

An Effective Standard for Developing Performance Portable Applications for Future Hybrid Systems

Supercomputing 2012
November 12, 2012

John Levesque
Director/CTO Office



First a confession

I have never written an application from scratch, everything I have done starts with existing applications which I restructure to run faster.

One might say that I am one of the reasons some legacy applications are still around.

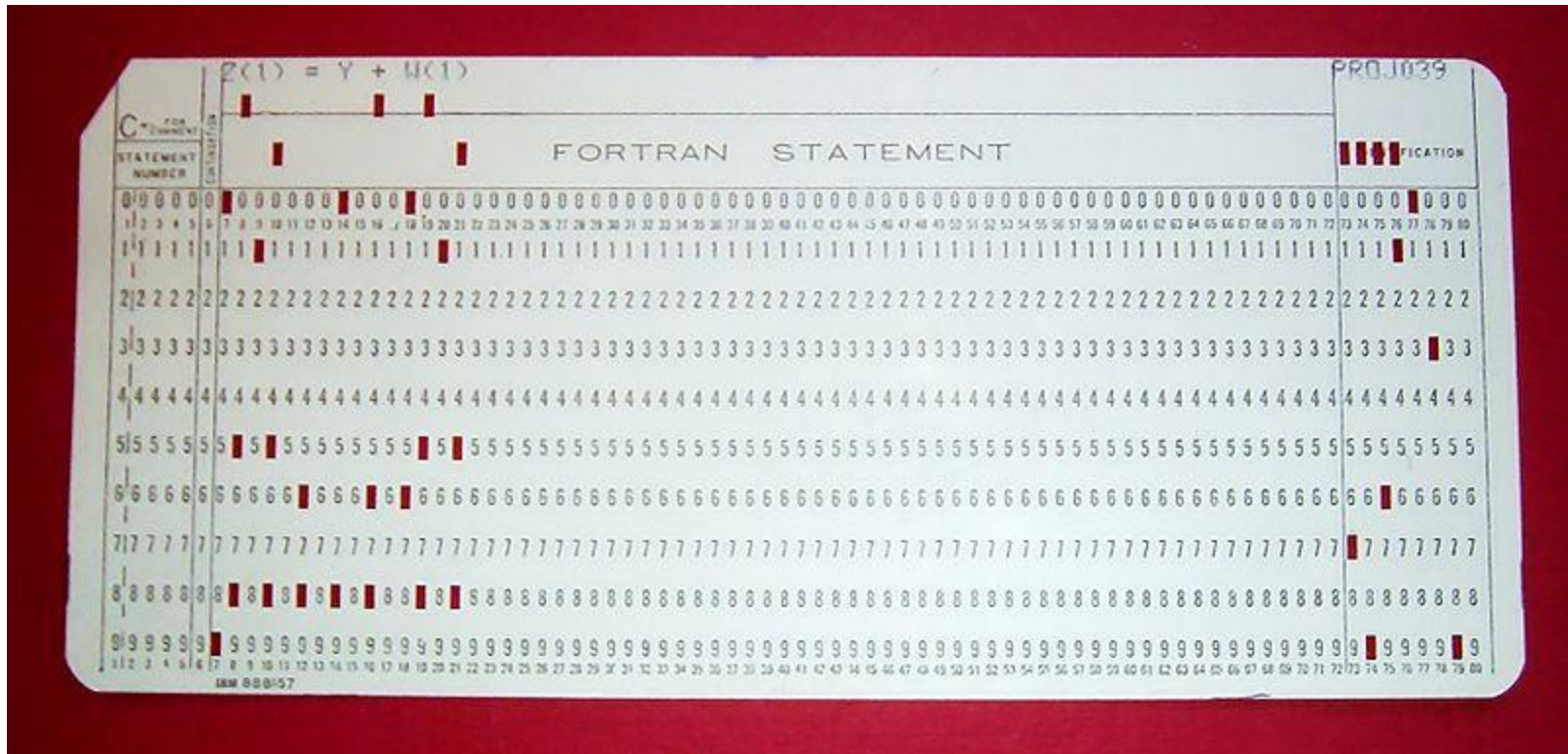
What Supercomputer Was this?



What are these?



Why was the Fortran line length 72?



Computational Steering in the 70s

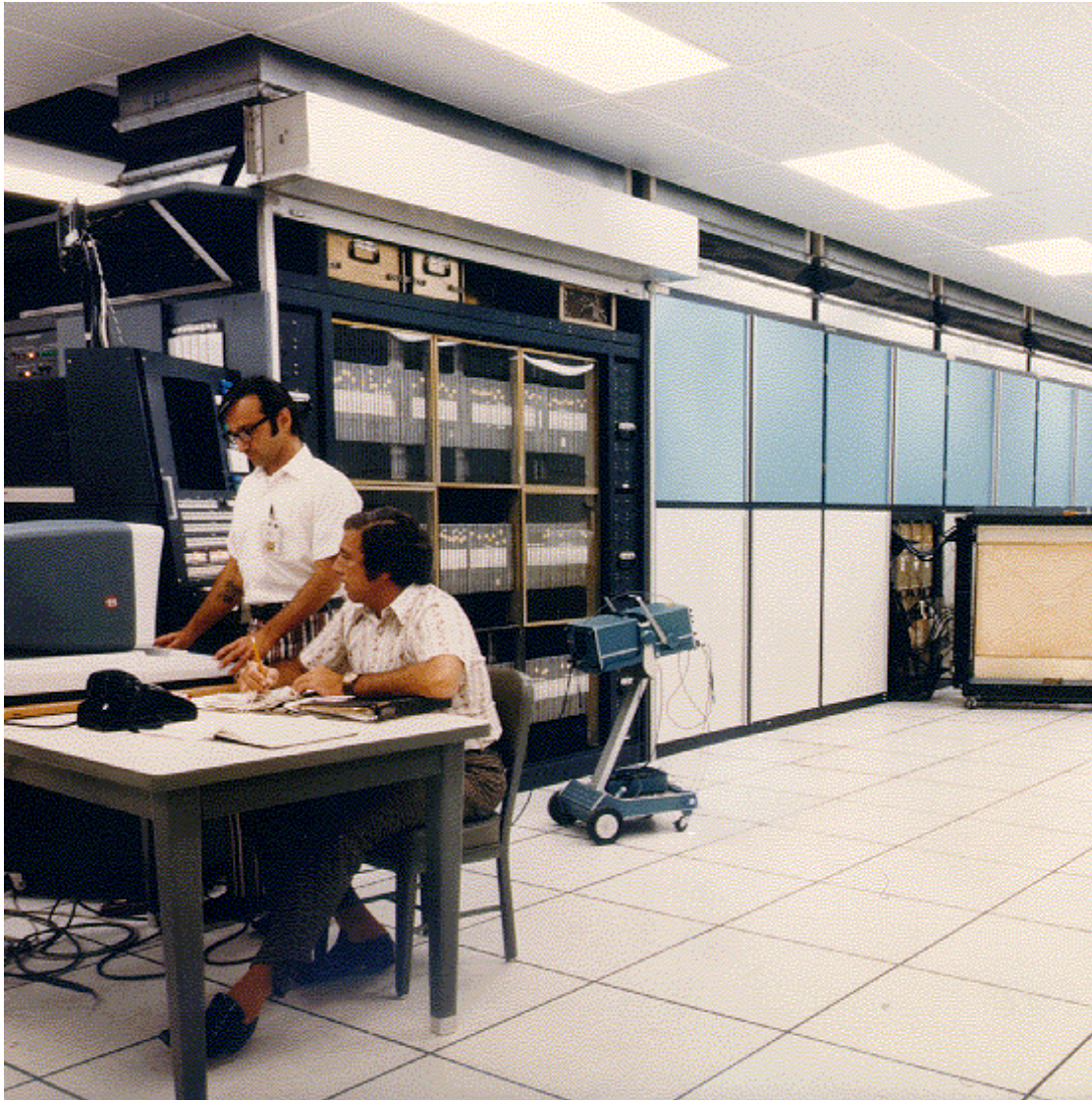


Set a sense switch, dump the velocity
Vector field to tape, take tape to Calcomp
Plotter.



What Supercomputer is This?

Anyone know why the door is open?



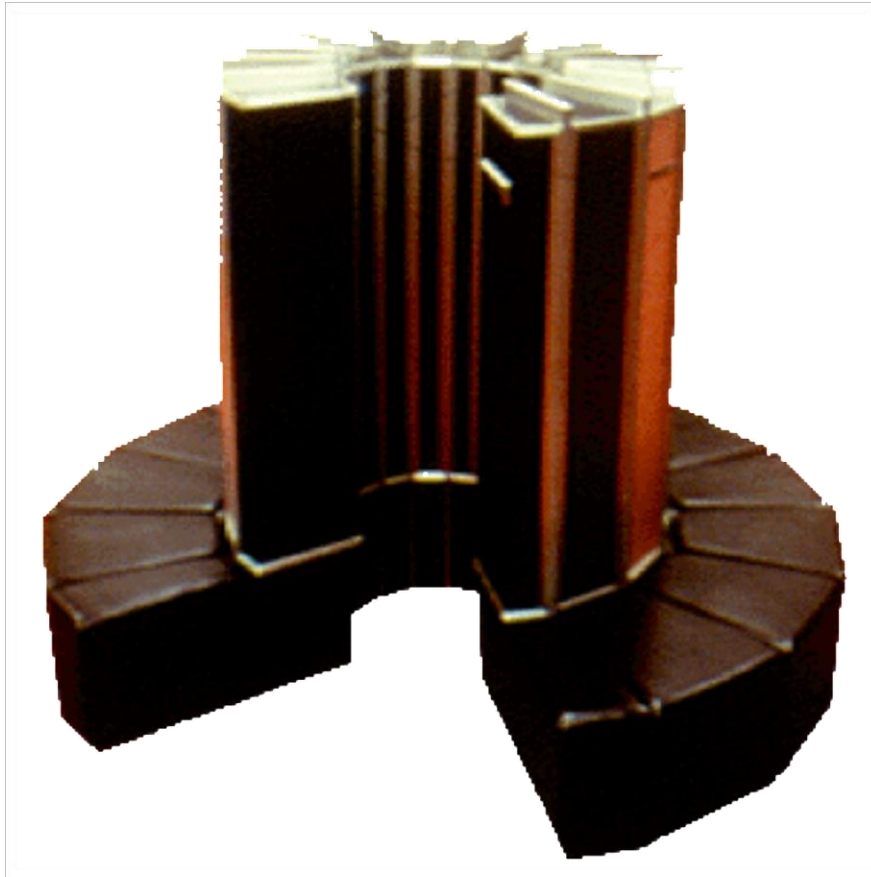
Who Invented the Internet?



