Refactoring Applications for the XK7 and Future Hybrid Architectures

John M Levesque (Cray Inc)
&
Jeff Larkin (Nvidia)

Are you ready for the future?

- **You must move to a hybrid (MPI, threading & vector) architecture to prepare for the future**
- **You must start considering how to manage your arrays to have them close to the computational engine when you need them there**
  - We are moving to a more complex memory hierarchy that will require user intervention to achieve good performance.
- **You must design your application to be performance portable across a wide range of systems**
  - Fortunately they will be similar enough that this will be possible
- **Bottom Line – you must understand your application extremely well to achieve acceptable performance on today's and tomorrow's architectures**

Outline of Workshop

- **Monday Morning (John Levesque)**
  - Introduction to this generation of Supercomputers
  - Using tools to understand your application
    - Where is the time spent
    - Where are the important parallel structures (DO Loops)
    - Where are the important arrays used
  - Demonstration of the steps for porting/optimizing a simple application
- **Monday Afternoon**
  - Hands on using the tools and getting ready to hybridize your application.

Outline of Workshop

- **Tuesday Morning**
  - Architecture of the XK7 (Jeff Larkin)
  - Reveal demonstration (Heidi Poxon)
  - OpenMP issues (John Levesque)
- **Tuesday Afternoon**
  - Use Reveal on your application and get the best OpenMP parallelization

Outline of Workshop

- **Wednesday Morning**
  - Introducing OpenACC to your application (John Levesque)
  - How OpenACC and CUDA play together (Jeff Larkin)
  - How to use the tools to optimize for the GPU (Larkin and Levesque)
- **Wednesday Afternoon**
  - Vampire Demonstration (Joseph Schuchart)
  - Get started putting OpenACC into your application

Outline of Workshop

- **Thursday Morning (John Levesque)**
  - Optimizing the OpenACC
    - Optimizing Data movement
    - Optimizing Kernel performance
    - Optimizing communicating between GPU nodes
- **Thursday Afternoon**
  - Hands on to work on your application

Files you should get

```
[levesque@home2 levesque]$ ls -ltr *
-rw-r--r-- 1 levesque levesque   72867 Jul 11 09:32 VH1_version1.tgz
-rw-r--r-- 1 levesque levesque   88362 Jul 11 10:57 himeno_workshop.tgz
-rw-rw-r-- 1 levesque levesque 4439874 Jul 14 10:08 Refactoring_for_XK7.pdf

OpenACC_workshop:
total 5576
-rw-rw-r-- 1 levesque levesque  204800 Jul  9 12:14 vh1_src12.tar
-rw-rw-r-- 1 levesque levesque  125622 Jul  9 12:42 VH1_version1-2.tgz
-rw-rw-r-- 1 levesque levesque     345 Jul  9 21:13 Document.rtf
-rw-rw-r-- 1 levesque levesque 5341192 Jul 10 18:14 Refactoring_for_XK7.pptx
drwxrwxr-x 2 levesque levesque    4096 Jul 12 00:00 VH1_version1
drwxrwxr-x 2 levesque levesque    4096 Jul 12 00:00 VH1_version2

additional_OpenACC_slides:
total 16176
-rw-rw-r-- 1 levesque levesque  817721 May 20 07:42 OpenACC-2.0-draft.pdf
-rw-rw-r-- 1 levesque levesque  798783 Jul 11 10:04 P1_sample_code.pptx
-rw-rw-r-- 1 levesque levesque 2167318 Jul 11 10:04 L9_interop_future.pptx
-rw-rw-r-- 1 levesque levesque  836779 Jul 11 10:04 L8_porting_MG.pptx
-rw-rw-r-- 1 levesque levesque 1568472 Jul 11 10:04 L7_parallel_himeno.pptx
-rw-rw-r-- 1 levesque levesque  964370 Jul 11 10:04 L6_advanced_openacc.pptx
-rw-rw-r-- 1 levesque levesque  560772 Jul 11 10:04 L5_Poznanovic_OpenACC-troubleshooting.pdf
-rw-rw-r-- 1 levesque levesque 1421039 Jul 11 10:04 L4_scalar_himeno.pptx
-rw-rw-r-- 1 levesque levesque 1036361 Jul 11 10:04 L3_intro_openacc.pptx
-rw-rw-r-- 1 levesque levesque  807418 Jul 11 10:04 L2_using_cray_pe.pptx
-rw-rw-r-- 1 levesque levesque 4648952 Jul 11 10:04 L1_intro.pptx
-rw-rw-r-- 1 levesque levesque  848640 Jul 11 10:04 P3_openacc_mg.docx
```

OpenACC®

DIRECTIVES FOR ACCELERATORS

CRAY THE SUPERCOMPUTER COMPANY

- 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, permaner
  - 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

CAPS CRAY THE SUPERCOMPUTER COMPANY NVIDIA The Portland Group

- C99, C++, F2003 Compilers
  - Optimizing
  - Vectorizing
  - Parallelizing
- Graphical parallel tools
  - PGDBG® debugger
  - PGPROF® profiler
- AMD, Intel, NVIDIA
- PGI Unified Binary™
- Linux, OS X, Windows
- Visual Studio & Eclipse integration
- GPGPU Features
  - PGI Accelerator™, OpenACC
  - CUDA Fortran/C/C++
  - CUDA-x86
  - OpenCL (ARM CPUs only)

www.pgroup.com

PGI CUDA Fortran

```
real, device, allocatable, dimension(:,:) ::
        Adev,Bdev,Cdev

. . .

allocate (Adev(N,M), Bdev(M,L), Cdev(N,L))
Adev = A(1:N,1:M)
Bdev = B(1:M,1:L)

call mm_kernel <<<dim3(N/16,M/16),dim3(16,16)>>>
              ( Adev, Bdev, Cdev, N, M, L)

C(1:N,1:L) = Cdev
deallocate ( Adev, Bdev, Cdev )

. . .
```

CPU Code

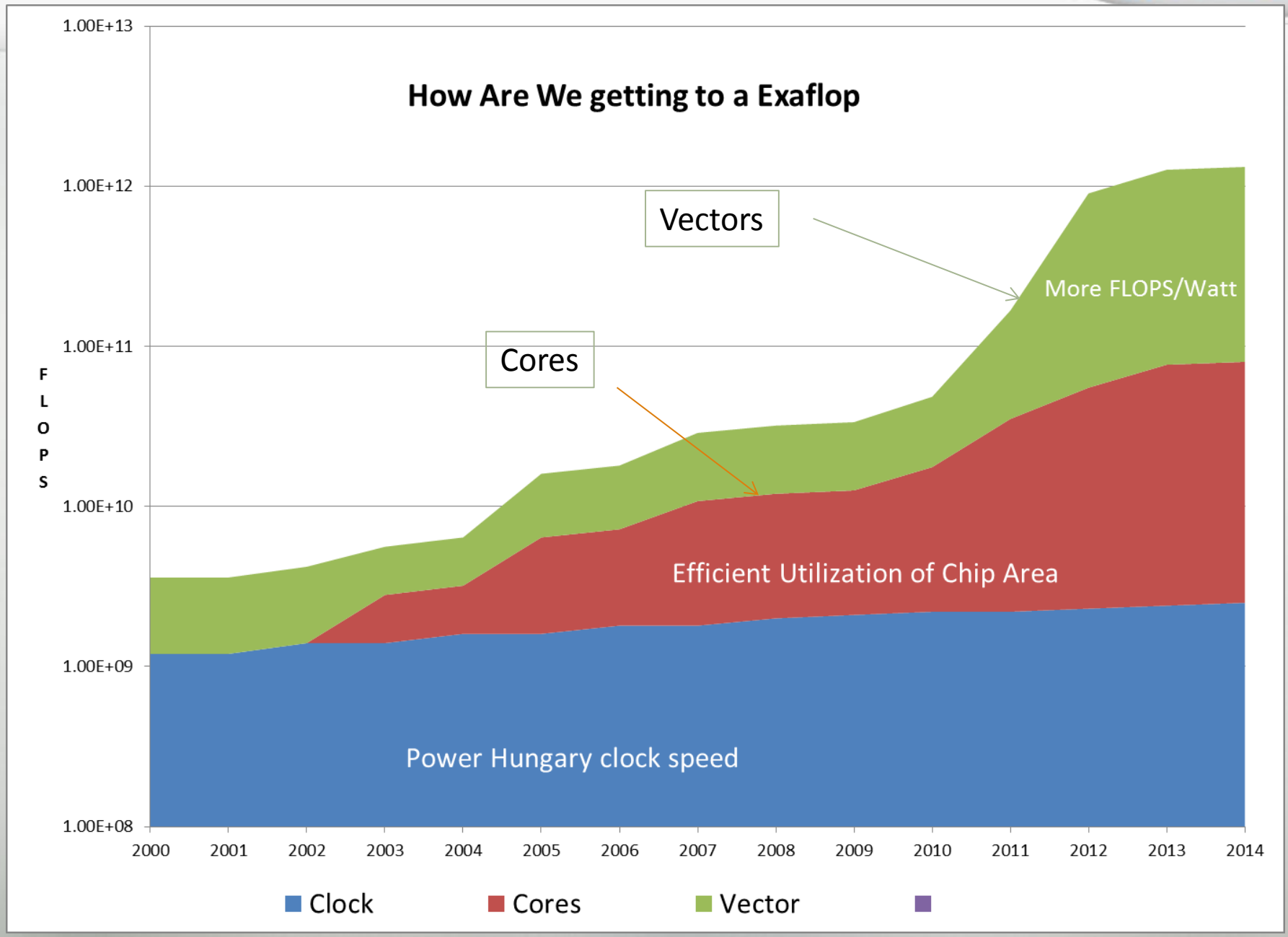
```
attributes(global) subroutine mm_kernel
                ( A, B, C, N, M, L )
real :: A(N,M), B(M,L), C(N,L), Cij
integer, value :: N, M, L
integer :: i, j, kb, k, tx, ty
real, shared :: Asub(16,16),Bsub(16,16)
tx = threadidx%x
ty = threadidx%y
i = blockidx%x * 16 + tx
j = blockidx%y * 16 + ty
Cij = 0.0
do kb = 1, M, 16
   Asub(tx,ty) = A(i,kb+tx-1)
   Bsub(tx,ty) = B(kb+ty-1,j)
   call syncthreads()
   do k = 1,16
      Cij = Cij + Asub(tx,k) * Bsub(k,ty)
   enddo
   call syncthreads()
enddo
C(i,j) = Cij
end subroutine mmul_kernel
```

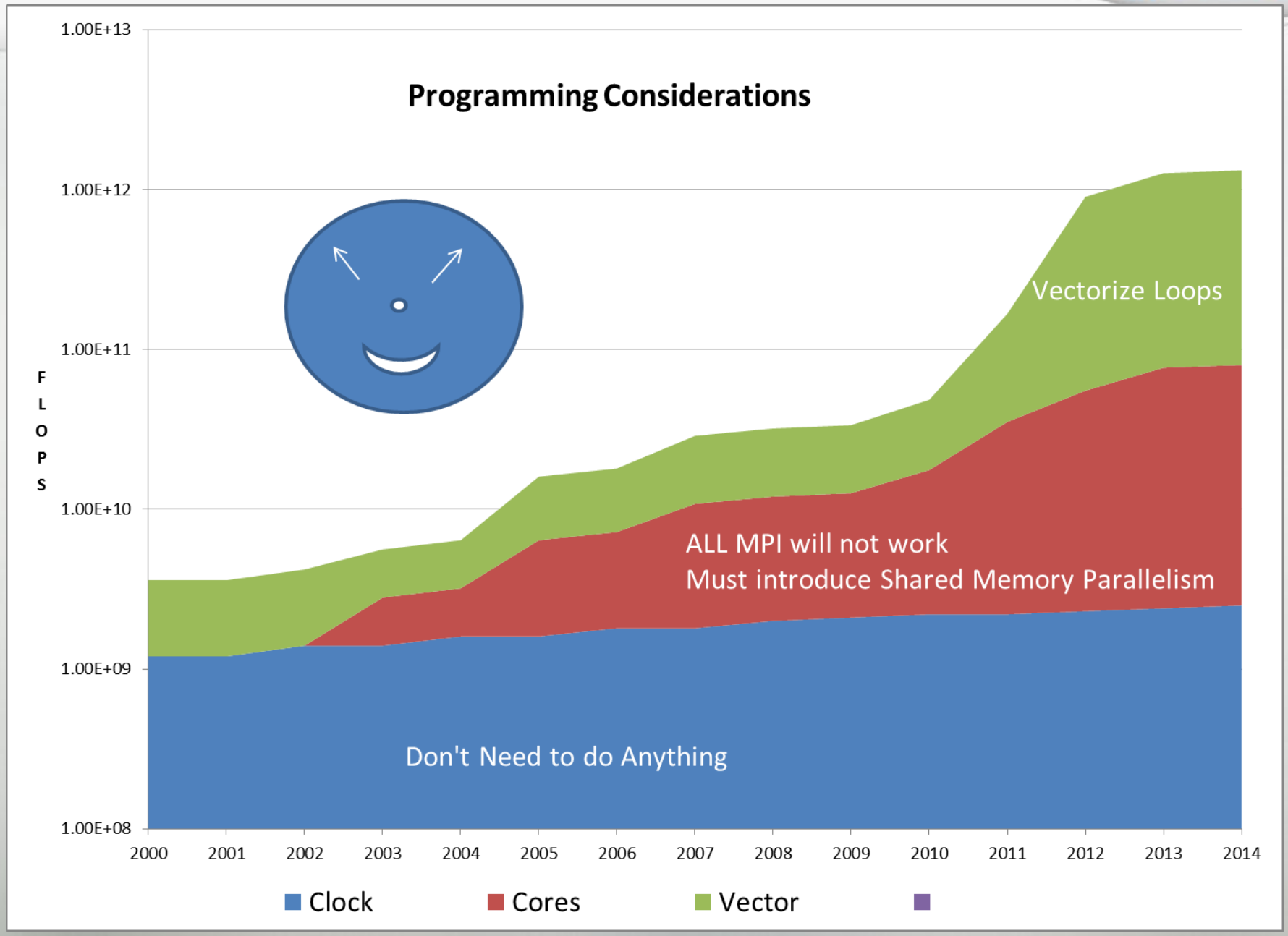
GPU Code

Strategic risk factors

- **Will there be machines to run my OpenACC code on?**
  - Now? Lots of Nvidia GPU accelerated systems
    - Cray XK7s: CSCS tödi, HLRS hermit, ORNL titan...
    - Lots of other GPU machines in Top100 (OpenACC is multi-vendor)
  - Future? OpenACC can be targetted at other accelerators
    - PGI and CAPS already target Intel Xeon Phi, AMD GPUs
  - Plus you can always run on CPUs using same codebase
- **Will OpenACC continue?**
  - Support? Cray and PGI (at least) are committed to support OpenACC
    - Lots of big customer pressure to continue to run OpenACC
  - Develop? OpenACC committee now finalising v2.0 of standard
    - Lots of new partners joined committee at end of last year
- **Will OpenACC be superseded by something else?**
  - Auto-accelerating compilers? If only!
    - Never really managed it for threading real HPC applications on the CPU
    - Data locality adds to the challenge
  - OpenMP accelerator directives? OpenACC work not wasted
    - Very similar programming model; can transition when these release if wish
    - Cray (co-chair), PGI very active in OpenMP accelerator subcommittee

Programming for Hybrid Architectures


How Are We getting to a Exaflop
1.00E+13
1.00E+12
1.00E+11
1.00E+10
1.00E+09
1.00E+08
FLOPS
Vectors
More FLOPS/Watt
Cores
Efficient Utilization of Chip Area
Power Hungary clock speed
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
Clock
Cores
Vector


Programming Considerations
1.00E+13
1.00E+12
1.00E+11
1.00E+10
1.00E+09
1.00E+08
FLOPS
Vectorize Loops
ALL MPI will not work
Must introduce Shared Memory Parallelism
Don't Need to do Anything
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
Clock
Cores
Vector

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)

Power-Constrained Computing

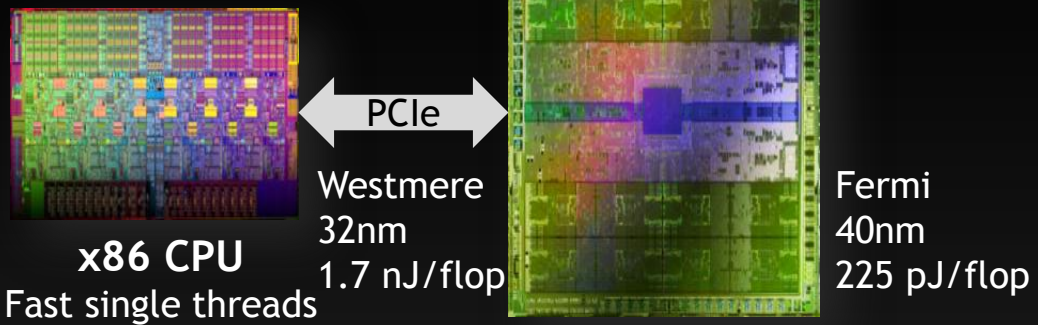
Classic voltage scaling is over

$\Rightarrow$ chips have become power (not area) constrained

and it's getting worse with each generation

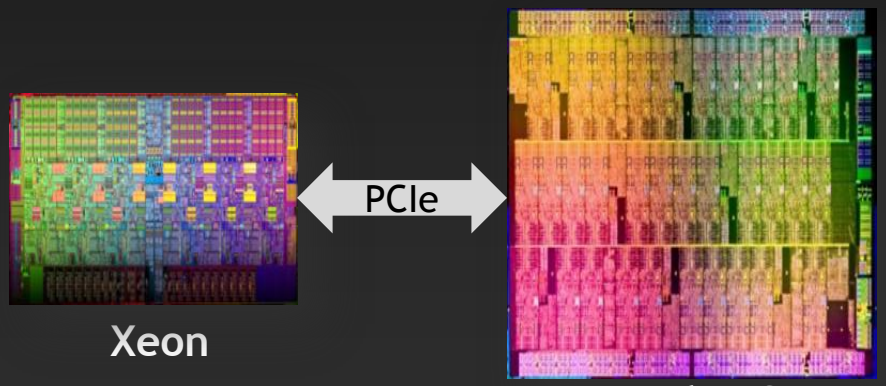
Steve Scott – Cray Technical Workshop Oct 2012

HPC Goes Hybrid



x86 CPU
Fast single threads (serial work)

GPU
Extreme power-efficiency (throughput work)



Xeon

Intel MIC

(AMD Fusion too)

- Can't optimize for both single thread performance and power efficiency
- Must do most work with cores optimized for **extreme energy efficiency**
- Still need a few cores optimized for **fast serial work**

Steve Scott – Cray Technical Workshop Oct 2012

Future Architectural Directions

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

Threading on the Node and Vectorization is becoming more important – need more parallelism on the node – just like Cyber 205 and Connection Machine

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
- User rewriting of loops is required for 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
  - 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 "state o the art" Xeons or Power 7
  - REQUIRES TOO MUCH POWER

Major architectural design that must be considered

- All systems will start having a secondary memory that is as large as we require; however, it will not have high bandwidth to the principal computational 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

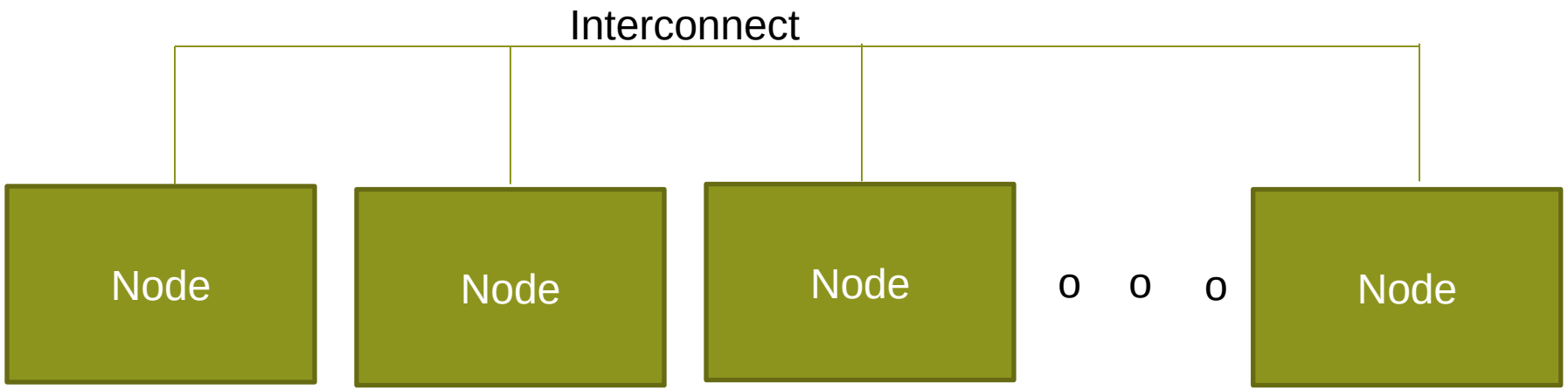
So how should we program for these new systems

- **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

So how should we program for these new systems

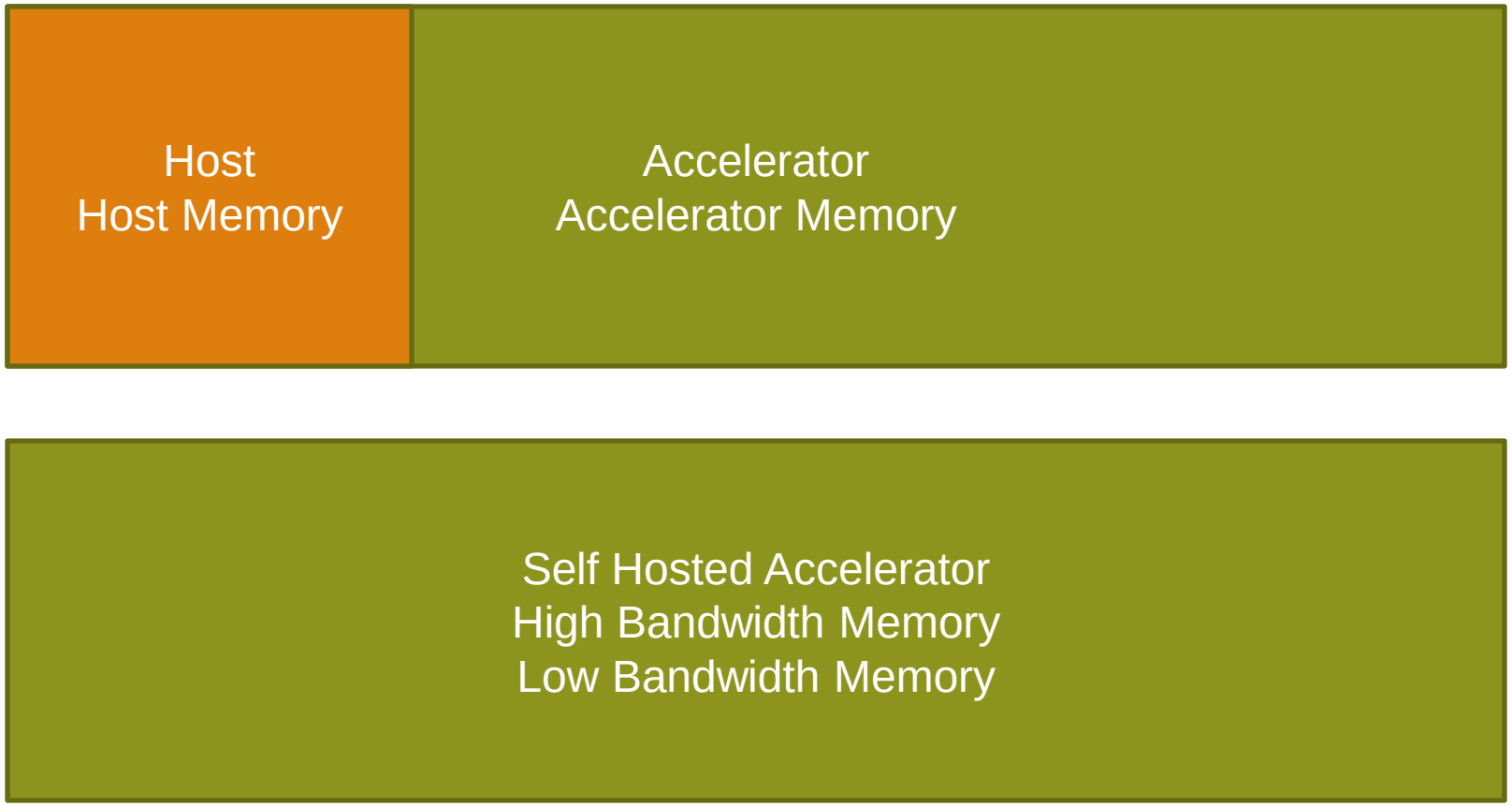
- **What to do – Good Threading (OpenMP)**
  - Must do high level threading
  - Thread must access close shared memory rather than distant shared memory
  - Load Balancing
- **What to do – Good Vectorization**
  - Vectorization advantage allows for introducing overhead to vectorize
    - Vectorization of Ifs
      - Conditional vector merge (too many paths??)
      - Gather/scatter (Too much data motion??)
      - Identification of strings

