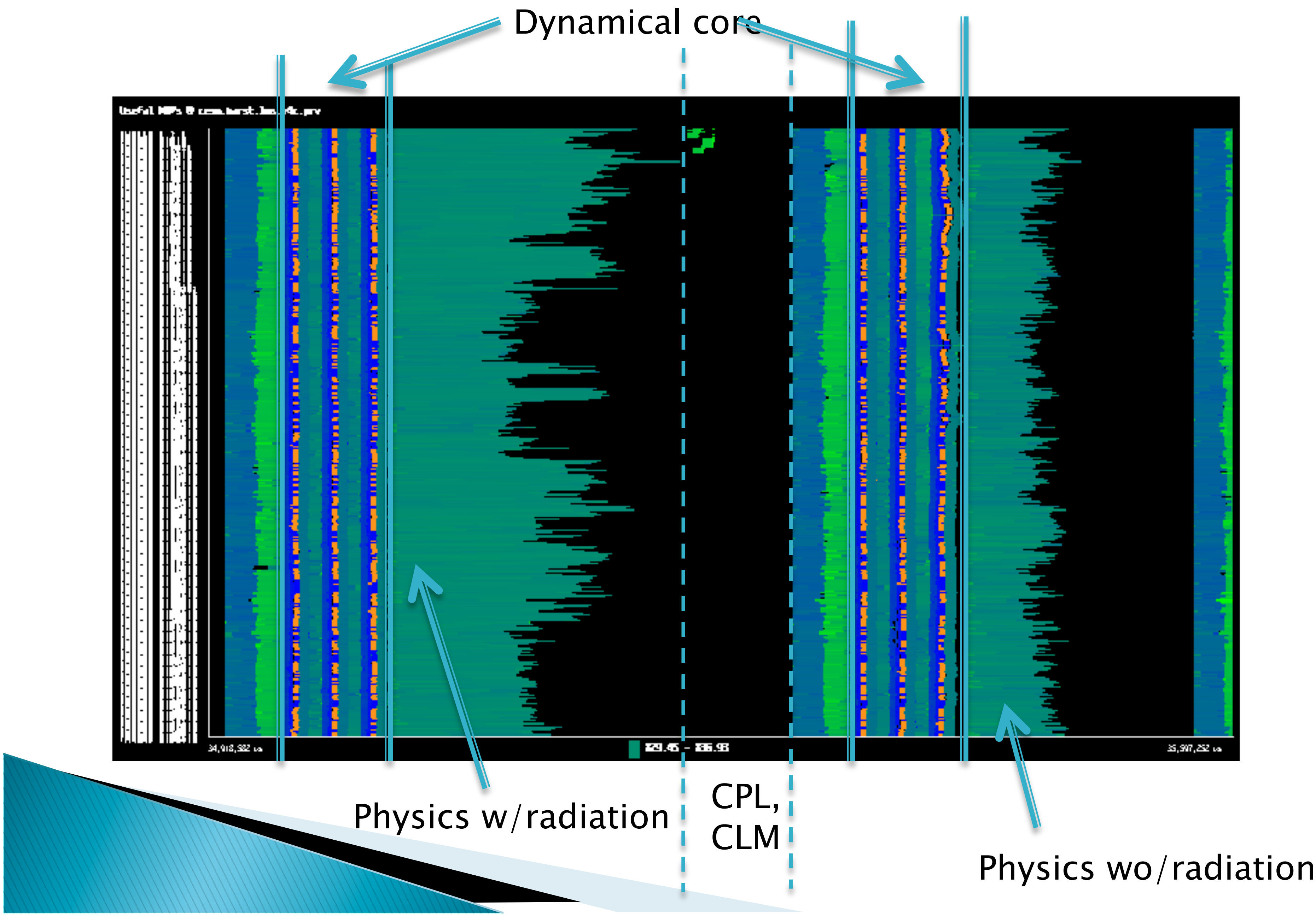
How to Identify poorly performing code?