Another Seymour Masterpiece



How much memory
did this system have?



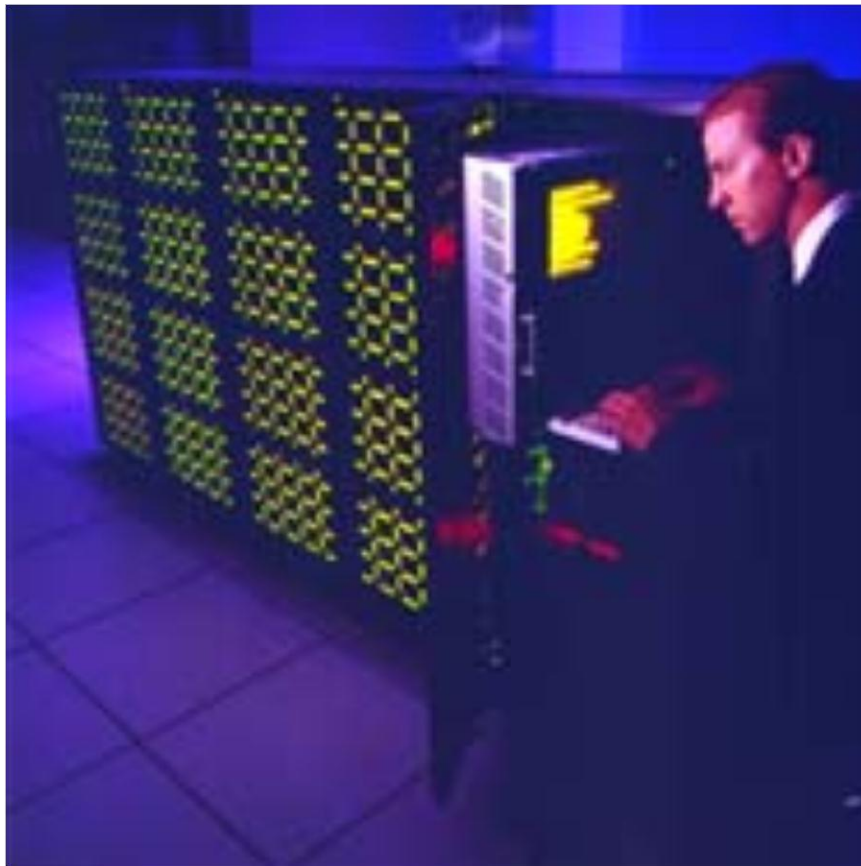
What Supercomputer is this?

My first laptop



Door Prize for anyone guessing what this is.





What Supercomputer is this?

