

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	
Compute Nodes	18,688
Login & I/O Nodes	512
Memory per node	32 GB + 6 GB
# of Fermi chips (2012)	960
# of NVIDIA K20 “Kepler” processor (2013)	14,592
Total System Memory	688 TB
Total System Peak Performance	20+ Petaflops
Cross Section Bandwidths	X=14.4 TB/s Y=11.3 TB/s Z=24.0 TB/s

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



	XK6 (w/ GPU) vs. XK6 (w/o GPU)	XK6 (w/ GPU) vs. XE6	Cray XK6: Fermi GPU plus Interlagos CPU Cray XE6: Dual Interlagos and no GPU
Application	Performance Ratio	Performance Ratio	Comment
S3D	1.5	1.4(now 1.9)	<ul style="list-style-type: none">Turbulent combustion6% of Jaguar workload
Denovo	3.5	3.3	<ul style="list-style-type: none">3D neutron transport for nuclear reactors2% of Jaguar workload
LAMMPS	6.5	3.2	<ul style="list-style-type: none">High-performance molecular dynamics1% of Jaguar workload
WL-LSMS	3.1	1.6	<ul style="list-style-type: none">Statistical mechanics of magnetic materials2% of Jaguar workload2009 Gordon Bell Winner
CAM-SE	2.6	1.5	<ul style="list-style-type: none">Community atmosphere model1% of Jaguar workload

These are all Fermi+ numbers – Cannot show Kepler

The Approach



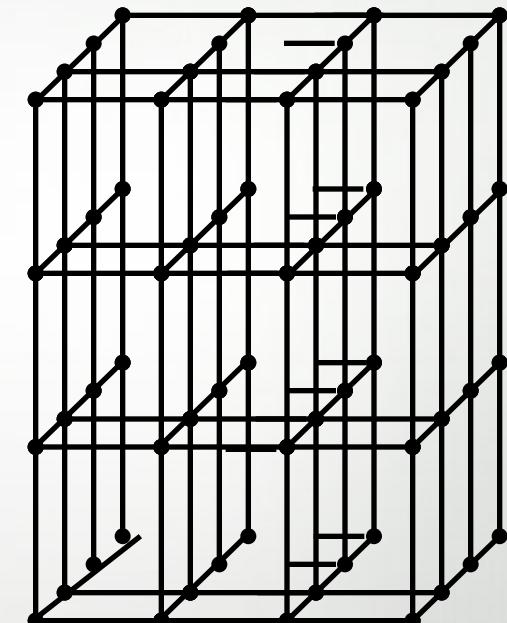


The Approach

- Formulate a target problem that does real science and is same work/node as all MPI

