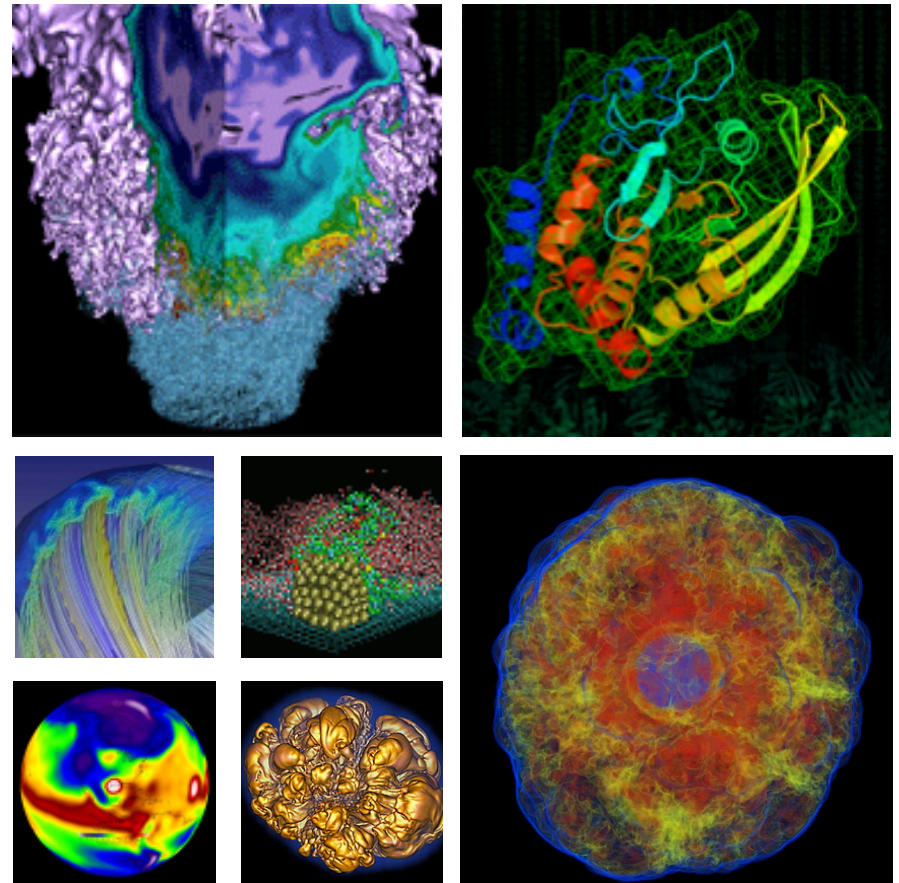


Adding OpenMP to Your Code Using Cray Reveal



Helen He
NERSC User Services Group

October 10, 2013

Current Architecture Trend



- **Multi-socket nodes with rapidly increasing core counts**
- **Memory per core decreases**
- **Memory bandwidth per core decreases**
- **Network bandwidth per core decreases**
- **Need a hybrid programming model with three levels of parallelism**
 - MPI between nodes or sockets
 - Shared memory (such as OpenMP) on the nodes/sockets
 - Increase vectorization for lower level loop structures

Advantages of hybrid MPI/OpenMP



- Reduce number of MPI ranks per node
- Minimize network injection contention
- Avoids the extra communication overhead with MPI within node
- Reduce memory footprint
- Chance of overlapping MPI communication with OpenMP thread computation

What is Reveal



- A tool developed by Cray to help developing the hybrid programming model
- Part of the Cray Perftools software package
- Only works under PrgEnv-cray
- Utilizes the Cray CCE program library for loopmark and source code analysis, combined with performance data collected from CrayPat
- Helps to identify top time consuming loops, with compiler feedback on dependency and vectorization
- Loop scope analysis provides variable scope and compiler directive suggestions for inserting OpenMP parallelism to a serial or pure MPI code