During this decade,

More money was spent on Disposal
Diapers than on Supercomputer



**What Supercomputer
is this?**

Famous Jurassic Park Prop



The best Co-Array machine



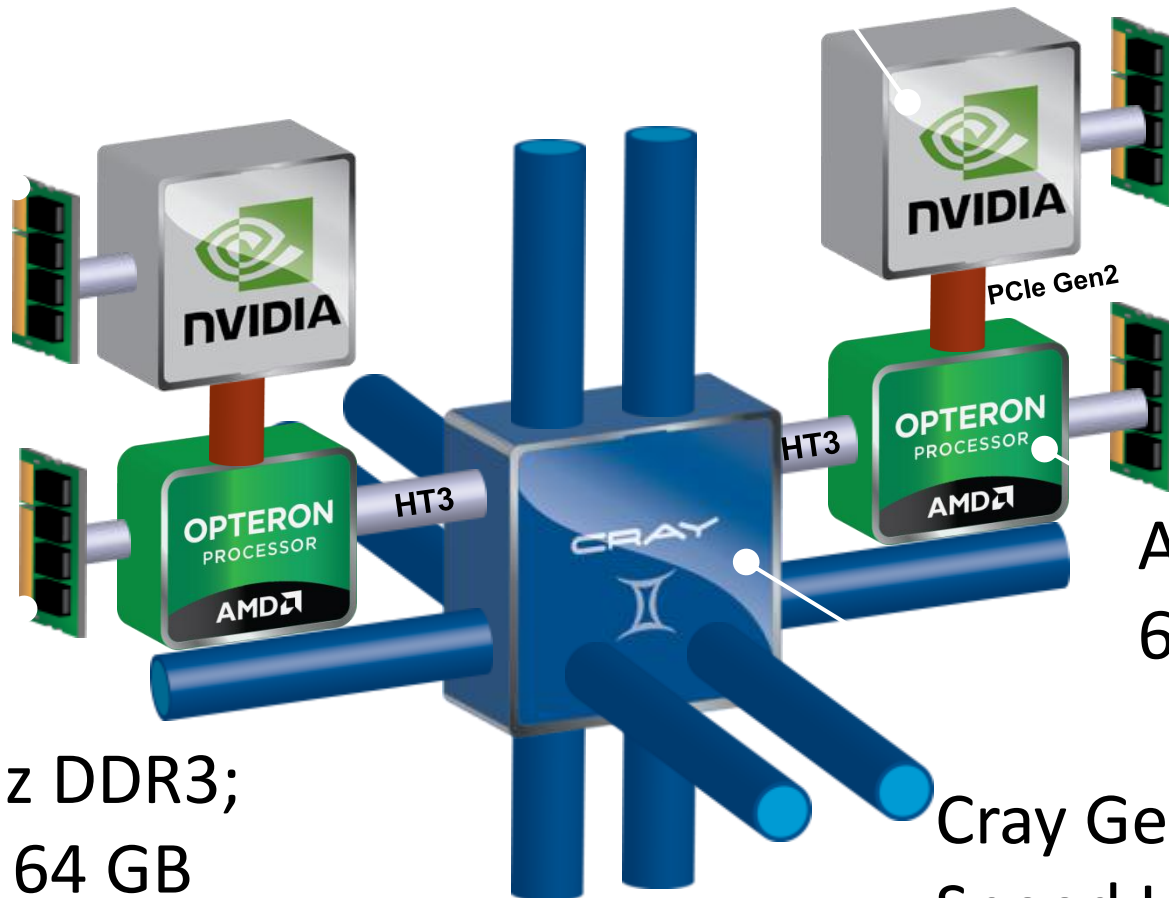
The System that shot down a satellite



Our Current Challenge

6GB GDDR5;
138 GB/s

NVIDIA Tesla GPU
with 665GF DPFP



1600 MHz DDR3;
16, 32 or 64 GB

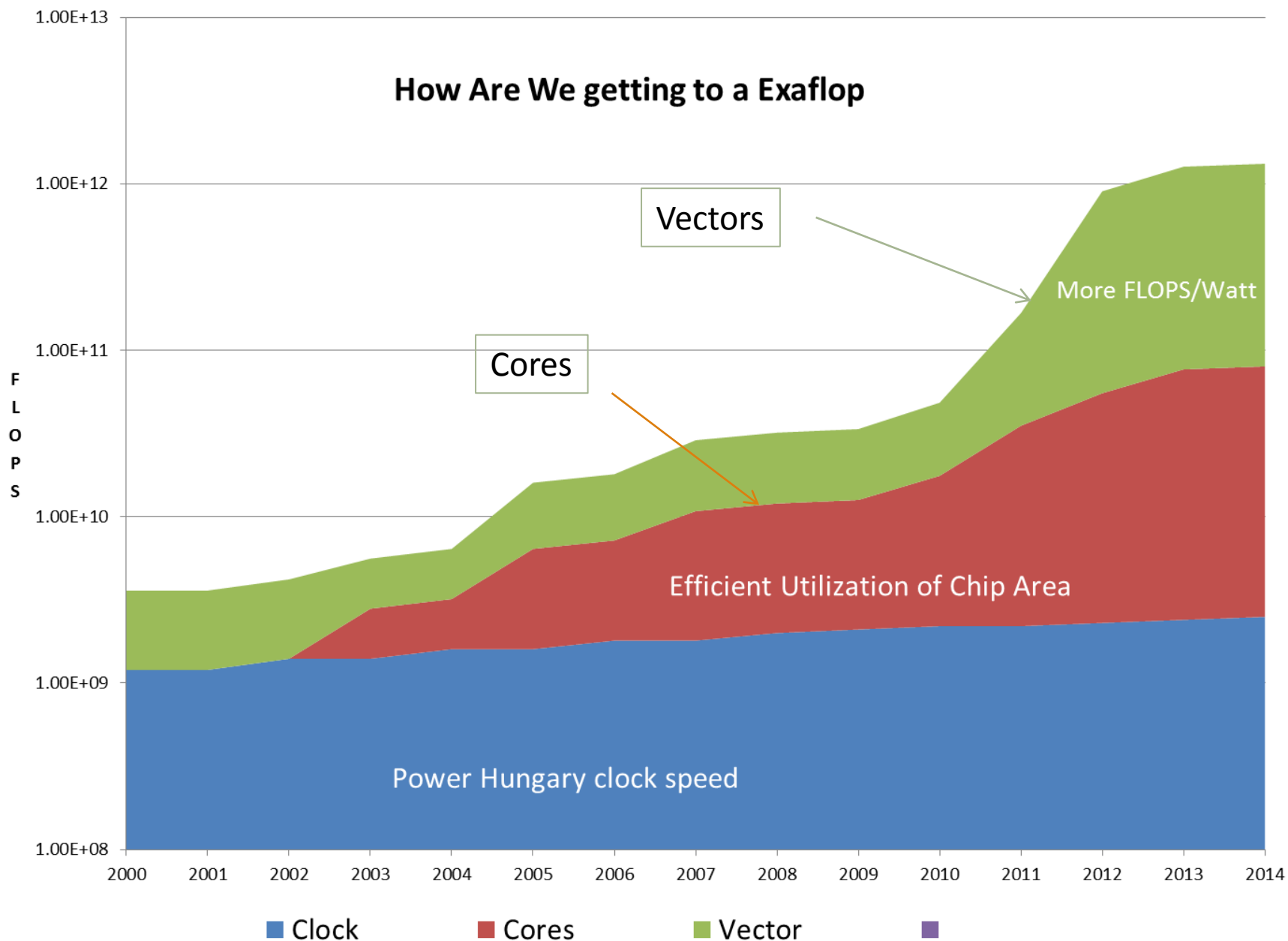
AMD Series
6200 CPU

Cray Gemini High
Speed Interconnect

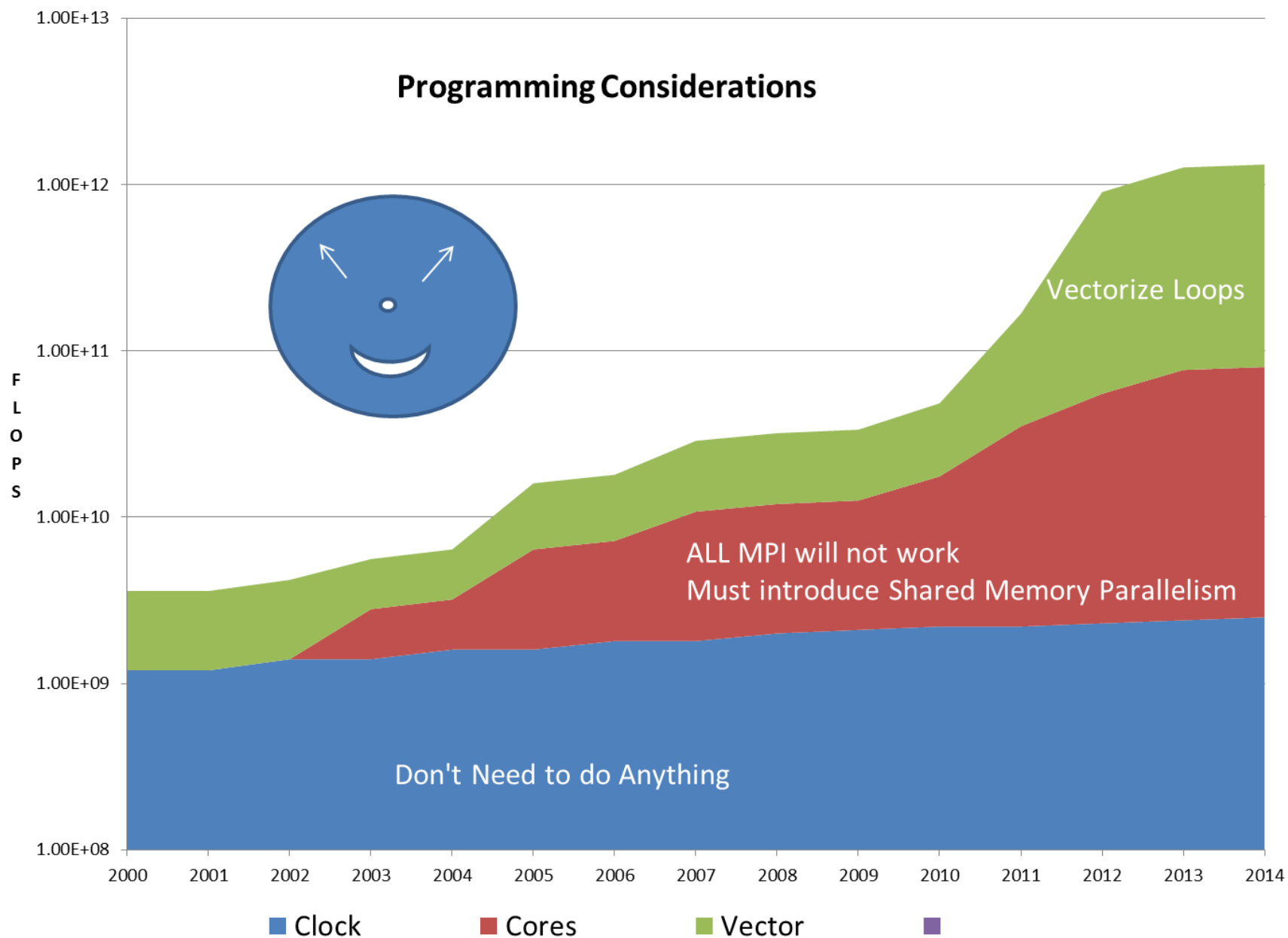
Outline

- Future Architectural Directions
 - Chips are not being designed for HPC
 - Power consumption is a major concern
 - What is heterogeneous Computing?
- Programming implications
 - All MPI is not an option
 - OpenMP and OpenACC

How Are We getting to a Exaflop



Programming Considerations



Potential System Architecture for Exaflop

| Systems | 2010 | 202? | Difference Today & 2018 |
|----------------------------|-----------------------------|--|----------------------------|
| System peak | 2 Pflop/s | 1 Eflop/s | O(1000) |
| Power | 6 MW | ~20 MW | |
| System memory | 0.3 PB | 32 - 64 PB [.03 Bytes/Flop] | O(100) |
| Node performance | 125 GF | 1,2 or 15TF | O(10) – O(100) |
| Node memory BW | 25 GB/s [.20 Bytes/Flop] | 2 - 4TB/s [.002 Bytes/Flop] | O(100) |
| Node concurrency | 12 | O(1k) or 10k | O(100) – O(1000) |
| Total Node Interconnect BW | 3.5 GB/s | 200-400GB/s (1:4 or 1:8 from memory BW) | O(100) |
| System size (nodes) | 18,700 | O(100,000) or O(1M) | O(10) – O(100) |
| Total concurrency | 225,000 | O(billion) [O(10) to O(100) for latency hiding] | O(10,000) |
| Storage | 15 PB | 500-1000 PB (>10x system memory is min) | O(10) – O(100) |
| IO | 0.2 TB | 60 TB/s (how long to drain the machine) | O(100) |
| MTTI | days | O(1 day) | - O(10) |

Future Architectural Directions

- Nodes are becoming much more powerful
 - More processors/node
 - More threads/processor
 - Vector lengths are getting longer
 - Memory hierarchy is becoming more complex
 - Scalar performance is not increasing

Threading on the Node and Vectorization is becoming more important

Today's Multi-Petascale Systems – Node Architecture

| | Cores on the node | Total threading | Vector Length | Programming Model |
|--------------|-------------------|-----------------|---------------|-------------------------|
| Blue Waters | (16) 32 | 32 | 8 (4) | OpenMP/MPI/ Vector |
| Blue Gene Q | 16 | 32 | 8 | OpenMP/MPI/ Vector |
| Magna-Cours | (12) 24 | (12) 24 | 4 | OpenMP/MPI/ Vector |
| Titan (ORNL) | 16 (16) | 16 (768*) | (8) (4) (32) | Threads/Cuda /Vector |
| Intel MIC | >50 | >200 | 16 | OpenMP/MPI/ Vector |
| Power 7 (??) | 16 | 32 | 8 | OpenMP/MPI/ Vector |

* Nvidia allows oversubscription to SIMT units

Vectorization is becoming more important

- ALL accelerated nodes require vectorization at a good size to achieve reasonable performance
 - Nvidia Kepler 32 length
 - Intel MIC >8
- All compilers other than Cray's CCE were designed for marginal vector performance, they do not understand current tradeoffs
 - Be sure to get listing indicating if loop vectorizes
- User refactoring of loop is paramount in gaining good performance on future systems

Memory Hierarchy is becoming more complex

- As processors get faster, memory bandwidth cannot keep up
 - More complex caches
 - Non Uniform Memory Architecture (NUMA) for shared memory on node
 - Operand alignment is becoming more important
- Going forward this will become even more complex – two memories within same address space
 - Fast expensive memory
 - Slow less expensive memory
 - More about this later

Scalar performance is not getting better

- Consider Intel's chips
 - Xeon line with more cores per node using traditional X86 instruction set
 - MIC line with many more cores of slower processors
- Hosted system – Xeon with MIC
 - Native mode – run complete app on the MIC
 - Scalar performance will be an issue
 - Non-vector code will be an issue
 - Off Load mode – use Xeon as host, major computation on MIC
 - Memory transfer to and from Host will be an issue

Scalar Performance is not getting better

- Consider Nvidia approach
- Looking at ARM chip as co-processor
 - Once again scalar is far below state of the art Xeon
- So why not build an Exascale system out of Xeons or Power 7
 - TOO MUCH POWER

Code Design Question?

- Should code designers be concerned with memory management like that required to utilize a hosted accelerator like XK7

YES

- This is not throw away work?