S3D – A DNS solver

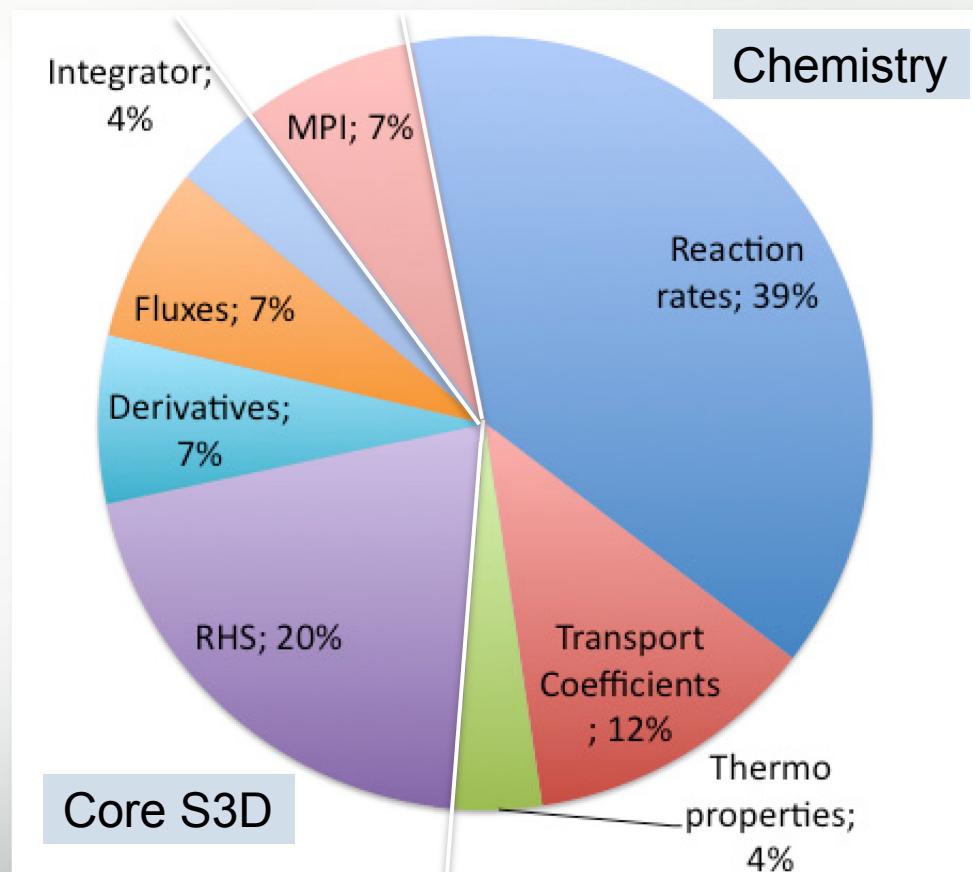
- Structured Cartesian mesh flow solver
- Solves compressible reacting Navier-Stokes, energy and species conservation equations.
 - 8th order explicit finite difference method
 - 4th order Runge-Kutta integrator with error estimator
- Detailed gas-phase thermodynamic, chemistry and molecular transport property evaluations
- Lagrangian particle tracking
- MPI-1 based spatial decomposition and parallelism
- Fortran code. Does not need linear algebra, FFT or solver libraries.



Developed and maintained at CRF, Sandia (Livermore) with BES and ASCR sponsorship. PI – Jacqueline H. Chen (jhchen@sandia.gov)

Benchmark Problem and Profile

- A benchmark problem was defined to closely resemble the target simulation
 - 52 species n-heptane chemistry and 48^3 grid points per node
 - $48^3 * 18,500$ 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



Acceleration Strategy

Team:

Ramanan Sankaran	ORNL
Ray Grout	NREL
John Levesque	Cray

Goals:

- Convert S3D to a hybrid multi-core application suited for a multi-core node with or without an accelerator.
- Be able to perform the computation entirely on the accelerator.
 - Arrays and data able to reside entirely on the accelerator.
 - Data sent from accelerator to host CPU for halo communication, I/O and monitoring only.

Strategy:

- To program using both hand-written and generated code.
 - Hand-written and tuned CUDA*.
 - Automated Fortran and CUDA generation for chemistry kernels
 - Automated code generation through compiler directives
- S3D is now a part of Cray's compiler development test cases

S3D – Weak Scaling Study

- All MPI mesh on a node
 - $15*15*15*32 = 108000$
- One MPI tasks/node
 - $48*48*48 = 110592$
- Two MPI tasks/node
 - $38*38*38*2=109744$

Share GPU with the two MPI tasks

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

Original S3D

		S3D	
Time Step		Solve_Drive	
Time Step	Runge K	Integrate	
Time Step	Runge K	RHS	
Time Step	Runge K	get mass fraction	I,j,k,n_spec loops
Time Step	Runge K	get_velocity	I,j,k,n_spec loops
Time Step	Runge K	calc_inv_avg	I,j,k,n_spec loops
Time Step	Runge K	calc_temp	I,j,k,n_spec loops
Time Step	Runge K	Compute Grads	I,j,k,n_spec loops
Time Step	Runge K	Diffusive Flux	I,j,k,n_spec loops
Time Step	Runge K	Derivatives	I,j,k,n_spec loops
Time Step	Runge K	reaction rates	I,j,k,n_spec loops