Steps to Use Reveal on Edison (1)



- **Load the user environment**
 - % module swap PrgEnv-intel PrgEnv-cray
 - % module unload darshan
 - % module load perftools (current default is version 6.1.2)
- **Generate loop work estimates**
 - % ftn -c -h profile_generate myprogram.f90
 - % ftn -o poisson_serial -h profile_generate myprogram.o
 - Good to separate compile and link to keep object files
 - Optimization flags disabled with -h profile-generate
 - % pat_build -w myprogram (-w enables tracing)
 - It will generate executable “myprogram+pat”
 - Run the program “myprogram+pat”
 - It will generate one or more myprogram+pat+...xf files
 - % pat_report myprogram+pat...xf > myprogram.rpt
 - It will generate myprogram+pat....ap2 file

Steps to Use Reveal on Edison (2)



- **Generate a program library**
 - `% ftn -O3 -hpl=myprogram.pl -c myprogram.f90`
 - Optimization flags can be used
 - Build one source code at a time, with “-c” flag
 - Use absolute path for program library if sources are in multiple directories
 - User needs to clean up program library from time to time
- **Launch Reveal**
 - `% reveal myprogram.pl myprogram+pat...ap2`

Steps to Use Reveal on Hopper



- **To use the newest version perftools/6.1.2, which is built upon cray-mpich/6.x.x**
 - % module unload cray-libsci cray-mpich2
 - % module load cray-libsci/12.1.01
 - % module load cray-mpich/6.1.0
 - % module unload darshan
 - % module load perftools-lite/6.1.2
- **To use perftools-lite/6.1.1 or older**
 - % module unload darshan
 - % module load perftools-lite/6.1.2
- **Follow the rest of steps for Edison**

Cray Reveal GUI



The screenshot displays the Cray Reveal GUI interface. At the top, there is a menu bar with 'File', 'Edit', 'View', and 'Help'. Below the menu bar, a window titled 'poisson_mpi.c' is open. The interface is divided into three main sections:

- Navigation:** A tree view on the left showing the program structure. The root is 'poisson_mpi.c', which contains several sub-items: 'allocate_arrays', 'jacobi', 'main', 'make_domains', and 'make_source'. Each sub-item is associated with a specific loop and processor ID, such as '0.0055 Loop@252' for 'allocate_arrays' and '33.5342 Loop@148' for 'main'.
- Source:** A large text area on the right for editing the source code. It includes navigation buttons for 'Up', 'Down', and 'Save'.
- Info:** A section at the bottom of the window, currently empty.

A callout box is centered in the 'Source' area, containing the text: **New to Reveal?** [Try "Getting Started" in the "Help" Menu](#).

Top loops with compiler loopmarks and feedback

File Edit View Help

poisson_mpi.c

Navigation

Top Loops

- 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_dom
 - 0.0000 Loop@404
 - make_source
 - 0.0056 Loop@495
 - timestamp

Compiler loopmarks

```
159 my_n = 0;
160
L 161 for ( i = i_min[my_rank]; i <= i_max[my_rank]; i++ )
162 {
Lr4 163   for ( j = 1; j <= N; j++ )
164   {
165     if ( u_new[INDEX(i,j)] != 0.0 )
166     {
167       my_change = my_change
168         + fabs ( 1.0 - u[INDEX(i,j)] / u_new[INDEX(i,j)] );
169
170       my_n = my_n + 1;
171     }
172   }
173 }
!I 174 MPI_Allreduce ( &my_change, &change, 1, MPI_DOUBLE, MPI_SUM,
175               MPI_COMM_WORLD );
!I 176
!I 177 MPI_Allreduce ( &my_n, &n, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD );
178
179 if ( n != 0 )
180 {
181   change = change / n;
```

