Using the Cray Programming Environment to Convert an all MPI code to a Hybrid-Multi-core Ready Application

John Levesque
Cray’s Supercomputing Center of Excellence
The Target
ORNL’s “Titan” System

- Upgrade of Jaguar from Cray XT5 to XK6
- Cray Linux Environment operating system
- Gemini interconnect
  - 3-D Torus
  - Globally addressable memory
  - Advanced synchronization features
- AMD Opteron 6274 processors (Interlagos)
- New accelerated node design using NVIDIA multi-core accelerators
  - 2011: 960 NVIDIA x2090 “Fermi” GPUs
  - 2012: 14,592 NVIDIA K20 “Kepler” GPUs
- 20+ PFlops peak system performance
- 600 TB DDR3 mem. + 88 TB GDDR5 mem

Titan Specs

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute Nodes</td>
<td>18,688</td>
</tr>
<tr>
<td>Login &amp; I/O Nodes</td>
<td>512</td>
</tr>
<tr>
<td>Memory per node</td>
<td>32 GB + 6 GB</td>
</tr>
<tr>
<td># of Fermi chips (2012)</td>
<td>960</td>
</tr>
<tr>
<td># of NVIDIA K20 “Kepler” processor (2013)</td>
<td>14,592</td>
</tr>
<tr>
<td>Total System Memory</td>
<td>688 TB</td>
</tr>
<tr>
<td>Total System Peak Performance</td>
<td>20+ Petaflops</td>
</tr>
<tr>
<td>Cross Section Bandwidths</td>
<td>X=14.4 TB/s, Y=11.3 TB/s, Z=24.0 TB/s</td>
</tr>
</tbody>
</table>
The Challenge

Not the first Six

INCITE Awardees
How Effective are GPUs on Scalable Applications?  
OLCF-3 Early Science Codes -- Current performance measurements on TitanDev

<table>
<thead>
<tr>
<th>Application</th>
<th>XK6 (w/ GPU) vs. XK6 (w/o GPU)</th>
<th>XK6 (w/ GPU) vs. XE6</th>
<th>Cray XK6: Fermi GPU plus Interlagos CPU Cray XE6: Dual Interlagos and no GPU</th>
<th>Comment</th>
</tr>
</thead>
</table>
| S3D         | 1.5                             | 1.4 (now 1.9)        | • Turbulent combustion  
• 6% of Jaguar workload                                                  |         |
| Denovo      | 3.5                             | 3.3                  | • 3D neutron transport for nuclear reactors  
• 2% of Jaguar workload                                                  |         |
| LAMMPS      | 6.5                             | 3.2                  | • High-performance molecular dynamics  
• 1% of Jaguar workload                                                  |         |
| WL-LSMS     | 3.1                             | 1.6                  | • Statistical mechanics of magnetic materials  
• 2% of Jaguar workload  
• 2009 Gordon Bell Winner                                                |         |
| CAM-SE      | 2.6                             | 1.5                  | • Community atmosphere model  
• 1% of Jaguar workload                                                  |         |

These are all Fermi+ numbers – Cannot show Kepler

NCAR Sept12-13
The Approach
The Approach

- Formulate a target problem that does real science and is same work/node as all MPI
S3D – Weak Scaling Study

- All MPI mesh on a node
  - $15 \times 15 \times 15 \times 32 = 108000$

- One MPI tasks/node
  - $48 \times 48 \times 48 = 110592$

- Two MPI tasks/node
  - Share GPU with the two MPI tasks
  - $38 \times 38 \times 38 \times 2 = 109744$
The Approach