Profile from Original S3D

Table 1: Profile by Function Group and Function

Time%	Time	Imb.	Imb.	Calls	Group
		Time	Time%		Function
					PE=HIDE
					Thread=HIDE
100.0%	284.732812	--	--	156348682.1	Total
92.1%	262.380782	--	--	155578796.1	USER
12.4%	35.256420	0.237873	0.7%	391200.0	ratt_i_.LOOPS
9.6%	27.354247	0.186752	0.7%	391200.0	ratx_i_.LOOPS
7.7%	21.911069	1.037701	4.5%	1562500.0	mcedif_.LOOPS
5.4%	15.247551	2.389440	13.6%	35937500.0	mceval4_
5.2%	14.908749	4.123319	21.7%	600.0	rhsf_.LOOPS
4.7%	13.495568	1.229034	8.4%	35937500.0	mceval4_.LOOPS
4.6%	12.985353	0.620839	4.6%	701.0	calc_temp\$thermchem_m_.LOOPS
4.3%	12.274200	0.167054	1.3%	1562500.0	mcavis_new\$transport_m_.LOOPS
4.0%	11.363281	0.606625	5.1%	600.0	computespeciesdiffflux\$transport_m_.LOOPS
2.9%	8.257434	0.743004	8.3%	21921875.0	mixcp\$thermchem_m_
2.9%	8.150646	0.205423	2.5%	100.0	integrate_.LOOPS
2.4%	6.942384	0.078555	1.1%	391200.0	qssa_i_.LOOPS
2.3%	6.430820	0.481475	7.0%	21921875.0	mixcp\$thermchem_m_.LOOPS
2.0%	5.588500	0.343099	5.8%	600.0	computeheatflux\$transport_m_.LOOPS
1.8%	5.252285	0.062576	1.2%	391200.0	rdwdot_i_.LOOPS
1.7%	4.801062	0.723213	13.1%	31800.0	derivative_x_calc_.LOOPS
1.6%	4.4461274	1.310813	22.7%	31800.0	derivative_y_calc_.LOOPS
1.5%	4.327627	1.290121	23.0%	31800.0	derivative_z_calc_.LOOPS
1.4%	3.963951	0.138844	3.4%	701.0	get_mass_frac\$variables_m_.LOOPS

Restructured S3D for multi-core systems

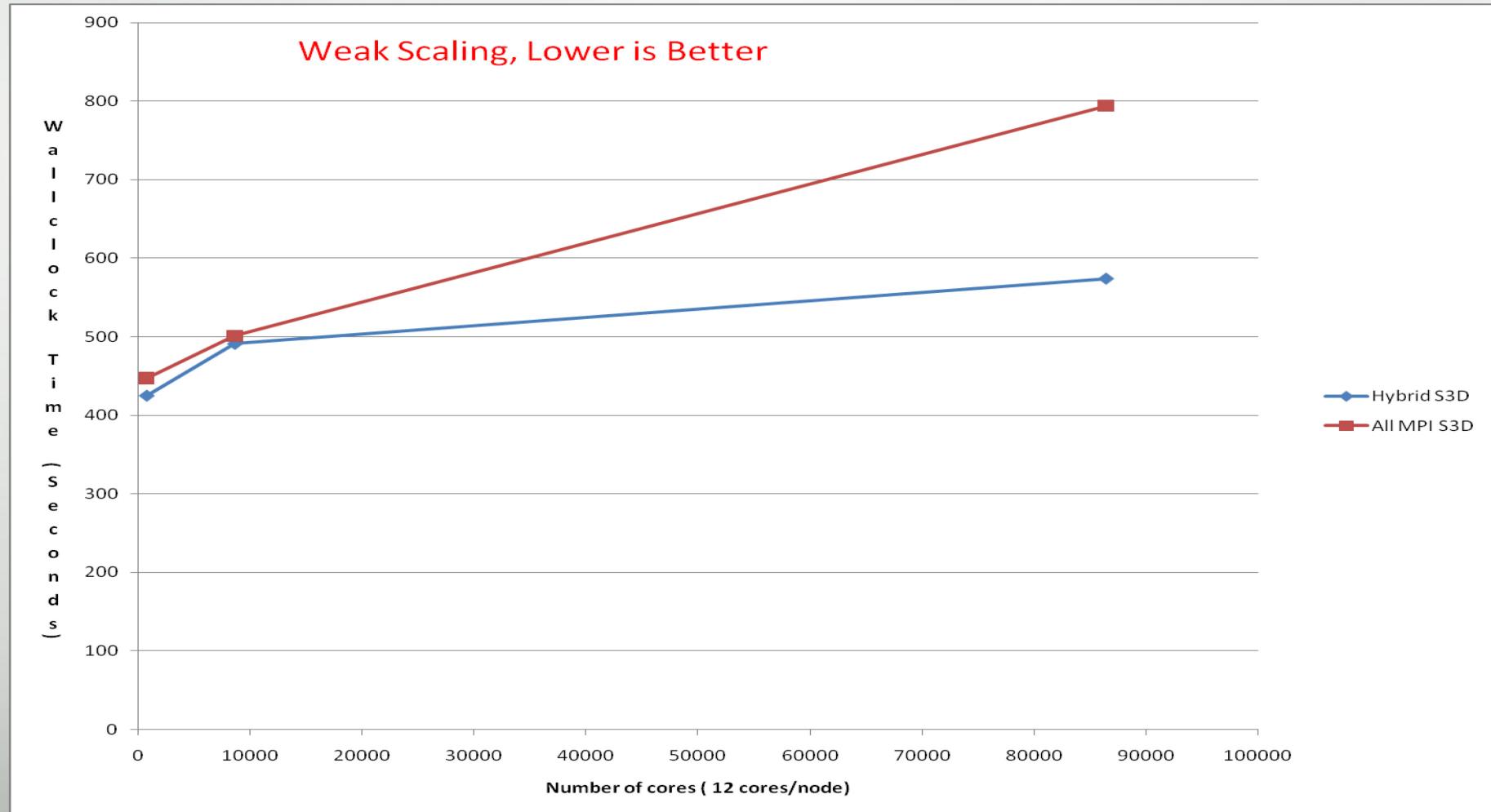
		S3D	
Time Step		Solve_Drive	
Time Step	Runge K	Integrate	
Time Step	Runge K	RHS	
Time Step	Runge K	grid loop -omp	get mass fraction
Time Step	Runge K	grid loop-omp	get_velocity
Time Step	Runge K	grid loop-omp	calc_inv_avg
Time Step	Runge K	grid loop-omp	calc_temp
Time Step	Runge K	grid loop-omp	Compute Grads
Time Step	Runge K	grid loop-omp	Diffusive Flux
Time Step	Runge K	grid loop-omp	Derivatives
Time Step	Runge K	grid loop-omp	reaction rates