- Automatic performance identification
  - Barcelona Supercomputer Center (BSC)
  - Polytechnic University of Catalonia (UPC)
  - H. Servat, J. Labarta, J. Gimenez
- Utilize BSC tools
  - extrae: trace collection
  - paraver: visualization client
  - clustering & folding tools

Extrae tracing (BSC)

- Enables very detailed tracing of application characteristics
- Creates a "performance database"
  - time in user code
  - time in MPI
  - time in OpenMP
  - hardware counters
  - etc…
- Browse performance database with Paraver
  - Timeline visual analysis
  - Statistical analysis

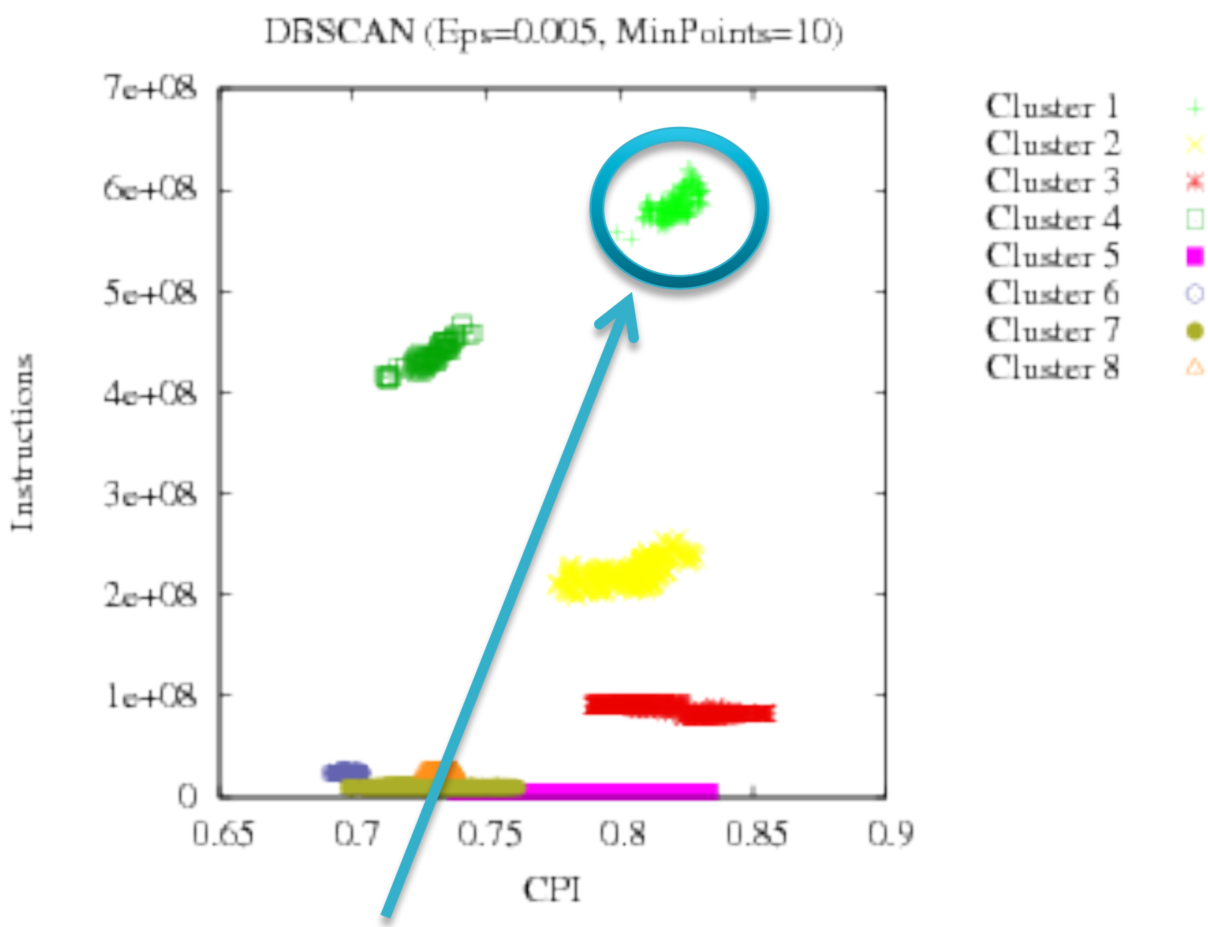
CAM5-SE: 2 degree (ne=16) on 384 cores



Extrae tracing (con't)