- Formulate a target problem that does real science and is same work/node as all MPI
- **Identify hotspots**
  - Using Craypat –hprofile_generate to identify loop structure
<table>
<thead>
<tr>
<th>Function</th>
<th>Time</th>
<th>Mbytes</th>
<th>Percentage</th>
<th>Compilation</th>
<th>Source</th>
<th>Function</th>
<th>Time</th>
<th>Mbytes</th>
<th>Percentage</th>
<th>Compilation</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER</td>
<td>88.723495</td>
<td>13542013.0</td>
<td>75.4%</td>
<td>-</td>
<td>-</td>
<td>USER</td>
<td>88.723495</td>
<td>13542013.0</td>
<td>75.4%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>parabola_</td>
<td>12.589734</td>
<td>2592000.0</td>
<td>10.7%</td>
<td>-</td>
<td>-</td>
<td>parabola_</td>
<td>12.589734</td>
<td>2592000.0</td>
<td>10.7%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>remap_.LOOPS</td>
<td>8.360290</td>
<td>1728000.0</td>
<td>7.1%</td>
<td>-</td>
<td>-</td>
<td>remap_.LOOPS</td>
<td>8.360290</td>
<td>1728000.0</td>
<td>7.1%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ppmlr</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>ppmlr</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>sweepx2_.LOOP.2.li.35</td>
<td>3.708452</td>
<td>768000.0</td>
<td>3.2%</td>
<td>-</td>
<td>-</td>
<td>sweepx2_.LOOP.2.li.35</td>
<td>3.708452</td>
<td>768000.0</td>
<td>3.2%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>sweepx2_.LOOP.1.li.34</td>
<td>3.663423</td>
<td>768000.0</td>
<td>3.1%</td>
<td>-</td>
<td>-</td>
<td>sweepx2_.LOOP.1.li.34</td>
<td>3.663423</td>
<td>768000.0</td>
<td>3.1%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ppmlr_</td>
<td>4.229443</td>
<td>864000.0</td>
<td>3.6%</td>
<td>-</td>
<td>-</td>
<td>ppmlr_</td>
<td>4.229443</td>
<td>864000.0</td>
<td>3.6%</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
The Approach

- Formulate a target problem that does real science and is same work/node as all MPI
- Identify hotspots
  - Using Craypat –hprofile_generate to identify loop structure
- Convert all- MPI into Hybrid, using high level OpenMP
  - Tune to beat the all MPI
  - Structure to be general purpose, n MPI tasks/node, m threads/node
  - Use OpenMP – easy path to the next step
  - This is important for all future multi-petaflop systems
Parallel Analysis, Scoping and Vectorization

- Current scoping tool, -homp_analyze, is meant to interface to a code restructuring GUI called “reveal”.
  - !dir$ omp_analyze_loop
- In order to utilize scoping tool for loops that contain procedures the program library need to be employed
  - -hwp –hpl=vhone.aid
    - This will do an initial pass of the code, checking for error and then at the load it will build the program library and perform the analysis
- Compiler will be very conservative
Main window of reveal

```plaintext
53  #endif
54  do j = 1, js
55  do i = 1, isz
56      radius = x[i]+myeye'*isz)
57      theta = zyc[i]+myeye'*js)
58      sttheta = sin(theta)
59      radius = radius * sttheta
60
61     ! Put state variables into 1D arrays, padding with 6 ghost zones
62     do m = 1, npez
63     do k = 1, ks
64         n = k + ks*(m-1) + 6
65         r(n) = recv3(1,j,k,i,m)
66         p(n) = recv3(2,j,k,i,m)
67         u(n) = recv3(3,j,k,i,m)
68         v(n) = recv3(4,j,k,i,m)
69         w(n) = recv3(5,j,k,i,m)
70         f(n) = recv3(6,j,k,i,m)
71        enddo
72      enddo
73      do m = 1, kmax
74         n = k + 6
75         x(n) = zza(k)
76         dx(n) = zdz(k)
77         x0(n) = zza(k)
```
Scoping window

![OpenMP Construct window with variables and their scopes]

- Name: zyc, Type: Scalar, Scope: Unknown
- Name: a, Type: Scalar, Scope: Private
- Name: ai, Type: Scalar, Scope: Private
- Name: amid, Type: Scalar, Scope: Private
- Name: ar, Type: Scalar, Scope: Private
- Name: b, Type: Scalar, Scope: Private
- Name: bi, Type: Scalar, Scope: Private
- Name: c, Type: Scalar, Scope: Private
- Name: cdtdx, Type: Scalar, Scope: Private
- Name: ci, Type: Scalar, Scope: Private
- Name: clft, Type: Scalar, Scope: Private
- Name: crgh, Type: Scalar, Scope: Private
Resultant Hybrid S3D Performance

