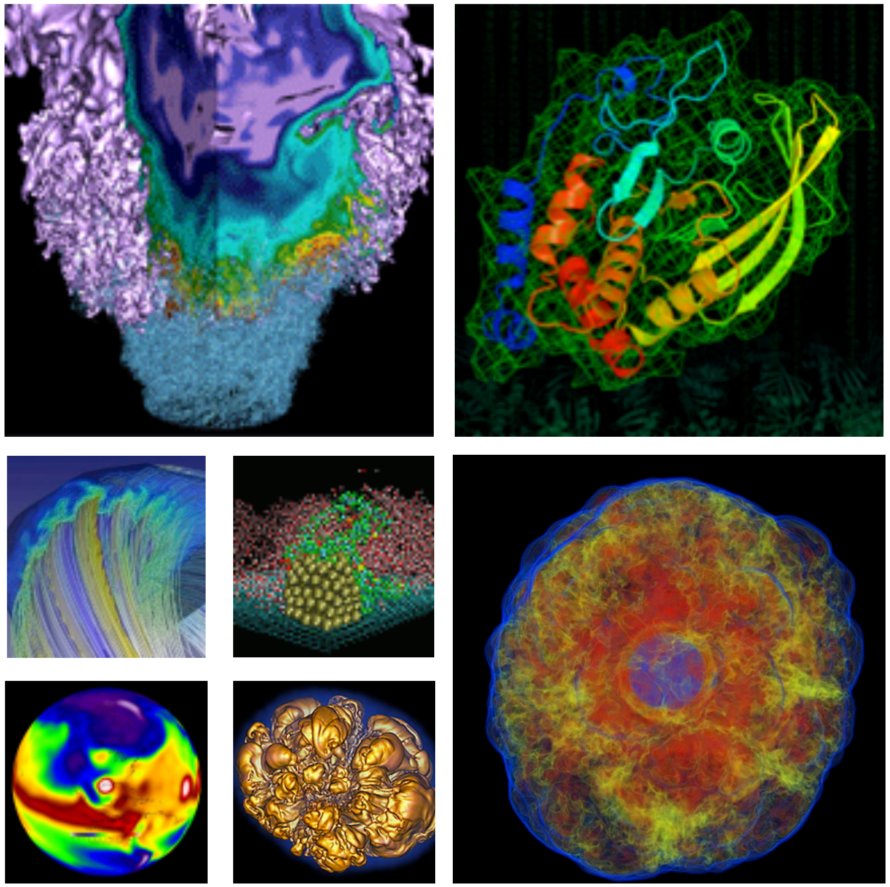
Advanced OpenMP and CESM Case Study



Helen He, NERSC

NERSC User Group Meeting
March 23, 2016

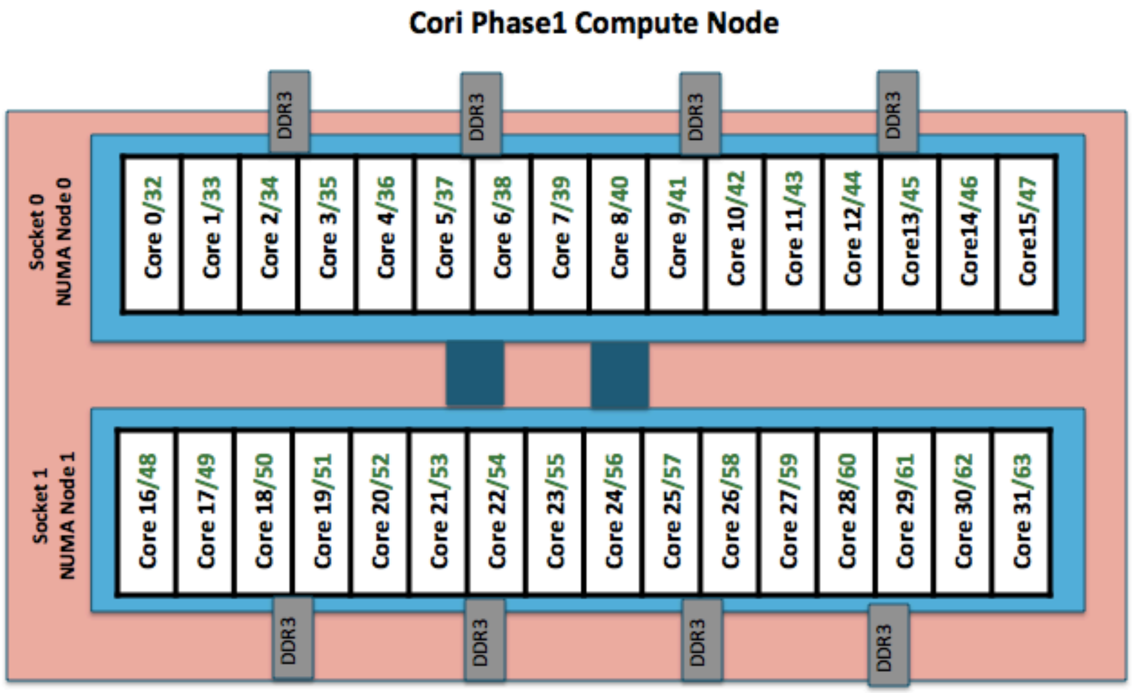
Outline

- Background
- What's New in OpenMP 4.0 and 4.5
- Nested OpenMP
- CESM MG2 Kernel Case Study

Hybrid MPI/OpenMP: Big Picture

- **Next NERSC system Cori is an Intel Xeon Phi KNL many-core system architecture.**
- **Application is very likely to run on KNL with simple porting, but high performance is harder to achieve.**
- **Many applications will not fit into the memory of a KNL node using pure MPI across all HW cores and threads because of the memory overhead for each MPI task.**
- **Applications need to explore more on-node parallelism with thread scaling and vectorization, also to utilize HBM and burst buffer options.**
- **Hybrid MPI/OpenMP is a recommended programming model for Cori. It is also a portable programing model (recommended over OpenACC) for running across various large DOE systems, whether many-core system architecture, or hybrid CPU/GPU system.**
- **Optimization on current NERSC systems will help to prepare for Cori Phase 2 KNL.**