Future hardware trends

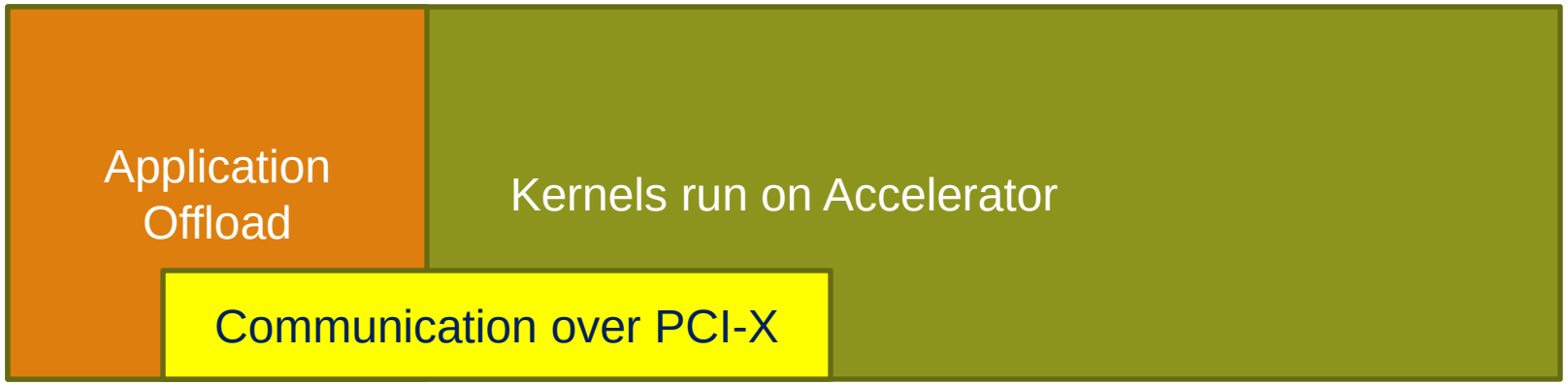


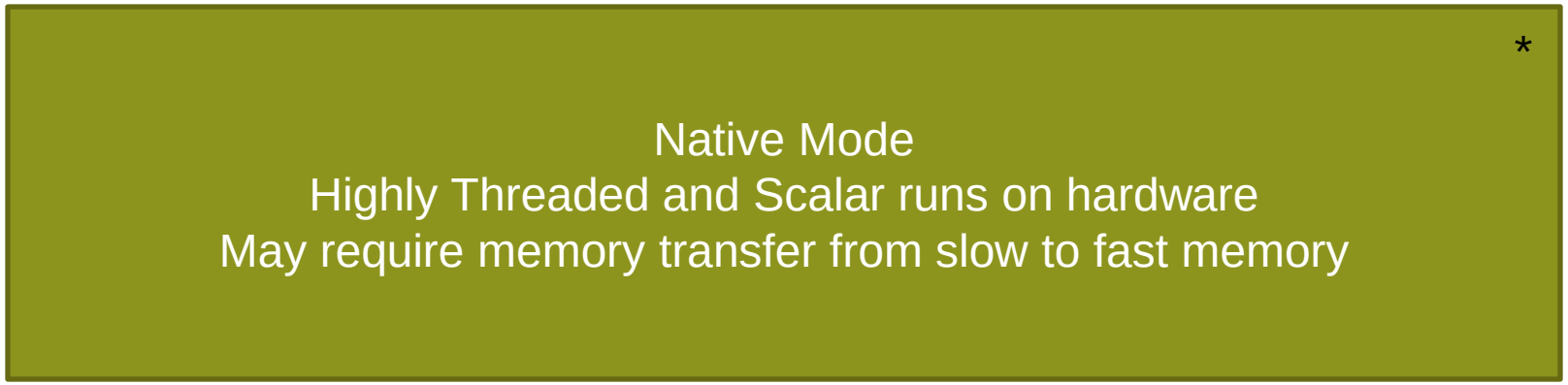
Everyone will be doing this

Node Architectures



Node Programming Paradigm's





* May include Multi-die multi-core node

Advantages & Disadvantages of the Node Architectures

	"hosted"	"self hosted"	Traditional Multi-core
Scalar Performance	Uses State of art Scalar Processor	Uses much slower scalar processor	Uses State of art Scalar Processor
Parallel Performance	Uses State of art Parallel Processor	Uses State of art Parallel Processor	Uses multi-core for performance
Data movement today	Significant Data Motion between Host/Accelerator	Minimal Data Motion	Minimal Data Motion
Data movement tomorrow	Significant Data Motion between Host/Accelerator	Significant Data Motion between Memory Levels	Minimal Data Motion
Amount of Memory	Limited on Accelerator	Limited fast Memory	Sufficient

Analyzing an application

- **What goes on within the time step loop?**
  - Where is computation
  - Where is communication
  - What data is used
- **What, if any computation can be moved?**
- **What, if any communication can be moved?**
- **Identification of potential overlap**
- **What about I/O**

Things we need to know about the application

- **Where are the major arrays allocated and how are they accessed**

  WHY – We need to understand how arrays can be allocated to assure most efficient access by major computational loops. (First touch, alignment, etc)

- **Where are the major computational and communication regions**

  WHY – We want to maintain a balance between computation and communication. How much time is spent in a computational region, what if any communication can be performed during that time

- **Where is the major I/O performed**

  WHY – Can we perform I/O asynchronously with computation/communication

Goals

- **Develop a single source code that implements OpenMP and OpenACC in such a way that application can be efficiently run on:**
  - Multi-core MPP systems
  - Multi-core MPP systems with companion accelerator
    - Nvida
    - Intel
    - AMD
    - Whatever
- **Clearly identify three levels of parallelism**
  - MPI/PGAS between NUMA/UMA memory regions
  - Threading within the NUMA/UMA memory region
    - How this is implemented is important – OpenMP/OpenACC is most portable
  - SIMDization at a low level
    - How this is coded is important – compilers have different capability
- **We do want a performance/portable application at the end**

Working through a simple example

- **Himeno**
  - This benchmark program takes measurements to proceed major loops in solving the Poisson's equation solution using the Jacobi iteration method.
  - Single computational loop
  - Easy to introduce OpenMP
  - Good to see how to implement OpenACC

Step 1

Profile the Application

Standard Profile

Table 1: Profile by Function Group and Function

Time%	Time	Imb. Time	Imb. Time%	Calls	Group / Function PE=HIDE
100.0%	66.812060	--	--	2372.1	Total
100.0%	66.811928	--	--	2167.0	USER
94.2%	62.965959	2.805426	4.3%	2.0	jacobi_.LOOPS
2.1%	1.385449	3.224803	71.1%	539.0	sendp1_
1.7%	1.167354	3.333783	75.2%	539.0	sendp2_
0.7%	0.461089	0.847765	65.8%	539.0	sendp3_
0.7%	0.455975	0.009942	2.2%	1.0	initmt_.LOOPS
0.4%	0.295538	0.302255	51.4%	1.0	initcomm_
0.1%	0.075184	0.015566	17.4%	1.0	himenobmtxp_
0.0%	0.005124	0.003026	37.7%	539.0	sendp_
0.0%	0.000094	0.000009	8.7%	1.0	initmax_.LOOPS
0.0%	0.000090	0.000010	9.7%	1.0	initmax_
0.0%	0.000042	0.000012	22.9%	1.0	exit
0.0%	0.000021	0.000005	20.2%	2.0	jacobi_
0.0%	0.000009	0.000005	37.5%	1.0	initmt_
0.0%	0.000088	--	--	2.0	MPI
0.0%	0.000087	0.000012	12.0%	1.0	mpi_finalize_
0.0%	0.000001	0.000000	22.0%	1.0	mpi_init_
0.0%	0.000044	--	--	203.1	ETC
0.0%	0.000043	0.000086	67.9%	203.1	__libc_csu_init
0.0%	0.000001	0.000068	100.0%	0.0	_STOP2

Loop Profile

Table 2: Loop Stats by Function (from -hprofile_generate)

Loop Incl Time Total	Loop Hit	Loop Trips Avg	Loop Trips Min	Loop Trips Max	Function=/.LOOP[.] PE=HIDE
65.985609	2	269.500	0	536	jacobi_.LOOP.1.li.206
52.004289	539	127.500	0	128	jacobi_.LOOP.2.li.209
51.999616	68722	127.500	0	128	jacobi_.LOOP.3.li.210
51.523908	8762119	255.500	0	256	jacobi_.LOOP.4.li.211
8.474166	539	127.500	0	128	jacobi_.LOOP.5.li.229
8.470748	68722	127.500	0	128	jacobi_.LOOP.6.li.230
7.510744	8762119	255.500	0	256	jacobi_.LOOP.7.li.231
0.302805	1	131	0	131	initmt_.LOOP.1.li.152
0.302796	131	131	0	131	initmt_.LOOP.2.li.153
0.302007	17161	259	0	259	initmt_.LOOP.3.li.154
0.153160	1	129.500	0	130	initmt_.LOOP.4.li.173
0.153145	129.500	129.500	0	130	initmt_.LOOP.5.li.174
0.152091	16770	257.500	0	258	initmt_.LOOP.6.li.175
0.000000	1	4	0	4	initmax_.LOOP.1.li.350
0.000000	1	4	0	4	initmax_.LOOP.3.li.365
0.000000	1	4	0	4	initmax_.LOOP.5.li.380
0.000000	1	4	0	4	initmax_.LOOP.6.li.387
0.000000	1	4	0	4	initmax_.LOOP.4.li.372
0.000000	1	4	0	4	initmax_.LOOP.2.li.357

Step 2

Use Reveal to scope the major computational loops

OpenMP directives on loop

```
! Directive inserted by Cray Reveal.  May be incomplete.
!$OMP  parallel do default(none)
!$OMP&   private (i,j,k,s0,ss)
!$OMP&   shared  (a,b,bnd,c,imax,jmax,kmax,omega,p,wrk1,wrk2)
!$OMP&   reduction (+:wgosa)
         DO K=2,kmax-1
            DO J=2,jmax-1
               DO I=2,imax-1
                  S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
     1                 +a(I,J,K,3)*p(I,J,K+1)
     2                 +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
     3                 -p(I-1,J+1,K)+p(I-1,J-1,K))
     4                 +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
     5                 -p(I,J+1,K-1)+p(I,J-1,K-1))
     6                 +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
     7                 -p(I+1,J,K-1)+p(I-1,J,K-1))
     8                 +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
     9                 +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
                  SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
                  WGOSA=WGOSA+SS*SS
                  wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
               enddo
            enddo
         enddo
```

OpenMP directives on loop – compiler listing

```
213.    1               ! Directive inserted by Cray Reveal.  May be incomplete.
214.  + 1                !$OMP  parallel do default(none)
215.    1                !$OMP&   private (i,j,k,s0,ss)
216.    1                !$OMP&   shared  (a,b,bnd,c,imax,jmax,kmax,omega,p,wrk1,wrk2)
217.    1                !$OMP&   reduction (+:wgosa)
218.  + 1 m----------<           DO K=2,kmax-1
219.  + 1 m 3--------<              DO J=2,jmax-1
220.    1 m 3 Vr3----<                 DO I=2,imax-1
221.    1 m 3 Vr3                         S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
222.    1 m 3 Vr3           1                  +a(I,J,K,3)*p(I,J,K+1)
223.    1 m 3 Vr3           2                  +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
224.    1 m 3 Vr3           3                  -p(I-1,J+1,K)+p(I-1,J-1,K))
225.    1 m 3 Vr3           4                  +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
226.    1 m 3 Vr3           5                  -p(I,J+1,K-1)+p(I,J-1,K-1))
227.    1 m 3 Vr3           6                  +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
228.    1 m 3 Vr3           7                  -p(I+1,J,K-1)+p(I-1,J,K-1))
229.    1 m 3 Vr3           8                  +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
230.    1 m 3 Vr3           9                  +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
231.    1 m 3 Vr3                         SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
232.    1 m 3 Vr3                         WGOSA=WGOSA+SS*SS
233.    1 m 3 Vr3                         wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
234.    1 m 3 Vr3---->                 enddo
235.    1 m 3-------->              enddo
236.    1 m---------->           enddo
```

OpenMP Version 1 versus Version 2

- Version 1 only had the OpenMP on the major kernel
- Version 2 had OpenMP on initialization
  - Helps on memory layout
- Version 2 had OpenMP on copy loop

Step 3

Introduce OpenACC

This should only be done once a very good OpenMP/MPI application is in hand

Introducing OpenACC

```
#ifdef GPU
!$ACC  parallel loop
!$ACC&   private (i,j,k,s0,ss)
!$ACC&   reduction (+:wgosa)
#else
!$OMP  parallel do default(none)
!$OMP&   private (i,j,k,s0,ss)
!$OMP&   shared  (a,b,bnd,c,imax,jmax,kmax,omega,p,wrk1,wrk2)
!$OMP&   reduction (+:wgosa)
#endif
         DO K=2,kmax-1
            DO J=2,jmax-1
               DO I=2,imax-1
                  S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
     1                 +a(I,J,K,3)*p(I,J,K+1)
     2                 +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
     3                 -p(I-1,J+1,K)+p(I-1,J-1,K))
     4                 +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
     5                 -p(I,J+1,K-1)+p(I,J-1,K-1))
     6                 +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
     7                 -p(I+1,J,K-1)+p(I-1,J,K-1))
     8                 +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
     9                 +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
                  SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
                  WGOSA=WGOSA+SS*SS
                  wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
               enddo
            enddo
         enddo
#ifdef GPU
!$ACC  end parallel loop
#endif
```

Introducing OpenACC –compiler listing

```
214.    1                #ifdef GPU
 215.  + 1 G------------< !$ACC  parallel loop
 216.    1 G              !$ACC&   private (i,j,k,s0,ss)
 217.    1 G              !$ACC&   reduction (+:wgosa)
 218.    1 G              #else
 219.    1 G              !$OMP  parallel do default(none)
 220.    1 G              !$OMP&   private (i,j,k,s0,ss)
 221.    1 G              !$OMP&   shared  (a,b,bnd,c,imax,jmax,kmax,omega,p,wrk1,wrk2)
 222.    1 G              !$OMP&   reduction (+:wgosa)
 223.    1 G              #endif
 224.    1 G g----------<           DO K=2,kmax-1
 225.  + 1 G g 4--------<              DO J=2,jmax-1
 226.    1 G g 4 g------<                 DO I=2,imax-1
 227.    1 G g 4 g                           S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
 228.    1 G g 4 g             1                  +a(I,J,K,3)*p(I,J,K+1)
 229.    1 G g 4 g             2                  +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
 230.    1 G g 4 g             3                  -p(I-1,J+1,K)+p(I-1,J-1,K))
 231.    1 G g 4 g             4                  +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
 232.    1 G g 4 g             5                  -p(I,J+1,K-1)+p(I,J-1,K-1))
 233.    1 G g 4 g             6                  +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
 234.    1 G g 4 g             7                  -p(I+1,J,K-1)+p(I-1,J,K-1))
 235.    1 G g 4 g             8                  +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
 236.    1 G g 4 g             9                  +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
 237.    1 G g 4 g                           SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
 238.    1 G g 4 g                           WGOSA=WGOSA+SS*SS
 239.    1 G g 4 g                           wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
 240.    1 G g 4 g------>                 enddo
 241.    1 G g 4-------->              enddo
 242.    1 G g---------->           enddo
 243.    1 G              #ifdef GPU
 244.    1 G------------> !$ACC  end parallel loop
 245.    1                #endif
```

Introducing OpenACC –compiler listing

```
ftn-6405 ftn: ACCEL File = himenoBMTxpr.f, Line = 215
  A region starting at line 215 and ending at line 244 was placed on the accelerator.

ftn-6418 ftn: ACCEL File = himenoBMTxpr.f, Line = 215
  If not already present: allocate memory and copy whole array "p" to accelerator, free at line 244 (acc_copyin).

ftn-6418 ftn: ACCEL File = himenoBMTxpr.f, Line = 215
  If not already present: allocate memory and copy whole array "a" to accelerator, free at line 244 (acc_copyin).

ftn-6418 ftn: ACCEL File = himenoBMTxpr.f, Line = 215
  If not already present: allocate memory and copy whole array "b" to accelerator, free at line 244 (acc_copyin).

ftn-6418 ftn: ACCEL File = himenoBMTxpr.f, Line = 215
  If not already present: allocate memory and copy whole array "c" to accelerator, free at line 244 (acc_copyin).

ftn-6418 ftn: ACCEL File = himenoBMTxpr.f, Line = 215
  If not already present: allocate memory and copy whole array "wrk1" to accelerator, free at line 244 (acc_copyin).

ftn-6418 ftn: ACCEL File = himenoBMTxpr.f, Line = 215
  If not already present: allocate memory and copy whole array "bnd" to accelerator, free at line 244 (acc_copyin).

ftn-6416 ftn: ACCEL File = himenoBMTxpr.f, Line = 215
  If not already present: allocate memory and copy whole array "wrk2" to accelerator, copy back at line 244 (acc_copy

ftn-6415 ftn: ACCEL File = himenoBMTxpr.f, Line = 215
  Allocate memory and copy variable "wgosa" to accelerator, copy back at line 244 (acc_copy).

ftn-6430 ftn: ACCEL File = himenoBMTxpr.f, Line = 224
  A loop starting at line 224 was partitioned across the thread blocks.
```

OpenACC Version 1 – Profile output

```
Table 1:  Profile by Function Group and Function

 Time%  |      Time  |     Imb.  |  Imb.  | Calls  |Group
        |            |     Time  | Time%  |        | Function
        |            |           |        |        |  PE=HIDE
        |            |           |        |        |   Thread=HIDE

 100.0% | 62.073674 |       -- |     -- | 7627.1 |Total
|-------------------------------------------------------------------------------
|  95.6% | 59.368154 |       -- |     -- | 2565.0 |USER
||------------------------------------------------------------------------------
||  81.0% | 50.284595 | 1.058725 |   2.1% |  284.0 |jacobi_.ACC_COPY@li.215
||   3.8% |  2.359195 | 0.094854 |   3.9% |  284.0 |jacobi_.ACC_COPY@li.244
||   3.3% |  2.055168 | 0.131544 |   6.1% |    2.0 |jacobi_
||   2.6% |  1.616893 | 0.457704 |  22.4% |  284.0 |jacobi_.ACC_SYNC_WAIT@li.244
||   2.5% |  1.535406 | 0.280701 |  15.7% |    1.0 |exit
||   2.3% |  1.410941 | 0.284518 |  17.0% |    1.0 |initmt_.LOOP@li.156
||   0.1% |  0.054959 | 0.001335 |   2.4% |    1.0 |initmt_
||   0.1% |  0.039723 | 0.005092 |  11.5% |  284.0 |jacobi_.ACC_ASYNC_KERNEL@li.215
||   0.0% |  0.003376 | 0.003567 |  52.2% |  284.0 |jacobi_.ACC_REGION@li.215
||   0.0% |  0.002758 | 0.000394 |  12.7% |  284.0 |sendp3_
||   0.0% |  0.002172 | 0.000380 |  15.1% |  284.0 |sendp_
||   0.0% |  0.001459 | 0.000189 |  11.7% |  284.0 |sendp1_
||   0.0% |  0.001155 | 0.000109 |   8.8% |  284.0 |sendp2_
||   0.0% |  0.000276 | 0.000590 |  69.2% |    1.0 |himenobmtxp_
||   0.0% |  0.000047 | 0.000007 |  13.3% |    1.0 |initcomm_
||   0.0% |  0.000025 | 0.000005 |  15.9% |    1.0 |initmax_
||   0.0% |  0.000006 | 0.000018 |  74.8% |    1.0 |initmt_.REGION@li.156
.
```

Steps 4, 5,6,7,8

Once one loop is analyzed, now look at next highest compute loop, perform steps 2 and 3

Moving to Version 2

- **Introduce !$acc data region in main routine**
  - Move data to the device and initialize data on the device
  - Now must move data back to the host for the message passing

OpenACC Version 2 – Using Data regions –a)

```
#ifdef GPU
!$acc data present(a,p,b,c,bnd,wrk1)present_or_create(wgosa)
#endif
C
      DO loop=1,nn
         gosa=0.0
         wgosa=0.0
! Directive inserted by Cray Reveal.  May be incomplete.
#ifdef GPU
!$ACC  parallel loop
!$ACC&   private (i,j,k,s0,ss)
!$ACC&   reduction (+:wgosa)
#else
!$OMP  parallel do default(none)
!$OMP&   private (i,j,k,s0,ss)
!$OMP&   shared  (a,b,bnd,c,imax,jmax,kmax,omega,p,wrk1,wrk2)
!$OMP&   reduction (+:wgosa)
#endif
         DO K=2,kmax-1
            DO J=2,jmax-1
               DO I=2,imax-1
                  S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
     1                 +a(I,J,K,3)*p(I,J,K+1)
     2                 +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
     3                 -p(I-1,J+1,K)+p(I-1,J-1,K))
     4                 +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
     5                 -p(I,J+1,K-1)+p(I,J-1,K-1))
     6                 +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
     7                 -p(I+1,J,K-1)+p(I-1,J,K-1))
     8                 +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
     9                 +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
                  SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
                  WGOSA=WGOSA+SS*SS
                  wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
               enddo
            enddo
         enddo
#ifdef GPU
!$ACC  end parallel loop
#endif
```