Weak Scaling, Lower is Better

- Blue line: Hybrid S3D
- Red line: All MPI S3D

Number of cores (12 cores/node) vs Wall clock time (seconds)
The Approach

- Formulate a target problem that does real science and is same work/node as all MPI
- Identify hotspots
  - Using Craypat –hprofile_generate to identify loop structure
- Convert all MPI into Hybrid, using high level OpenMP
  - This is important for all future multi-petaflop systems
- Move to accelerator using OpenACC
  - Insert OpenACC on major OpenMP loops
    - Examine Compiler feedback on data movement
• Things that are different between OpenMP and OpenACC
  • Cannot have CRITICAL REGION down callchain
  • Cannot have THREADPRIVATE
  • Vectorization is much more important
  • Cache/Memory Optimization much more important
  • No EQUIVALENCE
• Currently both OpenMP and OpenACC must be included in the source

```c
#ifdef GPU
 !$acc parallel loop private( k,j,i,n,r, p, e, q, u, v, w, svel0,&
 !$acc & xa, xa0, dx, dx0, dvol, f, flat, para,radius, theta, stheta)&
 !$acc & reduction(max:svel)
#else
 !$omp parallel do private( k,j,i,n,r, p, e, q, u, v, w, svel0,&
 !$omp & xa, xa0, dx, dx0, dvol, f, flat, para,radius, theta, stheta)&
 !$omp & reduction(max:svel)
#endif
```
**Compiler list for SWEEPX1**

45. ifdef GPU
46. G------------< !$acc parallel loop private( k,j,i,n,r, p, e, q, u, v, w, svel0,&
47. G           xa, xa0, dx, dx0, dvol, f, flat, para, radius, theta, stheta)&
48. G           !$acc & reduction(max:svel)
49. G #else
50. G           !$omp parallel do private( k,j,i,n,r, p, e, q, u, v, w, svel0,&
51. G           !$omp& xa, xa0, dx, dx0, dvol, f, flat, para, radius, theta, stheta)&
52. G           !$omp& & reduction(max:svel)
53. G #endif
55. G g----------< do k = 1, ks
56. G g 3--------< do j = 1, js
57. G g 3       theta=0.0
58. G g 3       stheta=0.0
59. G g 3       radius=0.0
62. G g 3 g------< do i = 1,imax
63. G g 3 g     n = i + 6
64. G g 3 g     r (n) = zro(i,j,k)
65. G g 3 g     p (n) = zpr(i,j,k)
66. G g 3 g     u (n) = zux(i,j,k)
67. G g 3 g     v (n) = zuy(i,j,k)
68. G g 3 g     w (n) = zuz(i,j,k)
69. G g 3 g     f (n) = zfl(i,j,k)
71. G g 3 g     xa0(n) = zxa(i)
72. G g 3 g     dx0(n) = zdx(i)
73. G g 3 g     xa (n) = zxa(i)
74. G g 3 g     dx (n) = zdx(i)
75. G g 3 g     p (n) = max(smallp,p(n))
76. G g 3 g     e (n) = p(n)/(r(n)*gamm)+0.5*(u(n)**2+v(n)**2+w(n)**2)
77. G g 3 g------> enddo
79. G g 3 ! Do 1D hydro update using PPMLR
80. G g 3 gr2 I--> call ppmlr (svel0, sweep, nmin, nmax, ngeom, nleft, nright,r, p, e, q, u, v, w, &
81. G g 3      xa, xa0, dx, dx0, dvol, f, flat, para, radius, theta, stheta)
A region starting at line 46 and ending at line 104 was placed on the accelerator.

If not already present: allocate memory and copy whole array "zro" to accelerator, free at line 104 (acc_copyin).

If not already present: allocate memory and copy whole array "zpr" to accelerator, free at line 104 (acc_copyin).

If not already present: allocate memory and copy whole array "zux" to accelerator, free at line 104 (acc_copyin).

If not already present: allocate memory and copy whole array "zuy" to accelerator, free at line 104 (acc_copyin).

If not already present: allocate memory and copy whole array "zuz" to accelerator, free at line 104 (acc_copyin).

If not already present: allocate memory and copy whole array "zfl" to accelerator, free at line 104 (acc_copyin).