Info - Line 161

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

Compiler feedback

Compiler feedback explanation



The screenshot shows an IDE window titled "poisson_mpi.c" with a "Navigation" pane on the left and a "Source" pane on the right. The "Navigation" pane shows a tree view of the file's structure, with "Loop@163" selected. The "Source" pane shows the following code:

```
161 for ( i = i_min[my_rank]; i <= i_max[my_rank]; i++ )
162 {
Lr4 163 for ( j = 1; j <= N; j++ )
164 {
165     MPI_SUM,
166     _COMM_WORLD );
167     u_new = u + my_change;
168 }
169 if ( my_rank == 0 && ( step % 1000 ) == 0 )
```

An "Explain" dialog box is open over line 163, displaying the following text:

Explain CC-6290

VECTOR: A loop was not vectorized because a recurrence was found between "var" and "var" at line num.

Scalar code was generated for the loop because it contains a linear recurrence. The following loop would cause this message to be issued:

```
for (i = 1; i < 100; i++) {
    b[i] = a[i-1];
    a[i] = b[i];
}
```

Buttons: Explain other message..., Close

Below the source pane, an "Info" bar shows the following messages:

- A loop was not vectorized because a recurrence was found between "u_new" and "my_change" at line 167.
- A loop was unrolled 4 times.

A callout box with a double arrow points to the "Info" bar, containing the text: "Double click to explain".

Compiler feedback explanation (2)



The screenshot shows a compiler feedback window with two panes: 'Reveal' and 'Explain'.

Reveal Pane:

- Navigation:** A tree view showing the file structure of `poisson_mpi.c`. The loop at line 163 is selected.
- Source:** The source code for `poisson_mpi.c` is displayed. Lines 161-183 are visible. Line 163 is highlighted in blue and labeled 'Lr4'. Lines 174 and 177 are labeled '!I'.
- Info:** A message at the bottom states: "A loop was unrolled 4 times."

Explain Pane:

- Title:** Explain CC-6005
- Message:** "SCALAR: A loop was unrolled." This message indicates that unroll-and-jam was performed with respect to the identified loop. A different message is issued when literal outer loop unrolling is performed, as this transformation is far less likely to be beneficial.
- Text:** "The compiler unrolled the loop. Unrolling creates a number of copies of the loop body. When unrolling an outer loop, the compiler attempts to fuse replicated inner loops - a transformation known as unroll-and-jam. The compiler will always employ the unroll-and-jam mode when unrolling an outer loop; literal outer loop unrolling may occur when unrolling to satisfy a user directive (pragma). For sake of illustration, the following contrasts unroll-and-jam with literal outer loop unrolling."
- Code Examples:** Three code snippets are shown to illustrate the difference between unroll-and-jam and literal outer loop unrolling.
- Bottom:** "Explain other message..." and "Close" buttons.

Reveal scoping assistance



The screenshot shows the Reveal IDE interface with a code editor and a navigation pane. The code editor displays the following C code snippet:

```
159 my_n = 0;
160
161 for ( i = i_min[my_rank]; i <= i_max[my_rank]; i++ )
162 {
163     for ( j = 1; j <= N; j++ )
164     {
165         if ( u_new[INDEX(i,j)] != 0.0 )
166         {
167             my_change = my_change
168                 + fabs ( 1.0 - u[INDEX(i,j)] / u_new[INDEX(i,j)] );
169
170             my_n = my_n + 1;
171         }
172     }
173
174 MPI_Allreduce ( &my_change, &change, 1, MPI_DOUBLE, MPI_SUM,
175               MPI_COMM_WORLD );
176
177 MPI_Allreduce ( &my_n, &n, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD );
178
179 if ( n != 0 )
```

The navigation pane on the left shows a tree view of loops, with 'Loop@161' selected. A callout box points to the 'Scope Loop' button in the navigation pane with the text: "Right click to select loops".