OpenACC Version 2 – Using Data regions –b)

```
C
#ifdef GPU
!$ACC  parallel loop
!$ACC&   private (i,j,k)
#else
!$OMP  parallel do default(none)
!$OMP&   private (i,j,k)
!$OMP&   shared  (p,wrk2)
#endif
         DO K=2,kmax-1
            DO J=2,jmax-1
               DO I=2,imax-1
                  p(I,J,K)=wrk2(I,J,K)
               enddo
            enddo
         enddo
#ifdef GPU
!$ACC  end parallel loop
!$acc update host(p,wgosa)
#endif
C
         call sendp(ndx,ndy,ndz)
C
#ifdef GPU
!$acc update device(p)
#endif
         call mpi_allreduce(wgosa,
     >                      gosa,
     >                      1,
     >                      mpi_real4,
     >                      mpi_sum,
     >                      mpi_comm_world,
     >                      ierr)
C
      enddo
#ifdef GPU
!$acc end data
#endif
```

OpenACC Version 2 – Using Data regions –a)

```
240.  + G------------< !$acc data present(a,p,b,c,bnd,wrk1)present_or_create(wgosa)
 241.    G              #endif
 242.    G              C
 243.  + G 2----------<       DO loop=1,nn
 244.    G 2                     gosa=0.0
 245.    G 2                     wgosa=0.0
 246.    G 2            ! Directive inserted by Cray Reveal.  May be incomplete.
 247.    G 2            #ifdef GPU
 248.  + G 2 G--------< !$ACC  parallel loop
 249.    G 2 G          !$ACC&   private (i,j,k,s0,ss)
 250.    G 2 G          !$ACC&   reduction (+:wgosa)
 251.    G 2 G          #else
 252.    G 2 G          !$OMP  parallel do default(none)
 253.    G 2 G          !$OMP&   private (i,j,k,s0,ss)
 254.    G 2 G          !$OMP&   shared  (a,b,bnd,c,imax,jmax,kmax,omega,p,wrk1,wrk2)
 255.    G 2 G          !$OMP&   reduction (+:wgosa)
 256.    G 2 G          #endif
 257.    G 2 G g------<           DO K=2,kmax-1
 258.  + G 2 G g 5----<             DO J=2,jmax-1
 259.    G 2 G g 5 g--<               DO I=2,imax-1
 260.    G 2 G g 5 g                    S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
 261.    G 2 G g 5 g         1               +a(I,J,K,3)*p(I,J,K+1)
 262.    G 2 G g 5 g         2               +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
 263.    G 2 G g 5 g         3               -p(I-1,J+1,K)+p(I-1,J-1,K))
 264.    G 2 G g 5 g         4               +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
 265.    G 2 G g 5 g         5               -p(I,J+1,K-1)+p(I,J-1,K-1))
 266.    G 2 G g 5 g         6               +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
 267.    G 2 G g 5 g         7               -p(I+1,J,K-1)+p(I-1,J,K-1))
 268.    G 2 G g 5 g         8               +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
 269.    G 2 G g 5 g         9               +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
 270.    G 2 G g 5 g                    SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
 271.    G 2 G g 5 g                    WGOSA=WGOSA+SS*SS
 272.    G 2 G g 5 g                    wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
 273.    G 2 G g 5 g-->               enddo
 274.    G 2 G g 5---->             enddo
 275.    G 2 G g------>           enddo
 276.    G 2 G          #ifdef GPU
 277.    G 2 G--------> !$ACC  end parallel loop.
```

OpenACC Version 2 – Using Data regions –b)

```
279.    G 2            C
 280.    G 2            #ifdef GPU
 281.  + G 2 G--------< !$ACC  parallel loop
 282.    G 2 G          !$ACC&   private (i,j,k)
 283.    G 2 G          #else
 284.    G 2 G          !$OMP  parallel do default(none)
 285.    G 2 G          !$OMP&   private (i,j,k)
 286.    G 2 G          !$OMP&   shared  (p,wrk2)
 287.    G 2 G          #endif
 288.    G 2 G g------<           DO K=2,kmax-1
 289.  + G 2 G g 5----<              DO J=2,jmax-1
 290.    G 2 G g 5 g--<                 DO I=2,imax-1
 291.    G 2 G g 5 g                      p(I,J,K)=wrk2(I,J,K)
 292.    G 2 G g 5 g-->                 enddo
 293.    G 2 G g 5---->              enddo
 294.    G 2 G g------>           enddo
 295.    G 2 G          #ifdef GPU
 296.    G 2 G--------> !$ACC  end parallel loop
 297.    G 2            !$acc update host(p,wgosa)
 298.    G 2            #endif
 299.    G 2            C
 300.  + G 2                     call sendp(ndx,ndy,ndz)
 301.    G 2            C
 302.    G 2            #ifdef GPU
 303.    G 2            !$acc update device(p)
 304.    G 2            #endif
 305.  + G 2                     call mpi_allreduce(wgosa,
 306.    G 2                  >                     gosa,
 307.    G 2                  >                     1,
 308.    G 2                  >                     mpi_real4,
 309.    G 2                  >                     mpi_sum,
 310.    G 2                  >                     mpi_comm_world,
 311.    G 2                  >                     ierr)
 312.    G 2            C
 313.    G 2---------->       enddo
 314.    G              #ifdef GPU
 315.    G------------> !$acc end data.
```

OpenACC Version 2 – Using Data regions –c)

```
ftn-6413 ftn: ACCEL File = himenoBMTxpr.f, Line = 240
  A data region was created at line 240 and ending at line 315.

ftn-6422 ftn: ACCEL File = himenoBMTxpr.f, Line = 240
  If not already present: allocate memory for variable "wgosa" on accelerator, free at line 315 (acc_share).

ftn-6288 ftn: VECTOR File = himenoBMTxpr.f, Line = 243
  A loop starting at line 243 was not vectorized because it contains a call to subroutine "sendp" on line 300.

ftn-6405 ftn: ACCEL File = himenoBMTxpr.f, Line = 248
  A region starting at line 248 and ending at line 277 was placed on the accelerator.

ftn-6416 ftn: ACCEL File = himenoBMTxpr.f, Line = 248
  If not already present: allocate memory and copy whole array "wrk2" to accelerator, copy back at line 277 (acc_copy

ftn-6430 ftn: ACCEL File = himenoBMTxpr.f, Line = 257
  A loop starting at line 257 was partitioned across the thread blocks.

ftn-6509 ftn: ACCEL File = himenoBMTxpr.f, Line = 258
  A loop starting at line 258 was not partitioned because a better candidate was found at line 259.

ftn-6412 ftn: ACCEL File = himenoBMTxpr.f, Line = 258
  A loop starting at line 258 will be redundantly executed.

ftn-6430 ftn: ACCEL File = himenoBMTxpr.f, Line = 259
  A loop starting at line 259 was partitioned across the 128 threads within a threadblock.

ftn-6405 ftn: ACCEL File = himenoBMTxpr.f, Line = 281
  A region starting at line 281 and ending at line 296 was placed on the accelerator.

ftn-6418 ftn: ACCEL File = himenoBMTxpr.f, Line = 281
  If not already present: allocate memory and copy whole array "wrk2" to accelerator, free at line 296 (acc_copyin).

ftn-6430 ftn: ACCEL File = himenoBMTxpr.f, Line = 288
  A loop starting at line 288 was partitioned across the thread blocks.
.
```

OpenACC Version 2 – Profile

```
Table 1:  Profile by Function Group and Function

 Time%  |     Time  |    Imb.   |  Imb.   |  Calls  |Group
        |           |    Time   | Time%   |         | Function
        |           |           |         |         |  PE=HIDE

 100.0% | 35.679931 |       -- |     -- | 36436.1 |Total
|-------------------------------------------------------------------------------
|  92.2% | 32.914300 |       -- |     -- | 19121.0 |USER
||------------------------------------------------------------------------------
||  34.9% | 12.445154 | 0.259110 |   2.1% |  1005.0 |jacobi_.ACC_COPY@li.303
||  31.5% | 11.256881 | 0.247093 |   2.2% |  1005.0 |jacobi_.ACC_COPY@li.297
||  17.1% |  6.089251 | 0.845903 |  12.4% |  1005.0 |jacobi_.ACC_SYNC_WAIT@li.277
||   8.2% |  2.916580 | 1.242691 |  30.4% |  1005.0 |jacobi_.ACC_SYNC_WAIT@li.296
||   0.2% |  0.076624 | 0.010203 |  11.9% |  1005.0 |jacobi_.ACC_ASYNC_KERNEL@li.248
||   0.1% |  0.049299 | 0.006400 |  11.7% |  1005.0 |jacobi_.ACC_ASYNC_KERNEL@li.281
||   0.1% |  0.019482 | 0.008660 |  31.3% |  1005.0 |jacobi_.ACC_COPY@li.248
||   0.0% |  0.009641 | 0.001109 |  10.5% |     2.0 |jacobi_.ACC_DATA_REGION@li.240
||   0.0% |  0.007366 | 0.000622 |   7.9% |  1005.0 |sendp3_
||   0.0% |  0.005379 | 0.000703 |  11.7% |  1005.0 |jacobi_.ACC_REGION@li.248
||   0.0% |  0.004826 | 0.000634 |  11.8% |  1005.0 |sendp1_
||   0.0% |  0.004560 | 0.000721 |  13.9% |  1005.0 |jacobi_.ACC_UPDATE@li.303
||   0.0% |  0.004302 | 0.000850 |  16.8% |  1005.0 |jacobi_.ACC_SYNC_WAIT@li.297
||   0.0% |  0.004042 | 0.000214 |   5.1% |  1005.0 |sendp_
||   0.0% |  0.003855 | 0.000379 |   9.1% |  1005.0 |sendp2_
.
```

Steps 9, 10,11, 12

Optimizing data transfers

Moving to Version 3

- **Pack MPI buffers on the accelerator**
  - Difficult here because MPI data types are used
  - Significantly reduce the data moved back and forth

OpenACC Version 3 – a)

```
C
#ifdef GPU
!$ACC  parallel loop
!$ACC&   private (i,j,k)
#else
!$OMP  parallel do default(none)
!$OMP&   private (i,j,k)
!$OMP&   shared  (p,wrk2)
#endif
         DO K=2,kmax-1
            DO J=2,jmax-1
               DO I=2,imax-1
                  p(I,J,K)=wrk2(I,J,K)
               enddo
            enddo
         enddo
#ifdef GPU
!$ACC  end parallel loop
#endif
#ifdef GPU
!$ACC  parallel loop
!$ACC&   private (ii,j,k)
#else
!$OMP  parallel do
!$OMP&   private (ii,j,k)
!$OMP&   shared  (p,wrk2)
#endif
         DO K=1,kmax
            DO J=1,jmax
                ii = j + (k-1)*jmax
                  pack_p11(ii)=p(2,j,k)
                  pack_p12(ii)=p(imax-1,j,k)
            enddo
         enddo
#ifdef GPU
!$ACC  end parallel loop
#endif
_
```

```
ifdef GPU
!$ACC  parallel loop
!$ACC&   private (i,jj,k)
#else
!$OMP  parallel do
!$OMP&   private (i,jj,k)
#endif
         DO K=1,kmax
            DO I=1,imax
                jj = i +(k-1)*imax
                  pack_p21(jj)=p(i,2,k)
                  pack_p22(jj)=p(i,jmax-1,k)
            enddo
         enddo
#ifdef GPU
!$ACC  end parallel loop
#endif
#ifdef GPU
!$ACC  parallel loop
!$ACC&   private (i,j,kk)
#else
!$OMP  parallel do
!$OMP&   private (i,j,kk)
#endif
         DO J=1,jmax
            DO I=1,imax
                kk = i + (j-1)*imax
                  pack_p31(kk)=p(i,j,2)
                  pack_p32(kk)=p(i,j,kmax-1)
            enddo
         enddo
#ifdef GPU
!$ACC  end parallel loop
#endif
#ifdef GPU
!$acc update host(pack_p11,pack_p12,pack_p21,pack_p22,
!$acc&              pack_p31,pack_p32,wgosa)
#endif
C
         call sendp(ndx,ndy,ndz,
     & pack_p11,pack_p12,pack_p21,pack_p22,
     & pack_p31,pack_p32,unpack_p11,unpack_p12,
     & unpack_p21,unpack_p22,unpack_p31,unpack_p32)
C
C
```

OpenACC Version 3 – b)

```
#ifdef GPU
!$acc update device(unpack_p11,unpack_p12,unpack_p21,unpack_p22,
!$acc&              unpack_p31,unpack_p32)
#endif
C
#ifdef GPU
!$ACC  parallel loop
!$ACC&   private (ii,j,k)
#else
!$OMP  parallel do
!$OMP&   private (ii,j,k)
!$OMP&   shared  (p,wrk2)
#endif
         DO K=1,kmax
            DO J=1,jmax
                ii = j + (k-1)*jmax
                  p(2,j,k)=unpack_p11(ii)
                  p(imax-1,j,k)=unpack_p12(ii)
            enddo
         enddo
#ifdef GPU
!$ACC  end parallel loop
#endif
#ifdef GPU
!$ACC  parallel loop
!$ACC&   private (i,jj,k)
#else
!$OMP  parallel do
!$OMP&   private (i,jj,k)
#endif
         DO K=1,kmax
            DO I=1,imax
                jj = i + (k-1)*imax
                  p(i,2,k)=unpack_p21(jj)
                  p(i,jmax-1,k)=unpack_p32(jj)
            enddo
         enddo
#ifdef GPU
!$ACC  end parallel loop
#endif
```

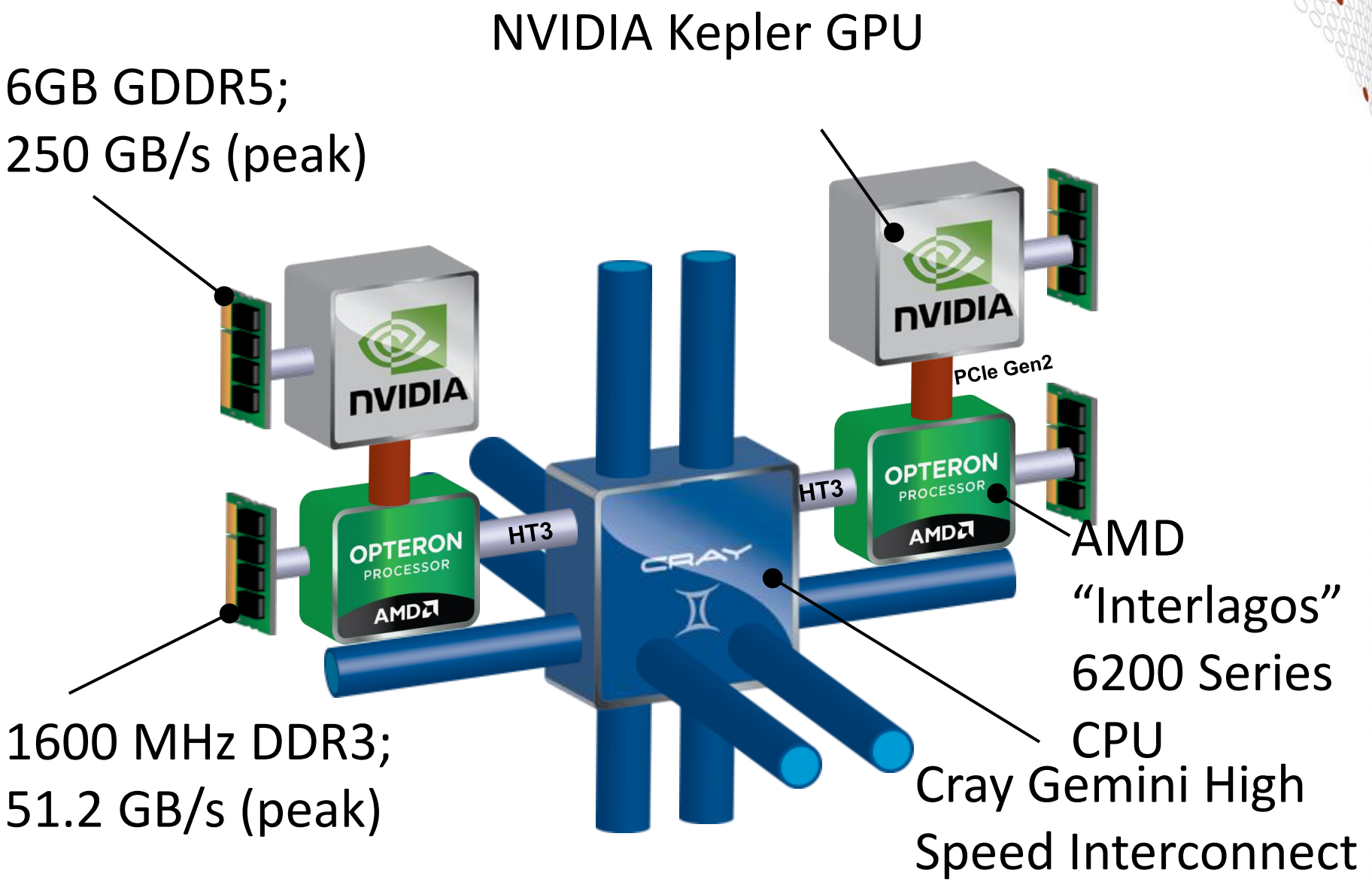
```
#ifdef GPU
!$ACC  parallel loop
!$ACC&   private (i,jj,k)
#else
!$OMP  parallel do
!$OMP&   private (i,jj,k)
#endif
         DO K=1,kmax
            DO I=1,imax
                jj = i + (k-1)*imax
                  p(i,2,k)=unpack_p21(jj)
                  p(i,jmax-1,k)=unpack_p22(jj)
            enddo
         enddo
#ifdef GPU
!$ACC  end parallel loop
#endif
#ifdef GPU
!$ACC  parallel loop
!$ACC&   private (i,j,kk)
#else
!$OMP  parallel do
!$OMP&   private (i,j,kk)
#endif
         DO J=1,jmax
            DO I=1,imax
                kk = i +(j-1)*imax
                  p(i,j,2)=unpack_p31(kk)
                  p(i,j,kmax-1)=unpack_p32(kk)
            enddo
         enddo
#ifdef GPU
!$ACC  end parallel loop
#endif
)
c
C
—
.
```

Comparisons of Himeno Versions

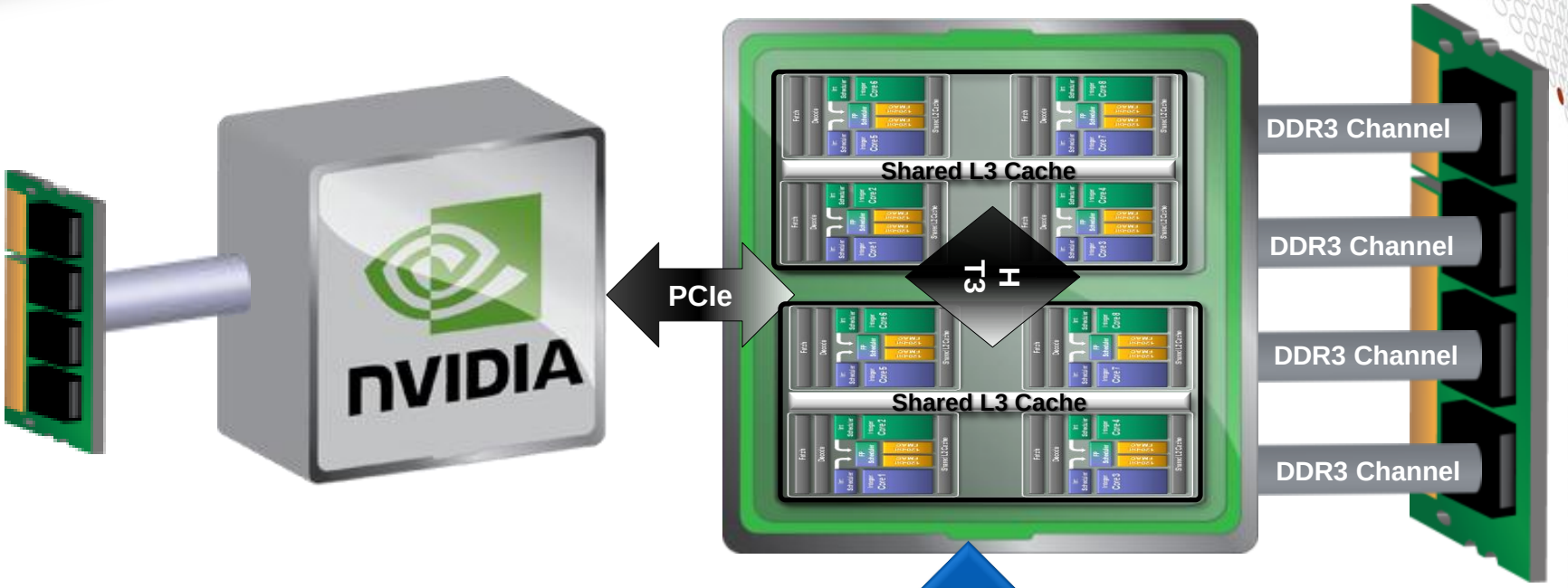
Version of Himeno	GFLOPS
Original running on 16 nodes 4 MPI/Node	78.75
OMP version1 running on 16 nodes 4 MPI/Node 4 threads/MPI task	169.5
OMP version2 running on 16 nodes 4 MPI/Node 4 threads/MPI task	180.9
OpenACC version 1 running on 16 nodes 4 MPI/Node 1 GPU	45.991
OpenACC version 2 running on 16 nodes 4 MPI/Node 1 GPU	256.9
OpenACC version 3 running on 16 nodes 4 MPI/Node 1 GPU	593.3
Last Version running in OpenMP mode	180.99

Cray XK7 Architecture

Cray XK7 Architecture



XK7 Node Details



- **1 Interlagos Processor, 2 Dies**
  - 8 “Compute Units”
  - 8 256-bit FMAC Floating Point Units
  - 16 Integer Cores
- **4 Channels of DDR3 Bandwidth to 4 DIMMs**
- **1 Nvidia Kepler Accelerator**
  - Connected via PCIe Gen 2

AMD Interlagos Single vs. Dual-Stream

- **Dual-stream mode allows for 16 threads of execution per CPU**
  - 16 MPI ranks
  - 16 OpenMP threads
  - Some combination between
- **Two threads share a 256-bit FPU**
  - Single FP scheduler determines how best to share
- **This is aprun's default behavior on most systems.**

- **Single-stream mode places 1 thread of execution per compute unit (maximum 8)**
  - 8 MPI ranks
  - 8 OpenMP threads
  - Some combination between
- **Each thread fully owns a 256-bit FPU**
  - AVX256 instructions required
- **This mode has same peak FP and memory performance**
  - 2X FLOPS & Bandwidth per thread
- **This can be enabled in aprun with –j1 flag**

AMD Interlagos Single vs. Dual-Stream

- **Dual-stream mode allows for 16 threads of execution per CPU**
  - 16 MPI ranks
  - 16 OpenMP threads
  - Some c…
- **Two threads sh… FPU**
  - Single F… how best to sha…
- **This is aprun … default behavior on most sys…ems.**

- **Single-stream mode places 1 …read of …xecution per …unit (m…imum 8)**
  - …reads
  - …tion between
- **…lly owns a …P**
  - …56 instructions required
- **This …ode has same peak FP and memory performance**
  - 2X FLOPS & Bandwidth per thread
- **This can be enabled in aprun with –j2 flag**

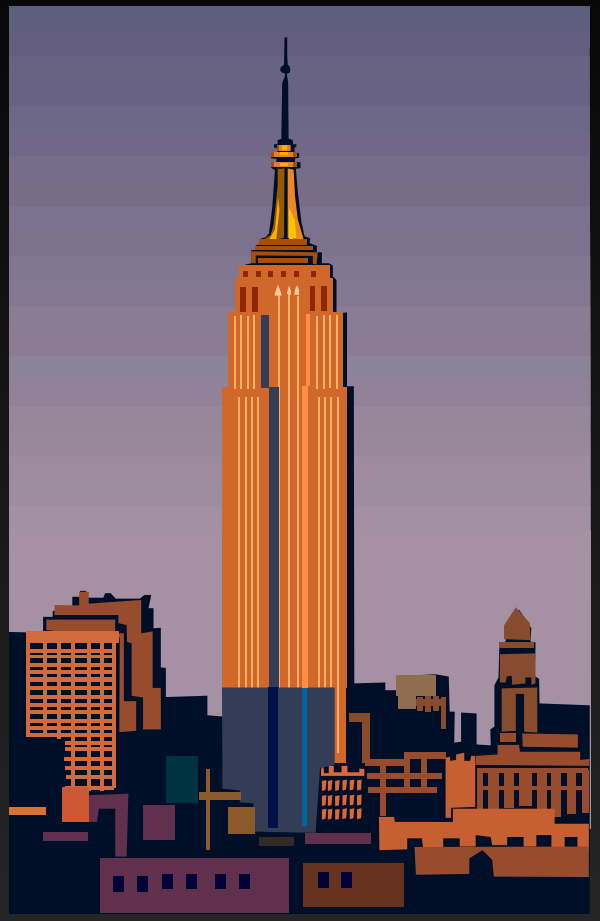
You have to experiment for yourself.