If not already present: allocate memory and copy whole array "send1" to accelerator, copy back at line 104 (acc_copy).
The Approach

- Formulate a target problem that does real science and is same work/node as all MPI
- Identify hotspots
  - Using Craypat –hprofile_generate to identify loop structure
- Convert all MPI into Hybrid, using high level OpenMP
  - This is important for all future multi-petaflop systems
- Move to accelerator using OpenACC
  - Insert OpenACC on major OpenMP loops
    - Examine Compiler feedback on data movement
- **Introduce data regions outside the timestep loop**
  - Now have to find all the code that accesses the arrays you want to reside on the accelerator
  - Looking at the PIN tool to help with that complex analysis. Especially when using derived types and C++ dynamic arrays
### Table 1: Time and Bytes Transferred for Accelerator Regions

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Host</th>
<th>Acc Copy</th>
<th>Acc Copy</th>
<th>Calls</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td></td>
<td>58.363</td>
<td>67.688</td>
<td>24006.022</td>
<td>16514.196</td>
<td>14007</td>
</tr>
<tr>
<td>30.3%</td>
<td>17.697</td>
<td>0.022</td>
<td>--</td>
<td>--</td>
<td>1000</td>
<td>sweepy_.ACC_KERNAL@li.47</td>
</tr>
<tr>
<td>22.0%</td>
<td>12.827</td>
<td>0.010</td>
<td>--</td>
<td>--</td>
<td>500</td>
<td>sweepx2_.ACC_KERNAL@li.46</td>
</tr>
<tr>
<td>21.2%</td>
<td>12.374</td>
<td>0.013</td>
<td>--</td>
<td>--</td>
<td>500</td>
<td>sweepz_.ACC_KERNAL@li.67</td>
</tr>
<tr>
<td>14.0%</td>
<td>8.170</td>
<td>0.013</td>
<td>--</td>
<td>--</td>
<td>500</td>
<td>sweepx1_.ACC_KERNAL@li.46</td>
</tr>
<tr>
<td>3.9%</td>
<td>2.281</td>
<td>1.161</td>
<td>12000.004</td>
<td>--</td>
<td>1000</td>
<td>sweepy_.ACC_COPY@li.47</td>
</tr>
<tr>
<td>2.0%</td>
<td>1.162</td>
<td>0.601</td>
<td>6000.002</td>
<td>--</td>
<td>500</td>
<td>sweepz_.ACC_COPY@li.67</td>
</tr>
<tr>
<td>1.6%</td>
<td>0.953</td>
<td>0.014</td>
<td>--</td>
<td>6000.004</td>
<td>1000</td>
<td>sweepy_.ACC_COPY@li.129</td>
</tr>
<tr>
<td>1.0%</td>
<td>0.593</td>
<td>0.546</td>
<td>3000.002</td>
<td>--</td>
<td>500</td>
<td>sweepx1_.ACC_COPY@li.46</td>
</tr>
<tr>
<td>1.0%</td>
<td>0.591</td>
<td>0.533</td>
<td>3000.002</td>
<td>--</td>
<td>500</td>
<td>sweepx2_.ACC_COPY@li.46</td>
</tr>
<tr>
<td>0.8%</td>
<td>0.494</td>
<td>0.015</td>
<td>--</td>
<td>3000.002</td>
<td>500</td>
<td>sweepx2_.ACC_COPY@li.107</td>
</tr>
<tr>
<td>0.8%</td>
<td>0.485</td>
<td>0.007</td>
<td>--</td>
<td>3000.002</td>
<td>500</td>
<td>sweepx1_.ACC_COPY@li.104</td>
</tr>
<tr>
<td>0.8%</td>
<td>0.477</td>
<td>0.007</td>
<td>--</td>
<td>3000.002</td>
<td>500</td>
<td>sweepz_.ACC_COPY@li.150</td>
</tr>
<tr>
<td>0.4%</td>
<td>0.250</td>
<td>0.016</td>
<td>--</td>
<td>1503.174</td>
<td>500</td>
<td>vhone_.ACC_COPY@li.251</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.005</td>
<td>0.005</td>
<td>6.012</td>
<td>--</td>
<td>1</td>
<td>vhone_.ACC_COPY@li.205</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.001</td>
<td>0.000</td>
<td>--</td>
<td>6.012</td>
<td>1</td>
<td>vhone_.ACC_COPY@li.283</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.001</td>
<td>0.000</td>
<td>--</td>
<td>5.000</td>
<td>1</td>
<td>vhone_.ACC_COPY@li.266</td>
</tr>
</tbody>
</table>
The Approach