Cori Phase 1 Compute Nodes



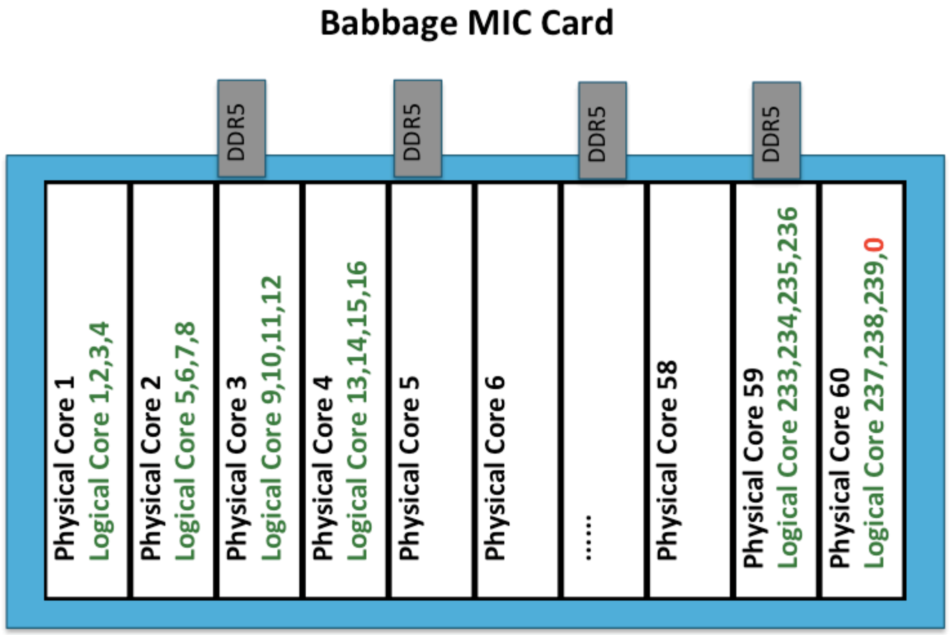
To obtain processor info:

Get on a compute node:
```
% salloc –N 1
```

Then:
```
% cat /proc/cpuinfo
```
or
```
% hwloc-ls
```

- **Cori Phase 1: NERSC Cray XC40, 1,630 nodes, 52,160 cores.**
  - **Each node has 2 Intel Xeon 16-core Haswell processors.**
  - **2 NUMA domains per node, 16 cores per NUMA domain. 2 hardware threads per core.**
- **Memory bandwidth is non-homogeneous among NUMA domains.**

Babbage MIC Card

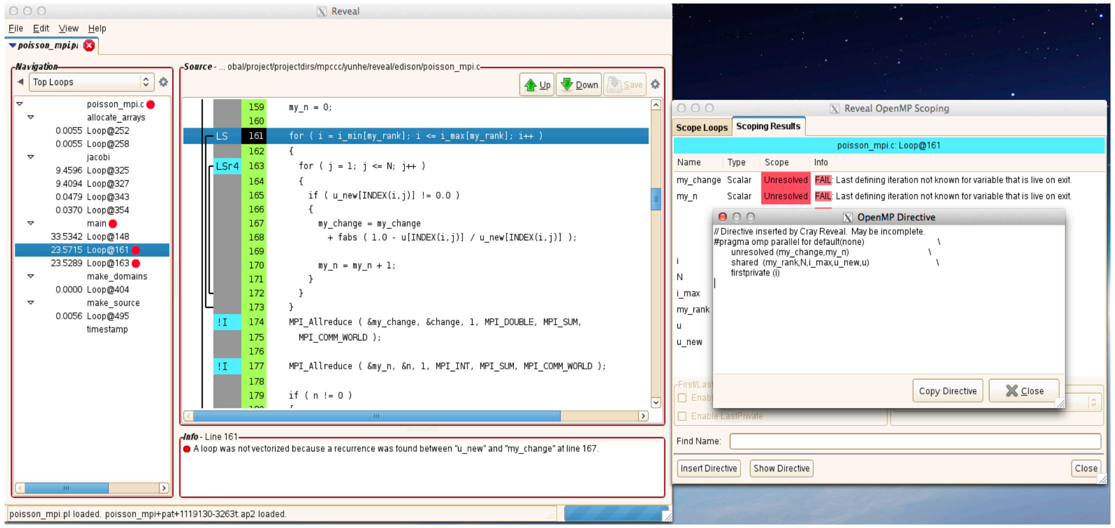


Babbage: NERSC Intel Xeon Phi testbed, 45 nodes. 2 MIC cards per node. Recommend to use the "native" mode.

- 1 NUMA domain per MIC card: 60 physical cores, 240 logical cores. OpenMP threading potential to 240-way. Recommend to use at least 2 threads per core to hide latency of in-order execution.
- KMP_AFFINITY, KMP_PLACE_THREADS, OMP_PLACES, OMP_PROC_BIND for thread affinity control
- I_MPI_PIN_DOMAIN for MPI/OpenMP process and thread affinity control.

Adding OpenMP to Your Program

- **On Cori/Edison, under Cray programming environment, Cray Reveal tool helps to perform scope analysis, and suggests OpenMP compiler directives to a pure MPI or serial code.**
  - Based on CrayPat performance analysis
  - Utilizes Cray compiler source code analysis and optimization information



	poisson_mpi.c
	allocate_arrays
0.0055	Loop@252
0.0055	Loop@258
	jacobi
9.4596	Loop@325
9.4094	Loop@327
0.0479	Loop@343
0.0370	Loop@354
	main
33.5342	Loop@148
23.5715	Loop@161
23.5289	Loop@163
	make_domains
0.0000	Loop@404
	make_source
0.0056	Loop@495
	timestamp

```
    my_n = 0;

    for ( i = i_min[my_rank]; i <= i_max[my_rank]; i++ )
    {
      for ( j = 1; j <= N; j++ )
      {
        if ( u_new[INDEX(i,j)] != 0.0 )
        {
          my_change = my_change
            + fabs ( 1.0 - u[INDEX(i,j)] / u_new[INDEX(i,j)] );

          my_n = my_n + 1;
        }
      }
    }
    MPI_Allreduce ( &my_change, &change, 1, MPI_DOUBLE, MPI_SUM,
      MPI_COMM_WORLD );

    MPI_Allreduce ( &my_n, &n, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD );

    if ( n != 0 )
```

Info - Line 161

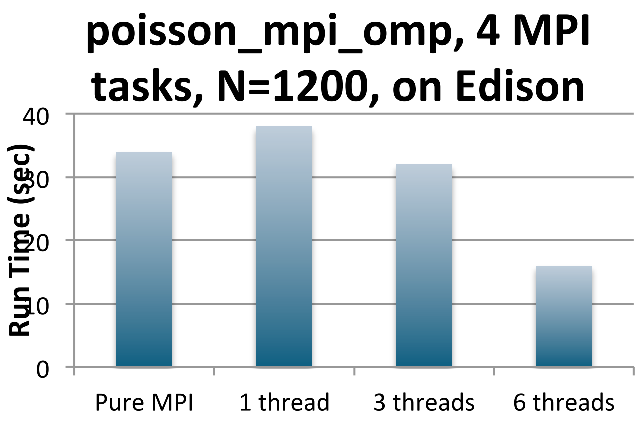
A loop was not vectorized because a recurrence was found between "u_new" and "my_change" at line 167.

poisson_mpi.pl loaded. poisson_mpi+pat+1119130-3263t.ap2 loaded.