How to Think Like a
GPU
NVIDIA

You've been hired to paint a building

You've been hired to paint a building

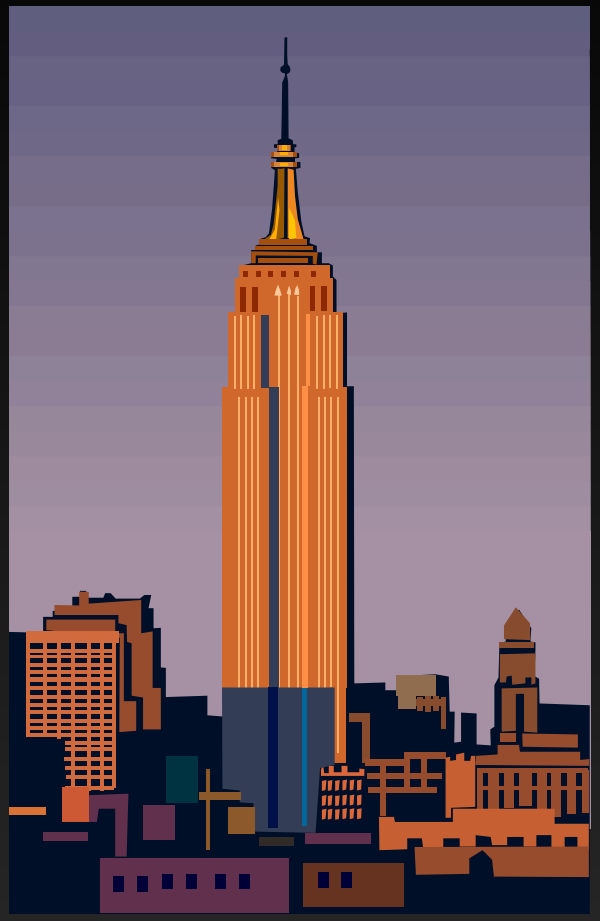
(A Big Building)



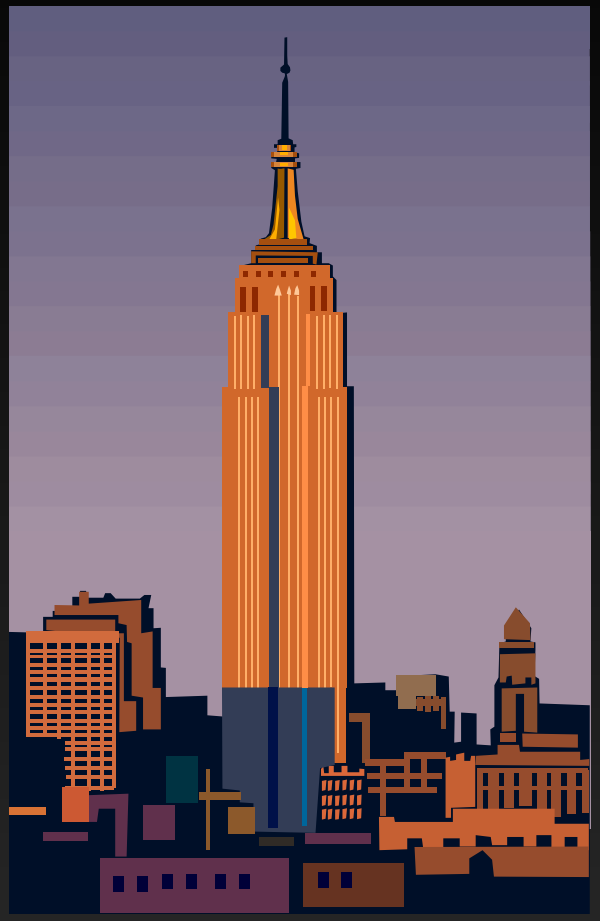
How can 1 painter paint faster?

1. Paint faster
   - One person's arm can only move so fast
2. Paint wider
   - A wider roller will cover more area, but rollers can only be made so wide
3. Minimize trips to paint bucket
   - A paint tray can be kept close by, but it can only realistically be so big

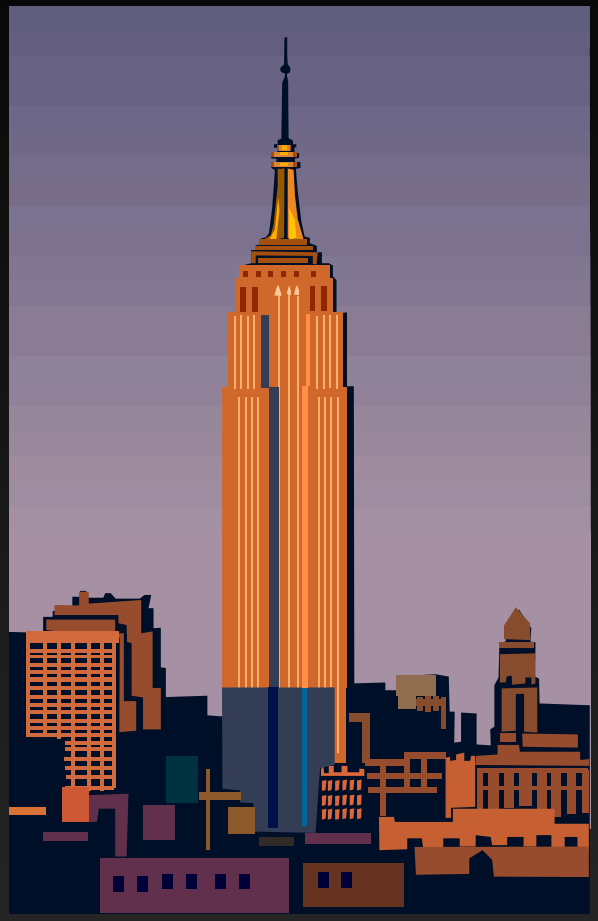
In order to paint it quickly, you keep your roller and paint close by and roll as quickly as possible



But, there's a limit to how quickly you can roll and how much paint you can keep near by.

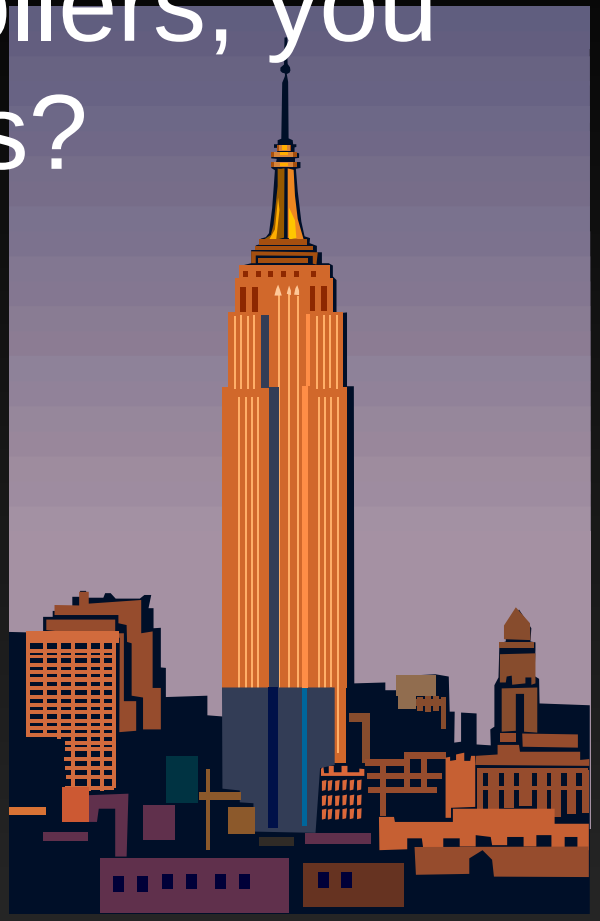


So you hire some help.



A well-organized team can paint nearly 4X faster.

What if, instead of buying more paint cans and wider rollers, you hire even more painters?



Now each painter is slower, but…

If we have enough painters, there will always be someone painting, so this won't matter.

Thread Performance vs. Throughput

- CPUs optimize for maximum performance from each thread.
  - Fast clocks
  - Big caches

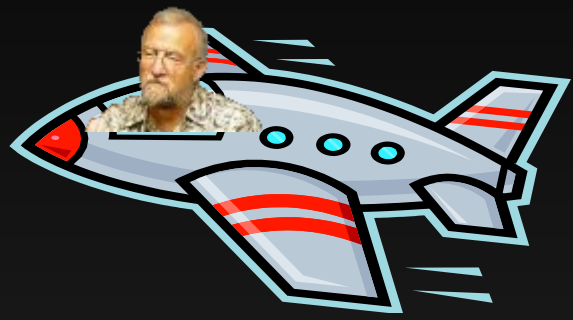
- GPUs optimize for maximum throughput.
  - Slower threads and smaller caches
  - Lots of threads active at once.

Another Example

Latency vs. Throughput

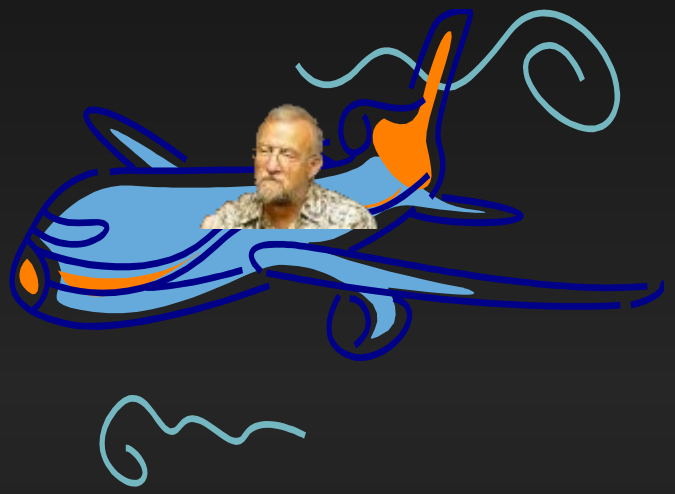
F-22 Rapter

- 1500 mph
- Knoxville to San Francisco in 1:25
- Seats 1



Boeing 737

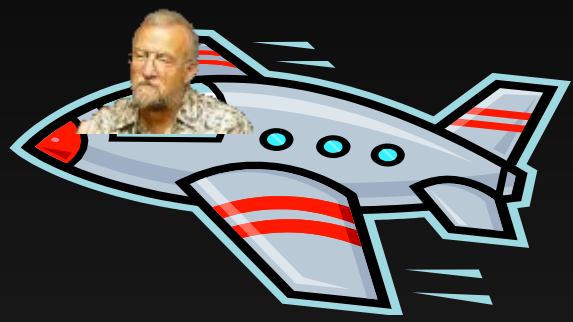
- 485 mph
- Knoxville to San Francisco in 4:20
- Seats 200



Latency vs. Throughput

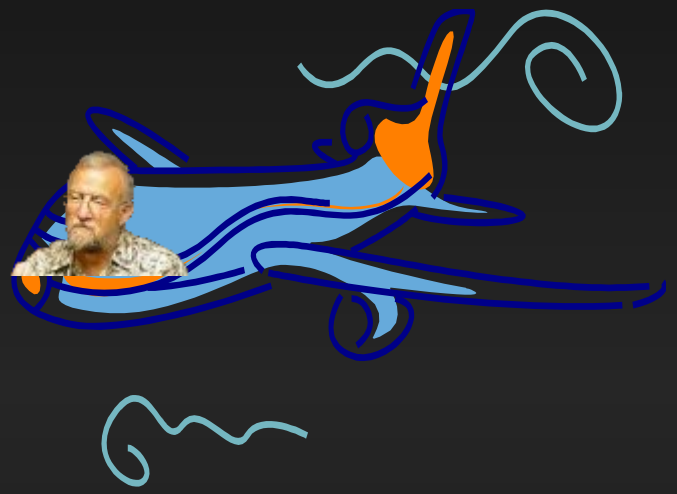
F-22 Rapter

- Latency – 1:25
- Throughput – 1 / 1.42 hours = 0.7 people/hr.



Boeing 737

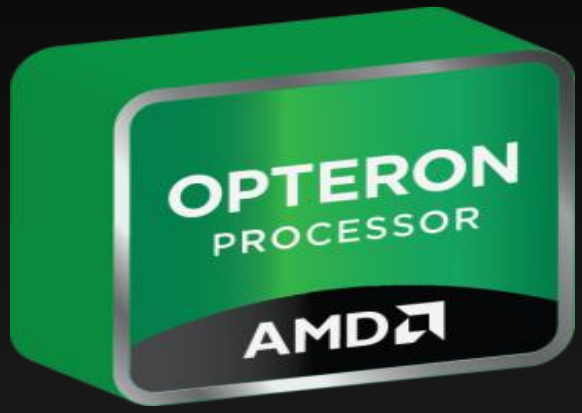
- Latency – 4:20
- Throughput – 200 / 4.33 hours = 46.2 people/hr.



Latency vs. Throughput

AMD Opteron

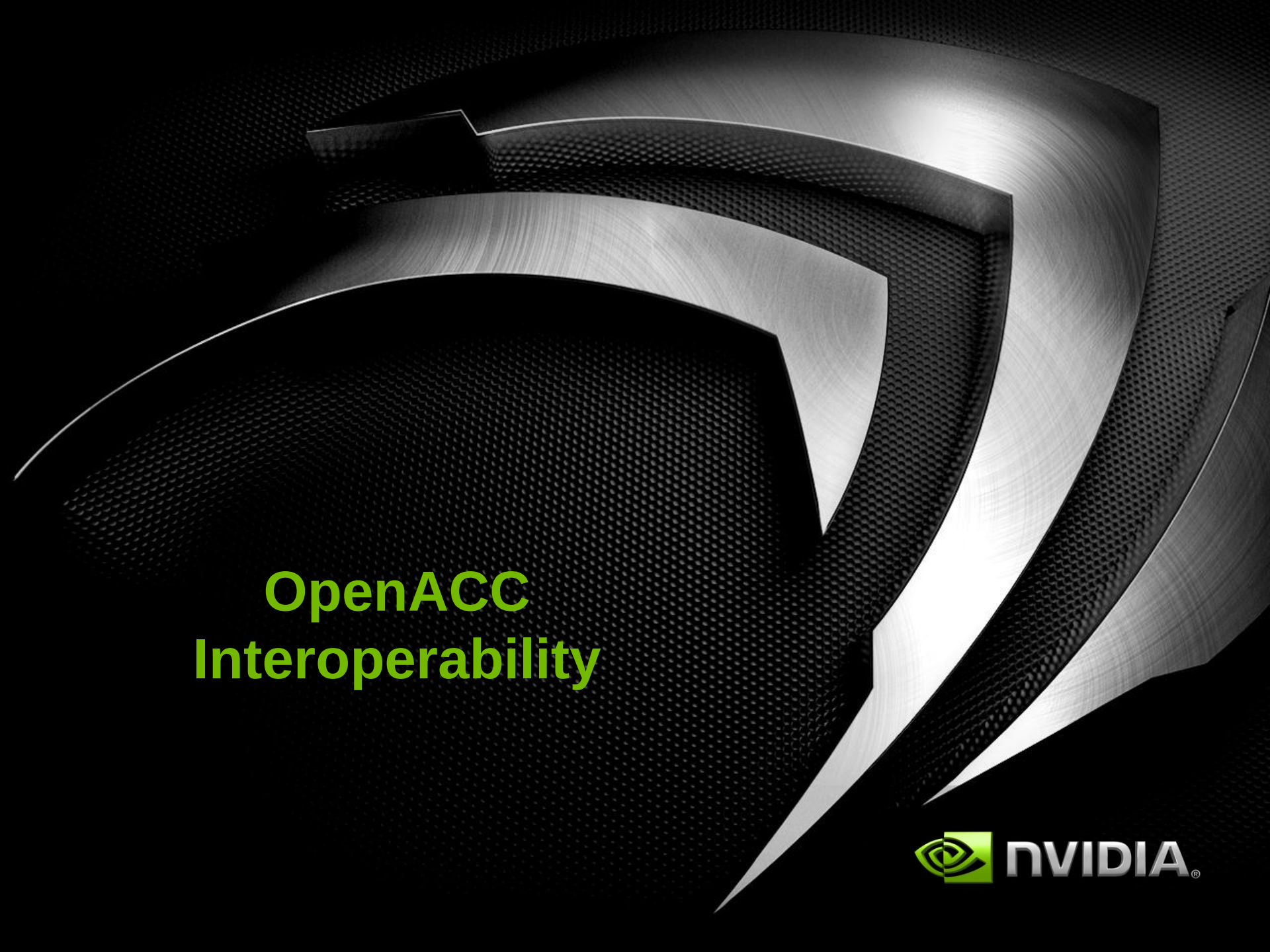
- Optimized for low latency
- For when time to complete an individual operation matters



NVIDIA Kepler

- Optimized for high throughput
- For when time to complete an operation on a lot of data matters

OpenACC Interoperability



OpenACC is not an Island

- OpenACC allows very high level expression of parallelism and data movement.
- It's still possible to leverage low-level approaches such as CUDA C, CUDA Fortran, and GPU Libraries.

Why Interoperate?

- Don’t reinvent the wheel
  - Lots of CUDA code and libraries already exist and can be leveraged.
- Maximum Flexibility
  - Some things can just be represented more easily in one approach or another.
- Maximum Performance
  - Sometimes hand-tuning achieves the most performance.

CUDA C Primer

Standard C

```
void saxpy(int n, float a,
           float *x, float *y)
{
  for (int i = 0; i < n; ++i)
    y[i] = a*x[i] + y[i];
}


int N = 1<<20;



// Perform SAXPY on 1M elements
saxpy(N, 2.0, x, y);
```

- Serial loop over 1M elements, executes 1M times sequentially.
- Data is resident on CPU.

Parallel C

```
__global__
void saxpy(int n, float a,
           float *x, float *y)
{
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < n) y[i] = a*x[i] + y[i];
}

int N = 1<<20;
cudaMemcpy(d_x, x, N, cudaMemcpyHostToDevice);
cudaMemcpy(d_y, y, N, cudaMemcpyHostToDevice);

// Perform SAXPY on 1M elements
saxpy<<<4096,256>>>(N, 2.0, d_x, d_y);

cudaMemcpy(y, d_y, N, cudaMemcpyDeviceToHost);
```

- Parallel kernel, executes 1M times in parallel in groups of 256 elements.
- Data must be copied to/from GPU.

http://developer.nvidia.com/cuda-toolkit

CUDA C Interoperability

OpenACC Main

```
program main
  integer, parameter :: N = 2**20
  real, dimension(N) :: X, Y
  real               :: A = 2.0

  !$acc data
  ! Initialize X and Y
  ...

  !$acc host_data use_device(x,y)
  call saxpy(n, a, x, y)
  !$acc end host_data
  !$acc end data

end program
```

CUDA C Kernel & Wrapper

```
__global__
void saxpy_kernel(int n, float a,
                  float *x, float *y)
{
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < n) y[i] = a*x[i] + y[i];
}

void saxpy(int n, float a, float *dx, float *dy)
{
  // Launch CUDA Kernel
  saxpy_kernel<<<4096,256>>>(N, 2.0, dx, dy);
}
```

- It's possible to interoperate from C/C++ or Fortran.
- OpenACC manages the data and passes device pointers to CUDA.

- CUDA kernel launch wrapped in function expecting device arrays.
- Kernel is launch with arrays passed from OpenACC in main.

CUDA C Interoperability (Reversed)

OpenACC Kernels

```
void saxpy(int n, float a, float *
restrict x, float * restrict y)
{
  #pragma acc kernels
deviceptr(x[0:n],y[0:n])
  {
    for(int i=0; i<n; i++)
    {
      y[i] += 2.0*x[i];
    }
  }
}
```

By passing a device pointer to an OpenACC region, it's possible to add OpenACC to an existing CUDA code.

CUDA C Main

```
int main(int argc, char **argv)
{
  float *x, *y, tmp;
  int n = 1<<20, i;

  cudaMalloc((void*)&x,(size_t)n*sizeof(float));
  cudaMalloc((void*)&y,(size_t)n*sizeof(float));

  ...

  saxpy(n, 2.0, x, y);
  cudaMemcpy(&tmp,y,(size_t)sizeof(float),
             cudaMemcpyDeviceToHost);
  return 0;
}
```

Memory is managed via standard CUDA calls.

CUDA Fortran

Standard Fortran

```
module mymodule contains
  subroutine saxpy(n, a, x, y)
    real :: x(:), y(:), a
    integer :: n, i
    do i=1,n
      y(i) = a*x(i)+y(i)
    enddo
  end subroutine saxpy
end module mymodule

program main
  use mymodule
  real :: x(2**20), y(2**20)
  x = 1.0, y = 2.0

  ! Perform SAXPY on 1M elements
  call saxpy(2**20, 2.0, x, y)

end program main
```

- Serial loop over 1M elements, executes 1M times sequentially.
- Data is resident on CPU.