- All systems will soon have a secondary memory that is as large as we require; however, it will not have high bandwidth to the principal compute engine.
- There will be a smaller faster memory that will supply the principal compute engine.
- While system software may manage the two memories for the user, the user will have to manage these desperate memories to achieve maximum performance

So What is Heterogeneous Computing

- I believe it will have more to do with different memories
 - If doing scalar processing, application can afford to access slower larger memory
 - Scalar processing may be significantly slower than state-of-the-art Xeon
 - If doing high speed compute, application must have major computational arrays in fast memory
 - Parallel vector processing need high memory bandwidth and larger caches/registers

So how should we program for these new systems

- **What to avoid**

- Excessive memory movement
 - Memory organization is the most important analysis for moving an application to these systems
- Avoid wide gaps between operands
 - Indirect addressing is okay, if it is localized
- Avoid scalar code
 - Think about Cyber 205, Connection Machine

- **What to do – Good Threading (OpenMP)**

- Must do high level threading
- Thread must access close shared memory rather than distant shared memory
- Load Balancing

- **What to do – Good Vectorization**

- Vectorization advantage allows for introducing overhead to vectorize
 - Vectorization of Ifs
 - Conditional vector merge (too many paths??)
 - Gather/scatter (Too much data motion??)
 - Identification of strings

- **Given the success of OpenMP extensions for accelerators, OpenACC and Intel's OffLoad Directives OpenMP offers an approach to develop a performance portable application that targets ALL future architecture**

Programming for Future Multi-Petaflop and Exaflop Computers

aka

Finding more parallelism in existing applications

Porting an existing application to new Systems



- **Converting to Hybrid OpenMP/MPI**
 - Identifying high level OpenMP loops
 - Using the Cray Scoping tool
 - Using the program library (-hwp)
 - NUMA effects on the XK6 node
 - Comparing Hybrid OpenMP/MPI to all MPI
 - Using the progress engine for overlapping MPI and computation
- **Looking at methods of acceleration**
 - Using Cuda with OpenACC and Cuda Fortran, Visual profiler, command line profiler, libsci being used with OpenACC
- **A systematic approach for converting a Hybrid OpenMP/MPI application to OpenACC**
 - Using OpenACC
 - First, let the compiler do most of the work
 - Using Craypat to identify the most time consuming portions of the accelerated code
 - Optimizing the OpenACC code
 - Most optimizations will improve OpenMP code
 - Employing Cuda and/or Cuda Fortran in an OpenACC application

- Fact
 - For the next decade all HPC system will basically have the same architecture
 - Message passing between nodes
 - Multi-threading within the node – MPI will not do
 - Vectorization at the lower level -
- Fact
 - Current petascale applications are not structured to take advantage of these architectures
 - Current – 80-90% of application use a single level of parallelism, message passing between the cores of the MPP system
 - Looking forward, application developers are faced with a significant task in preparing their applications for the future

Hybridization* of an All MPI Application

* Creation of an application that exhibits three levels of parallelism, MPI between nodes, OpenMP** on the node and vectorized looping structures

** Why OpenMP? To provide performance portability. OpenMP is the only threading construct that a compiler can analyze sufficiently to generate efficient threading on multi-core nodes and to generate efficient code for companion accelerators.

CAUTION!!

- Do not read “Automatic” into this presentation, the Hybridization of an application is difficult and efficient code only comes with a thorough interaction with the compiler to generate the most efficient code and
 - High level OpenMP structures
 - Low level vectorization of major computational areas
- Performance is also dependent upon the location of the data. Best case is that the major computational arrays reside on the accelerator. Otherwise computational intensity of the accelerated kernel must be significant

**Cray's Hybrid Programming Environment
supplies tools for addressing these issues**

Three levels of Parallelism required

- Developers will continue to use MPI between nodes or sockets
- Developers must address using a shared memory programming paradigm on the node
- Developers must vectorize low level looping structures
- While there is a potential acceptance of new languages for addressing all levels directly. Most developers cannot afford this approach until they are assured that the new language will be accepted and the generated code is within a reasonable performance range

Task 1 – Identification of potential accelerator kernels

- Identify high level computational structures that account for a significant amount of time (95-99%)
 - To do this, one must obtain global runtime statistics of the application
 - High level call tree with subroutines and DO loops showing inclusive/exclusive time, min, max, average iteration counts.
- Identify major computational arrays
- **Tools that will be needed**
 - **Advanced instrumentation to measure**
 - DO loop statistics, iteration counts, inclusive time
 - Routine level sampling and profiling

Normal Profile – default Craypat report

Table 1: Profile by Function Group and Function

| Time% | Time | Imb. | Imb. | Calls | Group |
|--------|-----------|----------|-------|-----------|----------------------|
| | | Time | Time% | | Function |
| | | | | | PE=HIDE |
| 100.0% | 50.553984 | -- | -- | 6922023.0 | Total |
| 52.1% | 26.353695 | -- | -- | 6915004.0 | USER |
| 16.9% | 8.540852 | 0.366647 | 4.1% | 2592000.0 | parabola_ |
| 8.0% | 4.034867 | 0.222303 | 5.2% | 288000.0 | remap_ |
| 7.1% | 3.612980 | 0.862830 | 19.3% | 288000.0 | riemann_ |
| 3.7% | 1.859449 | 0.094075 | 4.8% | 288000.0 | ppmlr_ |
| 3.3% | 1.666590 | 0.064095 | 3.7% | 288000.0 | evolve_ |
| 2.6% | 1.315145 | 0.119832 | 8.4% | 576000.0 | paraset_ |
| 1.8% | 0.923711 | 0.048359 | 5.0% | 864000.0 | volume_ |
| 1.8% | 0.890751 | 0.064695 | 6.8% | 288000.0 | states_ |
| 1.4% | 0.719636 | 0.079651 | 10.0% | 288000.0 | flatten_ |
| 1.0% | 0.513454 | 0.019075 | 3.6% | 864000.0 | forces_ |
| 1.0% | 0.508696 | 0.023855 | 4.5% | 500.0 | sweepz_ |
| 1.0% | 0.504152 | 0.027139 | 5.1% | 1000.0 | sweepy_ |
| 37.9% | 19.149499 | -- | -- | 3512.0 | MPI |
| 28.7% | 14.487564 | 0.572138 | 3.8% | 3000.0 | mpi_alltoall |
| 8.7% | 4.391205 | 2.885755 | 39.7% | 2.0 | mpi_comm_split |
| 10.0% | 5.050780 | -- | -- | 3502.0 | MPI_SYNC |
| 6.9% | 3.483206 | 1.813952 | 52.1% | 3000.0 | mpi_alltoall_(sync) |
| 3.1% | 1.567285 | 0.606728 | 38.7% | 501.0 | mpi_allreduce_(sync) |

Normal Profile – Using “setenv PAT_RT_HWPC 1”

```
=====
USER / parabola_
-----
```

```

Time%                                12.4%
Time                                9.438486 secs
Imb. Time                           0.851876 secs
Imb. Time%                          8.3%
Calls                               0.265M/sec    2592000.0 calls
PAPI_L1_DCM                         42.908M/sec    419719824 misses
PAPI_TLB_DM                         0.048M/sec      474094 misses
PAPI_L1_DCA                        1067.727M/sec   10444336795 refs
PAPI_FP_OPS                        1808.848M/sec   17693862446 ops
Average Time per Call               0.000004 secs
CrayPat Overhead : Time             75.3%
User time (approx)                  9.782 secs    21520125183 cycles 100.0% Time
HW FP Ops / User time              1808.848M/sec   17693862446 ops  10.3%peak(DP)
HW FP Ops / WCT                    1808.848M/sec
Computational intensity              0.82 ops/cycle    1.69 ops/ref
MFLOPS (aggregate)                 7409042.08M/sec
TLB utilization                     22030.09 refs/miss  43.028 avg uses
D1 cache hit,miss ratios            96.0% hits        4.0% misses
D1 cache utilization (misses)       24.88 refs/miss    3.111 avg hits
=====
```

Re-compiling with `-hprofile_generate "pat_report-O callers"`

```

100.0% | 117.646170 | 13549032.0 |Total
|-----
| 75.4% | 88.723495 | 13542013.0 |USER
||-----
|| 10.7% | 12.589734 | 2592000.0 |parabola_
|||-----
3|| 7.1% | 8.360290 | 1728000.0 |remap_.LOOPS
4|| | | | remap_
5|| | | | ppmlr_
||||-----
6|||| 3.2% | 3.708452 | 768000.0 |sweepx2_.LOOP.2.li.35
7|||| | | | sweepx2_.LOOP.1.li.34
8|||| | | | sweepx2_.LOOPS
9|||| | | | sweepx2_
10|||| | | | vhone_
6|||| 3.1% | 3.663423 | 768000.0 |sweepx1_.LOOP.2.li.35
7|||| | | | sweepx1_.LOOP.1.li.34
8|||| | | | sweepx1_.LOOPS
9|||| | | | sweepx1_
10|||| | | | vhone_
|||||=====
3|| 3.6% | 4.229443 | 864000.0 |ppmlr_
||||-----
4||| 1.6% | 1.880874 | 384000.0 |sweepx2_.LOOP.2.li.35
5||| | | | sweepx2_.LOOP.1.li.34
6||| | | | sweepx2_.LOOPS
7||| | | | sweepx2_
8||| | | | vhone_
4||| 1.6% | 1.852820 | 384000.0 |sweepx1_.LOOP.2.li.35
5||| | | | sweepx1_.LOOP.1.li.34
6||| | | | sweepx1_.LOOPS
7||| | | | sweepx1_
8||| | | | vhone_
||||=====