- Formulate a target problem that does real science and is same work/node as all MPI
- Identify hotspots
  - Using Craypat –hprofile_generate to identify loop structure
- Convert all- MPI into Hybrid, using high level OpenMP
  - This is important for all future multi-petaflop systems
- Move to accelerator using OpenACC
  - Insert OpenACC on major OpenMP loops
    - Examine Compiler feedback on data movement
- Introduce data regions outside the timestep loop
  - Now have to find all the code that accesses the arrays you want to reside on the accelerator
  - Looking at the PIN tool to help with that complex analysis. Especially when using derived types and C++ dynamic arrays
- Optimize the kernels
A common directive programming model for today’s GPUs

- Announced at SC11 conference
- Offers portability between compilers
  - Drawn up by: NVIDIA, Cray, PGI, CAPS
  - Multiple compilers offer portability, debugging, permanence
- Works for Fortran, C, C++
  - Standard available at www.OpenACC-standard.org
  - Initially implementations targeted at NVIDIA GPUs

- Current version: 1.0 (November 2011)

- Compiler support:
  - Cray CCE
  - PGI Accelerator
  - CAPS
### Creating arrays on Accelerator using runtime calls

```c
#define _OPENACC

fdsim_ac3dvti *sim_fwd;
sim_fwd = (fdsim_ac3dvti*)(pd.sim_fwd);
void *data;
int size;
for ( int isub=0; isub < pd.sim_fwd->nsubdom; isub++ ) {
    if(sim_fwd->pml_coef[isub] != 0){
        for(int i=0 ; i<12 ; i++){
            if(i>=0 && i<=3)size = sim_fwd->txx[isub]->ax1->ntot;
            if(i>=4 && i<=7)size = sim_fwd->txx[isub]->ax2->ntot;
            if(i>=8 && i<=11)size = sim_fwd->txx[isub]->ax3->ntot;
            data = sim_fwd->pml_coef[isub][i];
            cray_acc_create(data,size*sizeof(float));
        }
    }
}
```
#ifdef _OPENACCX
#include "openacc.h"

void fdsim_ac3dvti::acc_updatein_arrays()
{
    acc_update_arrays(1);
}

void fdsim_ac3dvti::acc_updateout_arrays()
{
    acc_update_arrays(2);
}

void fdsim_ac3dvti::acc_update_anarray(int inorout, void* data, size_t size)
{
    if(inorout==1)
    {
        cray_acc_copyin(data,size);
    }else{
        cray_acc_copyout(data,size);
    }
}
What does OpenACC look like

\[
\begin{align*}
! & X - \text{direction communication} - (+) \text{ side} \\
& \text{reqcount} = 0 \\
& \text{do } i = 1, \text{ derivcount} \\
& \quad \text{if( deriv\_x\_list(i)\%pos\_nbr } \geq 0 \ \text{and.} \ \& \\
& \quad \quad \text{deriv\_x\_list(i)\%inuse ) then} \\
& \quad \text{reqcount} = \text{reqcount} + 1 \\
& \quad \text{req(reqcount) = deriv\_x\_list(i)\%req(3)} \\
& \quad \text{endif} \\
& \text{enddo} \\
& \text{if( reqcount } > 0 \ ) \text{ then} \\
& \quad \text{!write(*,'(1i4,1a,1i4)') myid, 'x pos waiting on ', reqcount} \\
& \quad \text{call MPI\_WAITALL( reqcount, req, stat, ierr )} \\
& \text{#ifdef GPU\_ASYNC} \\
& \quad !$\text{acc update device(pos\_f\_x\_buf(:,;,:,1:reqcount)) async(1)} \\
& \text{#endif} \\
& \text{endif}
\end{align*}
\]
Using GPU Direct