Parallel Fortran

```
module mymodule contains
  attributes(global) subroutine saxpy(n, a, x, y)
    real :: x(:), y(:), a
    integer :: n, i
    attributes(value) :: a, n
    i = threadIdx%x+(blockIdx%x-1)*blockDim%x
    if (i<=n) y(i) = a*x(i)+y(i)
  end subroutine saxpy
end module mymodule

program main
  use cudafor; use mymodule
  real, device :: x_d(2**20), y_d(2**20)
  x_d = 1.0, y_d = 2.0

  ! Perform SAXPY on 1M elements
  call saxpy<<<4096,256>>>(2**20, 2.0, x_d, y_d)

end program main
```

- Parallel kernel, executes 1M times in parallel in groups of 256 elements.
- Data must be copied to/from GPU (implicit).

http://developer.nvidia.com/cuda-fortran

CUDA Fortran Interoperability

OpenACC Main

```
program main
  use mymodule
  integer, parameter :: N =
2**20
  real, dimension(N) :: X, Y

  X(:) = 1.0
  Y(:) = 0.0

  !$acc data copy(y) copyin(x)
  call saxpy(N, 2.0, x, y)
  !$acc end data

end program
```

CUDA Fortran Kernel & Launcher

```
module mymodule
  contains
  attributes(global) &
  subroutine saxpy_kernel(n, a, x, y)
    real :: x(:), y(:), a
    integer :: n, i
    attributes(value) :: a, n
    i = threadIdx%x+(blockIdx%x-1)*blockDim%x
    if (i<=n) y(i) = a*x(i)+y(i)
  end subroutine saxpy_kernel
  subroutine saxpy (n, a, x, y)
    use cudafor
    real, device :: x(:), y(:)
    real :: a
    integer :: n
    call saxpy_kernel<<<4096,256>>>(n, a, x, y)
  end subroutine saxpy
end module mymodule
```

- Thanks to the “device” attribute in saxpy, no host_data is needed.
- OpenACC manages the data and passes device pointers to CUDA.

- CUDA kernel launch wrapped in function expecting device arrays.
- Kernel is launch with arrays passed from OpenACC in main.

OpenACC with CUDA Fortran Main

Using the “deviceptr” data clause makes it possible to integrate OpenACC into an existing CUDA application.

CUDA C takes a few more tricks to compile, but can be done.

In theory, it should be possible to do the same with C/C++ (including Thrust), but in practice compiler incompatibilities make this difficult.

CUDA Fortran Main w/ OpenAcc Region

```
program main
  use cudafor
  integer, parameter :: N = 2**20
  real, device, dimension(N) :: X, Y
  integer :: i
  real :: tmp

  X(:) = 1.0
  Y(:) = 0.0

  !$acc kernels deviceptr(x,y)
  y(:) = y(:) + 2.0*x(:)
  !$acc end kernels

  tmp = y(1)
  print *, tmp
end program
```

CUBLAS Library

Serial BLAS Code

```
int N = 1<<20;


...

// Use your choice of blas library

// Perform SAXPY on 1M elements
blas_saxpy(N, 2.0, x, 1, y, 1);
```

Parallel cuBLAS Code

```
int N = 1<<20;


cublasInit();
cublasSetVector(N, sizeof(x[0]), x, 1, d_x, 1);
cublasSetVector(N, sizeof(y[0]), y, 1, d_y, 1);

// Perform SAXPY on 1M elements
cublasSaxpy(N, 2.0, d_x, 1, d_y, 1);

cublasGetVector(N, sizeof(y[0]), d_y, 1, y, 1);

cublasShutdown();
```

You can also call cuBLAS from Fortran, C++, Python, and other languages

http://developer.nvidia.com/cublas

CUBLAS Library & OpenACC

OpenACC can interface with existing GPU-optimized libraries (from C/C++ or Fortran).

This includes…

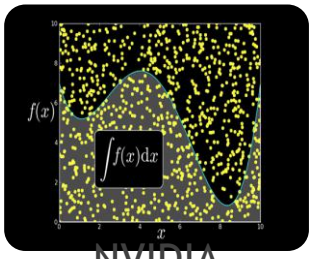
- CUBLAS
- Libsci_acc
- CUFFT
- MAGMA
- CULA
- …

OpenACC Main Calling CUBLAS

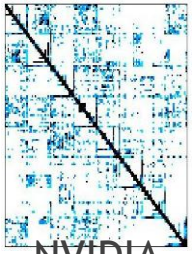
```
int N = 1<<20;
float *x, *y
// Allocate & Initialize X & Y
...

cublasInit();

#pragma acc data copyin(x[0:N]) copy(y[0:N])
{
  #pragma acc host_data use_device(x,y)
  {
    // Perform SAXPY on 1M elements
    cublasSaxpy(N, 2.0, d_x, 1, d_y, 1);
  }
}

cublasShutdown();
```

Some GPU-accelerated Libraries

NVIDIA cuBLAS



NVIDIA cuRAND



NVIDIA cuSPARSE

NVIDIA NPP

GPU VSIPL

Vector Signal Image Processing

CULA tools

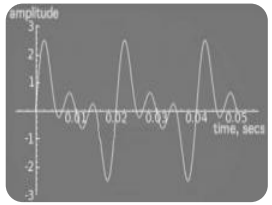
GPU Accelerated Linear Algebra

MAGMA

ICL UT

Matrix Algebra on GPU and Multicore

open source initiative



NVIDIA cuFFT

ROGUE WAVE SOFTWARE

IMSL Library

ArrayFire Matrix Computations

CUSP

Sparse Linear Algebra

open source initiative

Thrust

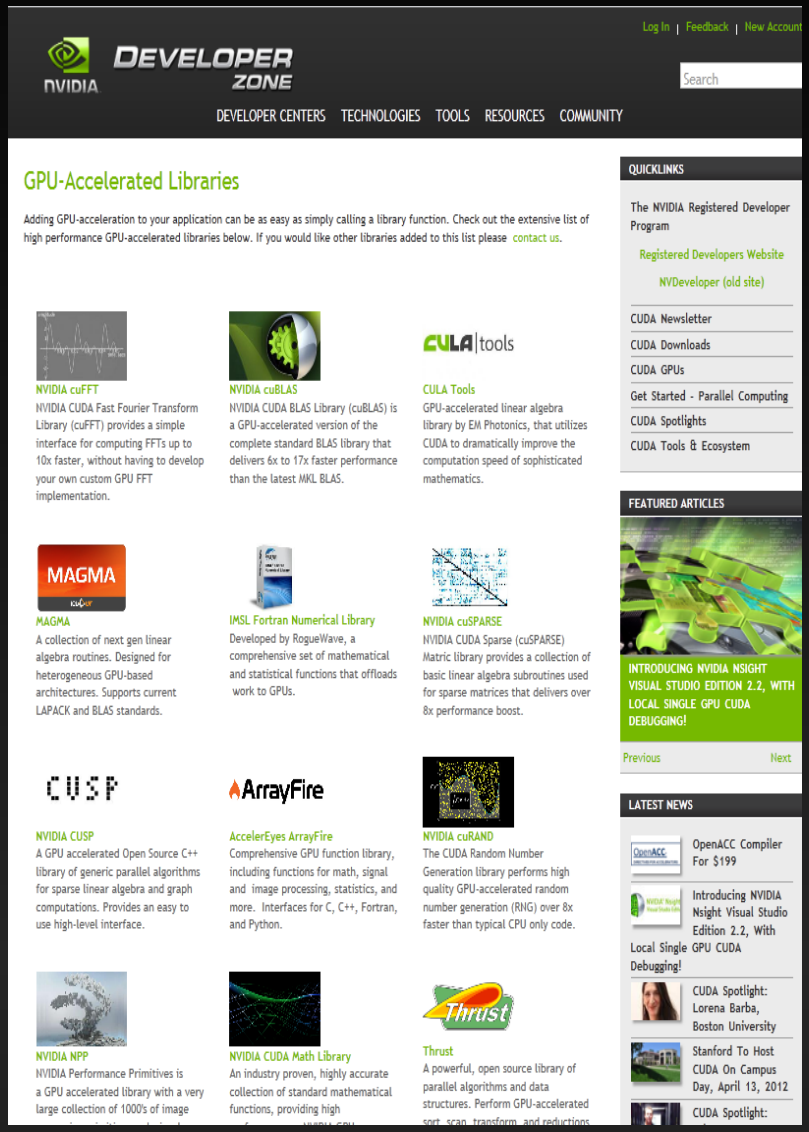
C++ STL Features for CUDA

open source initiative

Explore the CUDA (Libraries) Ecosystem

- CUDA Tools and Ecosystem described in detail on NVIDIA Developer Zone:

developer.nvidia.com/cuda-tools-ecosystem



Thrust C++ Template Library

Serial C++ Code with STL and Boost

```
int N = 1<<20;
std::vector<float> x(N), y(N);

...



// Perform SAXPY on 1M elements
std::transform(x.begin(), x.end(),
               y.begin(), y.end(),
                2.0f * _1 + _2);
```

www.boost.org/libs/lambda

Parallel C++ Code

```
int N = 1<<20;
thrust::host_vector<float> x(N), y(N);

...

thrust::device_vector<float> d_x = x;
thrust::device_vector<float> d_y = y;


// Perform SAXPY on 1M elements
thrust::transform(d_x.begin(), d_x.end(),
                  d_y.begin(),
d_y.begin(),
                  2.0f * _1 + _2);
```

http://thrust.github.com

Thrust C++ and OpenACC??

OpenACC Saxpy

```
void saxpy(int n, float a, float *
restrict x, float * restrict y)
{
#pragma acc kernels
deviceptr(x[0:n],y[0:n])
  {
    for(int i=0; i<n; i++)
    {
      y[i] += 2.0*x[i];
    }
  }
}
```

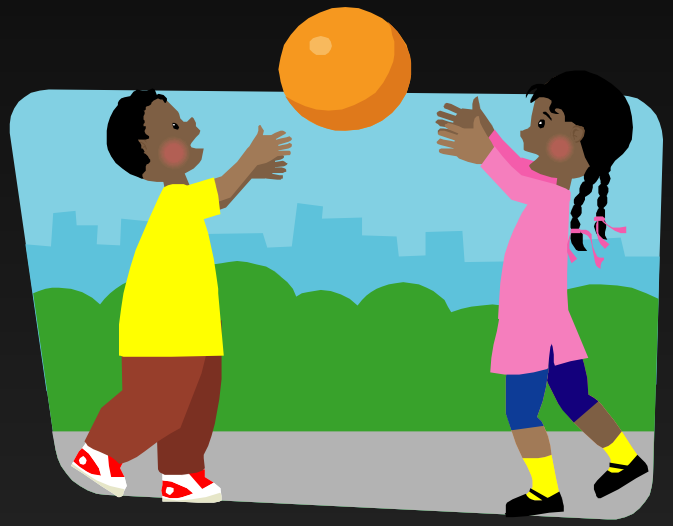
Thrust Main

```
int main(int argc, char **argv)
{
  int N = 1<<20;
  thrust::host_vector<float> x(N), y(N);
  for(int i=0; i<N; i++)
  {
    x[i] = 1.0f;
    y[i] = 1.0f;
  }

  thrust::device_vector<float> d_x = x;
  thrust::device_vector<float> d_y = y;
  thrust::device_ptr<float> p_x = &d_x[0];
  thrust::device_ptr<float> p_y = &d_y[0];

  saxpy(N,2.0,p_x.get(),p_y.get());

  y = d_y;
  return 0;
}
```

How to play well with others

My advice is to do the following:

1. Start with OpenACC
   - Expose high-level parallelism
   - Ensure correctness
   - Optimize away data movement last
2. Leverage other work that's available (even if it's not OpenACC)
   - Common libraries (good software engineering practice)
   - Lots of CUDA already exists
3. Share your experiences
   - OpenACC is still very new, best practices are still forming.
   - Allow others to leverage your work.




GPU Tools
I’m on the GPU, now what?
NVIDIA

CUDA-Memcheck

- You're hitting an error or getting wrong results, try cuda-memcheck first.
  - Reports OOB memory accesses
  - Reports errors from CUDA calls
  - https://developer.nvidia.com/cuda-memcheck
- Works with CUDA and OpenACC

```
$ aprun cuda-memcheck app.exe
```

CUDA-memcheck Output


```
========= CUDA-MEMCHECK
0.000000
========= Invalid __global__ read of size 4
=========     at 0x00000098 in saxpy$ck_L5_2
=========     by thread (0,0,0) in block (0,0,0)
=========     Address 0xb00c00000 is out of bounds
=========     Device Frame:<1 frames were hidden>
=========     Saved host backtrace up to driver entry point at kernel launch time
=========     Host Frame:<9 frames were hidden>
=========     Host Frame:/opt/cray/nvidia//default/lib64/libcuda.so.1 (cuLaunchKernel +
0x3ae) [0xc863e]
=========     Host Frame:/opt/cray/cce/8.1.7/craylibs/x86-64/libcrayacc.so.0
(__cray_acc_hw_start_kernel + 0x1072) [0x1b0a6]
=========     Host Frame:/opt/cray/cce/8.1.7/craylibs/x86-64/libcrayacc.so.0 [0x7c47]
=========     Host Frame:/opt/cray/cce/8.1.7/craylibs/x86-64/libcrayacc.so.0
(cray_start_acc_kernel + 0x114) [0x807e]
=========     Host Frame:./a.out [0xf01]
=========     Host Frame:./a.out [0xd81]
=========     Host Frame:/lib64/libc.so.6 (__libc_start_main + 0xe6) [0x1ec36]
=========     Host Frame:./a.out [0xac9]
========= ERROR SUMMARY: 3 errors
Application 219996 resources: utime ~6s, stime ~1s
```

Compiler Profiling Variables

- The Cray compiler provides automatic instrumentation when `CRAY_ACC_DEBUG=<1,2,3>` at runtime

```
ACC: Initialize CUDA
ACC: Get Device 0
ACC: Create Context
ACC: Set Thread Context
ACC: Start transfer 2 items from saxpy.c:17
ACC:       allocate, copy to acc 'x' (4194304 bytes)
ACC:       allocate, copy to acc 'y' (4194304 bytes)
ACC: End transfer (to acc 8388608 bytes, to host 0 bytes)
ACC: Execute kernel saxpy$ck_L17_1 blocks:8192 threads:128
async(auto) from saxpy.c:17
ACC: Wait async(auto) from saxpy.c:18
ACC: Start transfer 2 items from saxpy.c:18
ACC:       free 'x' (4194304 bytes)
ACC:       copy to host, free 'y' (4194304 bytes)
ACC: End transfer (to acc 0 bytes, to host 4194304 bytes)
```

Compiler Profiling Variables

- The PGI compiler provides automatic instrumentation when `PGI_ACC_TIME=1` at runtime

```
Accelerator Kernel Timing data
/home/jlarkin/kernels/saxpy/saxpy.c
  saxpy  NVIDIA  devicenum=0
        time(us): 3,256
        11: data copyin reached 2 times
             device time(us): total=1,619 max=892 min=727 avg=809
        11: kernel launched 1 times
            grid: [4096]  block: [256]
             device time(us): total=714 max=714 min=714 avg=714
            elapsed time(us): total=724 max=724 min=724 avg=724
        15: data copyout reached 1 times
             device time(us): total=923 max=923 min=923 avg=923
```

CUDA Profiler (nvprof)

- At its most basic, nvprof will instrument your application and provide information about all CUDA-related activity.
- It's also possible to use nvprof to gather data for the CUDA Visual Profiler for viewing on your machine.

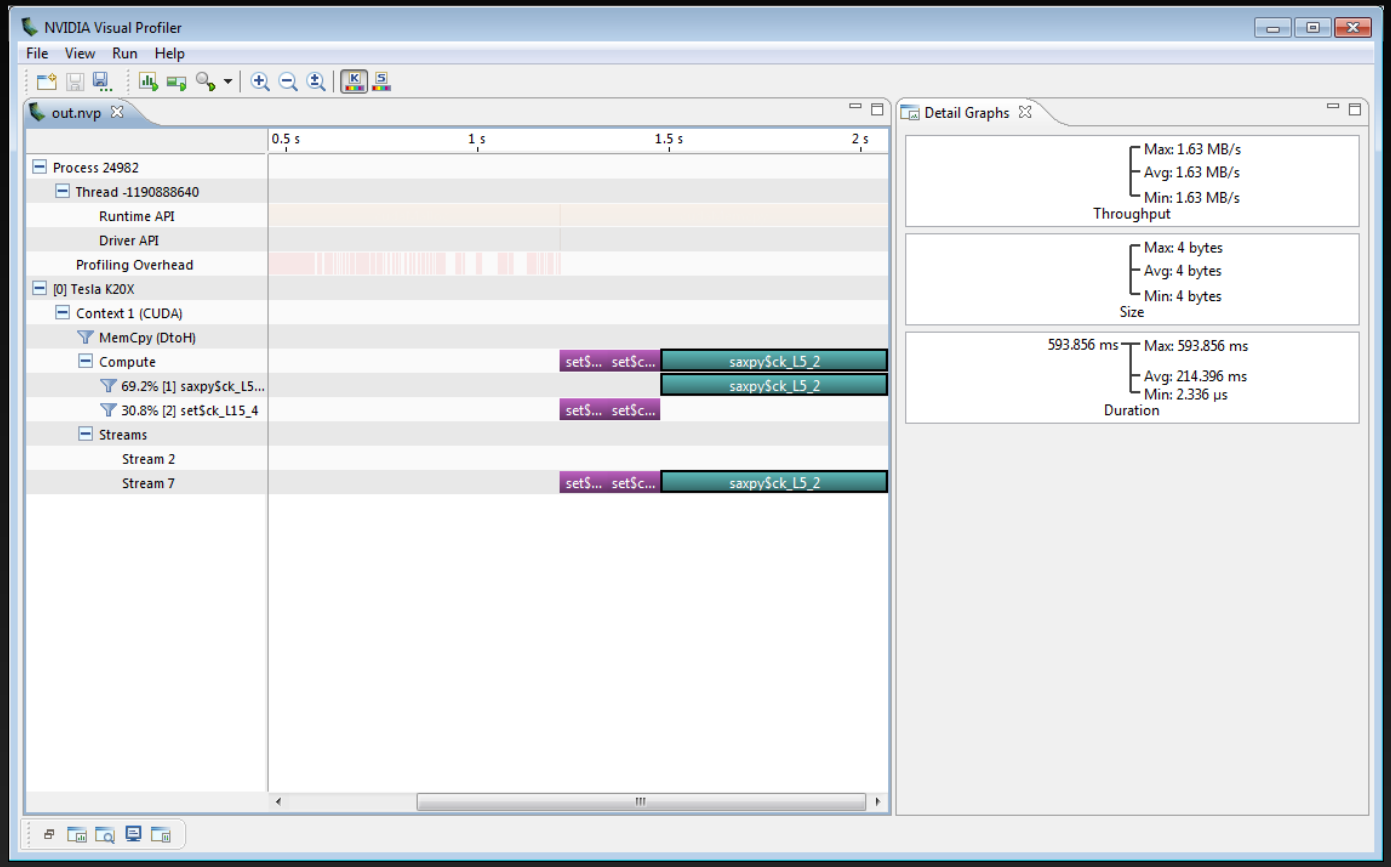
- NOTE: On Cray XK7, it's necessary to set the environment variable below to gather data.

```
export PMI_NO_FORK=1
setenv PMI_NO_FORK 1
```

NVProf Basic Output


```
$ aprun nvprof ./a.out
======== NVPROF is profiling a.out...
======== Command: a.out
2.000000
======== Profiling result:
 Time(%)      Time   Calls       Avg       Min       Max
Name
   70.20  594.27ms       1  594.27ms  594.27ms  594.27ms
saxpy$ck_L5_2
   29.80  252.26ms       2  126.13ms  126.13ms  126.13ms
set$ck_L15_4
    0.00    2.34us       1    2.34us    2.34us    2.34us
[CUDA memcpy DtoH]
```

Nvidia Visual Profiler



Instrument on compute node with: `aprun nvprof -o out.nvp a.out`
Then import into Visual Profiler on your local machine to analyze.

NVProf XK7 Trick

- When running a MPI app, all processes will write to the same file, but try this trick to get 1 per node:

```
#!/bin/bash
# USAGE: Add between aprun options and executable
# For Example: aprun -n 16 -N 1 ./foo arg1 arg2
# Becomes: aprun -n 16 -N 1 ./nvprof.sh ./foo arg1 arg2

# Give each *node* a separate file
LOG=profile_$(hostname).nvp

# Stripe each profile file by 1 to share the load on large runs
lfs setstripe -c 1 $LOG

# Execute the provided command.
exec nvprof -o $LOG $*
```

- Explanation: this script intercepts the call to your executable, determines a unique filename based on the compute node, and calls nvprof.

https://gist.github.com/jefflarkin/5503716

CUDA Command-Line Profiler

- Any CUDA or OpenACC program can also get a more detailed profile via the command-line profiler.

```
export COMPUTE_PROFILE=1
```

- Many performance counters are available.

```
export COMPUTE_PROFILE_CONFIG=events.txt
```

- Outputting to CSV allows importing into Visual Profiler

```
export COMPUTE_PROFILE_CSV=1
```

CLI Profiler Trick

- This trick matches the nvprof trick for getting a unique log file for each XK7 node.

```
#!/bin/bash
# USAGE: Add between aprun options and executable
# For Example: aprun -n 16 -N 1 ./foo arg1 arg2
# Becomes: aprun -n 16 -N 1 ./profile.sh ./foo arg1 arg2

# Enable command-line profiler
export COMPUTE_PROFILE=1

# Set output to CSV (optional)
export COMPUTE_PROFILE_CSV=1

# Give each *node* a separate file
export COMPUTE_PROFILE_LOG=cuda_profile_$(hostname).log

# Stripe each profile file by 1 to share the load on large runs
lfs setstripe -c 1 $COMPUTE_PROFILE_LOG

# Execute the provided command.
exec $*
```

https://gist.github.com/jefflarkin/5356512

OpenMP

Issues that impact Scaling

OpenMP issues

- **Notice himeno only got 2.4 increase out of 4 threads**
  - Scalability can suffer from numerous factors – some we can fix
    - Memory Locality
      - First touch
    - Load Imbalance
      - Choose a schedule that balances the work
    - Granularity
      - The higher the granularity the better
    - Memory bandwidth saturation
      - Better Cache utilization will improve scalability

Possible Approaches

- **Botton Up (Aim to parallelize some of the computation)**
  1. Identify looping structures that use the most time
  2. Identify what arrays are used in those loops
  3. Identify other loops that utilize those arrays
  4. Go to 2
  5. Can computation and/or communication be reorganized
- **Top Down (Aim to parallelize all computation)**
  1. Identify all arrays used within the time step loop
  2. Identify which loops access arrays
  3. Can computation and/or communication be reorganized

What goes on within time step loop?