poisson_mpi.c: Loop@161

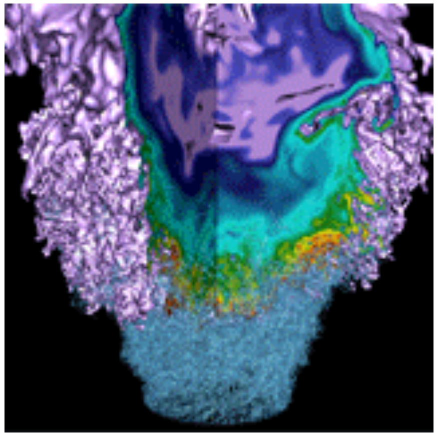
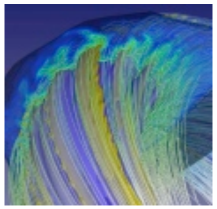
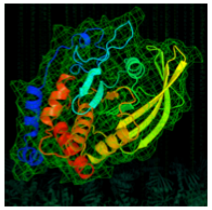
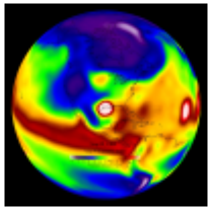
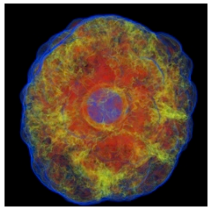
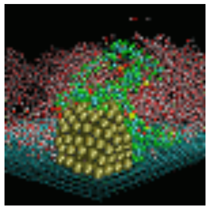
Name	Type	Scope	Info
my_change	Scalar	Unresolved	FAIL: Last defining iteration not known for variable that is live on exit.
my_n	Scalar	Unresolved	FAIL: Last defining iteration not known for variable that is live on exit.
i			
N			
i_max			
my_rank			
u			
u_new			

```
// Directive inserted by Cray Reveal.  May be incomplete.
#pragma omp parallel for default(none)                    \
        unresolved (my_change,my_n)                      \
        shared  (my_rank,N,i_max,u_new,u)                 \
        firstprivate (i)
```



- **On Babbage, Intel Advisor tool helps to guide threading design options.**

New in OpenMP 4.0 and 4.5

New Features in OpenMP 4.0

- OpenMP 4.0 was released in July 2013
- Device constructs for accelerators
- SIMD constructs for vectorization
- Task groups and dependencies
- Thread affinity control
- User defined reductions
- Cancellation construct
- Initial support for Fortran 2003
- OMP_DISPLAY_ENV for all internal variables

device Constructs for Accelerators

C/C++:

```
#pragma omp target map(to:B,C), map (tofrom: sum)
#pragma omp parallel for reduction(+,sum)
for (int i=0; i<N; i++) {
    sum += B[i] + C[i];
}
```

- **Use target directive to offload a region to device. Host and device share memory via mapping: to, from, tofrom.**

C/C++:

```
#pragma omp target teams distribute parallel for \
map (to:B,C), map (tofrom:sum) reduction(+:sum)
for (int i=0; i<N; i++) {
    sum += B[i] + C[i];
}
```

- **Use teams clause to create multiple master threads that can execute in parallel on multiple processors on device.**
- **Use “distribute” clause to spread iterations of a parallel loop across teams.**

"OMP target device" Works on Babbage

```
program test
use omp_lib
write(*,*) 'cpu max threads:',omp_get_max_threads()
!$omp target device(0)
write(*,*) 'mic max threads:',omp_get_max_threads()
!$omp parallel
!$omp master
write(*,*) 'mic nbr threads:',omp_get_num_threads()
!$omp end master
!$omp end parallel
!$omp end target

!$omp target device(0)
!$omp teams num_teams(1)
write(*,*) 'team', omp_get_team_num(), ' mic max
threads:',omp_get_max_threads()
!$omp parallel
!$omp master
write(*,*) 'team',omp_get_team_num(),' mic nbr
threads:',omp_get_num_threads()
!$omp end master
!$omp end parallel
!$omp end teams
!$omp end target
end program test
```

```
export KMP_AFFINITY=balanced
export OMP_NUM_THREADS=1
export MIC_ENV_PREFIX=MIC
export MIC_OMP_NUM_THREADS=60
```

```
% cat myjob.host.2680.out
cpu max threads: 1
mic max threads: 60
mic nbr threads: 60
team 0 mic max threads: 60
team 0 mic nbr threads: 236
```

Not recommended for preparing for Cori KNL, but it is good to know that it works and it is portable ☺

Code should default to run on Cori (host), but fails due to "device not found". Compiler bug filed.

Asynchronous Offloading with Tasking

Useful for people who need to write portable codes across DOE centers.

```
#pragma omp parallel
#pragma omp single
{
#pragma omp task
   {
#pragma omp target map(to:input[:N]) map(from:result[:N])
#pragma omp parallel for
      for (i=0; i<N; i++) {
         result[i] = some_computation(input[i], i);
      }

   }
#pragma omp task
   {
      do_something_important_on_host();
   }
}    // implicit taskwait at barrier
```

OpenMP Vectorization Support

- More architectures support longer vector length
- Vectorization: execute a single instruction on multiple data objects in parallel within a single CPU core
- Auto-vectorization can be hard for compilers (dependencies)
- Many compilers support SIMD directives to aid vectorization of loops
- OpenMP 4.0 provides a standardization for SIMD

OpenMP4 SIMD

- **Parallelize and Vectorize:**
  - Fortran: !$OMP do simd *[clauses]*
  - The loop is first divided across a thread team, then subdivide loop chunks to fit in a SIMD vector register.
- **SIMD Functions:**

C/C++:
```
#pragma omp declare simd
float min (float a, float b) {
    return a<b ? a:b;
}
```

  - Compilers may not be able to vectorize and inline function calls easily.
  - Compilers #pramga declare simd tells compiler to generate SIMD function
  - Useful to use “declare simd” for elemental functions that are called from within a loop, so compilers can vectorize the function.
- **Using OpenMP4 SIMD bypasses the compiler analysis**
  - Incorrect results and poor performance possible!

Clauses for simd Directive

- safelen(length): defines the max number of iterations can run concurrently without breaking dependence.
- linear: lists variables with a linear relationship to the iteration number.
- aligned: specifies byte alignment of the list items
- all regular clauses ….

linear(ref) Clause is Important

- In C, compiler places consecutive argument values in a vector register
- But Fortran passes arguments by reference
  - By default compiler places consecutive addresses in a vector register. Leads to a gather of the 4 addresses (slow)
  - LINEAR(REF(X)) tells the compiler that the addresses are consecutive; only need to dereference once and copy consecutive values to vector register
  - New in compiler version 16.0.1
- Same method could be used for C arguments passed by reference

```
subroutine test_linear(x, y)
!$omp declare simd
(test_linear) linear(ref(x, y))
   real(8),intent(in)  :: x
   real(8),intent(out) :: y
   y = 1. + sin(x)**3
end subroutine test_linear
…
Interface
…
do j = 1,n
    call test_linear(a(j), b(j))
enddo
```