Statistics from running S3D

Table 1: Profile by Function Group and Function

Time%	Time	Imb.	Imb.	Calls	Group
		Time	Time%		Function
85.3%	539.077983	--	--	144908.0	USER
<hr/>					
21.7%	136.950871	0.583731	0.5%	600.0	rhsf_
14.7%	93.237279	0.132829	0.2%	600.0	rhsf_.LOOP@li.1084
8.7%	55.047054	0.309278	0.6%	600.0	rhsf_.LOOP@li.1098
6.3%	40.129463	0.265153	0.8%	100.0	integrate_
5.8%	36.647080	0.237180	0.7%	600.0	rhsf_.LOOP@li.1211
5.6%	35.264114	0.091537	0.3%	600.0	rhsf_.LOOP@li.194
3.7%	23.624271	0.054666	0.3%	600.0	rhsf_.LOOP@li.320
2.7%	17.211435	0.095793	0.6%	600.0	rhsf_.LOOP@li.540
2.4%	15.471160	0.358690	2.6%	14400.0	derivative_y_calc_buff_r_.LOOP@li.1784
2.4%	15.113374	1.020242	7.2%	14400.0	derivative_z_calc_buff_r_.LOOP@li.1822
2.3%	14.335142	0.144579	1.1%	14400.0	derivative_x_calc_buff_r_.LOOP@li.1794
1.9%	11.794965	0.073742	0.7%	600.0	integrate_.LOOP@li.96
1.7%	10.747430	0.063508	0.7%	600.0	computespeciesdiffflux2\$transport_m_.LOOP
1.5%	9.733830	0.096476	1.1%	600.0	rhsf_.LOOP@li.247
1.2%	7.649953	0.043920	0.7%	600.0	rhsf_.LOOP@li.274
0.8%	5.116578	0.008031	0.2%	600.0	rhsf_.LOOP@li.398
0.6%	3.966540	0.089513	2.5%	1.0	s3d_
0.3%	2.027255	0.017375	1.0%	100.0	integrate_.LOOP@li.73
0.2%	1.318550	0.001374	0.1%	600.0	rhsf_.LOOP@li.376
0.2%	0.986124	0.017854	2.0%	600.0	rhsf_.REGION@li.1096
0.1%	0.700156	0.027669	4.3%	1.0	exit