DO WHILE TIME < MAXIMUM TIME

- Compute
- Communicate
- Compute
- I/O
- Communicate
- Compute
- Compute
- Compute
- I/O
- Compute
- Communicate
- Compute
- Communicate

END DO

Where are the looping structures

```
DO WHILE TIME < MAXIMUM TIME

    call crunch0 (Contains loops )
     DO
       call crunch1
       call crunch2
    END DO

      call crunch3 (Contains loops )
      call communicate0 (Contains MPI)

    DO
     call crunch4 (Contains loops, I/O and/or MPI )
    END DO
      call Inputouput (Contains I/O)

END DO
```

Once we understand the looping structures

- **Is the application well suited for OpenMP?**
  - Are the loop iteration counts appropriate?
  - Is the granularity sufficient?
- **Restructuring might be necessary**
  - Consider introducing a higher level, strip mining loop that can be run in parallel while lower loops are run in vector/SIMD mode
  - Combining loops is helpful
- **The next step is to introduce scoping directives**

Tools for performing this difficult task

- **Cray Perftool tool box has many elements that will be useful for this analysis**
  - ftn -h profile_generate
    - Tremendous overhead for instrumentation, only want to run this a few timestep.
    - Identifies loop characteristics – average, min, max loop iteration count
  - pat_build –u –g mpi,io
    - Identifies major routines, where MPI is used, where I/O is used, etc
- **Tools we need and are looking at**
  - Given an array or a set of arrays, show everywhere they are accessed and how they are accessed
    - This is difficult for Fortran, must consider aliasing through structures, routine arguments
    - This is extremely difficult and extremely useful for C and C++

How to use –hprofile_generate

- **Only do this once for a given problem – that is all you need to get the information you will need**
  - Be sure to load the perftools
  - Add -hprofile_generate to the compile and link line
  - Run pat_build –u (-w) –g mpi <executable>
  - Run application using <executable>+pat
  - Run pat_report –T …….. > profile_all
  - Run pat_report –Oca …….. > profile_ca
  - Run pat_report –Oct ……> profile_ct
  - Profile_all will have several tables, one of which will have a do loop table with min, max, avg iteration counts

1. First and foremost – Profile the application VH1

Table 1: Profile by Function Group and Function

Time%	Time	Imb. Time	Imb. Time%	Calls	Group Function PE=HIDE
100.0%	74.343236	--	--	6922221.1	Total
68.0%	50.560859	--	--	6915004.0	USER
15.0%	11.125597	1.127372	9.2%	288000.0	remap_
14.4%	10.742300	1.092106	9.2%	288000.0	ppmlr_
13.0%	9.629421	1.156963	10.7%	2592000.0	parabola_
7.0%	5.200492	0.573247	9.9%	288000.0	evolve_
5.4%	3.978226	1.112412	21.9%	288000.0	riemann_
2.5%	1.877102	0.244424	11.5%	288000.0	states_
2.1%	1.554790	0.279146	15.2%	576000.0	paraset_
1.8%	1.349213	0.395894	22.7%	864000.0	volume_
1.3%	0.969134	0.324846	25.1%	864000.0	forces_
1.1%	0.834536	0.144497	14.8%	288000.0	flatten_
1.0%	0.759212	0.091074	10.7%	500.0	sweepx1_
0.9%	0.671678	0.067951	9.2%	500.0	sweepx2_
0.8%	0.576190	0.067274	10.5%	1000.0	sweepy_
0.8%	0.569666	0.045713	7.4%	500.0	sweepz_
0.5%	0.368043	0.120640	24.7%	288000.0	boundary_
0.4%	0.331896	0.046669	12.3%	1.0	vhone_
0.0%	0.015684	0.004329	21.6%	500.0	dtcon_
0.0%	0.006907	1.146796	99.4%	1.0	prin_
0.0%	0.000706	0.000916	56.5%	1.0	init_
0.0%	0.000064	0.000660	91.2%	1.0	exit

1. Continued – VH1

Table 3: Loop Stats by Function (from -hprofile_generate)

Loop Incl Time Total	Loop Hit	Loop Trips Avg	Loop Trips Min	Loop Trips Max	Function=/.LOOP[.] PE=HIDE
17.277480	178	16	0	16	sweepx1_.LOOP.1.li.34
17.277266	2848	16	0	16	sweepx1_.LOOP.2.li.35
17.218758	178	16	0	16	sweepx2_.LOOP.1.li.34
17.218594	2848	16	0	16	sweepx2_.LOOP.2.li.35
10.508390	356	16	0	16	sweepy_.LOOP.1.li.38
10.507950	5696	4	0	4	sweepy_.LOOP.2.li.39
10.355416	178	16	0	16	sweepz_.LOOP.05.li.54
10.355199	2848	4	0	4	sweepz_.LOOP.06.li.55
2.081582	136704	135	0	263	riemann_.LOOP.2.li.63
1.178192	18455040	12	0	12	riemann_.LOOP.3.li.64
0.653297	1230336	132	0	264	parabola_.LOOP.6.li.67
0.514080	1230336	132	0	264	parabola_.LOOP.7.li.75
0.390579	1230336	133	0	265	parabola_.LOOP.4.li.44
0.386772	1230336	134	0	266	parabola_.LOOP.2.li.30
0.314141	136704	129	0	257	remap_.LOOP.7.li.83
0.290499	1230336	132	0	264	parabola_.LOOP.5.li.53
0.270040	1230336	132	0	264	parabola_.LOOP.8.li.84
0.210191	136704	135	0	263	riemann_.LOOP.1.li.44
0.194964	1230336	134	0	266	parabola_.LOOP.3.li.36
0.194560	1230336	135	0	267	parabola_.LOOP.1.li.24
0.171375	273408	136.500	0	268	paraset_.LOOP.1.li.117
0.167136	22784	256	0	256	sweepy_.LOOP.7.li.89
0.138138	136704	136	0	264	states_.LOOP.2.li.64

1. Continued VH1

```
Table 1:  Function Calltree View

 Time%  |     Time  |    Calls  |Calltree
        |           |           | PE=HIDE

 100.0% | 53.513557 | 6627213.1 |Total
|-------------------------------------------------------
| 100.0% | 53.513427 | 6627009.0 |vhone_
||------------------------------------------------------
||  28.8% | 15.419074 |  368500.0 |sweepz_
3|        |           |           | sweepz_.LOOPS
||||-----------------------------------------------------
4|||  16.0% |  8.553538 |     500.0 |sweepz_.LOOPS(exclusive)
4|||  12.8% |  6.865537 |  368000.0 |sweepz_.LOOP.05.li.54
5|||        |           |           | sweepz_.LOOP.06.li.55
6|||        |           |           |  ppmlr_
|||||||-----------------------------------------------------
7||||||   5.0% |  2.701293 |  144000.0 |remap_
8||||||        |           |           | remap_.LOOPS
|||||||||-----------------------------------------------------
9||||||||   3.4% |  1.832297 |   96000.0 |parabola_
10|||||||       |           |           | parabola_.LOOPS
9||||||||   1.2% |  0.665167 |   16000.0 |remap_.LOOPS(exclusive)
|||||||||=====================================================
7||||||   4.1% |  2.192975 |   16000.0 |riemann_
8||||||        |           |           | riemann_.LOOPS
7||||||   1.8% |  0.941416 |   48000.0 |parabola_
8||||||        |           |           | parabola_.LOOPS
||||=======================================================
```

1. Continued VH1

Table 1: Profile by Function and Callers

Time%	Time	Calls	Group Function Caller PE=HIDE
21.0%	11.235866	2592000.0	parabola_.LOOPS
			parabola_
13.8%	7.371909	1728000.0	remap_.LOOPS
			remap_
			ppmlr_
3.5%	1.876054	96000.0	sweepy_.LOOP.2.li.39
			sweepy_.LOOP.1.li.38
			sweepy_.LOOPS
			sweepy_
			vhone_
3.4%	1.839313	768000.0	sweepx2_.LOOP.2.li.35
			sweepx2_.LOOP.1.li.34
			sweepx2_.LOOPS
			sweepx2_
			vhone_
3.4%	1.832297	96000.0	sweepz_.LOOP.06.li.55
			sweepz_.LOOP.05.li.54
			sweepz_.LOOPS
			sweepz_
			vhone_
3.4%	1.824246	768000.0	sweepx1_.LOOP.2.li.35
			sweepx1_.LOOP.1.li.34
			sweepx1_.LOOPS
			sweepx1_
			vhone_

2. Continued

```
! module sweeps
!=======================================================================
! Data structures used in 1D sweeps, dimensioned maxsweep  (set in sweepsize.mod)
!-----------------------------------------------------------------------

use sweepsize

character(len=1) :: sweep                                  ! direction of sweep: x,y,z
integer :: nmin, nmax, ngeom, nleft, nright                ! number of first and last real zone
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

end module sweeps
```

For OpenMP these need to be made task_private, for OpenACC they must be passed down the call chain.

2. Continued

Original

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

Restructured

```
hdt    = 0.5*dt
Svel0 = 0.0
do n = nmin-4, nmax+4
  Cdtdx (n) = sqrt(gam*p(n)/r(n))/(dx(n)*radius)
  svel0(n)      = max(svel(n),Cdtdx(n))
  Cdtdx (n) = Cdtdx(n)*hdt
  fCdtdx(n) = 1. - fourthd*Cdtdx(n)
Enddo
!$omp critical
Do n = nmin-4, nmax +4
Svel = max(svel0(n),svel)
Enddo
!$omp end critical
```

For OpenMP need to have a critical region around setting of svel, for OpenACC This needs to be pulled up chain and made a reduction variable.

2. Continued

```
! Directive inserted by Cray Reveal.  May be incomplete.
!$OMP  parallel do default(none)                                          &
!$OMP&   unresolved (f,flat,p,q,radius)                                   &
!$OMP&   private (i,j,k,n,dxf,xaf,xwag,temp1,d,np,umidr,umidl,zrgh,zlft,  &
!$OMP&            pmold,l,uold,dm,dm0,fractn2,nn,fluxq,fluxe,fluxw,       &
!$OMP&            fluxv,fluxu,fluxr,delp2,delp1,shock,temp2,old_flat,     &
!$OMP&            onemfl,hdt,sinxf0,gamfac1,gamfac2,dtheta,deltx,fractn,  &
!$OMP&            ekin)                                                   &
!$OMP&   shared  (gamm,js,ks,ngeomx,nleftx,nrightx,send1,zdx,zfl,zpr,     &
!$OMP&            zro,zux,zuy,zuz,zxa)                                    &
!$OMP&   firstprivate (dx,dx0,e,r,u,v,w,xa,xa0,umid,pmid,rrgh,urgh,prgh,  &
!$OMP&            rlft,ulft,plft,ul,u6,du,rl,r6,dr,pl,p6,dp,steep,ci,c,   &
!$OMP&            bi,b,ai,a,scrch3,scrch2,scrch1,ar,da,diffa,fict,grav,   &
!$OMP&            fcdtdx,cdtdx,wrgh,wlft,prghi,plfti,crgh,clft,amid,       &
!$OMP&            fict1,grav1,fict0,grav0,xa3,xa2,upmid,xa1,dtbdm,dvol1,  &
!$OMP&            delta,dvol0,el,e6,de,ql,q6,dq,wl,w6,dw,vl,v6,dv)        &
!$OMP&   lastprivate (dx,dx0,e,r,u,v,w,xa,xa0)
```

Profile of OpenMP Version2 with 16nodes, 16 threads

Table 1: Profile by Function Group and Function

Time%	Time	Imb. Time	Imb. Time%	Calls	Group Function PE=HIDE Thread=HIDE
100.0%	19.705702	--	--	5941.7	Total
73.5%	14.476758	--	--	2137.9	USER
23.1%	4.553770	0.350832	7.6%	178.0	sweepx1_.LOOP@li.35
13.4%	2.640945	0.208529	7.8%	356.0	sweepy_.LOOP@li.39
12.2%	2.406295	0.152175	6.3%	178.0	sweepz_.LOOP@li.56
9.0%	1.764524	0.310248	16.0%	0.9	exit
8.2%	1.609402	0.117729	7.3%	178.0	sweepx2_.LOOP@li.35
6.8%	1.333533	0.195964	13.7%	1.0	vhone_
0.8%	0.159924	0.061475	29.6%	178.0	sweepz_.LOOP@li.23
0.0%	0.002929	0.000179	6.1%	356.0	sweepy_.REGION@li.39
0.0%	0.001400	0.000152	10.4%	178.0	sweepx2_.REGION@li.35
0.0%	0.001397	0.000450	26.0%	178.0	sweepz_.REGION@li.56
0.0%	0.001374	0.000173	11.9%	178.0	sweepx1_.REGION@li.35
0.0%	0.001266	0.000123	9.4%	178.0	sweepz_.REGION@li.23
13.0%	2.553876	--	--	1259.2	MPI
11.7%	2.309071	0.476361	18.2%	1068.0	mpi_alltoall_
0.7%	0.130259	0.055482	31.9%	1.0	mpi_finalize_
0.4%	0.069961	0.076013	55.5%	2.0	mpi_comm_split_
0.2%	0.034748	0.011043	25.7%	2.0	mpi_gather_
0.0%	0.009812	0.002771	23.5%	179.0	mpi_allreduce_
0.0%	0.000013	0.000001	7.2%	3.0	mpi_comm_size_
0.0%	0.000007	0.000001	8.8%	3.0	mpi_comm_rank_
0.0%	0.000002	0.000036	100.0%	0.1	mpi_wtime_
0.0%	0.000001	0.000016	100.0%	0.1	mpi_get_processor_name_
0.0%	0.000001	0.000000	15.0%	1.0	mpi_init_

Profile of OpenMP Version2 with 16nodes, 4 MPI tasks/node, 4 threads/MPI task

Table 1: Profile by Function Group and Function

Time%	Time	Imb. Time	Imb. Time%	Calls	Group Function PE=HIDE Thread=HIDE
100.0%	13.293169	--	--	5920.2	Total
81.3%	10.807448	--	--	2138.0	USER
21.6%	2.874494	0.311403	9.9%	178.0	sweepx1_.LOOP@li.35
18.8%	2.495491	0.097889	3.8%	356.0	sweepy_.LOOP@li.39
17.8%	2.371742	0.101864	4.2%	178.0	sweepz_.LOOP@li.56
11.9%	1.578197	0.081655	5.0%	178.0	sweepx2_.LOOP@li.35
9.1%	1.206978	0.234031	16.5%	1.0	exit
1.5%	0.205668	0.132494	39.8%	1.0	vhone_
0.5%	0.067165	0.006973	9.6%	178.0	sweepz_.LOOP@li.23
0.0%	0.002607	0.000228	8.2%	356.0	sweepy_.REGION@li.39
0.0%	0.001386	0.000111	7.5%	178.0	sweepx2_.REGION@li.35
0.0%	0.001285	0.000285	18.4%	178.0	sweepx1_.REGION@li.35
0.0%	0.001278	0.000095	7.1%	178.0	sweepz_.REGION@li.56
0.0%	0.001155	0.000090	7.3%	178.0	sweepz_.REGION@li.23
12.0%	1.593726	--	--	1259.0	MPI
9.5%	1.260809	0.092979	7.0%	1068.0	mpi_alltoall_
1.8%	0.239944	0.263621	53.2%	2.0	mpi_comm_split_
0.5%	0.070725	0.016897	19.6%	1.0	mpi_finalize_
0.1%	0.016589	0.021947	57.9%	2.0	mpi_gather_
0.0%	0.005632	0.000400	6.7%	179.0	mpi_allreduce_
0.0%	0.000018	0.000006	25.0%	3.0	mpi_comm_size_
0.0%	0.000007	0.000004	35.3%	3.0	mpi_comm_rank_
0.0%	0.000001	0.000004	81.8%	1.0	mpi_init_
0.0%	0.000001	0.000040	100.0%	0.0	mpi_wtime_
0.0%	0.000000	0.000021	100.0%	0.0	mpi_get_processor_name_

Comparisons of VH1 Versions

Version of VH1	Runtime (Seconds)
Original Version 1 running on 16 nodes 16 MPI/Node	.1500
OMP version2 running on 16 nodes, 1 MPI Task/node 16 threads/MPI task	.2223
OMP version2 running on 16 nodes 4 MPI/Node 4 threads/MPI task	.1748

1. First and foremost – Profile the application AWP-ODC

Table 1: Profile by Function Group and Function

Time%	Time	Imb. Time	Imb. Time%	Calls	Group Function PE=HIDE
100.0%	297.628078	--	--	503201.0	Total
75.6%	225.102220	--	--	72990.9	USER
48.0%	142.746772	13.330803	8.6%	5600.0	allq_
23.8%	70.952642	12.926321	15.4%	5600.0	dvelc_
2.3%	6.887266	26.999265	79.8%	3500.0	addcrjstr_loop_
1.0%	2.935374	10.522329	78.3%	3500.0	addcrjvel_loop_
0.2%	0.604887	0.550897	47.8%	1.0	$main_
0.1%	0.167412	0.073473	30.6%	5600.0	strswp_s_
0.0%	0.138389	0.061234	30.7%	5600.0	velswp_s_
0.0%	0.121721	1.300691	91.6%	700.0	fstr_
0.0%	0.099915	0.053316	34.9%	11200.0	wait_all_vel_
0.0%	0.092978	0.046297	33.3%	5600.0	strswp_r_
0.0%	0.079579	0.015334	16.2%	5600.0	velswp_r_
0.0%	0.056764	0.094206	62.5%	5600.0	addcrjstr_
0.0%	0.054230	0.017633	24.6%	1.0	vsemem_
0.0%	0.050369	0.082618	62.2%	5600.0	addcrjvel_
0.0%	0.037800	0.350154	90.4%	700.0	fvelz_
0.0%	0.030961	0.009635	23.8%	5600.0	dstrqc_
0.0%	0.023482	0.290860	92.7%	700.0	fvelxy_
0.0%	0.006425	0.075536	92.3%	700.0	fs2n_

1. Continued – AWP-ODC

Table 2: Loop Stats by Function (from -hprofile_generate)

Loop Incl Time Total	Loop Hit	Loop Trips Avg	Loop Trips Min	Loop Trips Max	Function=/.LOOP[.] PE=HIDE
142.614163	5600	4	0	4	allq_.LOOP.4.li.90
142.603928	22400	8	0	8	allq_.LOOP.5.li.91
142.583550	179200	16	0	16	allq_.LOOP.6.li.92
141.783744	2867200	8	0	8	allq_.LOOP.7.li.94
25.490044	5600	4	0	4	uxx1_.LOOP.1.li.113
25.477328	22400	8	0	8	uxx1_.LOOP.2.li.114
25.431219	179200	16	0	16	uxx1_.LOOP.3.li.115
25.246590	2867200	8	0	8	uxx1_.LOOP.4.li.116
23.160269	22937600	64	0	64	uxx1_.LOOP.5.li.117
22.741977	5600	4	0	4	vyy1_.LOOP.1.li.164
22.726513	22400	8	0	8	vyy1_.LOOP.2.li.165
22.688198	179200	16	0	16	vyy1_.LOOP.3.li.166
22.675890	5600	4	0	4	wzz1_.LOOP.1.li.217
22.665458	22400	8	0	8	wzz1_.LOOP.2.li.218
22.622475	179200	16	0	16	wzz1_.LOOP.3.li.219
22.515997	2867200	8	0	8	vyy1_.LOOP.4.li.167
22.450233	2867200	8	0	8	wzz1_.LOOP.4.li.220
20.494067	22937600	64	0	64	vyy1_.LOOP.5.li.168
20.460083	22937600	64	0	64	wzz1_.LOOP.5.li.221
6.951569	3500	3.400	0	4	addcrjstr_loop_.LOOP.1.li.273
6.945284	11900	5.162	0	8	addcrjstr_loop_.LOOP.2.li.274
6.930554	61425	15.470	0	16	addcrjstr_loop_.LOOP.3.li.275
6.844632	950250	7.431	0	8	addcrjstr_loop_.LOOP.4.li.276
4.761125	22937600	64	0	64	allq_.LOOP.8.li.97

1. Continued –AWP-ODC

```
Table 1:   Function Calltree View

 Time%  |      Time  |   Calls  |Calltree
        |            |          | PE=HIDE

 100.0% | 297.885224 | 551518.9 |Total
|-------------------------------------------------------
| 100.0% | 297.885101 | 551308.9 |$main_
||------------------------------------------------------
||  47.9% | 142.766354 |  16800.0 |dstrqc_
|||-----------------------------------------------------
3||  47.9% | 142.746260 |  11200.0 |allq_
||||----------------------------------------------------
4|||  47.9% | 142.682109 |   5600.0 |allq_.LOOPS
4|||   0.0% |   0.064151 |   5600.0 |allq_(exclusive)
||||====================================================
3||   0.0% |   0.020094 |   5600.0 |dstrqc_(exclusive)
|||=====================================================
||  23.9% |  71.342626 |  39200.0 |dvelc_
|||-----------------------------------------------------
3||   8.6% |  25.603272 |  11200.0 |uxx1_
||||----------------------------------------------------
4|||   8.6% |  25.543466 |   5600.0 |uxx1_.LOOPS
4|||   0.0% |   0.059806 |   5600.0 |uxx1_(exclusive)
||||====================================================
3||   7.7% |  22.876337 |  11200.0 |vyy1_
||||----------------------------------------------------
4|||   7.7% |  22.794174 |   5600.0 |vyy1_.LOOPS
4|||   0.0% |   0.082162 |   5600.0 |vyy1_(exclusive)
||||====================================================
3||   7.7% |  22.813118 |  11200.0 |wzz1_
||||----------------------------------------------------
```

1. Continued – AWP-ODC

```
Table 1:  Function Calltree View

 Time%  |     Time  |    Calls  |Calltree
        |           |           | PE=HIDE

 100.0% | 297.885224 | 551518.9 |Total
|-------------------------------------------------------
|  75.8% | 225.748891 | 121308.8 |USER
||------------------------------------------------------
||  47.9% | 142.682109 |   5600.0 |allq_.LOOPS
3|        |            |          | allq_
4|        |            |          |  dstrqc_
5|        |            |          |   $main_
||   8.6% |  25.543466 |   5600.0 |uxx1_.LOOPS
3|        |            |          | uxx1_
4|        |            |          |  dvelc_
5|        |            |          |   $main_
||   7.7% |  22.794174 |   5600.0 |vyy1_.LOOPS
3|        |            |          | vyy1_
4|        |            |          |  dvelc_
5|        |            |          |   $main_
||   7.6% |  22.728873 |   5600.0 |wzz1_.LOOPS
3|        |            |          | wzz1_
4|        |            |          |  dvelc_
5|        |            |          |   $main_
||   2.3% |   6.983995 |   3500.0 |addcrjstr_loop_.LOOPS
3|        |            |          | addcrjstr_loop_
4|        |            |          |  addcrjstr_
5|        |            |          |   $main_
||   1.0% |   3.035483 |   3500.0 |addcrjvel_loop_.LOOPS
3|        |            |          | addcrjvel_loop_
4|        |            |          |  addcrjvel_
5|        |            |          |   $main_
```

Comparisons of AWP-ODC Versions

Version of AWP-ODC	Time per timestep (Seconds)
Original Version 1 running on 64 nodes 4 MPI/Node	
OMP version2 running on 16 nodes, 1 MPI Task/node 16 threads/MPI task	
OMP version2 running on 16 nodes 4 MPI/Node 4 threads/MPI task	

OpenACC

Numerous examples to illustrate what to do

Relationship between OpenMP and OpenACC

```
DO WHILE TIME < MAXIMUM TIME
  !$omp parallel default(shared) private(……)
            call crunch0 (Contains OpenMP  )
    !$omp do

            call crunch1
            call crunch2

    !$omp end do
          call communicate0 (Contains MPI)

     !$omp do
    call crunch3 (Contains OpenMP  )
  !$omp end parallel

END DO
```

Relationship between OpenMP and OpenACC

```
!$acc data copyin(OpenMP shared data…
!$acc        present_or_create( OpenMP private data…
DO WHILE TIME < MAXIMUM TIME
  !$omp parallel default(shared)  private(……)
              call crunch0 (Contains OpenMP and OpenACC)
    !$omp do
    !$acc parallel loop
              call crunch1
              call crunch2
    !$acc parallel loop
    !$omp end do
          call communicate0 (Contains MPI)

     !$omp do
    call crunch3 (Contains OpenMP and OpenACC)
  !$omp end parallel

END DO
!$acc end data
```

Relationship between OpenMP and OpenACC

```
DO WHILE TIME < MAXIMUM TIME
  !$omp parallel do default(shared) private(……)
     DO K = 1, KMAX
        call crunch0

        DO J = 1, JMAX
           call crunch1
           call crunch2
        END DO

     call communicate0 (Contains MPI)
     call crunch3
     END DO
  !$omp end parallel do
END DO
```