```

Converting the MPI application to a Hybrid OpenMP/MPI application

Task 2 Parallel Analysis, Scoping and Vectorization

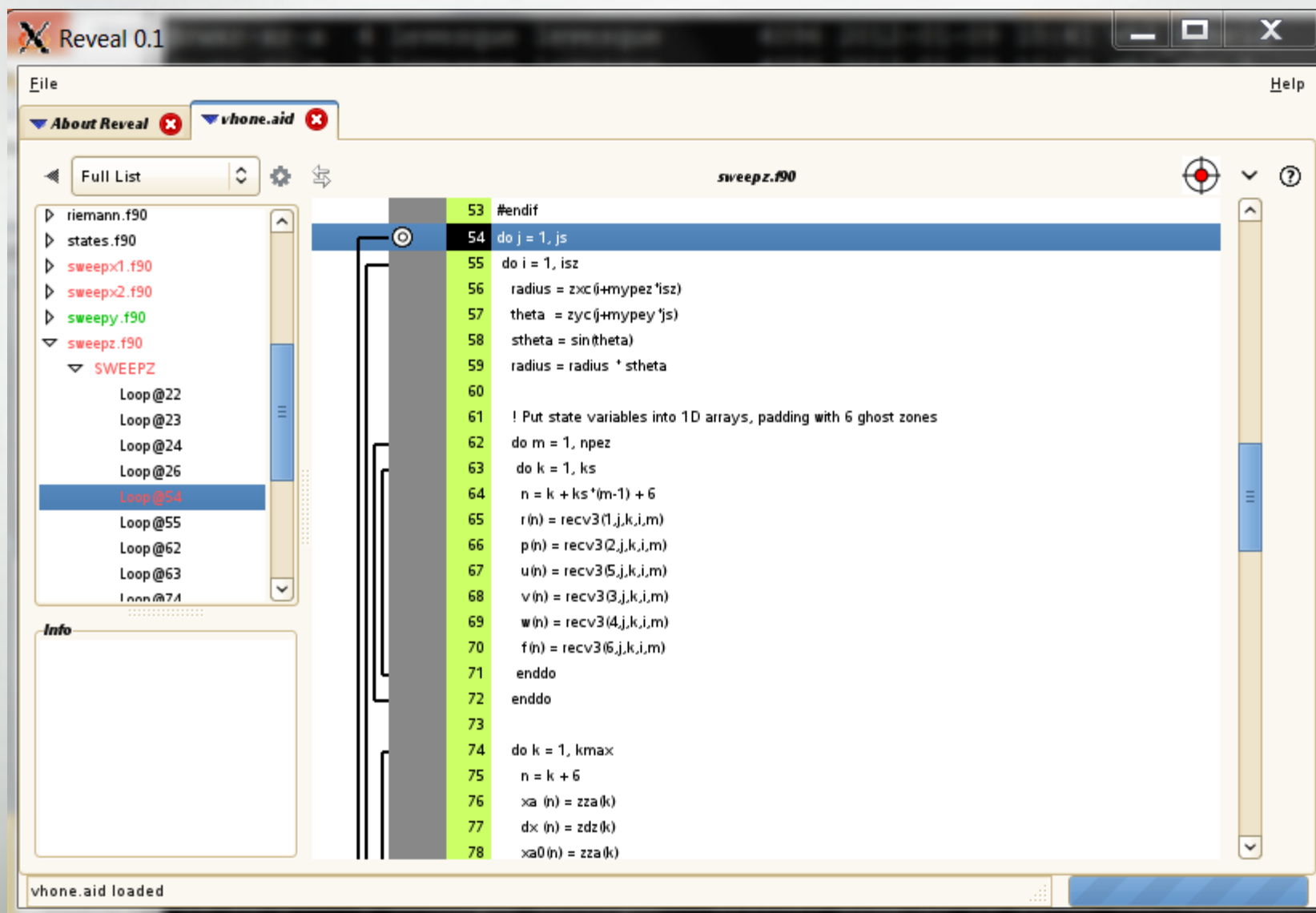
- Investigate parallelizability of high level looping structures
 - Often times one level of loop is not enough, must have several parallel loops
 - User must understand what high level DO loops are in fact independent.
 - Without tools, variable scoping of high level loops is very difficult
 - Loops must be more than independent, their variable usage must adhere to private data local to a thread or global shared across all the threads
- Investigate vectorizability of lower level Do loops
 - Cray compiler has been vectorizing complex codes for over 30 years

Converting the MPI application to a Hybrid OpenMP/MPI application


Task 2 Parallel Analysis, Scoping and Vectorization (Cont)

- Current scoping tool, -homp_analyze, is meant to interface to a code restructuring GUI called “reveal”. At this time, we need to use cryptic output and massage it with editor/script.
 - !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
 - <object_message kind="warn">LastPrivate of array may be very expensive.</object_message>

Main window of reveal



Scoping window


OpenMP Construct


| Name | Type | Scope | Info |
|-------|--------|---------|------|
| zyc | Scalar | Unknown | |
| a | Scalar | Private | |
| ai | Scalar | Private | |
| amid | Scalar | Private | |
| ar | Scalar | Private | |
| b | Scalar | Private | |
| bi | Scalar | Private | |
| c | Scalar | Private | |
| cdtdx | Scalar | Private | |
| ci | Scalar | Private | |
| clft | Scalar | Private | |
| crgh | Scalar | Private | |

First/Last Private—
☐ Enable First Private
☐ Enable Last Private

Reduction—
 None

Search:

Dump Data


 Close

At this point we should have some idea of the major arrays

- 1) Which arrays are use in the major computational routines?
- 2) Where else are these arrays used?
- 3) Are other arrays used with identified arrays
- 4) Go to 1

This is extremely difficult in Fortran and more so in C and C++. We could really used a tool that identified where in the code certain range of memory was used.

What we end up finding out

Private Variables in module, need to use Threadprivate

```
!$omp threadprivate (r, p, e, q, u, v, w, xa, xa0, dx, dx0, dvol, f, flat, para, radius, theta,
stheta)
real, dimension(maxsweep) :: r, p, e, q, u, v, w           ! fluid variables
real, dimension(maxsweep) :: xa, xa0, dx, dx0, dvol        ! coordinate values
real, dimension(maxsweep) :: f, flat                      ! flattening parameter
real, dimension(maxsweep,5) :: para                       ! parabolic interpolation
coefficients
real :: radius, theta, stheta
```

Reduction variable down callchain, need to use !\$OMP CRITICAL;!\$OMP END CRITICAL

```
hdt    = 0.5*dt
do n = nmin-4, nmax+4
  Cdtdx (n) = sqrt(gam*p(n)/r(n))/(dx(n)*radius)
enddo
!$omp critical
do n = nmin-4, nmax+4
  svel      = max(svel, Cdtdx(n))
enddo
!$omp end critical
do n = nmin-4, nmax+4
  Cdtdx (n) = Cdtdx(n)*hdt
  fCdtdx(n) = 1. - fourthd*Cdtdx(n)
enddo
```

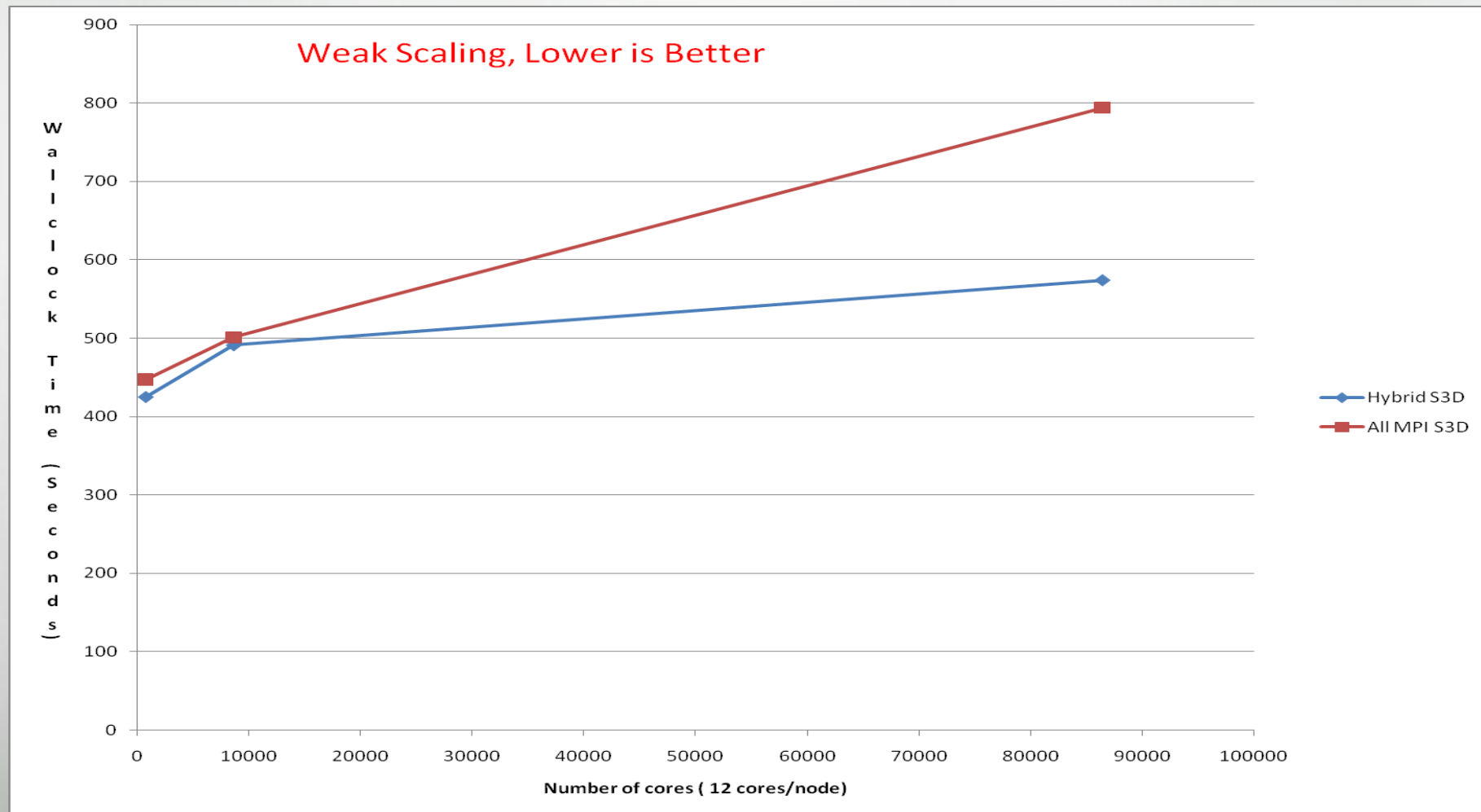
Task 3 Moving from OpenMP to OpenACC

- 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

```

#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
  