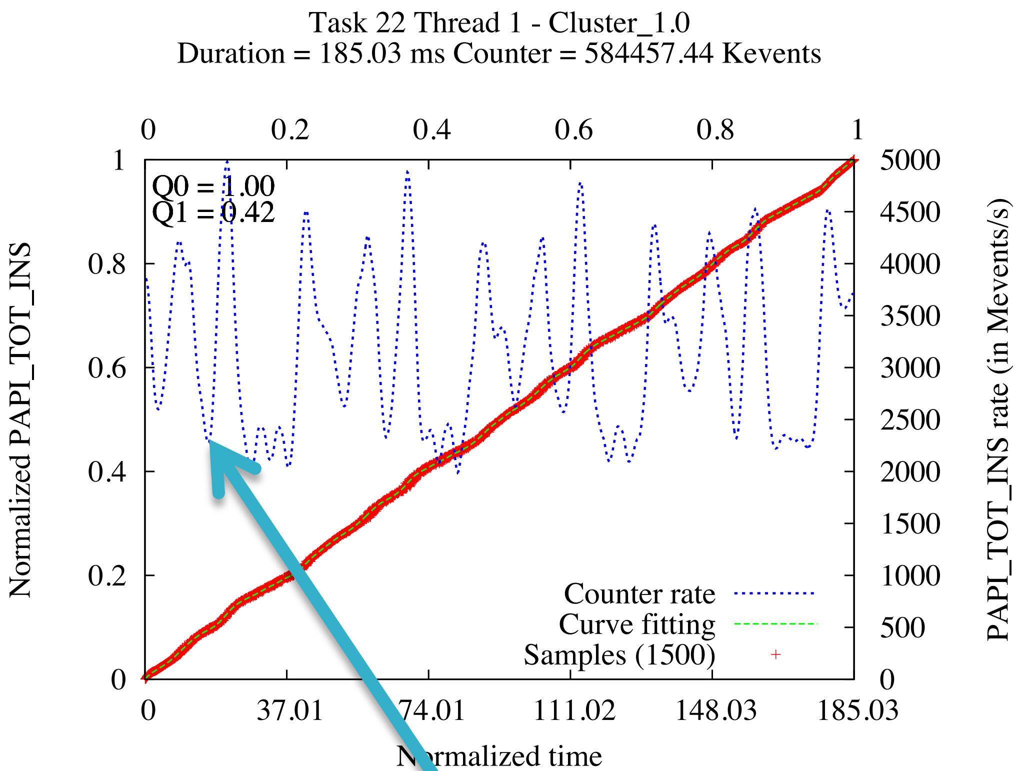
- Traces of non-trivial codes can become large
- Need method to reduce data to simplify analysis
- Automatic performance identification
- Sampled CESM at periodic intervals
- Identified repeating computational bursts (clusters)
- Create synthetic traces to simplify analysis
- Look for inefficient sections of code