Resultant Hybrid S3D Performance



Advantage of raising loops

- Create good granularity OpenMP Loop
- Improves cache re-use
- Reduces Memory usage significantly
- Creates a good potential kernel for an accelerator

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

		S3D	
Time Step – acc_data		Solve_Drive	
Time Step– acc_data	Runge K	Integrate	
Time Step– acc_data	Runge K	RHS	
Time Step– acc_data	Runge K	grid loop -ACC	get mass fraction
Time Step– acc_data	Runge K	grid loop-ACC	get_velocity
Time Step– acc_data	Runge K	grid loop-ACC	calc_inv_avg
Time Step– acc_data	Runge K	grid loop-ACC	calc_temp
Time Step– acc_data	Runge K	grid loop-ACC	Compute Grads
Time Step– acc_data	Runge K	grid loop-ACC	Diffusive Flux
Time Step– acc_data	Runge K	grid loop-ACC	Derivatives
Time Step– acc_data	Runge K	grid loop-ACC	reaction rates

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

Technique for using DATA and PRESENT

Solve_driver

!\$acc data copying

Timestep Loop

Integrate

!\$acc data present

Runge Kutta Loop

Rhsf

!\$acc data present

Call to major
Computations

!\$acc end data

End of Runge Kutta Loop

Computations in Integrate

!\$acc end data

End of Timestep loop

!\$acc end data

End of Solve_driver



What does OpenACC look like (Solve_driver

```
#ifdef GPU
!$acc data copyin(avmolwt, cpCoef_aa, cpCoef_bb, cpmix, enthCoef_aa, enthCoef_bb, &
!$acc& gamma, invEnthInc, iorder, lrmcwrt, mixMW, molwt_c, molwt, n_spec, neighbor, nsc, pressure, &
!$acc& neg_f_x_buf, neg_f_y_buf, neg_f_z_buf, pos_f_x_buf, pos_f_y_buf, pos_f_z_buf, &
!$acc& neg_fs_x_buf, neg_fs_y_buf, neg_fs_z_buf, pos_fs_x_buf, pos_fs_y_buf, pos_fs_z_buf, &
!$acc& rk_alpha, rk_beta, rk_err, rmcwrk, Ru, temp, temp_hibound, temp_lobound, tstep, &
!$acc& u, vary_in_x, vary_in_y, vary_in_z, volum, yspecies, q, q_err)
#endif
```



What does OpenACC look like (Integrate_erk)

```
#ifdef GPU
 !$acc data present_or_create( avmolwt, cpccoef_aa, cpccoef_bb, cpmix, enthcoef_aa, enthcoef_bb, &
 !$acc&   gamma, invEnthInc, lrmcwrt, molwt_c, molwt, n_spec, pressure, q, neighbor, nsc, &
 !$acc&   rhs, rmcwrk, Ru, temp, temp_hibound, temp_lobound, u, vary_in_x, vary_in_y, &
 !$acc&   vary_in_z, volum, yspecies, ds_mxvg, diffflux, tmmp2n, sumf1, sumf2, &
 !$acc&   diffusion, grad_mixmw, grad_t, grad_u, grad_ys, h_spec, lambdax, &
 !$acc&   rr_r, rs_therm_diff, tmmp, tmmp2, tmmpdx, volttmp, vscsty, &
 !$acc&   neg_fs_x_buf, neg_fs_y_buf, neg_fs_z_buf, pos_fs_x_buf, pos_fs_y_buf, pos_fs_z_buf, &
 !$acc$   buffer41, buffer42, buffer43, buffer44, buffer45, &
 !$acc&   buffer31, buffer32, buffer33, buffer34, buffer35, buffer36, buffer37, &
 !$acc&   neg_f_x_buf, neg_f_y_buf, neg_f_z_buf, pos_f_x_buf, pos_f_y_buf, pos_f_z_buf, mixmw)
#endif
```

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

What does OpenACC look like

```
#ifdef GPU
!$acc parallel loop gang private(i,ml,mu)
#else
!$omp parallel private(i, ml, mu)
!$omp do
#endif
do i = 1, nx*ny*nz, ms
  ml = i
  mu = min(i+ms-1, nx*ny*nz)
  call calc_gamma_r( gamma, cpmix, avmolwt, ml, mu)
  call calc_press_r( pressure, q(1,1,1,4), temp, avmolwt, ml, mu )
  call calc_specEnth_allpts_r(temp, h_spec, ml, mu)
end do
#endif GPU
!$acc end parallel loop
#else
!$omp end parallel
#endif
```