```

Resultant Hybrid S3D Performance

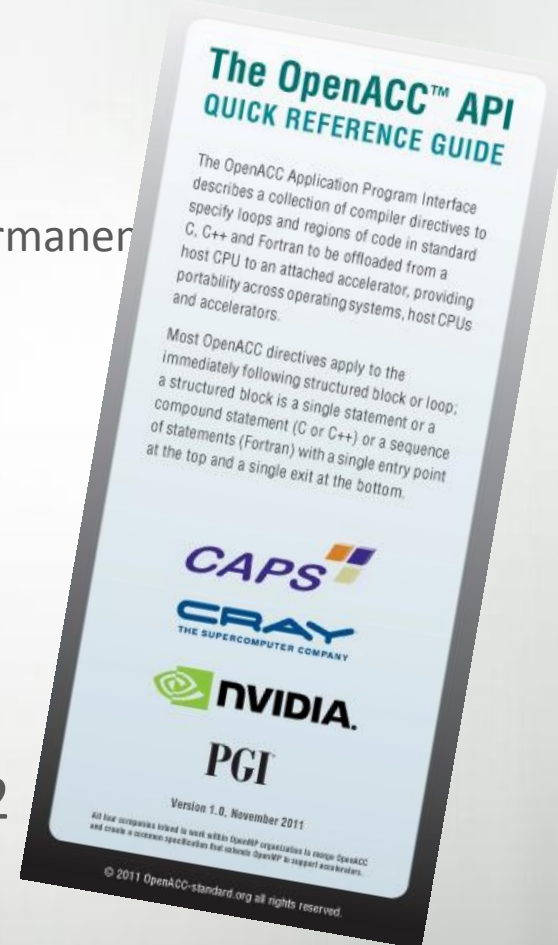


NVIDIA, Cray, PGI, CAPS Unveil 'OpenACC' Programming Standard for Parallel Computing

*Directives-based Programming Makes
Accelerating Applications Using
CPUs and GPUs Dramatically Easier*



- 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, permanent
 - 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: partial now, complete in 2012
 - PGI Accelerator: released product in 2012
 - CAPS: released product in Q1 2012



Using directives to give the compiler information

- Developing efficient OpenMP regions is not an easy task; however, the performance will definitely be worth the effort
- The next step will be to add OpenACC directives to allow for compilation of the same OpenMP regions to accelerator by the compiler.
 - With OpenACC data transfers between multi-core socket and the accelerator as well as utilization of registers and shared memory can be optimized.
 - With OpenACC user can control the utilization of the accelerator memory and functional units.

Task 3 Correctness Debugging

- Run transformed application on the accelerator and investigate the correctness and performance
 - Run as OpenMP application on multi-core socket
 - Use multi-core socket Debugger - DDT
 - Run as Hybrid multi-core application across multi-core socket and accelerator
- Tools That will be needed
 - Information that was supplied by the directives/user's interaction with the compiler

Task 4 Letting the Compiler do all the work

- The only requirement for using the !\$acc parallel loop is that the user specify the private variables and the compiler will do the rest.
 - If subroutine calls are contained in the loop, -hwp must be used.

```
#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
```

- The Compiler will then show:
 - All data motion required to run the loop on the accelerator.
 - Show how it handled the looping structures in the parallel region

Compiler list for SWEEPX1

```

45.          #ifdef GPU
46.  G-----< !$acc parallel loop private( k,j,i,n,r, p, e, q, u, v, w, svel0,&
47.  G          !$acc&      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)
82.  G g 3

```


Compiler list for SWEEPX1

ftn-6405 ftn: ACCEL File = sweepx1.f90, Line = 46

A region starting at line 46 and ending at line 104 was placed on the accelerator.

ftn-6418 ftn: ACCEL File = sweepx1.f90, Line = 46

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

ftn-6418 ftn: ACCEL File = sweepx1.f90, Line = 46

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

ftn-6418 ftn: ACCEL File = sweepx1.f90, Line = 46

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

ftn-6418 ftn: ACCEL File = sweepx1.f90, Line = 46

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

ftn-6418 ftn: ACCEL File = sweepx1.f90, Line = 46

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

ftn-6418 ftn: ACCEL File = sweepx1.f90, Line = 46

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

ftn-6416 ftn: ACCEL File = sweepx1.f90, Line = 46

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

Task 5 Fine tuning of accelerated program

- Understand current performance bottlenecks
 - Is data transfer between multi-core socket and accelerator a bottleneck?
 - Is shared memory and registers on the accelerator being used effectively?
 - Is the accelerator code utilizing the MIMD parallel units?
 - Is the shared memory parallelization load balanced?
 - Is the low level accelerator code vectorized?
 - Are the memory accesses effectively utilizing the memory bandwidth?

Profile of Accelerated Version 1

Table 1: Time and Bytes Transferred for Accelerator Regions

| Acc Time% | Acc Time | Host Time | Acc Copy In (MBytes) | Acc Copy Out (MBytes) | Calls | Function PE=HIDE Thread=HIDE |
|--------------|-------------|--------------|----------------------------|-----------------------------|-------|------------------------------------|
| 100.0% | 58.363 | 67.688 | 24006.022 | 16514.196 | 14007 | Total |
| 30.3% | 17.697 | 0.022 | -- | -- | 1000 | sweepy_.ACC_KERNEL@li.47 |
| 22.0% | 12.827 | 0.010 | -- | -- | 500 | sweepx2_.ACC_KERNEL@li.46 |
| 21.2% | 12.374 | 0.013 | -- | -- | 500 | sweepz_.ACC_KERNEL@li.67 |
| 14.0% | 8.170 | 0.013 | -- | -- | 500 | sweepx1_.ACC_KERNEL@li.46 |
| 3.9% | 2.281 | 1.161 | 12000.004 | -- | 1000 | sweepy_.ACC_COPY@li.47 |
| 2.0% | 1.162 | 0.601 | 6000.002 | -- | 500 | sweepz_.ACC_COPY@li.67 |
| 1.6% | 0.953 | 0.014 | -- | 6000.004 | 1000 | sweepy_.ACC_COPY@li.129 |
| 1.0% | 0.593 | 0.546 | 3000.002 | -- | 500 | sweepx1_.ACC_COPY@li.46 |
| 1.0% | 0.591 | 0.533 | 3000.002 | -- | 500 | sweepx2_.ACC_COPY@li.46 |
| 0.8% | 0.494 | 0.015 | -- | 3000.002 | 500 | sweepx2_.ACC_COPY@li.107 |
| 0.8% | 0.485 | 0.007 | -- | 3000.002 | 500 | sweepx1_.ACC_COPY@li.104 |
| 0.8% | 0.477 | 0.007 | -- | 3000.002 | 500 | sweepz_.ACC_COPY@li.150 |
| 0.4% | 0.250 | 0.016 | -- | 1503.174 | 500 | vhone_.ACC_COPY@li.251 |
| 0.0% | 0.005 | 0.005 | 6.012 | -- | 1 | vhone_.ACC_COPY@li.205 |
| 0.0% | 0.001 | 0.000 | -- | 6.012 | 1 | vhone_.ACC_COPY@li.283 |
| 0.0% | 0.001 | 0.000 | -- | 5.000 | 1 | vhone_.ACC_COPY@li.266 |

Differences in runtime

All MPI on 4096 cores

43.01 seconds

Hybrid 256 nodesx16 threads

45.05 seconds

Rest Hybrid 256x16 threads

47.58 seconds

OpenACC 256xgpu

105.92 seconds

Task 4 Fine tuning of accelerated program

- Tools that will be needed:
 - Compiler feedback on parallelization and vectorization of input application
 - Hardware counter information from the accelerator to identify bottlenecks in the execution of the application.
 - Information on memory utilization
 - Information on performance of SIMT units

Several other vendors are supplying similar performance gathering tools

Useful tools contd.

- Craypat profiling
 - Tracing: "pat_build -u <executable>" (can do APA sampling first)
 - "pat_report -O accelerator <.xf file>"; -T also useful
 - Other pat_report tables (as of perftools/5.2.1.7534)
 - acc_fu flat table of accelerator events
 - acc_time call tree sorted by accelerator time
 - acc_time_fu flat table of accelerator events sorted by accelerator time
 - acc_show_by_ct regions and events by calltree sorted alphabetically

Run and gather runtime statistics

Table 1: Profile by Function Group and Function

| Time % | Time | Imb. Time | Imb. Time % | Calls | Group | Function |
|--------|-----------|-----------|-------------|-------|---------------------------------|---------------|
| | | | | | | PE='HIDE' |
| | | | | | | Thread='HIDE' |
| 100.0% | 83.277477 | -- | -- | 851.0 | Total | |
| ----- | | | | | | |
| 51.3% | 42.762837 | -- | -- | 703.0 | ACCELERATOR | |
| ----- | | | | | | |
| 18.8% | 15.672371 | 1.146276 | 7.3% | 20.0 | recolor_.SYNC_COPY@li.790 | ←not good |
| 16.3% | 13.585707 | 0.404190 | 3.1% | 20.0 | recolor_.SYNC_COPY@li.793 | ←not good |
| 7.5% | 6.216010 | 0.873830 | 13.1% | 20.0 | lbm3d2p_d_.ASYNC_KERNEL@li.116 | |
| 1.6% | 1.337119 | 0.193826 | 13.5% | 20.0 | lbm3d2p_d_.ASYNC_KERNEL@li.119 | |
| 1.6% | 1.322690 | 0.059387 | 4.6% | 1.0 | lbm3d2p_d_.ASYNC_COPY@li.100 | |
| 1.0% | 0.857149 | 0.245369 | 23.7% | 20.0 | collisionb_.ASYNC_KERNEL@li.586 | |
| 1.0% | 0.822911 | 0.172468 | 18.5% | 20.0 | lbm3d2p_d_.ASYNC_KERNEL@li.114 | |
| 0.9% | 0.786618 | 0.386807 | 35.2% | 20.0 | injection_.ASYNC_KERNEL@li.1119 | |
| 0.9% | 0.727451 | 0.221332 | 24.9% | 20.0 | lbm3d2p_d_.ASYNC_KERNEL@li.118 | |