Computational burst clusters



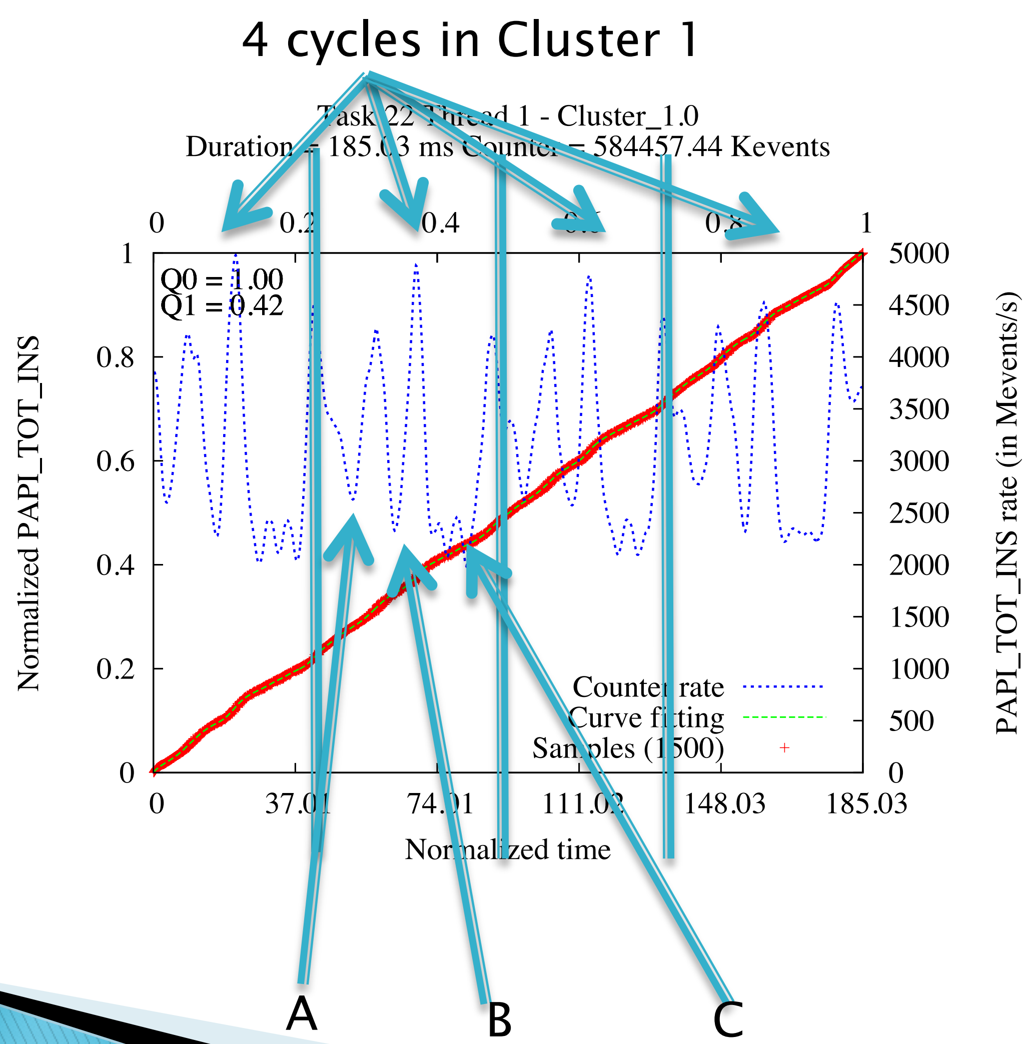
Most expensive computational cluster

Total Instructions: Cluster 1



Notice drops in Instruction rates

Total Instructions: Cluster 1



Underperforming subroutines Cluster 1

- Group A:
  - conden: 2.7%
  - compute_uwshcu: 3.3%
  - rtrnmc: 1.75%
- Group B:
  - micro_mg_tend: 1.36% (1.73%)
  - wetdepa_v2: 2.5% ← Focus effort on one subroutine
- Group C:
  - reftra_sw: 1.71%
  - spcvmc_sw: 1.21%
  - vrtqdr_sw 1.43%

Optimizing (vectorizing) wetdepa_v2

- Consists of a double nested loop
  - Very long ~400 lines
  - Unnecessary branches with inhibit vectorization
- Restructuring wetdepa_v2
  - Break up long loop to simplify vectorization
  - Promote scalar to vector temporaries
  - Common expression elimination

wetdepa_v2 (driver)