What OpenACC looks like

```
#ifdef GPU
!$acc parallel loop gang collapse(2) private(n,i,j,k)
#else
 !$omp parallel do private(n,i,j,k)
#endiff
 do n=1,n_spec
   do k = 1,nz
 #ifdef GPU
 !$acc loop vector collapse(2)
#endiff
   do j = 1,ny
     do i = 1,nx
       grad_Ys(i,j,k,n,1) = 0.0
       grad_Ys(i,j,k,n,2) = 0.0
       grad_Ys(i,j,k,n,3) = 0.0
       if(i.gt.iorder/2 .and. i.le.nx-iorder/2) then
         grad_Ys(i,j,k,n,1) = scale1x(i)*(aex *( yspecies(i+1,j,k,n)-yspecies(i-1,j,k,n) )      &
 + bex *( yspecies(i+2,j,k,n)-yspecies(i-2,j,k,n) )                                &
 + cex *( yspecies(i+3,j,k,n)-yspecies(i-3,j,k,n) )                                &
 + dex *( yspecies(i+4,j,k,n)-yspecies(i-4,j,k,n) ))
       endif
       if(j.gt.iorder/2 .and. j.le.ny-iorder/2) then
         grad_Ys(i,j,k,n,2) = scalely(j)*(aey *( yspecies(i,j+1,k,n)-yspecies(i,j-1,k,n) )      &
 + bey *( yspecies(i,j+2,k,n)-yspecies(i,j-2,k,n) )                                &
 + cey *( yspecies(i,j+3,k,n)-yspecies(i,j-3,k,n) )                                &
 + dey *( yspecies(i,j+4,k,n)-yspecies(i,j-4,k,n) ))
       endif
       if(k.gt.iorder/2 .and. k.le.nz-iorder/2) then
         grad_Ys(i,j,k,n,3) = scalelz(k)*(aez *( yspecies(i,j,k+1,n)-yspecies(i,j,k-1,n) ) &
 + bez *( yspecies(i,j,k+2,n)-yspecies(i,j,k-2,n) ) &
 + cez *( yspecies(i,j,k+3,n)-yspecies(i,j,k-3,n) ) &
 + dez *( yspecies(i,j,k+4,n)-yspecies(i,j,k-4,n) ))
       endif
     end do ! i
   end do ! j
#endiff
 ifdef GPU
 !$acc end loop
#endiff
 end do ! k
 end do ! n
#endiff
 ifdef GPU
 !$acc end parallel loop
#endiff
```

What does OpenACC look like

```
#ifdef GPU
 !$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)

#ifndef GPU
 !$acc end host_data
#endif
#endif
if(lnbr(2)>=0) then
    ! get ghost cells from neighbor on (+x) side
#ifndef GPU
 !$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)

#ifndef GPU
 !$acc end host_data
#endif
#endif
```

What does OpenACC look like

```
#ifdef GPU
#ifndef GPU_STREAMS
 !$acc update host(pos_fs_x_buf(:, :, :, idx)) async(istr)
#else
 !$acc host_data use_device(pos_fs_x_buf)
#endif
#endif
#ifndef GPU_STREAMS
    call cray_mpif_isend_openacc(c_loc(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,istr,deriv_x_list(idx)%req(2),ierr)
#else
    call MPI_ISend(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
#endif
#ifndef GPU
#ifndef GPU_STREAMS
 !$acc end host_data
#endif
#endif
```

Most Recent Profile from Host's view

Table 1: Profile by Function Group and Function

Time%	Time	Imb.	Imb.	Calls	Group
		Time	Time%		Function
					PE=HIDE
					Thread=HIDE
100.0%	161.015945	--	--	3634780.0	Total