Keep data on the accelerator with acc_data region

```

!$acc data copyin(cix,ci1,ci2,ci3,ci4,ci5,ci6,ci7,ci8,ci9,ci10,ci11,&
!$acc& ci12,ci13,ci14,r,b,uxyz,cell,rho,grad,index_max,index,&
!$acc& ciy,ciz,wet,np,streaming_sbuf1, &
!$acc& streaming_sbuf1,streaming_sbuf2,streaming_sbuf4,streaming_sbuf5,&
!$acc& streaming_sbuf7s,streaming_sbuf8s,streaming_sbuf9n,streaming_sbuf10s,&
!$acc& streaming_sbuf11n,streaming_sbuf12n,streaming_sbuf13s,streaming_sbuf14n,&
!$acc& streaming_sbuf7e,streaming_sbuf8w,streaming_sbuf9e,streaming_sbuf10e,&
!$acc& streaming_sbuf11w,streaming_sbuf12e,streaming_sbuf13w,streaming_sbuf14w, &
!$acc& streaming_rbuf1,streaming_rbuf2,streaming_rbuf4,streaming_rbuf5,&
!$acc& streaming_rbuf7n,streaming_rbuf8n,streaming_rbuf9s,streaming_rbuf10n,&
!$acc& streaming_rbuf11s,streaming_rbuf12s,streaming_rbuf13n,streaming_rbuf14s,&
!$acc& streaming_rbuf7w,streaming_rbuf8e,streaming_rbuf9w,streaming_rbuf10w,&
!$acc& streaming_rbuf11e,streaming_rbuf12w,streaming_rbuf13e,streaming_rbuf14e, &
!$acc& send_e,send_w,send_n,send_s,recv_e,recv_w,recv_n,recv_s)
do ii=1,ntimes
  o o o
  call set_boundary_macro_press2
  call set_boundary_micro_press
  call collisiona
  call collisionb
  call recolor

```

Now when we do communication we have to update the host

```

!$acc parallel_loop private(k,j,i)
  do j=0,local_ly-1
    do i=0,local_lx-1
      if (cell(i,j,0)==1) then
        grad (i,j,-1) = (1.0d0-wet)*db*press
      else
        grad (i,j,-1) = db*press
      end if
      grad (i,j,lz)    = grad(i,j,lz-1)
    end do
  end do
!$acc end parallel_loop
!$acc update host(grad)
  call mpi_barrier(mpi_comm_world,ierr)
  call grad_exchange
!$acc update device(grad)
  
```

But we would rather not send the entire grad array back – how about

```
!$acc data present(grad,recv_w,recv_e,send_e,send_w,recv_n,&
!$acc&                      recv_s,send_n,send_s)
!$acc parallel_loop
  do k=-1,lz
    do j=-1,local_ly
      send_e(j,k) = grad(local_lx-1,j          ,k)
      send_w(j,k) = grad(0          ,j          ,k)
    end do
  end do
!$acc end parallel_loop
!$acc update host(send_e,send_w)
  call mpi_irecv(recv_w, bufsize(2),mpi_double_precision,w_id, &
    tag(25),mpi_comm_world,irequest_in(25),ierr)
    o o o
  call mpi_isend(send_w, bufsize(2),mpi_double_precision,w_id, &
    tag(26),& mpi_comm_world,irequest_out(26),ierr)
  call mpi_waitall(2,irequest_in(25),istatus_req,ierr)
  call mpi_waitall(2,irequest_out(25),istatus_req,ierr)
!$acc update device(recv_e,recv_w)
!$acc parallel
!$acc loop
  do k=-1,lz
    do j=-1,local_ly
      grad(local_lx ,j          ,k) = recv_e(j,k)
      grad(-1      ,j          ,k) = recv_w(j,k)
```

Final Profile - bulk of time in kernel execution

| | | | | | | | | | | | |
|--|-------|--|------------|--|-----------|--|-------|--|---------|--|---|
| | 37.9% | | 236.592782 | | -- | | -- | | 11403.0 | | ACCELERATOR |
| | ----- | | | | | | | | | | |
| | 15.7% | | 98.021619 | | 43.078137 | | 31.0% | | 200.0 | | lbm3d2p_d_.ASYNC_KERNEL@li.129 |
| | 3.7% | | 23.359080 | | 2.072147 | | 8.3% | | 200.0 | | lbm3d2p_d_.ASYNC_KERNEL@li.127 |
| | 3.6% | | 22.326085 | | 1.469419 | | 6.3% | | 200.0 | | lbm3d2p_d_.ASYNC_KERNEL@li.132 |
| | 3.0% | | 19.035232 | | 1.464608 | | 7.3% | | 200.0 | | collisionb_.ASYNC_KERNEL@li.599 |
| | 2.6% | | 16.216648 | | 3.505232 | | 18.1% | | 200.0 | | lbm3d2p_d_.ASYNC_KERNEL@li.131 |
| | 2.5% | | 15.401916 | | 8.093716 | | 35.0% | | 200.0 | | injection_.ASYNC_KERNEL@li.1116 |
| | 1.9% | | 11.734026 | | 4.488785 | | 28.1% | | 200.0 | | recolor_.ASYNC_KERNEL@li.786 |
| | 0.9% | | 5.530201 | | 2.132243 | | 28.3% | | 200.0 | | collisionb_.SYNC_COPY@li.593 |
| | 0.8% | | 4.714995 | | 0.518495 | | 10.1% | | 200.0 | | collisionb_.SYNC_COPY@li.596 |
| | 0.6% | | 3.738615 | | 2.986891 | | 45.1% | | 200.0 | | collisionb_.ASYNC_KERNEL@li.568 |
| | 0.4% | | 2.656962 | | 0.454093 | | 14.8% | | 1.0 | | lbm3d2p_d_.ASYNC_COPY@li.100 |
| | 0.4% | | 2.489231 | | 2.409892 | | 50.0% | | 200.0 | | streaming_exchange_.ASYNC_COPY@li.810 |
| | 0.4% | | 2.487132 | | 2.311190 | | 48.9% | | 200.0 | | streaming_exchange_.ASYNC_COPY@li.625 |
| | 0.2% | | 1.322791 | | 0.510645 | | 28.3% | | 200.0 | | streaming_exchange_.SYNC_COPY@li.622 |
| | 0.2% | | 1.273771 | | 0.288743 | | 18.8% | | 200.0 | | streaming_exchange_.SYNC_COPY@li.574 |
| | 0.2% | | 1.212260 | | 0.298053 | | 20.0% | | 200.0 | | streaming_exchange_.SYNC_COPY@li.759 |
| | 0.2% | | 1.208250 | | 0.422182 | | 26.3% | | 200.0 | | streaming_exchange_.SYNC_COPY@li.806 |
| | 0.1% | | 0.696120 | | 0.442372 | | 39.5% | | 200.0 | | streaming_exchange_.ASYNC_KERNEL@li.625 |
| | 0.1% | | 0.624982 | | 0.379697 | | 38.4% | | 200.0 | | streaming_exchange_.ASYNC_KERNEL@li.525 |

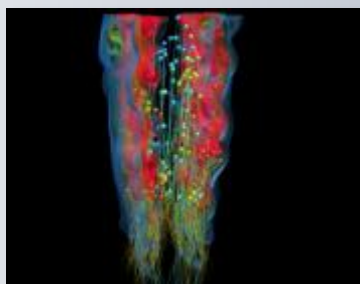
Cray GPU Programming Environment

- Objective: Enhance productivity related to porting applications to hybrid multi-core systems
- Four core components
 - Cray Statistics Gathering Facility on host and GPU
 - Cray Optimization Explorer – Scoping Tools (COE)
 - Cray Compilation Environment (CCE)
 - Cray GPU Libraries

Titan: Early Science Applications

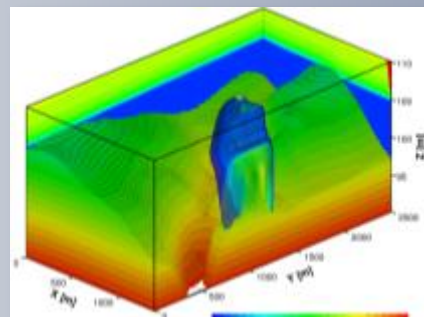
WL-LSMS

Role of material disorder, statistics, and fluctuations in nanoscale materials and systems.



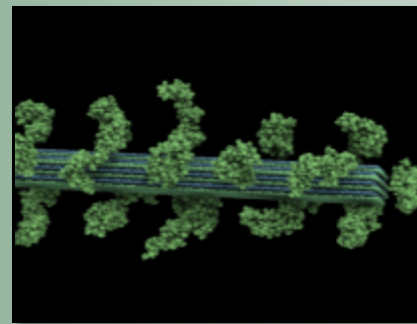
S3D

How are going to efficiently burn next generation diesel/bio fuels?



PFLOTRAN

Stability and viability of large scale CO₂ sequestration; predictive containment groundwater transport

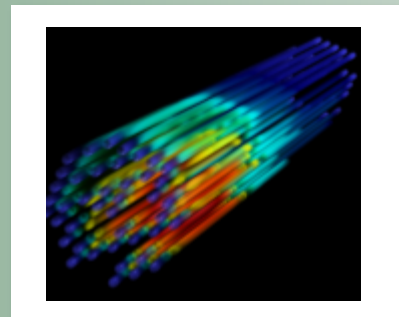
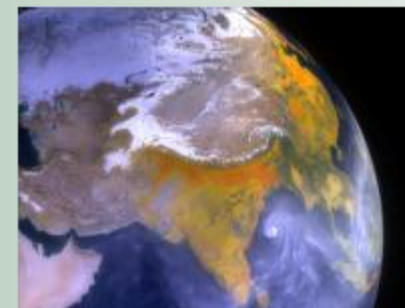


LAMMPS

Biofuels: An atomistic model of cellulose (blue) surrounded by lignin molecules comprising a total of 3.3 million atoms. Water not shown.