Approximate speed-up for double precision array of 1M elements	
No DECLARE SIMD	1.0
DECLARE SIMD but no LINEAR(REF)	0.9
DECLARE SIMD with LINEAR(REF) clause	3.6

The results above were obtained on an Intel® Xeon® E7-4850 v3 system, frequency 2.2 GHz, running Red Hat* Enterprise Linux* version 7.1 and using the Intel® Fortran Compiler version 16.0.1.

taskgroup Directive

- **OpenMP 4.0 extends the tasking support.**
- **The taskgroup directive waits for all descendant tasks to complete as compared to taskwait which only waits for direct children.**

Task Dependencies

```
#pragma omp task depend (out:a)
{ ….}
#pragma omp task depend (out:b)
{…}
#pragma omp task depend (in:a,b)
{…}
```

- **The first two tasks can execute in parallel**
- **The third task can only start after both of the first two are complete.**

Better Thread Affinity Control

- **OpenMP 3.1 only has OMP_PROC_BIND, either TRUE or FALSE.**
- **OpenMP 4.0 still allows the above. Can now provide a list.**
  - spread: Bind threads as evenly distributed (spreaded) as possible
  - close: Bind threads close to the master thread
  - master: Bind threads the same place as the master thread
- **Added OMP_PLACES environment variable: a list of places that threads can be pinned on**
  - threads: Each place corresponds to a single hardware thread on the target machine.
  - cores: Each place corresponds to a single core (having one or more hardware threads) on the target machine.
  - sockets: Each place corresponds to a single socket (consisting of one or more cores) on the target machine.
  - A list with explicit place values, such as:
    - "{0,1,2,3},{4,5,6,7},{8,9,10,11},{12,13,14,15}”
    - “{0:4},{4:4},{8:4},{12:4}”
- **Examples:**
  - export OMP_PLACES=threads
  - export OMP_PROC_BIND=“spread, close” (for nested levels)

User Defined Reductions

```
#pragma omp declare reduction (merge: std::vector<int>
 : omp_out.insert(omp_out.end(), omp_in.begin(), omp_in.end()))
```

- **OpenMP 3.1 can not do reductions on objects or structures.**
- **OpenMP 4.0 can now define own reduction operations with declare reduction directive.**
- **“merge” is now a reduction operator.**

Construct Cancellation

FORTRAN:

```
!$OMP PARALLEL DO PRIVATE (sample)
   do i = 1, n
       sample = testing(i,…)
!$OMP CANCEL PARALLEL IF (sample)
   enddo
!$OMP END PARALLEL DO
```

- **cancel / cancellation point is a clean way of early termination of an OpenMP construct.**
- **First thread exits when TRUE. Other threads exit when reaching the cancel directive.**

OMP_DISPLAY_ENV

- export OMP_DISPLAY_ENV=true
- Displays the OpenMP version number
- Displays the value of ICVs associated with ENV
- Useful for users to find out default settings

```
OPENMP DISPLAY ENVIRONMENT BEGIN
  _OPENMP='201307'
  [host] OMP_CANCELLATION='FALSE'
  [host] OMP_DISPLAY_ENV='TRUE'
  [host] OMP_DYNAMIC='FALSE'
  [host] OMP_MAX_ACTIVE_LEVELS='2147483647'
  [host] OMP_NESTED='FALSE'
  [host] OMP_NUM_THREADS='8'
  [host] OMP_PLACES: value is not defined
  [host] OMP_PROC_BIND='false'
  [host] OMP_SCHEDULE='static'
  [host] OMP_STACKSIZE='4M'
  [host] OMP_THREAD_LIMIT='2147483647'
  [host] OMP_WAIT_POLICY='PASSIVE'
OPENMP DISPLAY ENVIRONMENT END
```

New Features in OpenMP 4.5

- OpenMP 4.5 was released in November 2015
- Significantly improved support for devices
- Support for *doacross* loops
- New *taskloop* construct
- Reductions for C/C++ arrays
- New hint mechanisms
- Thread affinity support
- Improved support for Fortran 2003
- SIMD extensions
- New *linear* clause for loop construct
- Support for *if* clause on combined/composite constructs
- Addition of *schedule* modifiers

OpenMP 4.5 Focused on Device Support

- **OpenMP now provides:**
  - Unstructured data mapping,
  - Asynchronous execution
  - Runtime routines for device memory management: allocate, copy, and free.
- **More similar features/capabilities as in OpenACC**
  - Scalar variables are firstprivate by default
  - Improvements for C/C++ array sections
  - Clauses to support device pointers
  - Ability to map structure elements
  - New combined constructs
  - New way to map global variables: omp declare target

doacross Loops

- A natural mechanism to parallelize loops with well-structured dependences is provided.
- The source and sink dependence types were added to the depend clause to support doacross loop nests.

taskloop Constructs

- **Support to divide loops into tasks, avoiding the requirement that all threads execute the loop.**

- **Parallelize a loop using OpenMP tasks**
  - Cut loop into chunks
  - Create a task for each loop chunk

- **Syntax (C/C++)**

```
#pragma omp taskloop [simd] [clause[[,] clause],…]
for-loops
```

- **Syntax (Fortran)**

```
!$omp taskloop[simd] [clause[[,] clause],…]
do-loops
[!$omp end taskloop [simd]]
```

Reductions for C/C++ Arrays

- Semantics for reductions on C/C++ array sections were added and restrictions on the use of arrays and pointers in reductions were removed.

New Hint Mechanisms

- Hint mechanisms can provide guidance on the relative priority of tasks and on preferred synchronization implementations.
- The priority clause was added to the task construct to support hints that specify the relative execution priority of explicit tasks.
- The hint clause for omp lock was added to the critical construct

Thread Affinity Support

- It is now possible to use runtime functions to determine the effect of thread affinity clauses.
- Query functions for OpenMP thread affinity were added
  - omp_get_num_places
  - omp_get_place_num_procs
  - omp_get_place_proc_ids
  - omp_get_place_num
  - omp_get_partition_num_places
  - omp_get_partition_place_nums

SIMD Extensions

- The simdlen clause was added to the simd construct to support specification of the exact number of iterations desired per SIMD chunk.
- These extensions include the ability to specify exact SIMD width and additional data-sharing attributes.