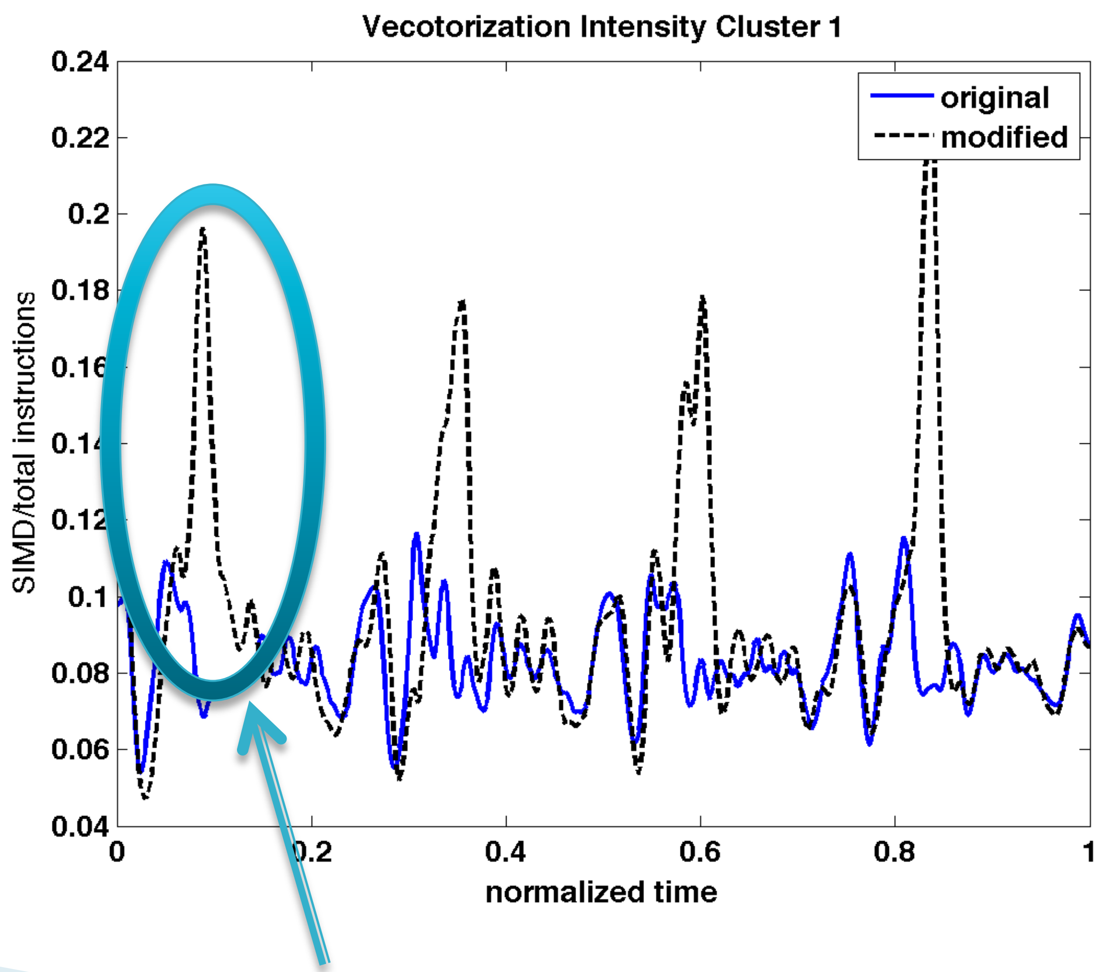
	Intel Phi (Intel 13.1.1)			Intel Sandybridge (Intel 13.1.2)		
	-O2	-O3	-O3 -fast	-O2	-O3	-O3 -fast
orig	42.85	41.24	3.74	3.43	3.32	0.97
mod	6.50	6.61	[illegible].58	1.09	1.12	1.04