The 'Info' pane at the bottom of the code editor displays the following message:

```
Info - Line 161
● A loop was not vectorized because a recurrence was found between "u_new" and "my_change" at line 167.
```

On the right side, the 'Reveal OpenMP Scoping' dialog box is shown. It has two tabs: 'Scope Loops' and 'Scoping Results'. The 'Scoping Results' tab is active, showing a table of loops to be scoped:

Scope?	Line #	File or Source Line
<input checked="" type="checkbox"/>	161	/global/project/projectdirs/mpccc/yunhe/reveal/edison/poisson_mpi.c for (i = i_min[my_rank]; i <= i_max[my_rank]; i++)
<input checked="" type="checkbox"/>	163	for (j = 1; j <= N; j++)

At the bottom of the dialog box, there are buttons for 'Start Scoping', 'Cancel', and 'Close'. A callout box points to the 'Start Scoping' button with the text: "Start Scoping".

Scoping Results



The screenshot shows a code editor with a file named `poisson_mpi.c`. The source code is as follows:

```
159 my_n = 0;
160
161 for ( i = i_min[my_rank]; i <= i_max[my_rank]; i++ )
162 {
163     for ( j = 1; j <= N; j++ )
164     {
165         if ( u_new[INDEX(i,j)] != 0.0 )
166         {
167             my_change = my_change
168                 + fabs ( 1.0 - u[INDEX(i,j)] / u_new[INDEX(i,j)] );
169
170             my_n = my_n + 1;
171         }
172     }
173 }
174 MPI_Allreduce ( &my_change, &change, 1, MPI_DOUBLE, MPI_SUM,
175               MPI_COMM_WORLD );
176
177 MPI_Allreduce ( &my_n, &n, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD );
178
179 if ( n != 0 )
180 {
```

The `Scoping Results` window displays the following table:

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.
			FAIL: Private/Shared Scope Conflict.
			FAIL: Value/Shared Scope Conflict.
			FAIL: unable to determine last iteratin that defines object.
i	Scalar	Private	
N	Scalar	Shared	
i_max	Scalar	Shared	
my_rank	Scalar	Shared	
u	Scalar	Shared	
u_new	Scalar	Shared	

Below the table, there are controls for `First/Last Private` and `Reduction`.

An `Info` window at the bottom of the editor shows the following message:

Line 161: A loop was not vectorized because a recurrence was found between "u_new" and "my_change" at line 167.

Suggested OpenMP directives



The screenshot displays the Cray Reveal IDE interface. The main window shows the source code for `poisson_mpi.c`. A loop starting at line 161 is highlighted in blue, indicating it is the current scope. The code includes nested loops for `i` and `j`, and a calculation involving `u_new` and `my_change`. An information window at the bottom states: "Info - Line 161: A loop was not vectorized because a recurrence was found between 'u_new' and 'my_change' at line 167." To the right, the "Reveal OpenMP Scoping" window is open, showing the "Scoping Results" for the selected loop. It lists variables `my_change` and `my_n` as "Unresolved" with a "FAIL" status, indicating that the last defining iteration is not known for variables that live on exit. An "OpenMP Directive" dialog box is also visible, showing the suggested directive: `#pragma omp parallel for default(none) unresolved(my_change,my_n) shared(my_rank,N,i_max,u_new,u) firstprivate(i)`. The IDE status bar at the bottom indicates that `poisson_mpi.pl` and `poisson_mpi+pat+1119130-3263t.ap2` are loaded.