New linear Clause for Loop Construct

- **Syntax (C/C++)**

```
#pragma omp for [clause[[,] clause],…]
[linear(list[ : linear-step])
for-loops
```

- **Syntax (Fortran)**

```
!$omp do [clause[[,] clause],…] [linear(list[ :
linear-step])
do-loops
[!$omp end do [nowait]]
```

- **Other usual clauses include: private, firstprivate, lastprivate, reduction, schedule, collapse, ordered, nowait (*C/C++ only*).**

schedule Modifiers

- **Schedule clause can be: static, dynamic, guided, auto, runtime.**
- **New schedule modifiers are added:**
  - monotonic: each thread executes its assigned chunks in increasing logical iteration order.
  - nonmonotonic: chunks are assigned to threads in any order
  - Simd: when a loop is associated with SIMD, the new chunk size becomes *[chunk_size/simd_width] * simd_width.*

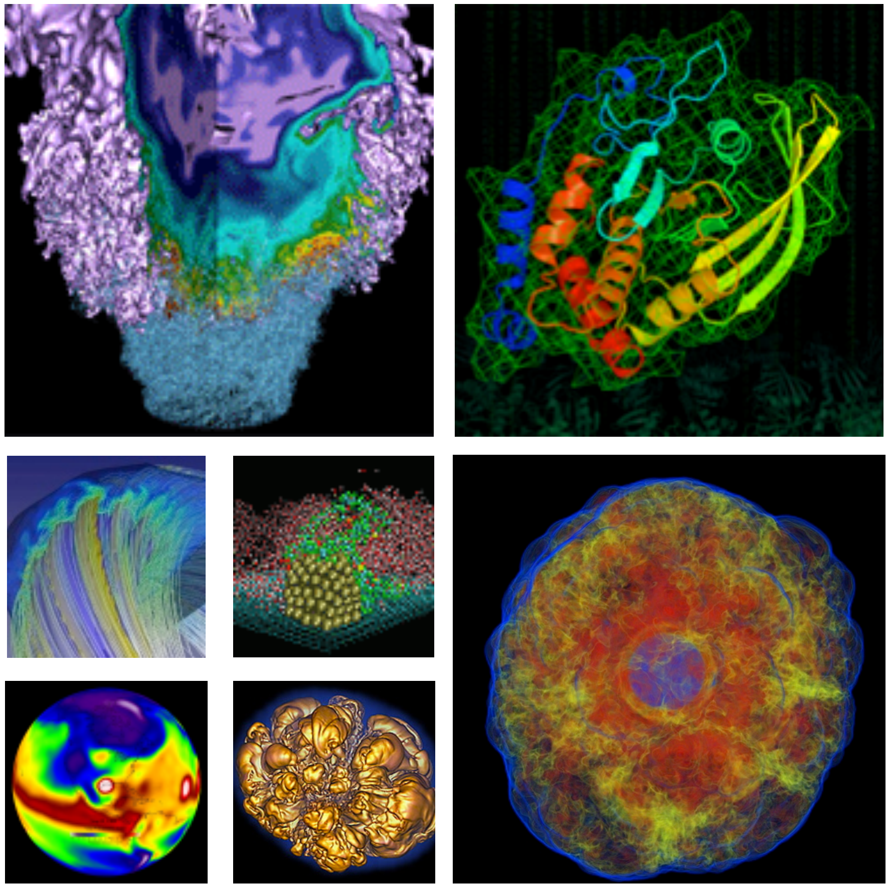
OpenMP 4.0/4.5 Support in Compilers

- **GNU compiler**
  - From gcc/4.9.0 for C/C++; OpenMP 4.0
  - From gcc/4.9.1 for Fortran: OpenMP 4.0
  - From gcc/6.0: most OpenMP 4.5 features
  - From gcc/6.1: full OpenMP 4.5 for C/C++ (not Fortran)
- **Intel compiler**
  - From intel/15.0: most OpenMP 4.0 features
  - From Intel/16.0: full OpenMP 4.0
  - From intel/16.0 Update 2: some OpenMP4.5 SIMD features
- **Cray compiler**
  - From cce/8.4.0: full OpenMP 4.0

Major OpenMP 5.0 Topics

- Support for event loops: Major tasking advances?
- Memory locality, affinity and working with complex memory hierarchies
- Performance and debugging tools support
- Updates to support latest C/C++ standards, completion of Fortran 2003
- Continued improvements to device support and tasking
- Interoperability and composability
- Many other potential smaller topics

Nested OpenMP



NERSC 40 YEARS at the FOREFRONT 1974-2014

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB Lawrence Berkeley National Laboratory

Sample Nested OpenMP Program

```
#include <omp.h>
#include <stdio.h>
void report_num_threads(int level)
{
   #pragma omp single {
       printf("Level %d: number of threads in the
team: %d\n", level, omp_get_num_threads());
      }
}
int main()
{
   omp_set_dynamic(0);
   #pragma omp parallel num_threads(2) {
      report_num_threads(1);
      #pragma omp parallel num_threads(2) {
         report_num_threads(2);
         #pragma omp parallel num_threads(2) {
            report_num_threads(3);
         }
      }
   }
   return(0);
}
```

```
% a.out
Level 1: number of threads in the team: 2
Level 2: number of threads in the team: 1
Level 3: number of threads in the team: 1
Level 2: number of threads in the team: 1
Level 3: number of threads in the team: 1
```

```
% setenv OMP_NESTED TRUE
% a.out
Level 1: number of threads in the team: 2
Level 2: number of threads in the team: 2
Level 2: number of threads in the team: 2
Level 3: number of threads in the team: 2
Level 3: number of threads in the team: 2
Level 3: number of threads in the team: 2
Level 3: number of threads in the team: 2
```

Level 0: P0
Level 1: P0 P1
Level 2: P0 P2; P1 P3
Level 3: P0 P4; P2 P5; P1 P6; P3 P7

When to Use Nested OpenMP

- **Beneficial to use nested OpenMP to allow more fine-grained thread parallelism.**
- **Some application teams are exploring with nested OpenMP to allow more fine-grained thread parallelism.**
  - Hybrid MPI/OpenMP not using node fully packed
  - Top level OpenMP loop does not use all available threads
  - Multiple levels of OpenMP loops are not easily collapsed
  - Certain computational intensive kernels could use more threads
  - MKL can use extra cores with nested OpenMP

Process and Thread Affinity in Nested OpenMP

- Achieving best process and thread affinity is crucial in getting good performance with nested OpenMP, yet it is not straightforward to do so.
- A combination of OpenMP environment variables and run time flags are needed for different compilers and different batch schedulers on different systems.

Example: Use Intel compiler with SLURM on Edison:

```
setenv OMP_NESTED true
setenv  OMP_NUM_THREADS 4,3
setenv OMP_PROC_BIND spread,close
srun -n 2 -c 12 ./nested.intel.edison
```