if(lnbr(1)>=0) then
    ! get ghost cells from neighbor on (-x) side
    #ifdef GPU_DIRECT
    !$acc host_data use_device(neg_f_x_buf)
    #endif
    call MPI_IRecv(neg_f_x_buf(1,1,1,idx),(my*mz*iorder/2),&
    MPI_REAL8,deriv_x_list(idx)%neg_nbr,idx,&
    gcomm,deriv_x_list(idx)%req(1),ierr)
    #ifdef GPU_DIRECT
    !$acc end host_data
    #endif
endif
if(lnbr(2)>=0) then
    ! get ghost cells from neighbor on (+x) side
    #ifdef GPU_DIRECT
    !$acc host_data use_device(pos_f_x_buf)
    #endif
    call MPI_IRecv(pos_f_x_buf(1,1,1,idx),(my*mz*iorder/2),&
    MPI_REAL8,deriv_x_list(idx)%pos_nbr,idx+deriv_list_size,&
    gcomm,deriv_x_list(idx)%req(3),ierr)
    #ifdef GPU_DIRECT
    !$acc end host_data
    #endif
endif

Do not have to update host with data prior to communication
Using GPU Direct

if ( deriv_x_list(idx)%packed ) then

! assume pos_fx_x_buffer and neg_fs_x_buffer have been filled somewhere
! else
if(deriv_x_list(idx)%neg_nbr>=0) then
! send ghost cells to neighbor on (-x) side
#endif GPU_DIRECTS
!$acc host_data use_device(pos_fs_x_buf)
#endif

call MPI_I Send(pos_fs_x_buf(1,1,1,idx),(my*mz*iorder/2), &
MPI_REAL8,deriv_x_list(idx)%neg_nbr,idx+deriv_list_size,&
gcomm, deriv_x_list(idx)%req(2), ierr)
#endif GPU_DIRECTS
!$acc end host_data
#endif
endif

if(deriv_x_list(idx)%pos_nbr>=0) then
! send ghost cells to neighbor on (+x) side
nm = mx + 1 - iorder/2
#endif GPU_DIRECTS
!$acc host_data use_device(pos_fs_x_buf)
#endif

call MPI_I Send(neg_fs_x_buf(1,1,1,idx),(my*mz*iorder/2), &
MPI_REAL8, deriv_x_list(idx)%pos_nbr,idx, &
gcomm, deriv_x_list(idx)%req(4), ierr)
#endif GPU_DIRECTS
!$acc end host_data
#endif
endif

do not have to update host with data prior to communication
And the principal good thing about S3D
A benchmark problem was defined to closely resemble the target simulation

- 52 species n-heptane chemistry and $48^3$ grid points per node

- $48^3 \times 18,500$ nodes = 2 billion grid points
- Target problem would take two months on today’s Jaguar

- Code was benchmarked and profiled on dual-hex core XT5
- Several kernels identified and extracted into stand-alone driver programs
Comparisons of Various systems running S3D

Original Version

Number of Nodes
<table>
<thead>
<tr>
<th>Acc</th>
<th>Acc</th>
<th>Host</th>
<th>Acc Copy</th>
<th>Acc Copy</th>
<th>Events</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time%</td>
<td>Time</td>
<td>Time</td>
<td>In</td>
<td>Out</td>
<td>PE=HIDE</td>
<td>Thread=HIDE</td>
</tr>
<tr>
<td>(MBytes)</td>
<td>(MBytes)</td>
<td></td>
<td></td>
<td></td>
<td>Total</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>105.999</td>
<td>172.083</td>
<td>55959</td>
<td>100021</td>
<td>1472704</td>
<td>Total</td>
</tr>
</tbody>
</table>