Save the directives



The screenshot shows the Cray Reveal IDE interface. On the left is a 'Navigation' pane with a 'Program View' tree showing the file structure of 'poisson_mpi.c'. The main window displays the source code for 'poisson_mpi.c'. The code includes MPI-related functions and a Jacobi update loop. A callout box points to the 'Save' button in the top right of the source editor, with the text 'Save directives to the original file'. The status bar at the bottom indicates 'poisson_mpi.pl loaded. poisson_mpi+pat+1128393-4576t.ap2 loaded.'

```
316         right_proc[my_rank], 2, MPI_COMM_WORLD,  
317         request + requests++ );  
318     MPI_Isend ( u + INDEX(i_max[my_rank], 1), N, MPI_DOUBLE,  
319               right_proc[my_rank], 0, MPI_COMM_WORLD,  
320               request + requests++ );  
321 }  
322 /*  
323 Jacobi update for internal vertices in my domain.  
324 */  
325 // Directive inserted by Cray Reveal. May be incomplete.  
326 #pragma omp parallel for default(none)  
327     shared (f,N,u,u_new,i,h)  
328     for ( i = i_min[my_rank] + 1; i <= i_max[my_rank] - 1; i++ )  
329     {  
330         for ( j = 1; j <= N; j++ )  
331         {  
332             u_new[INDEX(i,j)] =  
333                 0.25 * ( u[INDEX(i-1,j)] + u[INDEX(i+1,j)] +  
334                       u[INDEX(i,j-1)] + u[INDEX(i,j+1)] +  
335                       h * h * f[INDEX(i,j)] );  
336         }  
337     }
```

Save directives to the original file

Extensive “Help” topics in Reveal



The screenshot displays the Reveal application interface with several panels open:

- Navigation Panel:** Shows a tree view of the code structure for `poisson_mpi.pl`. The `main` function is expanded, showing loops like `Loop@148` through `Loop@495`. The `Loop@163` is highlighted in blue.
- Code Editor:** Displays C code from `poisson_mpi.c`. Lines 164-184 are visible, showing a loop with an `if` statement and `MPI_Allreduce` calls. Line 167 is highlighted in green, and lines 166 and 177 are highlighted in cyan.
- Info Panel:** Located at the bottom of the code editor, it provides details for Line 163:
 - A loop was not vectorized because a recurrence was found between "u_new" and "my_change" at line 167.
 - A loop was unrolled 4 times.
- Loopmark Legend:** A panel on the right side listing various loop characteristics and their corresponding icons:
 - A Pattern Matched
 - C Collapsed
 - D Deleted
 - E Cloned
 - G Accelerated
 - I Inlined
 - II Not Inlined
 - L Loop
 - M Multithreaded
 - R Region
 - S Scoping Analysis
 - V VectorizedBelow this list are letters `a` through `w` representing different scoping or optimization states.
- Extended Help:** A panel at the bottom right titled "Extended Help" with a "Program Library" section. It contains text about scoping loops and creating OpenMP clauses or directives.

Reveal helps to start adding OpenMP



- Only under PrgEnv-cray, with CCE compiler
- Start from most time consuming loops first
- Insert OpenMP directives
 - Make sure to save a copy of the original code first, since the saved new file will overwrite the original code
- There will be unresolved and incomplete variable scopes
- There maybe more incomplete and incorrect variables identified when compiling the resulted OpenMP codes
- User still needs to understand OpenMP, and resolves the issues.
- Verify correctness and performance
- Repeat as necessary
- No OpenMP tasks, barrier, critical, atomic regions, etc

More work after reveal (1)



Reveal suggests:

```
#pragma omp parallel for default(none) \
    unresolved (my_change,my_n) \
    shared (my_rank,N,i_max,u_new,u) \
    firstprivate (i)

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;
        }
    }
}
```

Final code:

(Lots of changes from Reveal suggestions, but will still make the code slower than without OpenMP directives, so will not use any directives)

```
#pragma omp parallel for default(none) \
    private (my_change,my_n) \
    shared (my_rank,N,i_min,i_max,u_new,u) \
    private (j) \
    private (i)

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 )
        #pragma omp critical
        {
            my_change = my_change
                + fabs ( 1.0 - u[INDEX(i,j)] / u_new[INDEX(i,j)] );

            my_n = my_n + 1;
        }
    }
}
```

More work after Reveal (2)