92.9%	149.509741	--	--	2483332.9	USER

13.9%	22.380861	0.002732	0.0%	600.0	reaction_rate_vec_.ACC_SYNC_WAIT@li.5008
6.3%	10.165194	0.051670	0.6%	600.0	rhsf_.ACC_SYNC_WAIT@li.2054
5.9%	9.539418	0.068807	0.8%	600.0	rhsf_.ACC_DATA_REGION@li.256
4.1%	6.663584	0.007798	0.1%	600.0	integrate_.ACC_SYNC_WAIT@li.122
3.7%	5.963459	0.007257	0.1%	600.0	rhsf_.ACC_SYNC_WAIT@li.2857
3.3%	5.305605	0.009377	0.2%	600.0	save_bc_deriv1\$rhsf_.ACC_COPY@li.248
2.0%	3.215194	0.008362	0.3%	100.0	integrate_.ACC_SYNC_WAIT@li.74
1.9%	3.026219	0.033275	1.2%	700.0	calc_primary_vars_.ACC_COPY@li.153
1.8%	2.939701	0.041773	1.6%	560.0	filter\$filter_m_
1.6%	2.554719	0.001722	0.1%	100.0	computecoefficients_r_.ACC_SYNC_WAIT@li.231
1.3%	2.115169	0.005334	0.3%	600.0	rhsf_.ACC_COPY@li.256
1.3%	2.074830	0.0000994	0.1%	100.0	computecoefficients_r_.ACC_COPY@li.1745
1.2%	1.8555853	0.036900	2.2%	100.0	controller\$rk_m_
1.0%	1.633700	0.008128	0.6%	600.0	rhsf_.LOOP@li.2325
1.0%	1.596338	0.022872	1.6%	600.0	rhsf_.LOOP@li.2255
0.9%	1.506629	0.004267	0.3%	100800.0	deriv_inplane_1_
0.9%	1.503893	0.001494	0.1%	700.0	impose_hard_bc_.ACC_SYNC_WAIT@li.192

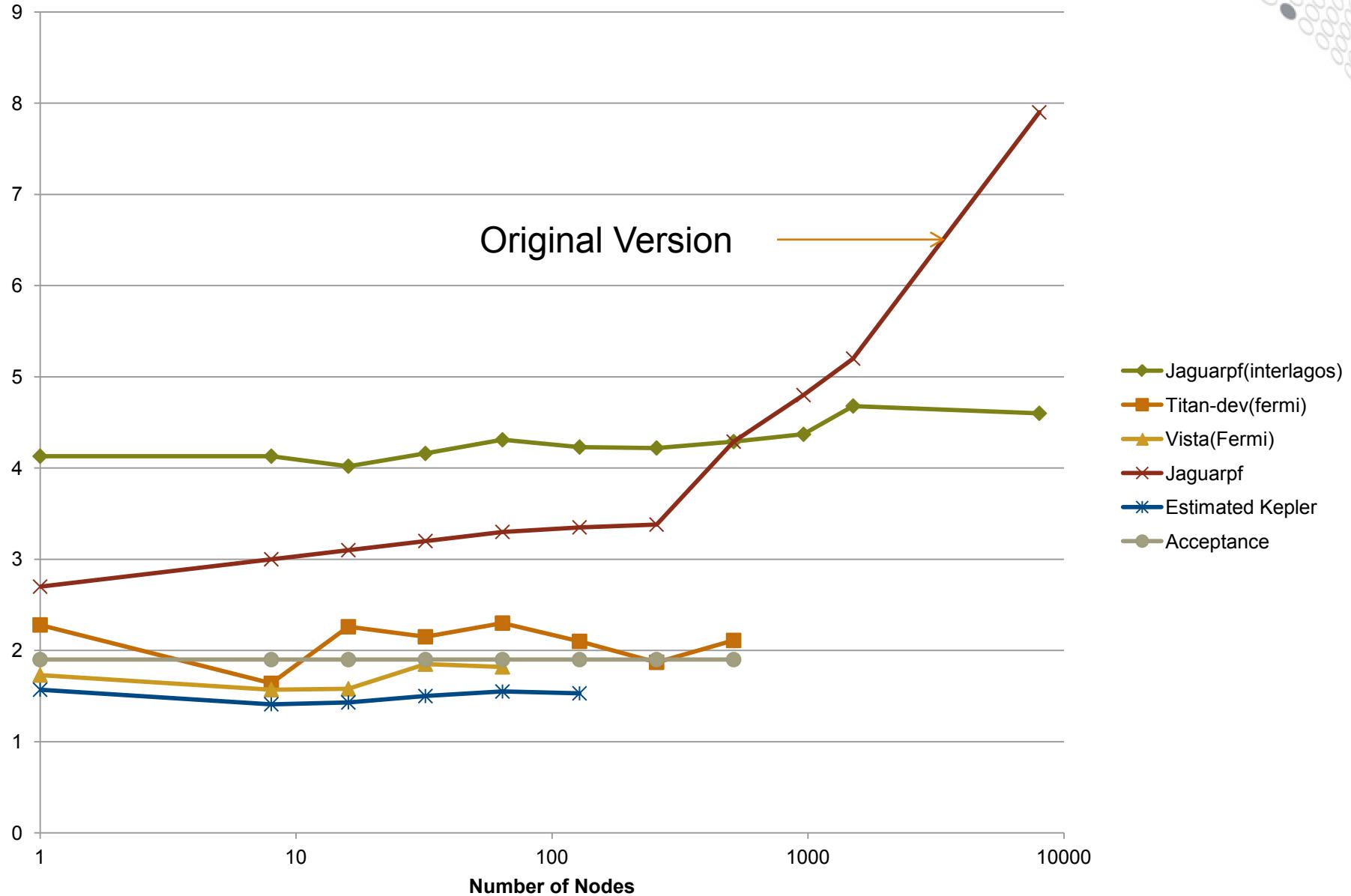
Most Recent Profile from Accelerator's view