- **21.2%** | **22.503** | 0.077 | -- | -- | **600** | **reaction_rate_vec_.ACC_KERNEL@li.167** |
- **3.7%** | **3.891** | 0.035 | -- | -- | **600** | **rhfs_.ACC_KERNEL@li.1756** |
- **3.5%** | **3.672** | 0.020 | -- | -- | **600** | **rhfs_.ACC_KERNEL@li.1820** |
- **3.3%** | **3.451** | 0.021 | -- | -- | **600** | **rhfs_.ACC_KERNEL@li.1875** |
- **2.9%** | **3.083** | 0.009 | -- | **19491** | **100** | **integrate_.ACC_COPY@li.74** |
- **2.8%** | **3.005** | 10.686 | -- | **6497** | **92400** | **derivative_z_pack_np_.ACC_COPY@li.573** |
- **2.8%** | **2.996** | 10.636 | -- | **6497** | **92400** | **derivative_z_pack_np_.ACC_COPY@li.623** |
- **2.6%** | **2.783** | 0.068 | -- | -- | **600** | **rhfs_.ACC_KERNEL@li.439** |
- **2.5%** | **2.605** | 0.019 | -- | -- | **600** | **rhfs_.ACC_KERNEL@li.654** |
- **2.4%** | **2.559** | 0.004 | -- | -- | **100** | **computecoefficients_r_.ACC_KERNEL@li.148** |
- **2.3%** | **2.444** | 0.035 | **13162** | -- | **600** | **rhfs_.ACC_COPY@li.417** |
- **2.3%** | **2.431** | 0.041 | **12909** | -- | **600** | **rhfs_.ACC_COPY@li.1870** |
- **2.3%** | **2.419** | 0.039 | **12909** | -- | **600** | **rhfs_.ACC_COPY@li.1871** |
- **2.3%** | **2.404** | 5.235 | -- | **7341** | **600** | **save_bc_deriv1$rhsf_.ACC_COPY@li.248** |
- **2.1%** | **2.254** | 2.138 | 0.721 | -- | **600** | **rhfs_.ACC_COPY@li.256** |
- **1.9%** | **2.028** | 0.029 | -- | -- | **700** | **calc_primary_vars_.ACC_KERNEL@li.42** |
- **1.8%** | **1.949** | 0.004 | -- | -- | **100** | **computecoefficients_r_.ACC_KERNEL@li.234** |
- **1.7%** | **1.766** | 10.701 | -- | -- | **92400** | **derivative_z_pack_np_.ACC_KERNEL@li.557** |
- **1.7%** | **1.760** | 0.014 | -- | -- | **600** | **rhfs_.ACC_KERNEL@li.992** |
- **1.6%** | **1.727** | 0.245 | -- | **6497** | **1800** | **derivative_x_pack_np_.ACC_COPY@li.658** |
- **1.6%** | **1.690** | 10.646 | -- | -- | **92400** | **derivative_z_pack_np_.ACC_KERNEL@li.607** |
- **1.4%** | **1.434** | 0.235 | -- | **6497** | **1800** | **derivative_y_pack_np_.ACC_COPY@li.624** |
- **1.3%** | **1.387** | 0.078 | -- | -- | **600** | **rhfs_.ACC_KERNEL@li.488** |
- **1.3%** | **1.328** | 1.433 | **2953** | -- | **700** | **calc_primary_vars_.ACC_COPY@li.42** |
- **1.2%** | **1.247** | 0.223 | -- | **6497** | **1800** | **derivative_x_pack_np_.ACC_COPY@li.613** |
- **1.1%** | **1.137** | 0.019 | -- | -- | **600** | **integrate_.ACC_KERNEL@li.123** |
- **1.0%** | **1.069** | 0.245 | -- | **6497** | **1800** | **derivative_y_pack_np_.ACC_COPY@li.672** |
And that is not all

Titan will be delivered with Nvidia Kepler, which will give us better performance
- Clock Rate
- Memory Bandwidth
- More Registers

GPU direct will be available from MPI with final delivery
This will give us a very good increase in performance
Cuda Proxy allows for multiple MPI tasks to share GPU on the XK system
Cray Technical Workshop on XK6 Programming

On October 9-10, 2012, ORNL and Cray will be hosting a two day XK6 programming workshop. The intent is to bring together users of XK6 systems around the world and share experiences. We would very much like your participation in this workshop. In addition to user talks we will have expert panels covering programming models for the XK6 including OpenACC, Cuda, OpenCL, Cuda Fortran, etc.

Attendance will be limited to 75 people, so please sign up early. The workshop will be held in the JICS auditorium on the ORNL campus and foreign nationals will need to submit visitor requests six weeks prior to the event. Please consider attending and if you know of others who would be interested in either attending or speaking, please forward this email.

Registration is required for this event. Information about travel and accommodations in the Oak Ridge area are provided below. Refreshments (not lunch) will be provided during the event. Additionally, the event will be webcast. Please indicate if you intend to participate in person or via webcast on the registration form.