Reveal suggests:

```
#pragma omp parallel for default(none) \
    shared (f,N,u,u_new,i,h)

for ( i = i_min[my_rank] + 1; i <= i_max[my_rank] - 1; i+
+)
{
    for ( j = 1; j <= N; j++ )
    {
        u_new[INDEX(i,j)] =
            0.25 * ( u[INDEX(i-1,j)] + u[INDEX(i+1,j)] +
                u[INDEX(i,j-1)] + u[INDEX(i,j+1)] +
                h * h * f[INDEX(i,j)] );
    }
}
```

Final code:

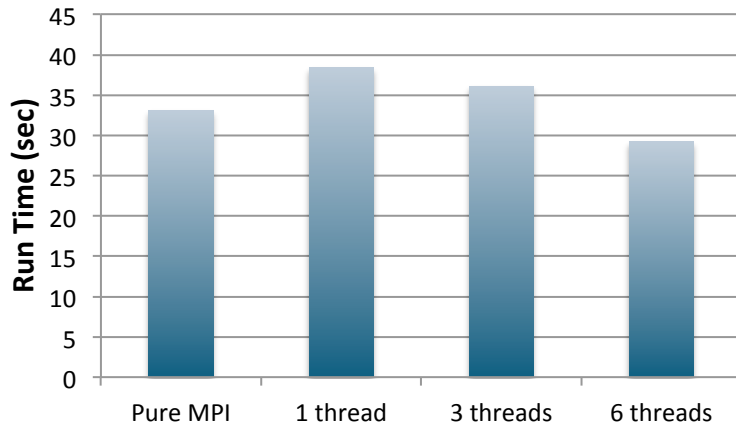
```
#pragma omp parallel for default(none) \
    private (my_rank,j,i) \
    shared (f,N,u,u_new,h,i_min,i_max)

for ( i = i_min[my_rank] + 1; i <= i_max[my_rank] - 1; i++ )
{
    for ( j = 1; j <= N; j++ )
    {
        u_new[INDEX(i,j)] =
            0.25 * ( u[INDEX(i-1,j)] + u[INDEX(i+1,j)] +
                u[INDEX(i,j-1)] + u[INDEX(i,j+1)] +
                h * h * f[INDEX(i,j)] );
    }
}
```

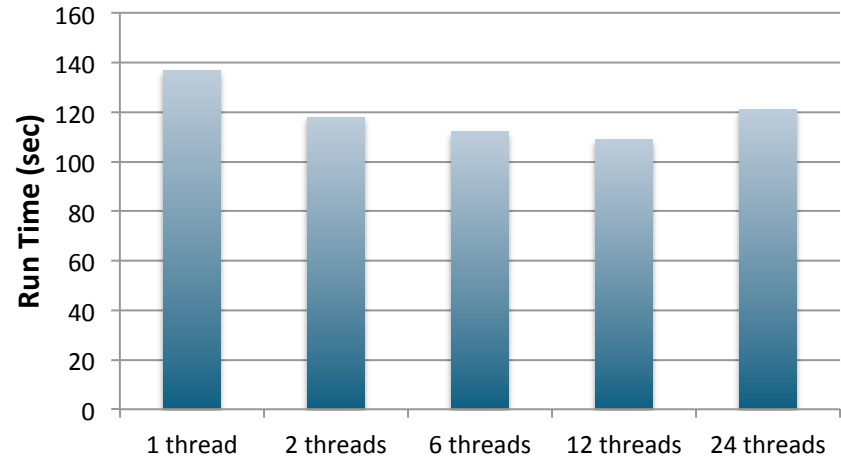
Performance with OpenMP added



Poisson_mpi_omp, 4 MPI tasks,
N=1200, on Edison



poisson_omp
nx=ny=1201, on Edison



- **% module load training**
- **See example codes, reports, detailed steps in README at:**
 - \$EXAMPLES/Edison2013/reveal
- **Documentations:**
 - % man reveal (when the “perftools” module is loaded)
 - Using Cray Performance Measurement and Analysis Tools
<http://docs.cray.com/books/S-2376-612/S-2376-612.pdf>



National Energy Research Scientific Computing Center