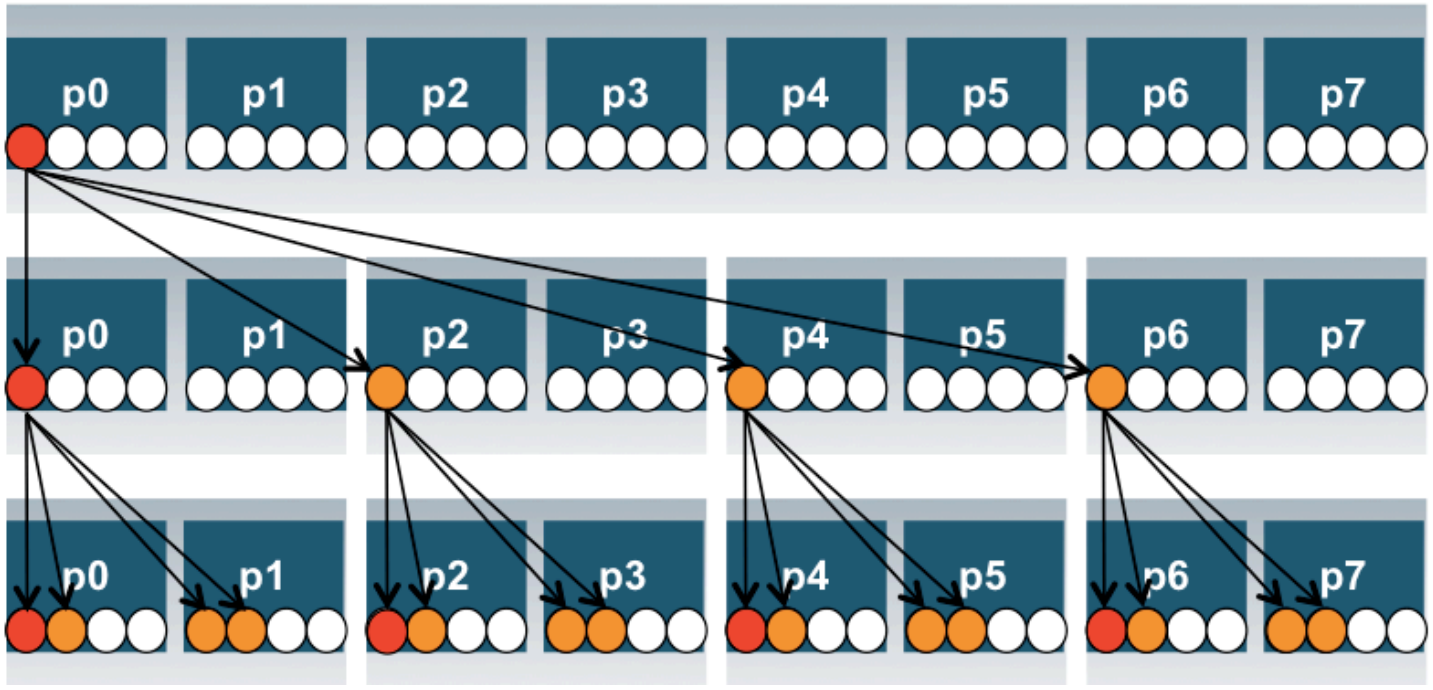
- Use num_threads clause in source codes to set threads for nested regions. For most other non-nested regions, use OMP_NUM_THREADS env for simplicity and flexibility.

Nested OpenMP Thread Affinity Illustration

```
setenv OMP_PLACES threads
Setenv OMP_NUM_THREADS 4,4
setenv OMP_PROC_BIND spread,close
```

Spread

close



Edison/Cori/Babbage: Run Time Environment Variables

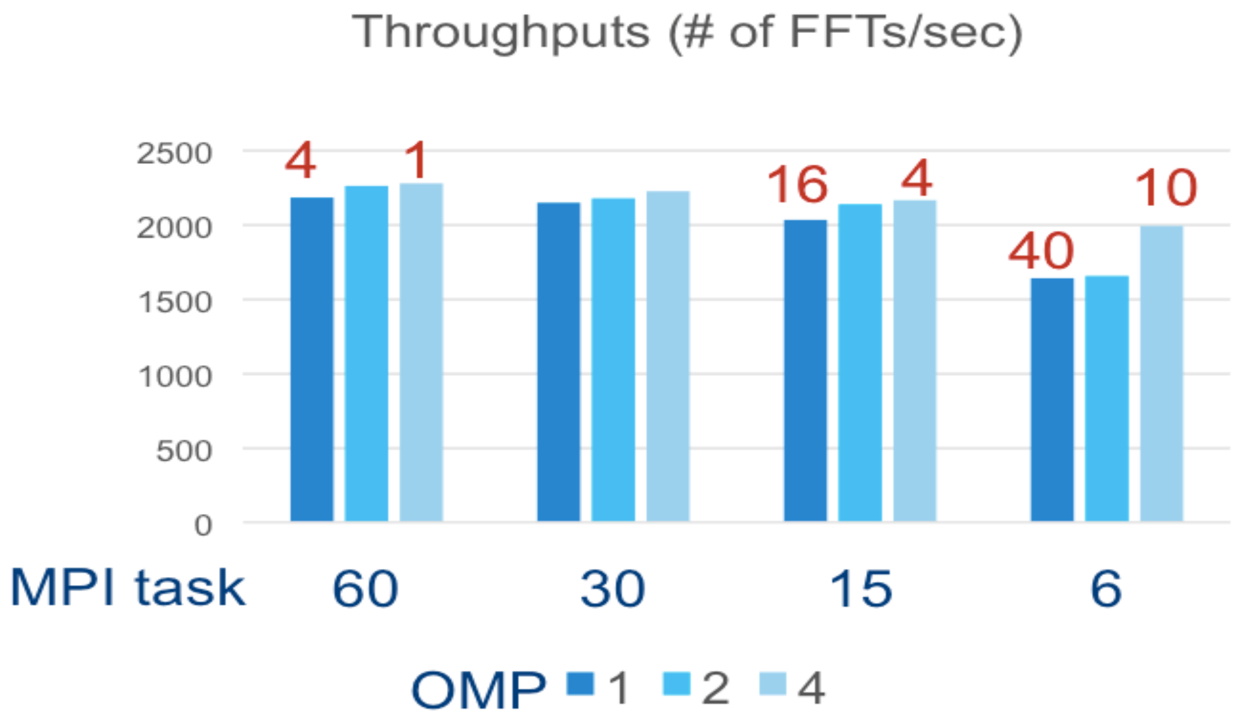
- **setenv OMP_NESTED true**
  - Default is false for most compilers
- **setenv OMP_MAX_ACTIVE_LEVELS 2**
  - The default was 1 for CCE prior to cce/8.4.0
- **setenv OMP_NUM_THREADS 4,3**
- **setenv OMP_PROC_BIND spread,close**
- **setenv KMP_HOT_TEAMS 1**
  - Intel only env. Default is false
- **setenv KMP_HOT_TEAMS_MAX_LEVELS 2**
  - Intel only env. Allow nested level OpenMP threads to stay alive instead of being destroyed and created again to reduce thread creation overhead.
- **Edison/Cori:**
  - srun -n 2 -c 12 ./nested.intel.edison
  - Use -c for total number of threads (products of num_threads from all levels).
- **Babbage:**
  - Set I_MPI_PIN_DOMAIN=auto to get basic MPI process affinity
  - Do not set KMP_AFFINITY, otherwise OMP_PROC_BIND will be ignored.
  - mpirun.mic -n 2 -host bc1109-mic0 ./xthi-nested.mic |sort