Table 1: Time and Bytes Transferred for Accelerator Regions

Acc Time%	Acc Time	Host Time	Acc Copy In (MBytes)	Acc Copy Out (MBytes)	Events	Function
PE=HIDE Thread=HIDE						
100.0%	88.013	130.413	14109	77239	842104	Total

25.3%	22.308	0.072	--	--	600	reaction_rate_vec_.ACC_KERNEL@li.167
4.4%	3.881	0.067	--	--	600	rhsf_.ACC_KERNEL@li.1776
4.2%	3.679	0.067	--	--	600	rhsf_.ACC_KERNEL@li.1840
3.8%	3.354	0.018	--	--	600	rhsf_.ACC_KERNEL@li.1891
3.5%	3.069	0.010	--	19491	100	integrate_.ACC_COPY@li.74
3.1%	2.754	0.065	--	--	600	rhsf_.ACC_KERNEL@li.441
2.9%	2.564	0.021	--	--	600	rhsf_.ACC_KERNEL@li.656
2.9%	2.552	0.004	--	--	100	computecoefficients_r_.ACC_KERNEL@li.148
2.7%	2.401	5.306	--	7341	600	save_bc_deriv1\$rhsf_.ACC_COPY@li.248
2.4%	2.112	2.115	0.721	--	600	rhsf_.ACC_COPY@li.256
2.3%	2.066	0.003	--	--	100	computecoefficients_r_.ACC_KERNEL@li.234
2.3%	2.059	0.029	--	--	700	calc_primary_vars_.ACC_KERNEL@li.42
2.0%	1.755	0.067	--	--	600	rhsf_.ACC_KERNEL@li.994
1.7%	1.500	0.229	--	3038	1800	derivative_z_send_np_.ACC_COPY@li.908
1.6%	1.397	1.399	2953	--	700	calc_primary_vars_.ACC_COPY@li.42
1.6%	1.386	0.066	--	--	600	rhsf_.ACC_KERNEL@li.490
1.4%	1.257	0.210	--	3038	1800	derivative_z_send_np_.ACC_COPY@li.877
1.3%	1.139	0.017	--	--	600	integrate_.ACC_KERNEL@li.123
1.2%	1.051	0.221	--	2700	1800	derivative_y_send_np_.ACC_COPY@li.947
1.1%	0.988	3.026	--	2953	700	calc_primary_vars_.ACC_COPY@li.153
1.0%	0.859	0.775	--	2953	600	rhsf_.ACC_COPY@li.2340
1.0%	0.856	0.772	--	2953	600	rhsf_.ACC_COPY@li.2270
1.0%	0.856	0.966	--	2953	600	rhsf_.ACC_COPY@li.2212
1.0%	0.851	0.907	--	--	600	reaction_rate_vec_.ACC_COPY@li.167
0.9%	0.834	0.003	4725	--	100	integrate_.ACC_COPY@li.172

Comparisons of Various systems running S3D



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