9.3 x

3.5x

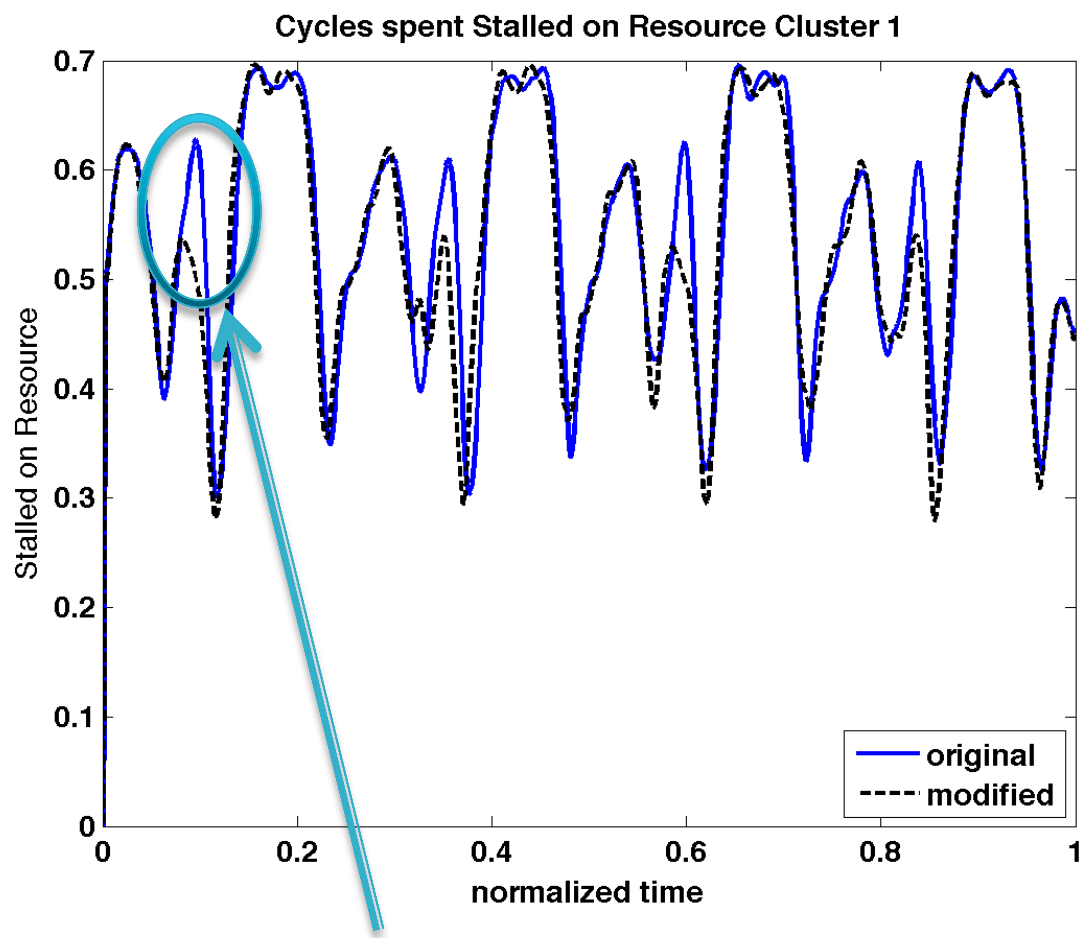
Significant potential for reducing execution time !

Vectorization Intensity Cluster #1



Increase in code vectorization

Stalls on Resources Cluster #1



Reduction in cycles stalled on resources

wetdepa_v2 (CESM)

- CESM B-case, NE=16, 570 cores
- Yellowstone, Intel (13.1.1) -O2
- Original version:
  - 2.5% total time
  - 492.6 ms
- Modified version:
  - 0.73% total time
  - 121.1 ms
- Actual improvement: 4.07x

Observations about wetdepa_v2

- Simple loop was vectorized using aggressive optimization (–O3 –fast)
- Correctness issues are problematic at high optimization levels
- Effort to extract wetdepa_v2 much larger then actual time to optimize
- Code restructuring will be necessary in general

Accelerator strategy (medical version)

- Identify “healthy” patient [DG-kernel]
  - Perform a panel of medical tests [PAPI + extrae]
- Perform panel of medical tests on large application (CESM/WRF/MPAS/DART)
  - Look at tests for sections of full application differs from “healthy” patient
  - Diagnose performance problems based on groups of “symptom”
  - Address identified performance problems
- Generic approach, suitable for all platforms
  - Intel SNB, Intel Phi, AMD Interlagos, nVidia Kepler, IBM A2
- Exact nature of tests may differ

How can you help speed up CESM?

- Write code that vectorizes
  - Don’t do this:

```
do i=1,pcols
   call sub1()
   call sub2()
   call sub3()
   …
```

Will never vectorize

  - Instead

```
do i=1,pcols
   srcc(i) = srcs1(i) + srcs2(i)  ! convective tend by both processes
   finc(i) = srcs1(i)/(srcc(i) + eps) ! fraction in-cloud
   srcs1(i) = 0._r8
   odds(i) = precabs(i)/max(cldvst(i,k),1.e-5_r8)*scavcoef(i,k)*deltat
               odds(i) = max(min(1._r8,odds(i)),0._r8)
   …
```

How can you help speed up CESM? (con't)

- Create/use drivers or unit tests for all new code
  - Simplifies development and debugging
  - Simple performance testing and restructuring of code
- Unit tests for parameterization
  - Community Ocean Vertical Mixing (CVMix) Project
  - CLUBB, UNICORN?

Conclusions

- Dedicated group within CISL to address many-core challenges
- Significant performance improvement possible for all architectures
- Equivalent performance for Intel Phi and nVidia 2070Q on DG-Kernel
- Possible to identify poorly performing code for CESM
- Possible to significantly increase performance through vectorization: 4 – 9x
- Strategy for continuous improvement of CESM performance