Relationship between OpenMP and OpenACC

```
!$acc data copyin(OpenMP shared data…
!$acc        present_or_create( OpenMP private data…
DO WHILE TIME < MAXIMUM TIME
  !$omp parallel do default(shared) private(……)
     DO K = 1, KMAX
        call crunch0(Contains OpenACC with ASYNC(K))
   !$ acc parallel loop present(….)  ASYNC(K)
        DO J = 1, JMAX
           call crunch1
           call crunch2
        END DO
   !$acc end parallel loop
   !$acc wait(K)
     call communicate0 (Contains MPI)
     call crunch3   (Contains OpenACC with ASYNC(K) )
  !$omp end parallel loop
END DO
```

3. Use OpenACC to identify data motion require to run with companion accelerator

```
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
54.  + G g-----------< do k = 1, ks
55.  + G g 3---------<  do j = 1, js
56.    G g 3               theta=0.0
57.    G g 3               stheta=0.0
58.    G g 3               radius=0.0
59.    G g 3
60.    G g 3             ! Put state variables into 1D arrays, padding with 6 ghost zones
61.  + G g 3 4-------<    do m = 1, npey
62.  + G g 3 4 r4----<     do i = 1, isy
63.    G g 3 4 r4             n = i + isy*(m-1) + 6
64.    G g 3 4 r4             r(n) = recv2(1,k,i,j,m)
65.    G g 3 4 r4             p(n) = recv2(2,k,i,j,m)
66.    G g 3 4 r4             u(n) = recv2(3,k,i,j,m)
67.    G g 3 4 r4             v(n) = recv2(4,k,i,j,m)
68.    G g 3 4 r4             w(n) = recv2(5,k,i,j,m)
69.    G g 3 4 r4             f(n) = recv2(6,k,i,j,m)
70.    G g 3 4 r4---->     enddo
71.    G g 3 4------->    enddo
72.    G g 3
73.    G g 3 g-------<    do i = 1,imax
74.    G g 3 g              n = i + 6
75.    G g 3 g              xa0(n) = zxa(i)
76.    G g 3 g              dx0(n) = zdx(i)
77.    G g 3 g              xa (n) = zxa(i)
78.    G g 3 g              dx (n) = zdx(i)
79.    G g 3 g              p  (n) = max(smallp,p(n))
80.    G g 3 g              e  (n) = p(n)/(r(n)*gamm)+0.5*(u(n)**2+v(n)**2+w(n)**2)
```

3. Continued

```
 72.     G g 3
 73.     G g 3 g-------<     do i = 1,imax
 74.     G g 3 g               n = i + 6
 75.     G g 3 g               xa0(n) = zxa(i)
 76.     G g 3 g               dx0(n) = zdx(i)
 77.     G g 3 g               xa (n) = zxa(i)
 78.     G g 3 g               dx (n) = zdx(i)
 79.     G g 3 g               p  (n) = max(smallp,p(n))
 80.     G g 3 g               e  (n) = p(n)/(r(n)*gamm)+0.5*(u(n)**2+v(n)**2+w(n)**2)
 81.     G g 3 g------->     enddo
 82.     G g 3
 83.     G g 3               ! Do 1D hydro update using PPMLR
 84.   + G g 3 gr2 Ip-->     call ppmlr (svel0, sweep, nmin, nmax, ngeom, nleft, nright,r, p, e, q, u, v,
 85.     G g 3                  xa, xa0, dx, dx0, dvol, f, flat, para,radius, theta, stheta)
 86.     G g 3 g-------<   do n = nmin-4, nmax+4
 87.     G g 3 g           svel      = max(svel,svel0(n))
 88.     G g 3 g------->   enddo
 89.     G g 3           #ifdef DEBUGX
 90.     G g 3             print *,'In sweepx2',svel
 91.     G g 3           #endif
 92.     G g 3
 93.     G g 3               ! Put updated values back into 3D arrays, dropping ghost zones
 94.     G g 3 g-------<     do i = 1, imax
 95.     G g 3 g               n = i + 6
 96.     G g 3 g               zro(i,j,k) = r(n)
 97.     G g 3 g               zpr(i,j,k) = p(n)
 98.     G g 3 g               zux(i,j,k) = u(n)
 99.     G g 3 g               zuy(i,j,k) = v(n)
100.     G g 3 g               zuz(i,j,k) = w(n)
101.     G g 3 g               zfl(i,j,k) = f(n)
102.     G g 3 g------->     enddo
103.     G g 3
104.     G g 3--------->  enddo
105.     G g----------> enddo
```

3. Continued

```
ftn-6405 ftn: ACCEL File = sweepx2.f90, Line = 46
  A region starting at line 46 and ending at line 107 was placed on the accelerator.

ftn-6418 ftn: ACCEL File = sweepx2.f90, Line = 46
  If not already present: allocate memory and copy whole array "recv2" to accelerator, free at line 107
  (acc_copyin).

ftn-6418 ftn: ACCEL File = sweepx2.f90, Line = 46
  If not already present: allocate memory and copy whole array "zxa" to accelerator, free at line 107
  (acc_copyin).

ftn-6418 ftn: ACCEL File = sweepx2.f90, Line = 46
  If not already present: allocate memory and copy whole array "zdx" to accelerator, free at line 107
  (acc_copyin).

ftn-6416 ftn: ACCEL File = sweepx2.f90, Line = 46
  If not already present: allocate memory and copy whole array "zro" to accelerator, copy back at line 107
  (acc_copy).

ftn-6416 ftn: ACCEL File = sweepx2.f90, Line = 46
  If not already present: allocate memory and copy whole array "zpr" to accelerator, copy back at line 107
  (acc_copy).

ftn-6416 ftn: ACCEL File = sweepx2.f90, Line = 46
  If not already present: allocate memory and copy whole array "zux" to accelerator, copy back at line 107
  (acc_copy).
```

4. Once one loop is analyze, now look at next highest compute loop, perform steps 2 and 3.

```
Table 1:  Profile by Function and Callers

 Time%  |     Time  |    Calls  |Group
        |           |           | Function
        |           |           |  Caller
        |           |           |   PE=HIDE
||  21.0% | 11.235866 | 2592000.0 |parabola_.LOOPS
3|       |           |           | parabola_
||||-------------------------------------------------
4|||  13.8% |  7.371909 | 1728000.0 |remap_.LOOPS
5|||        |           |           | remap_
6|||        |           |           |  ppmlr_
|||||||-----------------------------------------------
7||||||   3.5% |  1.876054 |   96000.0 |sweepy_.LOOP.2.li.39
8||||||        |           |           | sweepy_.LOOP.1.li.38
9||||||        |           |           |  sweepy_.LOOPS
10|||||        |           |           |   sweepy_
11|||||        |           |           |    vhone_
7||||||   3.4% |  1.839313 |  768000.0 |sweepx2_.LOOP.2.li.35
8||||||        |           |           | sweepx2_.LOOP.1.li.34
9||||||        |           |           |  sweepx2_.LOOPS
10|||||        |           |           |   sweepx2_
11|||||        |           |           |    vhone_
7||||||   3.4% |  1.832297 |   96000.0 |sweepz_.LOOP.06.li.55
8||||||        |           |           | sweepz_.LOOP.05.li.54
9||||||        |           |           |  sweepz_.LOOPS
10|||||        |           |           |   sweepz_
11|||||        |           |           |    vhone_
7||||||   3.4% |  1.824246 |  768000.0 |sweepx1_.LOOP.2.li.35
8||||||        |           |           | sweepx1_.LOOP.1.li.34
9||||||        |           |           |  sweepx1_.LOOPS
10|||||        |           |           |   sweepx1_
11|||||        |           |           |    vhone_
|||||||===============================================
```

5. Soon multiple loops can be combined within a OpenACC data region for elminating transfers to and from the host.

```
!################################################################################
!                          MAIN COMPUTATIONAL LOOP
#ifdef GPU
!$acc data copy(zro,zpr,zux,zuy,zuz,zfl,zxa,zdx,zxc,zya,zdy,zyc,zza,zdz,zzc)
#endif
do while (ncycle < ncycend)
!  if(mype == 0) write(*,*) 'STEP = ',ncycle
  ncycle = ncycle + 2
  ncycp  = ncycp  + 2
  ncycd  = ncycd  + 2
  ncycm  = ncycm  + 2
  olddt  = dt
  svel   = 0.
  if ( time + 2*dt > endtime ) then ! set dt to land on endtime
    if(mype==0) write(8,*) 'cutting to the end...', ncycle, ncycend
    dt = 0.5*(endtime - time)
    ncycend = ncycle-1
    ncycp   = nprin
    ncycd   = ndump
  else if ( timep+2.0*dt > tprin ) then ! set dt to land on tprin
    dt = 0.5*(tprin - timep)
    ncycp = nprin
  else if ( timem+2*dt > tmovie ) then ! set dt to land on tmovie
    dt = 0.5*(tmovie - timem)
    ncycm = nmovie
  endif
! Alternate sweeps to approximate 2nd order operator splitting
  call sweepx1(svel)
  call sweepy(svel)
  if (ndim==3) then
   call sweepz  (svel)
   call sweepy(svel)
  endif
  call sweepx2(svel)
```

6. Work outward until a data region encompasses a communication, I/O or looping structure more suited for the host

- **Now the hard part**
  - Must now account for update host and device
    - When message passing is done by the host, must update the host prior to the transfer and update the device after the transfer
    - When any computation is performed on the host, must update the host prior to the transfer and update the device after the transfer
    - When I/O is performed on the host, must update the host prior to the transfer and update the device after the transfer

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

7. Move data region outside time step loop

8. Test versions after each step – don't worry about performance yet – just accuracy

9. The compiler may introduce data transfer so look at –rm listing for each individual OpenACC loop.

```
ftn-6417 ftn: ACCEL File = computeCoefficients_r.f90, Line = 151
  Allocate memory and copy whole array "sqrtq" to accelerator, free at line 1752 (acc_copyin).

ftn-6418 ftn: ACCEL File = computeCoefficients_r.f90, Line = 151
  If not already present: allocate memory and copy whole array "wt" to accelerator,
  free at line  1752 (acc_copyin).

ftn-6422 ftn: ACCEL File = computeCoefficients_r.f90, Line = 151
  If not already present: allocate memory for whole array "xxwt" on accelerator,
  free at line 1752 (acc_share).

ftn-6422 ftn: ACCEL File = computeCoefficients_r.f90, Line = 151
  If not already present: allocate memory for whole array "y" on accelerator,
  free at line 1752 (acc_share).

ftn-6422 ftn: ACCEL File = computeCoefficients_r.f90, Line = 151
  If not already present: allocate memory for whole array "x" on accelerator,
  free at line 1752 (acc_share).

ftn-6405 ftn: ACCEL File = computeCoefficients_r.f90, Line = 156
  A region starting at line 156 and ending at line 223 was placed on the accelerator.

ftn-6416 ftn: ACCEL File = computeCoefficients_r.f90, Line = 156
  If not already present: allocate memory and copy whole array "lambdax" to accelerator,
  copy back at line 223 (acc_copy).

ftn-6416 ftn: ACCEL File = computeCoefficients_r.f90, Line = 156
  If not already present: allocate memory and copy whole array "vscsty" to accelerator,
  copy back at line 223 (acc_copy).

ftn-6418 ftn: ACCEL File = computeCoefficients_r.f90, Line = 156
  If not already present: allocate memory and copy whole array "mixmx" to accelerator,
  free at line 223 (acc_copyin).
```

9. Useful information can be obtained by using setenv CRAY_ACC_DEBUG 1 or 2 or 3

```
ACC: Start transfer 1 items async(auto) from ../source/f90_files/solve/integrate_erk.f90:99
ACC:   flags: AUTO_ASYNC
ACC:    async_info: 0x2aaab89bb720
ACC:
ACC:   Trans 1
ACC:       Simple transfer of 'q' (168 bytes)
ACC:            host ptr 7fffffff6ab8
ACC:            acc  ptr b28f80000
ACC:            flags: DOPE_VECTOR DV_ONLY_DATA COPY_ACC_TO_HOST
ACC:            Transferring dope vector
ACC:              dim:1 lowbound:1 extent:48 stride_mult:1
ACC:              dim:2 lowbound:1 extent:48 stride_mult:48
ACC:              dim:3 lowbound:1 extent:48 stride_mult:2304
ACC:              dim:4 lowbound:1 extent:56 stride_mult:110592
ACC:              dim:5 lowbound:3 extent:1 stride_mult:6193152
ACC:              DV size=49545216 (scale:8, elem_size:8)
ACC:              total mem size=49545216 (dv:0 obj:49545216)
ACC:            async copy acc to host (b28f80000 to 100324f08c0) async_info 0x2aaab89bb720
ACC:                split copy acc to host (100324f08c0 to b28f80000) size = 49545216
ACC:
ACC: End transfer (to acc 0 bytes, to host 49545216 bytes)
```

9. Useful information can be obtained by using setenv CRAY_ACC_DEBUG 1 or 2 or 3

```
ACC: Start transfer 89 items async(auto) from ../source/f90_files/solve/rhsf.f90:256
ACC:       allocate, copy to acc 'aex' (8 bytes)
ACC:       allocate, copy to acc 'aey' (8 bytes)
ACC:       allocate, copy to acc 'aez' (8 bytes)
ACC:       present 'avmolwt' (884736 bytes)
ACC:       allocate, copy to acc 'bex' (8 bytes)
ACC:       allocate, copy to acc 'bey' (8 bytes)
ACC:       allocate, copy to acc 'bez' (8 bytes)
ACC:       allocate 'buffer31' (110592 bytes)
ACC:       allocate 'buffer32' (110592 bytes)
ACC:       allocate 'buffer33' (110592 bytes)
ACC:       allocate 'buffer34' (110592 bytes)
ACC:       allocate 'buffer35' (110592 bytes)
ACC:       allocate 'buffer36' (110592 bytes)
ACC:       allocate 'buffer37' (110592 bytes)
ACC:       allocate 'buffer41' (6193152 bytes)
ACC:       allocate 'buffer42' (6193152 bytes)
ACC:       allocate 'buffer43' (6193152 bytes)
ACC:       allocate 'buffer44' (6193152 bytes)
ACC:       allocate 'buffer45' (6193152 bytes)
ACC:       allocate, copy to acc 'cex' (8 bytes)
ACC:       allocate, copy to acc 'cey' (8 bytes)
ACC:       allocate, copy to acc 'cez' (8 bytes)
ACC:       present 'cpcoef_aa' (832416 bytes)
ACC:       present 'cpcoef_bb' (832416 bytes)
ACC:       present 'cpmix' (884736 bytes)
ACC:       allocate, copy to acc 'dex' (8 bytes)
```

9. Useful information can be obtained by using setenv CRAY_ACC_DEBUG 1 or 2 or 3

```
ACC: Transfer 46 items (to acc 831869196 bytes, to host 0 bytes) async(auto)
from ../source/drivers/solve_driver.f90:176
ACC: Transfer 49 items (to acc 0 bytes, to host 0 bytes) async(auto)
from ../source/f90_files/solve/integrate_erk.f90:68
ACC: Transfer 7 items (to acc 0 bytes, to host 204374016 bytes) async(auto)
from ../source/f90_files/solve/integrate_erk.f90:75
ACC: Wait async(auto) from ../source/f90_files/solve/integrate_erk.f90:75
ACC: Execute kernel integrate_$ck_L86_1 async(auto)
from ../source/f90_files/solve/integrate_erk.f90:86
ACC: Transfer 1 items (to acc 0 bytes, to host 49545216 bytes) async(auto)
from ../source/f90_files/solve/integrate_erk.f90:99
ACC: Wait async(auto) from ../source/f90_files/solve/integrate_erk.f90:99
ACC: Transfer 89 items (to acc 1260 bytes, to host 0 bytes) async(auto)
from ../source/f90_files/solve/rhsf.f90:256
ACC: Transfer 15 items (to acc 0 bytes, to host 0 bytes) async(auto)
from ../source/f90_files/solve/calc_primitive_var.f90:42
ACC: Transfer 4 items (to acc 0 bytes, to host 0 bytes) async(auto)
from ../source/f90_files/solve/calc_primitive_var.f90:47
ACC: Execute kernel calc_primary_vars_$ck_L47_1 async(auto)
from ../source/f90_files/solve/calc_primitive_var.f90:47
ACC: Wait async(auto) from ../source/f90_files/solve/calc_primitive_var.f90:157
ACC: Transfer 4 items (to acc 0 bytes, to host 0 bytes) async(auto)
from ../source/f90_files/solve/calc_primitive_var.f90:157
```

9. Useful information can be obtained by using setenv CUDA_PROFILE 1

```
method=[ integrate_$ck_L86_1 ] gputime=[ 1265.536 ] cputime=[ 26.000 ] occupancy=[ 1.000 ]
method=[ memcpyDtoHasync ] gputime=[ 7381.152 ] cputime=[ 8.000 ]
method=[ memcpyHtoDasync ] gputime=[ 4.672 ] cputime=[ 13.000 ]
method=[ memcpyHtoDasync ] gputime=[ 2.496 ] cputime=[ 6.000 ]
method=[ memcpyHtoDasync ] gputime=[ 2.496 ] cputime=[ 6.000 ]
method=[ memcpyHtoDasync ] gputime=[ 2.528 ] cputime=[ 6.000 ]
method=[ memcpyHtoDasync ] gputime=[ 2.496 ] cputime=[ 10.000 ]
method=[ memcpyHtoDasync ] gputime=[ 2.528 ] cputime=[ 6.000 ]
method=[ memcpyHtoDasync ] gputime=[ 3.616 ] cputime=[ 8.000 ]
method=[ memcpyHtoDasync ] gputime=[ 2.528 ] cputime=[ 6.000 ]
method=[ memcpyHtoDasync ] gputime=[ 2.528 ] cputime=[ 6.000 ]
method=[ memcpyHtoDasync ] gputime=[ 2.528 ] cputime=[ 6.000 ]
method=[ memcpyHtoDasync ] gputime=[ 2.528 ] cputime=[ 6.000 ]
method=[ memcpyHtoDasync ] gputime=[ 2.528 ] cputime=[ 6.000 ]
method=[ memcpyHtoDasync ] gputime=[ 3.616 ] cputime=[ 8.000 ]
method=[ memcpyHtoDasync ] gputime=[ 3.808 ] cputime=[ 8.000 ]
method=[ memcpyHtoDasync ] gputime=[ 3.872 ] cputime=[ 8.000 ]
method=[ memcpyHtoDasync ] gputime=[ 2.688 ] cputime=[ 6.000 ]
method=[ memcpyHtoDasync ] gputime=[ 2.624 ] cputime=[ 6.000 ]
```

10. Optimize/Minimize data transfers first by using present on data clause.

- **PRESENT**
  - This is for variables that have been copyin, copy or created up the call chain
    - If you forget this – you could be creating an error. Compiler will copy in the host version when you are expecting the device version
- **PRESENT_OR_CREATE**
  - This is for variables that are only going to be used on the device
- **PRESENT_OR_COPYIN**
  - This is for variables that must be copied in from the host; however, they do not change after the first copyin

11. Gather perftools statistics on code and identify bottlenecks

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 Thread=HIDE
100.0%	130.491	140.390	50831	96209	897204	Total
17.1%	22.301	0.118	--	--	600	reaction_rate_vec_.ACC_ASYNC_KERNEL@li.167
8.9%	11.634	0.069	--	--	600	rhsf_.ACC_ASYNC_KERNEL@li.916
5.1%	6.594	0.810	6961	--	8400	derivative_xyz_wait_np$derivative_m_.ACC_ASYNC_COPY@li.
4.5%	5.815	0.004	--	--	100	computecoefficients_r_.ACC_ASYNC_KERNEL@li.155
3.7%	4.829	0.820	6961	--	8400	derivative_xyz_wait_np$derivative_m_.ACC_ASYNC_COPY@li.
3.5%	4.503	0.872	6961	--	8400	derivative_xyz_wait_np$derivative_m_.ACC_ASYNC_COPY@li.
3.1%	4.022	0.176	--	6497	1800	derivative_z_pack_np_.ACC_ASYNC_COPY@li.577
2.9%	3.842	0.241	--	6497	1800	derivative_z_pack_np_.ACC_ASYNC_COPY@li.619
2.9%	3.809	0.018	--	--	600	rhsf_.ACC_ASYNC_KERNEL@li.1737
2.8%	3.598	0.071	--	--	600	rhsf_.ACC_ASYNC_KERNEL@li.1680
2.7%	3.517	2.074	6961	--	8400	derivative_xyz_wait_np$derivative_m_.ACC_ASYNC_COPY@li.
2.3%	3.060	0.009	--	19491	100	integrate_.ACC_ASYNC_COPY@li.75
2.2%	2.856	0.174	--	6497	1800	derivative_y_pack_np_.ACC_ASYNC_COPY@li.650
2.1%	2.801	0.175	--	6497	1800	derivative_y_pack_np_.ACC_ASYNC_COPY@li.608
1.9%	2.529	0.068	--	--	600	rhsf_.ACC_ASYNC_KERNEL@li.624
1.9%	2.526	0.080	--	--	600	rhsf_.ACC_ASYNC_KERNEL@li.1636
1.8%	2.402	0.084	--	--	600	rhsf_.ACC_ASYNC_KERNEL@li.400
1.8%	2.399	0.066	--	--	600	rhsf_.ACC_ASYNC_KERNEL@li.450
1.8%	2.375	2.799	--	7341	600	save_bc_deriv1$rhsf_.ACC_ASYNC_COPY@li.248
1.7%	2.251	0.777	6961	--	8400	derivative_xyz_wait_np$derivative_m_.ACC_ASYNC_COPY@li.
1.6%	2.145	0.770	6961	--	8400	derivative_xyz_wait_np$derivative_m_.ACC_ASYNC_COPY@li.
1.6%	2.043	0.043	--	--	100	computecoefficients_r_.ACC_ASYNC_KERNEL@li.225
1.5%	1.938	0.066	--	--	600	rhsf_.ACC_ASYNC_KERNEL@li.493
1.4%	1.877	0.172	--	6497	1800	derivative_x_pack_np_.ACC_ASYNC_COPY@li.640
1.3%	1.734	1.674	3544	--	600	rhsf_.ACC_ASYNC_COPY@li.2436
1.1%	1.444	1.270	--	464.062	6600	derivative_x_pack_.ACC_ASYNC_COPY@li.463
1.0%	1.254	0.027	--	--	700	calc_primary_vars_.ACC_ASYNC_KERNEL@li.47
1.0%	1.247	0.160	--	6497	1800	derivative_x_pack_np_.ACC_ASYNC_COPY@li.598