CAM / HOMME

Answer questions about specific climate change adaptation and mitigation scenarios; realistically represent features like precipitation patterns/statistics and tropical storms

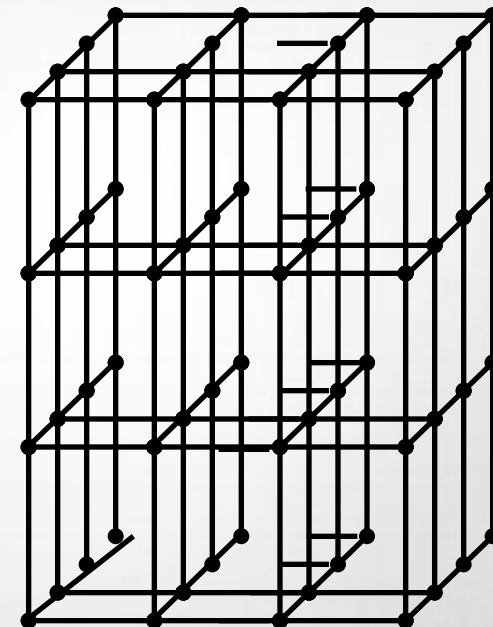


Denovo

Unprecedented high-fidelity radiation transport calculations that can be used in a variety of nuclear energy and technology applications.

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.



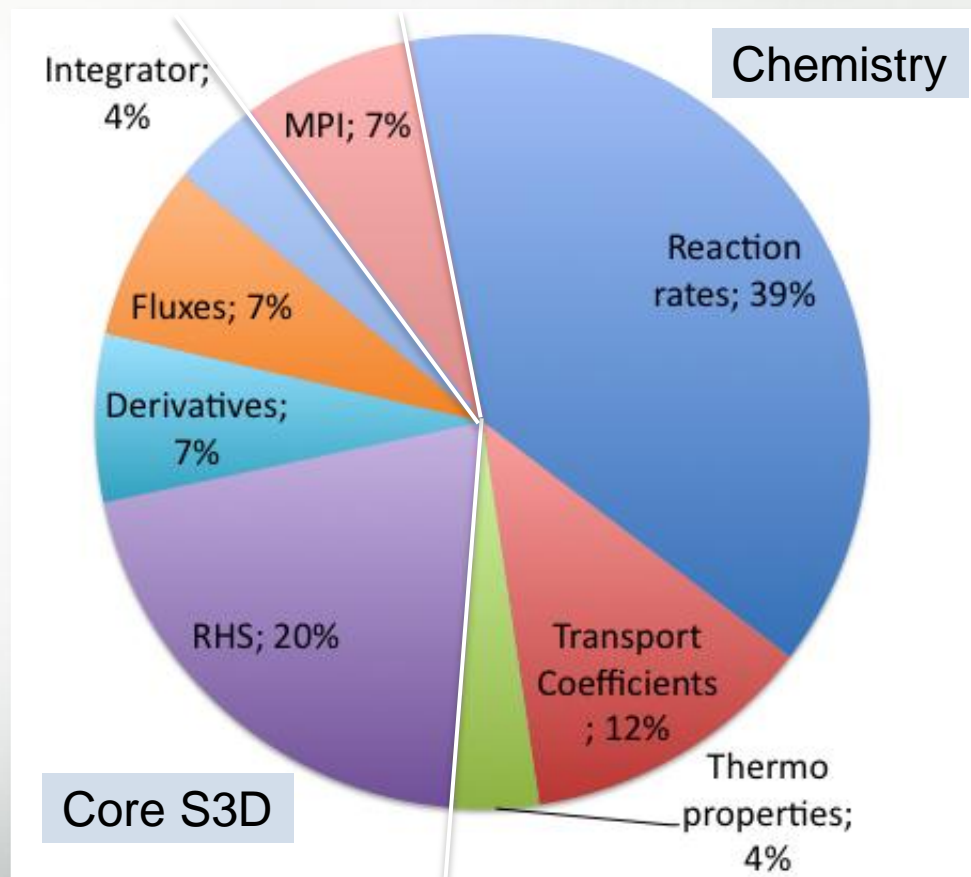
Developed and maintained at CRF, Sandia (Livermore) with BES and ASCR sponsorship. PI – Jacqueline H. Chen (jhchen@sandia.gov)

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

Original S3D

S3D

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

Profile from Original S3D

Table 1: Profile by Function Group and Function

| Time% | Time | Imb. Time | Imb. Time% | Calls | Group 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% | 4.461274 | 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 |

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 |
| ----- | | | | | |
| 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 |

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

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

What does OpenACC look like

```
#ifdef GPU
!$acc data copyin(avmolwt, cpCoef_aa, cpCoef_bb, cpmix, enthCoef_aa, enthCoef_bb, &
!$acc& gamma, invEnthInc, iorder, lrmcwrk, 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

```
#ifdef GPU
!$acc data present_or_create( avmolwt, cpcoef_aa, cpcoef_bb, cpmix, enthcoef_aa, enthcoef_bb, &
!$acc&   gamma, invEnthInc, lrmcwrk, 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,voltmp, 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)&
!$acc&   copyin( jstage,scale1x,scale1y,scale1z,aex,bex,cex,dex,ds,aey,bey,cey,dey,aez,bez,cez,dez)
#endif
```

What does OpenACC look like

```

#ifdef GPU
!$acc parallel loop gang private(i,ml,mu)
#else
!$omp parallel private(i, ml, mu)
!$omp do
#endif
  do i = 1, nx*ny*nz, ms
    ml = i
    mu = min(i+ms-1, nx*ny*nz)
    call calc_gamma_r( gamma, cpmix, avmolwt, ml, mu)
    call calc_press_r( pressure, q(1,1,1,4), temp, avmolwt, ml, mu )
    call calc_specEnth_allpts_r(temp, h_spec, ml, mu)
  end do
#ifdef GPU
!$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)
#endif
do n=1,n_spec
do k = 1,nz
#ifdef GPU
!$acc loop vector collapse(2)
#endif
do j = 1,ny
do i = 1,nx
grad_Ys(i,j,k,n,1) = 0.0
grad_Ys(i,j,k,n,2) = 0.0
grad_Ys(i,j,k,n,3) = 0.0
if(i.gt.iorder/2 .and. i.le.nx-iorder/2) then
grad_Ys(i,j,k,n,1) = scale1x(i)*(aex *( yspecies(i+1,j,k,n)-yspecies(i-1,j,k,n) ) &
+ bez *( yspecies(i+2,j,k,n)-yspecies(i-2,j,k,n) ) &
+ cex *( yspecies(i+3,j,k,n)-yspecies(i-3,j,k,n) ) &
+ dex *( yspecies(i+4,j,k,n)-yspecies(i-4,j,k,n) ))
endif
if(j.gt.iorder/2 .and. j.le.ny-iorder/2) then
grad_Ys(i,j,k,n,2) = scale1y(j)*(aey *( yspecies(i,j+1,k,n)-yspecies(i,j-1,k,n) ) &
+ bey *( yspecies(i,j+2,k,n)-yspecies(i,j-2,k,n) ) &
+ cey *( yspecies(i,j+3,k,n)-yspecies(i,j-3,k,n) ) &
+ dey *( yspecies(i,j+4,k,n)-yspecies(i,j-4,k,n) ))
endif
if(k.gt.iorder/2 .and. k.le.nz-iorder/2) then
grad_Ys(i,j,k,n,3) = scale1z(k)*(aez *( yspecies(i,j,k+1,n)-yspecies(i,j,k-1,n) ) &
+ bez *( yspecies(i,j,k+2,n)-yspecies(i,j,k-2,n) ) &
+ cez *( yspecies(i,j,k+3,n)-yspecies(i,j,k-3,n) ) &
+ dez *( yspecies(i,j,k+4,n)-yspecies(i,j,k-4,n) ))
endif
end do ! i
end do ! j
#endif GPU
!$acc end loop
#endif
end do ! k
end do ! n
#ifdef GPU
!$acc end parallel loop
#endif
```



What does OpenACC look like

```
#ifdef GPU
#ifdef GPU_STREAMS
!$acc update host(pos_fs_x_buf(:, :, :, idx)) async(istr)
#else
!$acc host_data use_device(pos_fs_x_buf)
#endif
#endif
#ifdef GPU_STREAMS
    call cray_mpiif_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
#ifdef GPU
#ifdef GPU_STREAMS
!$acc end host_data
#endif
#endif
```



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)

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

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

!\$acc host_data use_device

```
#ifdef GPU
!$acc data present(f)
!$acc host_data use_device(f)
#endif
if( deriv_z_list(idx)%packed ) then
  deriv_z_list(idx)%packed = .false.
  if(deriv_z_list(idx)%neg_nbr>=0) then
    call MPI_Isend(f(1,1,1), (mx*my*iorder/2), &
                  MPI_REAL8, deriv_z_list(idx)%neg_nbr, deriv_list_size + idx, &
                  gcomm, deriv_z_list(idx)%req(2), ierr)
  endif
  if(deriv_z_list(idx)%pos_nbr>=0) then
    ! send ghost cells to neighbor on (+z) side
    nm = mz + 1 - iorder/2
    call MPI_Isend(f(1,1,nm), (mx*my*iorder/2), &
                  MPI_REAL8, deriv_z_list(idx)%pos_nbr, idx, &
                  gcomm, deriv_z_list(idx)%req(4), ierr)
  endif
else
  if(deriv_z_list(idx)%neg_nbr>=0) then
    call MPI_Isend(f(1,1,1), (mx*my*iorder/2), &
                  MPI_REAL8, deriv_z_list(idx)%neg_nbr, deriv_list_size + idx, &
                  gcomm, deriv_z_list(idx)%req(2), ierr)
  endif

  if(deriv_z_list(idx)%pos_nbr>=0) then
    ! send ghost cells to neighbor on (+z) side
    nm = mz + 1 - iorder/2
    call MPI_Isend(f(1,1,nm), (mx*my*iorder/2), &
                  MPI_REAL8, deriv_z_list(idx)%pos_nbr, idx, &
                  gcomm, deriv_z_list(idx)%req(4), ierr)
  endif
endif
endif
#endif
!$acc end host_data
!$acc end data
#endif
```

Thank You!