Use Multiple Threads in MKL

- **By Default, in OpenMP parallel regions, only 1 thread will be used for MKL calls.**
  - MKL_DYNAMICS is true by default
- **Nested OpenMP can be used to enable multiple threads for MKL calls. Treat MKL as a nested inner OpenMP region.**
- **Sample settings**

```
export OMP_NESTED=true
export OMP_PLACES=cores
export OMP_PROC_BIND=close
export OMP_NUM_THREADS=6,4
export MKL_DYNAMICS=false
export KMP_HOT_TEAMS=1
export KMP_HOT_TEAMS_MAX_LEVELS=2
```

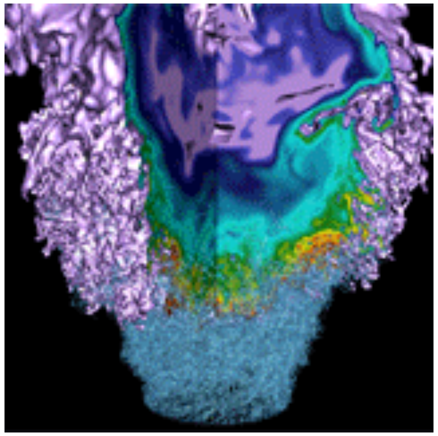
FFT3D on KNC, Ng=64^3

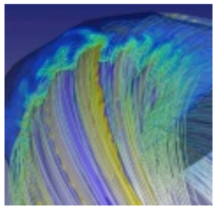


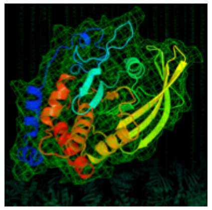
$$N_{MKL} = 240/(N_{MPI} * OMP)$$

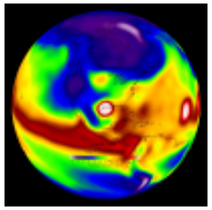
Courtesy of Jeongnim Kim, Intel

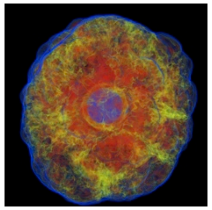
CESM MG2 Kernel Case Study

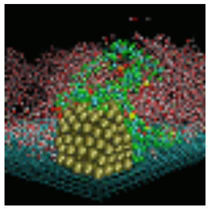












NESAP CESM Team:

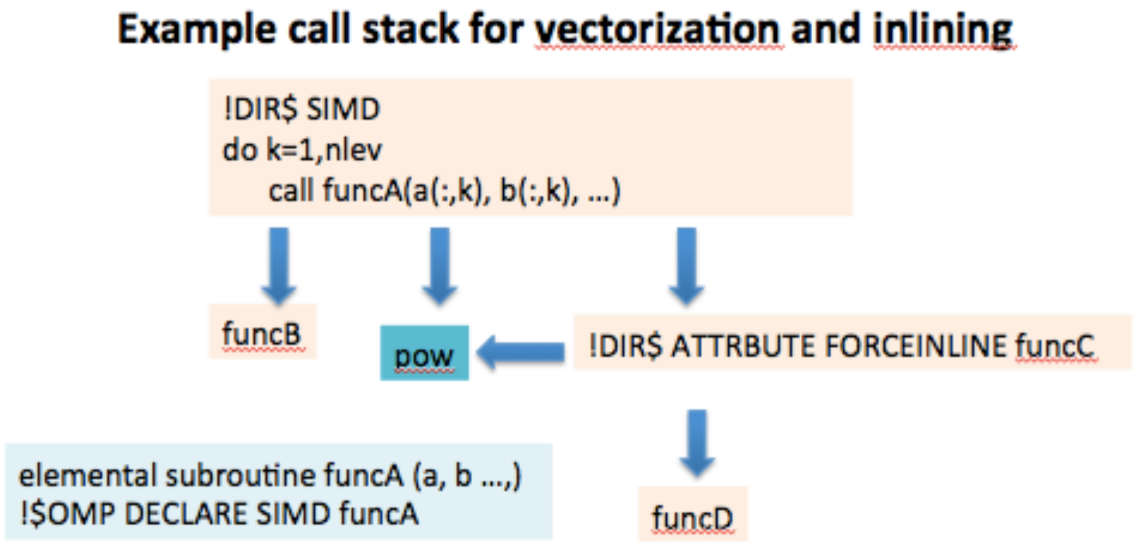
NCAR CESM developers: John Dennis (PI), **Christopher Kerr**, Sean Santos
Intel engineers: Nadezhda Plotnikova, Martyn Corden
Cray Center of Excellence: Marcus Wagner
NERSC Liaison: Helen He

MG2 Kernel

- **MG2 is a kernel for CESM that represents version 2 of the Morrison-Gettleman micro-physics package. Typically consumes about 10% of CESM run time.**
  - Brought to Dungeon Session in March 2015
- **Kernel is core bound**
  - Not bandwidth limited at all
  - Shows very little vectorization
    - Some loop bounds are short (e.g. 10)
    - Dependent sequence of instructions
  - Heavy use of math instrinsics that do not vectorize
    - pow(), gamma(), log10().
    - Intel intrinsic gamma() is 2.6x slower than MG2 internal function
- **Kernel has long complex loops with interleaved conditionals and elemental function calls.**
  - Mixed conditionals and non-inlined functions inhibit vectorization
  - Some send array sections to elemental functions

MG2 Vectorization Prototype

- **Use compiler report to check and make sure key functions are vectorized (and all functions on the call stack are vectorized too)**
  - Elemental functions need to be inlined
  - "-qopt-report=5" reports highest level of details.
  - "-ipo" is needed if functions are in different source codes.
- **Add !$OMP DECLARE SIMD and !DIR$ ATTRIBUTE FORCEINLINE when needed.**


Example call stack for vectorization and inlining
!DIR$ SIMD
do k=1,nlev
call funcA(a(:,k), b(:,k), ...)
funcB
pow
!DIR$ ATTRBUTE FORCEINLINE funcC
elemental subroutine funcA (a, b ...,)
!$OMP DECLARE SIMD funcA
funcD

Recommendations from Dungeon Session

- Divide major loops when possible and localize vectorization: work to be done by MG2 developers.
- Implement inlining as close to hotspot as possible; or use vector functions on the low level
- Follow up with MKL team on Gamma function vectorization.
- Work with compiler team for a flag to replace FORCEINLINE, and portable options for other compilers.

Changes Made to Improve Performance (1)

-- Routines with 'elemental' attribute don't inline

-- Without 'elemental' attribute routines still don't inline!