11. Continued

```
Table 1:  Profile by Function Group and Function

 Time%  |      Time  |    Imb.   |  Imb.   |    Calls  |Group
        |            |    Time   | Time%   |           | Function
        |            |           |         |           |  Thread=HIDE

 100.0% | 174.160022 |       -- |     -- | 4867603.0 |Total
|-------------------------------------------------------------------------------------------------------------------
|  92.4% | 160.926071 |       -- |     -- | 2676780.0 |USER
||------------------------------------------------------------------------------------------------------------------
||  12.8% |  22.319336 | 0.000000 |   0.0% |     600.0 |reaction_rate_vec_.ACC_SYNC_WAIT@li.5008
||  10.3% |  17.997279 | 0.000000 |   0.0% |     600.0 |rhsf_.ACC_DATA_REGION@li.256
||   7.6% |  13.238744 | 0.000000 |   0.0% |    6600.0 |derivative_z_pack_.ACC_ASYNC_KERNEL@li.479
||   3.4% |   5.842934 | 0.000000 |   0.0% |    3000.0 |derivative_xyz_wait_np$derivative_m_.ACC_SYNC_WAIT@li.567
||   3.3% |   5.817360 | 0.000000 |   0.0% |     100.0 |computecoefficients_r_.ACC_SYNC_WAIT@li.222
||   2.7% |   4.743826 | 0.000321 |   0.0% |     600.0 |rhsf_.LOOP@li.2268
||   2.3% |   3.991119 | 0.000000 |   0.0% |    6600.0 |derivative_x_send_.ACC_SYNC_WAIT@li.717
||   1.8% |   3.072952 | 0.000000 |   0.0% |     100.0 |integrate_.ACC_SYNC_WAIT@li.75
||   1.7% |   3.040157 | 0.000000 |   0.0% |  201600.0 |deriv_inplane_1_
||   1.7% |   3.024576 | 0.000000 |   0.0% |     560.0 |filter$filter_m_
||   1.7% |   3.019308 | 0.000000 |   0.0% |     700.0 |calc_primary_vars_.ACC_SYNC_WAIT@li.157
||   1.6% |   2.798427 | 0.000000 |   0.0% |     600.0 |save_bc_deriv1$rhsf_.ACC_ASYNC_COPY@li.248
||   1.2% |   2.111812 | 0.000000 |   0.0% |  201600.0 |deriv_inplane_2_
||   1.2% |   2.071792 | 0.000000 |   0.0% |    8400.0 |derivative_xyz_wait_np$derivative_m_.ACC_ASYNC_COPY@li.544
||   1.2% |   2.006773 | 0.000000 |   0.0% |     100.0 |computecoefficients_r_.ACC_SYNC_WAIT@li.1748
||   1.1% |   1.975207 | 0.000000 |   0.0% |    6600.0 |derivative_z_pack_.ACC_ASYNC_COPY@li.454
||   1.1% |   1.914216 | 0.000000 |   0.0% |     100.0 |controller$rk_m_
||   1.0% |   1.673879 | 0.000000 |   0.0% |     600.0 |rhsf_.ACC_ASYNC_COPY@li.2436
||   0.9% |   1.615192 | 0.000000 |   0.0% |     600.0 |rhsf_.ACC_ASYNC_COPY@li.2187
||   0.9% |   1.598921 | 0.000000 |   0.0% |     600.0 |rhsf_.ACC_ASYNC_COPY@li.2189
||   0.9% |   1.586929 | 0.000000 |   0.0% |     600.0 |rhsf_.ACC_ASYNC_COPY@li.2191
||   0.7% |   1.268257 | 0.000000 |   0.0% |    6600.0 |derivative_x_pack_.ACC_ASYNC_COPY@li.463
||   0.6% |   1.080301 | 0.001090 |   0.1% |     600.0 |rhsf_.LOOP@li.2411
||   0.5% |   0.949635 | 0.000000 |   0.0% |     100.0 |integrate_.ACC_SYNC_WAIT@li.145
||   0.5% |   0.892484 | 0.000000 |   0.0% |   67200.0 |point_der1_y_
||   0.5% |   0.888298 | 0.000000 |   0.0% |   67200.0 |point_der1_x_
||   0.5% |   0.870532 | 0.000000 |   0.0% |     100.0 |integrate_.ACC_SYNC_WAIT@li.99
```

12. If bottleneck is data copies and you did a good job on 9. - look at packing buffers on the accelerator

```
if (vary_in_x==1) then
      call derivative_x_pack_np( nx,ny,nz, yspecies(1,1,1,1), 5, 'yspecies-x',n_spec,25)
   endif
   if (vary_in_y==1) then
      call derivative_y_pack_np( nx,ny,nz, yspecies(1,1,1,1),5, 'yspecies-y',n_spec,27)
   endif
   if (vary_in_z==1) then
      call derivative_z_pack_np( nx,ny,nz, yspecies(1,1,1,1),5, 'yspecies-z',n_spec,29)
   endif
   if (vary_in_x==1) then
      call derivative_x_pack( nx,ny,nz, temp(1,1,1),4,'temp-x',19)
      call derivative_x_pack( nx,ny,nz, u(1,1,1,1), 1,  'u-x1',1)
      call derivative_x_pack( nx,ny,nz, u(1,1,1,2), 2,  'u-x2',7)
      call derivative_x_pack( nx,ny,nz, u(1,1,1,3), 3,  'u-x3',13)
   endif
   if (vary_in_y==1) then
      call derivative_y_pack( nx,ny,nz, temp(1,1,1),4,'temp-y',21)
      call derivative_y_pack( nx,ny,nz, u(1,1,1,1), 1, 'u-y1',3)
      call derivative_y_pack( nx,ny,nz, u(1,1,1,2), 2, 'u-y2',9)
      call derivative_y_pack( nx,ny,nz, u(1,1,1,3), 3, 'u-y3',15)
   endif
   if (vary_in_z==1) then
      call derivative_z_pack( nx,ny,nz, temp(1,1,1),4,'temp-z',23)
      call derivative_z_pack( nx,ny,nz, u(1,1,1,1), 1, 'u-z1',5)
      call derivative_z_pack( nx,ny,nz, u(1,1,1,2), 2, 'u-z2',11)
      call derivative_z_pack( nx,ny,nz, u(1,1,1,3), 3, 'u-z3',17)
   endif
```

13. If bottleneck is kernel performance

- You absolutely have to vectorize on a good vector length; that is, greater than or equal to 32 (32 is a warp, 128 is 4 warps)
- You need to have thousands of the warps waiting to kick off to amortize latency to memory
- Watch out for register spills
- Watch out for overflowing shared memory
- Jeff – what the heck is occupancy?

A Simdized OpenACC loop

```
do i = 1, nx*ny*nz, ms
  ml = i
  mu =  min(i+ms-1, nx*ny*nz)
DIRECTION: do m=1,3
diffFlux(ml:mu,1,1,n_spec,m) = 0.0
grad_mixMW(ml:mu,1,1,m)=grad_mixMW(ml:mu,1,1,m)&
    *avmolwt(ml:mu,1,1)
SPECIES: do n=1,n_spec-1
   diffFlux(ml:mu,1,1,n,m)=-Ds_mixavg(ml:mu,1,1,n)&
     *(grad_Ys(ml:mu,1,1,n,m)+Ys(ml:mu,1,1,n)*&
       grad_mixMW(ml:mu,1,1,m) )
diffFlux(ml:mu,1,1,n_spec,m)=&
         diffFlux(ml:mu,1,1,n_spec,m)-&
         diffFlux(ml:mu,1,1,n,m)

enddo SPECIES
enddo DIRECTION
enddo
```

A Better Simdized OpenACC loop

```
do i = 1, nx*ny*nz, ms
  ml = i
  mu =  min(i+ms-1, nx*ny*nz)
   difftemp1 = 0.0
   difftemp2 = 0.0
   difftemp3 = 0.0
   grad_mixMW(i,1,1,1)= grad_mixMW(i,1,1,1)* avmolwt(i,1,1)
   grad_mixMW(i,1,1,2)= grad_mixMW(i,1,1,2)* avmolwt(i,1,1)
   grad_mixMW(i,1,1,3)= grad_mixMW(i,1,1,3)* avmolwt(i,1,1)
   do n=1,n_spec-1
     diffFlux(i,1,1,n,1)=- ds_mxvg(i,1,1,n)* ( grad_Ys(i,1,1,n,1) + yspecies(i,1,1,n)*grad_mixMW(i,1,1,1) )
     diffFlux(i,1,1,n,2) =-ds_mxvg(i,1,1,n)* ( grad_Ys(i,1,1,n,2) + yspecies(i,1,1,n)*grad_mixMW(i,1,1,2) )
     diffFlux(i,1,1,n,3) = - ds_mxvg(i,1,1,n)*( grad_Ys(i,1,1,n,3) +yspecies(i,1,1,n)*grad_mixMW(i,1,1,3) )
     difftemp1 = difftemp1-diffFlux(i,1,1,n,1)
     difftemp2 = difftemp2-diffFlux(i,1,1,n,2)
     difftemp3 = difftemp3-diffFlux(i,1,1,n,3)
   enddo      ! n
   diffFlux(i,1,1,n_spec,1) =  difftemp1
   diffFlux(i,1,1,n_spec,2) =  difftemp2
   diffFlux(i,1,1,n_spec,3) =  difftemp3
   grad_T(i,1,1,1)=-lambda(i,1,1)* grad_T(i,1,1,1)
   grad_T(i,1,1,2)=-lambda(i,1,1)* grad_T(i,1,1,2)
   grad_T(i,1,1,3)=-lambda(i,1,1)* grad_T(i,1,1,3)
   do n=1,n_spec
     grad_T(i,1,1,1)=grad_T(i,1,1,1)+ h_spec(i,1,1,n)*diffFlux(i,1,1,n,1)
     grad_T(i,1,1,2)=grad_T(i,1,1,2)+ h_spec(i,1,1,n)*diffFlux(i,1,1,n,2)
     grad_T(i,1,1,3)=grad_T(i,1,1,3)+ h_spec(i,1,1,n)*diffFlux(i,1,1,n,3)
enddo          ! i                                ! k
```

Temperature Interpolation loop

```
tmp1(ml:mu) = e0(ml:mu) - 0.5*tmp1(ml:mu)
LOOPM: DO m = ml, mu
   icount = 1
   r_gas = Ru*avmolwt(m)
   yspec(:) = ys(m, :)
   ITERATION: DO
    cpmix(m) = mixCp( yspec, temp(m) )
    enthmix  = mixEnth( yspec, temp(m) )
    deltat =   &
       ( tmp1(m) - (enthmix- &
         r_gas*temp(m)) )  &
          / ( cpmix(m) - r_gas )
    temp(m) = temp(m) + deltat
      IF( ABS(deltat) < atol ) THEN
         cpmix(m) = mixCp( yspec, &
     temp(m) )
      EXIT ITERATION
   ELSEIF( icount > icountmax ) THEN
     STOP
   ELSE
     icount = icount + 1
   ENDIF
  ENDDO ITERATION
ENDDO LOOPM
```

Temperature Interpolation loop

```
 ITERATION: do
do m = ml, mu
      !-- compute mixture heat capacity and enthalpy for this temperature
      n = max(1,min(2001,int((temp(m)-temp_lobound)*invEnthInc)+1))
      cpmix(m) = 0.0
      do mm=1,n_spec
        cpmix(m) = cpmix(m) +  &
            yspecies(m,mm)*(cpCoef_aa(mm,n) * temp(m) + cpCoef_bb(mm,n) )
      enddo
      enthmix(m) = 0.0
      do mm=1,n_spec
        enthmix(m) = enthmix(m) + yspecies(m,mm)*(enthCoef_aa(mm,n)*temp(m) + enthCoef_bb(mm,n))
      enddo

      !-- calculate deltat, new temp
      !   remember tmp1 holds the internal energy
      deltat(m) = ( tmp1(m) - (enthmix(m)-Ru*avmolwt(m)*temp(m)) )  &
                /( cpmix(m) - Ru*avmolwt(m) )
      if(iconverge(m).eq.0)temp(m) = temp(m) + deltat(m)
    enddo
    do m = ml, mu
      !-- check for convergence
!     if( abs(deltat(m)) < atol.and.iconverge(m).eq.0 ) then  ! converged
!          BUG- FIX AUG-16-04 - cpmix was not updated after successful convergence
        iconverge(m) = 1
        n = max(1,min(2001,int((temp(m)-temp_lobound)*invEnthInc)+1))
        cpmix(m) = 0.0
        do mm=1,n_spec
          cpmix(m) = cpmix(m) +  &
              yspecies(m,mm)*(cpCoef_aa(mm,n) * temp(m) + cpCoef_bb(mm,n) )
        enddo
!     endif
    enddo
```

Temperature Interpolation loop

```
if(all(iconverge(ml:mu).eq.1))EXIT ITERATION
        EXIT ITERATION
        do m = ml,mu
        if(iconverge(m).eq.0)then
         if( icount(m) > icountmax ) then  ! maximum count violation
           write(6,*)'calc_temp cannot converge after 100 iterations'
           write(6,*) 'for processor with rank =',myid
           write(6,*) 'm=',m
           stop  !ugly termination but that's the way it is without doing a broadcast
         else
           icount(m) = icount(m) + 1
         endif
        endif
        enddo
     enddo ITERATION
 end do
```

14. Consider introducing CUDA streams

```
if (vary_in_x==1) then
      call derivative_x_pack_np( nx,ny,nz, yspecies(1,1,1,1), 5, 'yspecies-x',n_spec,25)
   endif
   if (vary_in_y==1) then
      call derivative_y_pack_np( nx,ny,nz, yspecies(1,1,1,1),5, 'yspecies-y',n_spec,27)
   endif
   if (vary_in_z==1) then
      call derivative_z_pack_np( nx,ny,nz, yspecies(1,1,1,1),5, 'yspecies-z',n_spec,29)
   endif
   if (vary_in_x==1) then
      call derivative_x_pack( nx,ny,nz, temp(1,1,1),4,'temp-x',19)
      call derivative_x_pack( nx,ny,nz, u(1,1,1,1), 1,  'u-x1',1)
      call derivative_x_pack( nx,ny,nz, u(1,1,1,2), 2,  'u-x2',7)
      call derivative_x_pack( nx,ny,nz, u(1,1,1,3), 3,  'u-x3',13)
   endif
   if (vary_in_y==1) then
      call derivative_y_pack( nx,ny,nz, temp(1,1,1),4,'temp-y',21)
      call derivative_y_pack( nx,ny,nz, u(1,1,1,1), 1, 'u-y1',3)
      call derivative_y_pack( nx,ny,nz, u(1,1,1,2), 2, 'u-y2',9)
      call derivative_y_pack( nx,ny,nz, u(1,1,1,3), 3, 'u-y3',15)
   endif
   if (vary_in_z==1) then
      call derivative_z_pack( nx,ny,nz, temp(1,1,1),4,'temp-z',23)
      call derivative_z_pack( nx,ny,nz, u(1,1,1,1), 1, 'u-z1',5)
      call derivative_z_pack( nx,ny,nz, u(1,1,1,2), 2, 'u-z2',11)
      call derivative_z_pack( nx,ny,nz, u(1,1,1,3), 3, 'u-z3',17)
   endif
```

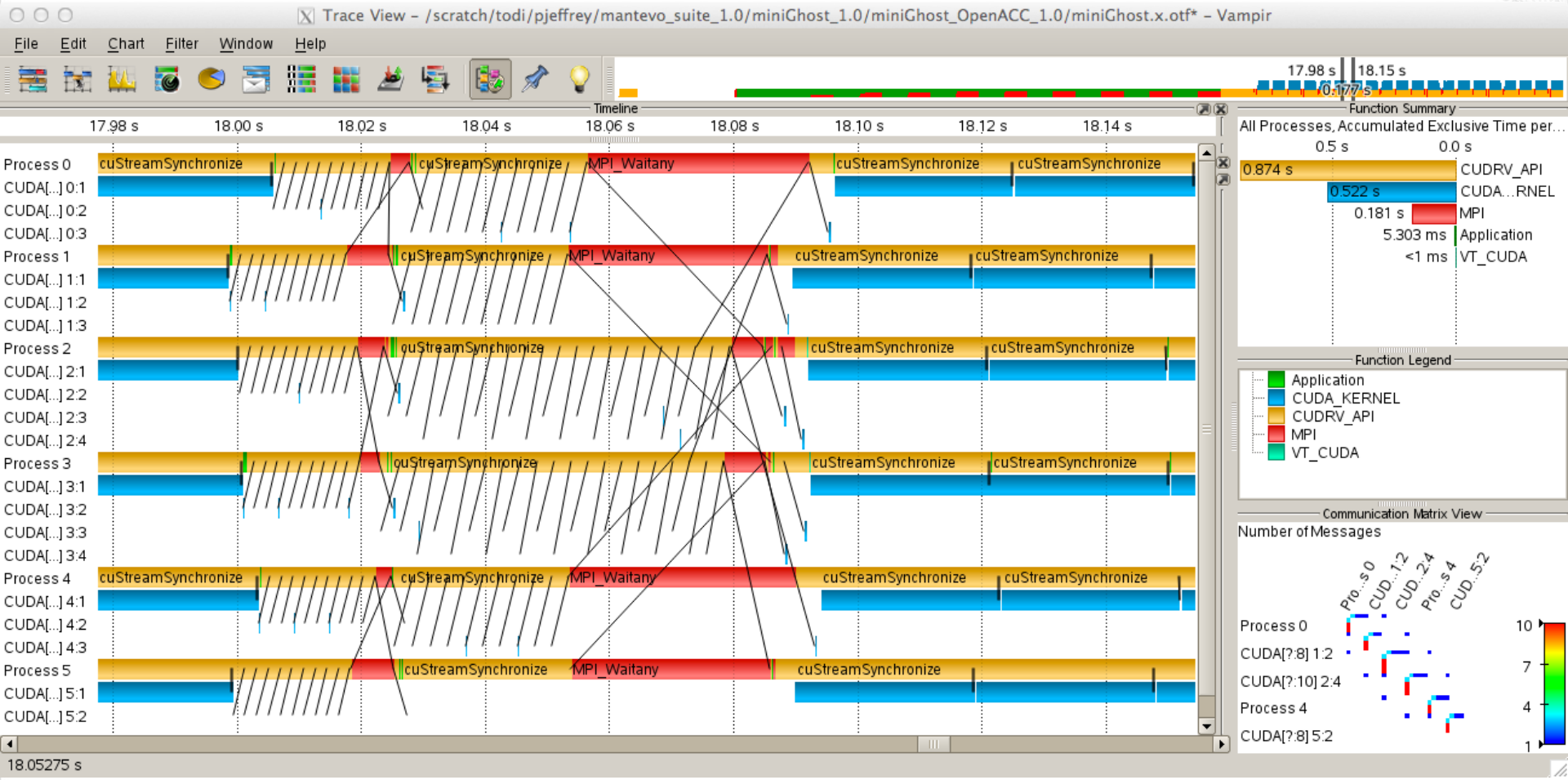
14. Continued

```
! Start communication - the _prep routines do posts and sends using buffer
! identified by itmp
call computeScalarGradient_prep_np(yspecies(1,1,1,1), 5,25,n_spec)
itmp = 4
istr = 19
call computeScalarGradient_prep( temp, itmp, istr )
itmp = 1
istr = 1
call computeVectorGradient_prep( u, itmp,istr )     endif
```

14. Continued

```
   do i = 1, reqcount
        call MPI_WAITANY(reqcount,req,index, stat, ierr )
if(direction(index).eq.1)then
!$acc update device(pos_f_x_buf(:,:,:,idx(index):idx(index)+nb(index)-1)) async(isync)
endif
if(direction(index).eq.2)then
!$acc update device(neg_f_x_buf(:,:,:,idx(index):idx(index)+nb(index)-1)) async(isync)
endif
if(direction(index).eq.3)then
!$acc update device(pos_f_y_buf(:,:,:,idx(index):idx(index)+nb(index)-1)) async(isync)
endif
if(direction(index).eq.4)then
!$acc update device(neg_f_y_buf(:,:,:,idx(index):idx(index)+nb(index)-1)) async(isync)
endif
if(direction(index).eq.5)then
!$acc update device(pos_f_z_buf(:,:,:,idx(index):idx(index)+nb(index)-1)) async(isync)
endif
if(direction(index).eq.6)then
!$acc update device(neg_f_z_buf(:,:,:,idx(index):idx(index)+nb(index)-1)) async(isync)
endif
isync=isync+1
    enddo
```

15. Start looking at timelines showing communication, host execution and accelerator



Strategy for refactoring the application

1. **First and foremost – Profile the application**
   Must identify looping structure within the time step loop
   Use –h profile_generate on compile and –Ocalltree or –Ocallers
2. **Use Reveal to identify scoping of variables in the major loop – may call subroutines and functions**
   The idea is to first generate OpenMP version of the loop and then add some OpenACC
3. **Use OpenACC to identify data motion require to run with companion accelerator**
   Once scoping is obtained, the OpenACC compiler will indicate what data would need to be moved to run on the accelerator – user must have the variable scoping correct
4. **Once one loop is analyze, now look at next highest compute loop, perform steps 2 and 3.**
5. **Soon multiple loops can be combined within a OpenACC data region for elminating transfers to and from the host.**

Strategy for refactoring the application

6. **Work outward until a data region encompasses a communication, I/O or looping structure more suited for the host**
   a. Must use updates to move data to and from the host to supply host with up-to-date data
7. **Move data region outside time step loop**
   a. Now must account for all updates to keep host and accelerator with consistent data
8. **Test versions after each step – don't worry about performance yet – just accuracy**
9. **The compiler may introduce data transfer so look at –rm listing for each individual OpenACC loop.**
10. **Optimize/Minimize data transfers first by using present on data clause.**

Strategy for refactoring the application

11. **Gather perftools statistics on code and identify bottlenecks**
12. **If bottleneck is data copies look at step 9**
13. **If bottleneck is kernel performance**
    A. Look at –rm and see what the compiler did to optimize the loop
    B. Ideally we want three levels of parallelism, gang, worker, vector
    C. Inner loop needs to be g on listing
    D. If inner loop is indicated by a loop level, that means that it is running in scalar – BAD
14. **Consider introducing CUDA streams**
    A. Either by taking an outer loop that cannot be parallelized due to communication and running that in a streaming mode
    B. Taking several independent operations and running that in a stream mode
15. **Start looking at timelines showing communication, host execution and accelerator**
    A. What can be overlapped

Analyze the code

- **Considering getting the maximum overlap of computation and communication**
  - Can some computation be delayed to allow for overlap of computation and communication

```
  call get_mass_frac( q, volum, yspecies )              ! get Ys from rho*Ys, volum from rho
  call get_velocity_vec( u, q, volum )                  ! fill the velocity vector
  call calc_inv_avg_mol_wt( yspecies, avmolwt )         ! set inverse of mixture MW
  call calc_temp(temp, q(:,:,:,5)*volum, u, yspecies )  ! set T, Cp_mix
  call calc_gamma( gamma, cpmix, mixMW )                ! set gamma
  call calc_press( pressure, q(:,:,:,4), temp, mixMW )  ! set pressure
!!!  Initiate Communication of temp, u and yspecies,mixMW
!!!  Wait for communication of halos of temp, u, yspecies,mixMW
```

  - Originally in S3D, all of the above computation was grouped and then halos temp,u and yspecies where communicated

```
  call get_mass_frac( q, volum, yspecies )              ! get Ys from rho*Ys, volum from rho
  call get_velocity_vec( u, q, volum )                  ! fill the velocity vector
  call calc_temp(temp, q(:,:,:,5)*volum, u, yspecies )  ! set T, Cp_mix
!!!  Initiate Communication of temp, u and yspecies
  call calc_inv_avg_mol_wt( yspecies, avmolwt )         ! set inverse of mixture MW
  call calc_gamma( gamma, cpmix, mixMW )                ! set gamma
  call calc_press( pressure, q(:,:,:,4), temp, mixMW )  ! set pressure
!!!  Wait for communication of halos of temp, u, yspecies
!!!  Initiate Communication of mixMW
```