- Remove 'elemental' attribute and move the 'mgncol' loop inside routine

Before change:

```
elemental function
wv_sat_svp_to_qsat(es, p)
result(qs)

  real(r8), intent(in) :: es  !
SVP
  real(r8), intent(in) :: p
real(r8) :: qs

  ! If pressure is less than SVP,
set qs to maximum of 1.
  if ( (p - es) <= 0._r8 ) then
     qs = 1.0_r8
  else
     qs = epsilo*es / (p -
omeps*es)
  end if

end function wv_sat_svp_to_qsat
```

After change:

```
function wv_sat_svp_to_qsat(es, p,
mgncol) result(qs)
  integer,
intent(in) :: mgncol
  real(r8), dimension(mgncol),
intent(in) :: es  ! SVP
  real(r8), dimension(mgncol),
intent(in) :: p
  real(r8), dimension(mgncol) :: qs
  integer :: i
  do i = 1, mgncol
  if ( (p(i) - es(i)) <= 0._r8 ) then
     qs(i) = 1.0_r8
  else
     qs(i) = epsilo*es(i) / (p(i) -
omeps*es(i))
  end if
  enddo
end function wv_sat_svp_to_qsat
```

Impact of Code Changes for Elemental Functions

- No changes to algorithm
- Algorithm gives same answers
- Code readability not effected
- Revised code looks almost identical to original
- Provide scalar and vector version of functions
- Overload function names to maintain single naming convention

Changes Made to Improve Performance (2)

- **Structure routine: don’t use assumed-shaped arrays:**

```
Before change:
subroutine size_dist_param_liq(qcic, …,)
          real, intent(in) :: qcic(:)
     do i = 1, SIZE(qcic)
```

```
After change:
subroutine size_dist_param_liq(qcic, …, mgncol)
          real, dimension(mgncol), intent(in) :: qcic
     do I = 1, mgncol
```

Changes Made to Improve Performance (3)

- Divide loop blocks into manageable sizes. Allows compiler to vectorize loops. Can fuse loops during optimization step.
- Remove array syntax: plev(:,:) and replace with loops
- Replace divides: 1.0/plev(i,k) with *plev_inv(i,k)
- Remove initialization of variables that are over written

Changes Made to Improve Performance (4)

- **Rearrange loop order to allow for data alignment**

```
Before change:
do i=1,mgncol
     do k=1,nlev
       plev(i,k) = …
```

```
After change:
Do k=1,nlev
     do i=1,mgncol
       plev(i,k) = …
```

- **Use more aggressive compiler options**
  - `-O3 -xAVX -fp-model fast=2 -no-prec-div -no-prec-sqrt -ip -fimf-precision=low -override-limits -qopt-report=5 -no-inline-max-total-size -inline-factor=200`
- **Use Profile-guided Optimization (PGO) to improve code performance**
- **Compare performance of code with different vendors compilers**

Changes Made to Improve Performance (5)

- **Align data on specific byte boundaries; directive based approach with OMP directive:**
  - Portable solution:

    !$OMP SIMD ALIGNED (qc,qcn,nc,ncn,qi,qin,ni,nin,qr,qrn,nr,nrn,qs,qsn,ns,nsn)
    - Tells the compiler that the arrays are aligned
    - Asserts that there are no dependencies
    - Requires to use PRIVATE or REDUCTION clauses to ensure correctness
    - Forces the compiler to vectorize, whether or not it thinks if it is a good idea or not
  - As compared to:

    !DIR$ VECTOR ALIGNED
    - Tells the compiler that the arrays are aligned
    - Intel compiler specific, not portable
- **!$OMP SIMD ALIGNED is independent of vendor, however it can be overly intrusive in code.**

OMP SIMD ALIGNED

- **Using the “ALIGNED” attribute achieved 8% performance gain when the list is explicitly provided.**
- **However, the process is tedious and error-prone, and often times impossible in large real applications.**
  - !$OMP SIMD ALIGNED added in 48 loops in MG2 kernel *(by Christopher Kerr)*, many with list of 10+ variables

!$OMP SIMD ALIGNED	!$OMP SIMD	!dir$ VECTOR ALIGNED	-align array64byte	-openmp	Time per iteration (usec) on Edison
x			x	x	**444**
x				x	446
	x		x	x	**484**
	x			x	482
		x	x		452
		x			456
					473

OMP SIMD ALIGNED

- **How can compilers know better which arrays are aligned so users do not have to specify?**
  - A variable can be declared as aligned
  - A variable can be set to aligned with a compiler flag
  - When in scope, hopefully complier should know
- **Inquired with Fortran Standard:**
  - Equivalent of “!$DIR ATTRIBUTES ALIGNED: 64 :: A”
    - C/C++ standard: float A[1000] __attribute__((aligned(64)));
    - Not in Fortran standard yet
  - Equivalent of the “-align array64byte” compiler flag
    - Exist in Intel (Fortran only) and Cray compilers
    - What about other compilers?

MG2 Optimization Steps

Version 1

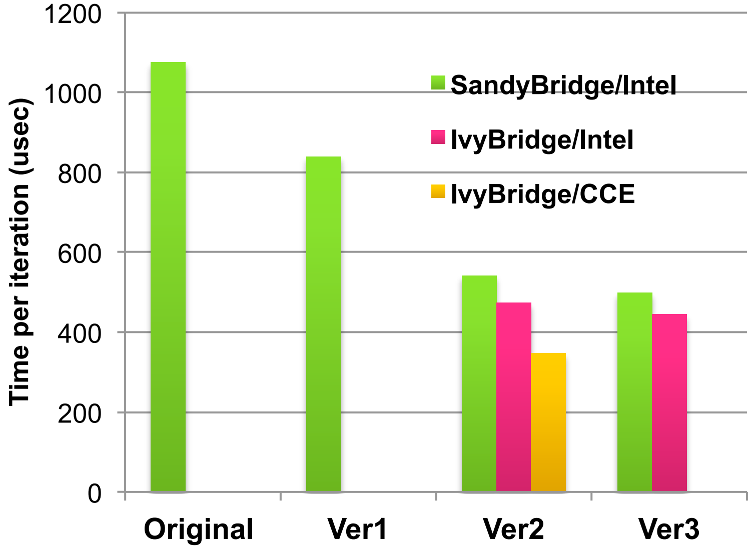
- Simplify expressions to minimize #operations
- Use internal GAMMA function

Version 2

- Remove “elemental” attribute, move loop inside.
- Inline subroutines. Divide, fuse, exchange loops.
- Replace assumed shaped arrays with loops
- Replace division with inversion of multiplication
- Remove initialization of loops to be overwritten later
- Use more aggressive compiler flags. Try different compilers.
- Use profile-guided optimization (PGO)

Version 3 (Intel compiler only)

- Use !$OMP SIMD ALIGNED to force vectorization



MG2 Summary

- Directives and flags can be helpful, however not a replacement for programmers' work on code modifications.
- Break up loops and push loops into functions where vectorization can be dealt with directly and can expose logic to compiler.
- Incremental improvements not necessary a BIG win from any one thing. Accumulative results matter.
- Performance and portability is a major goal: use !$OMP SIMD proves to be beneficial but very hard to use regarding the need of providing the aligned list.
- Requested optional alignment declaration in Fortran Language Standard.
- See case study at https://www.nersc.gov/users/computational-systems/cori/application-porting-and-performance/application-case-studies/cesm-case-study/

Thank you.