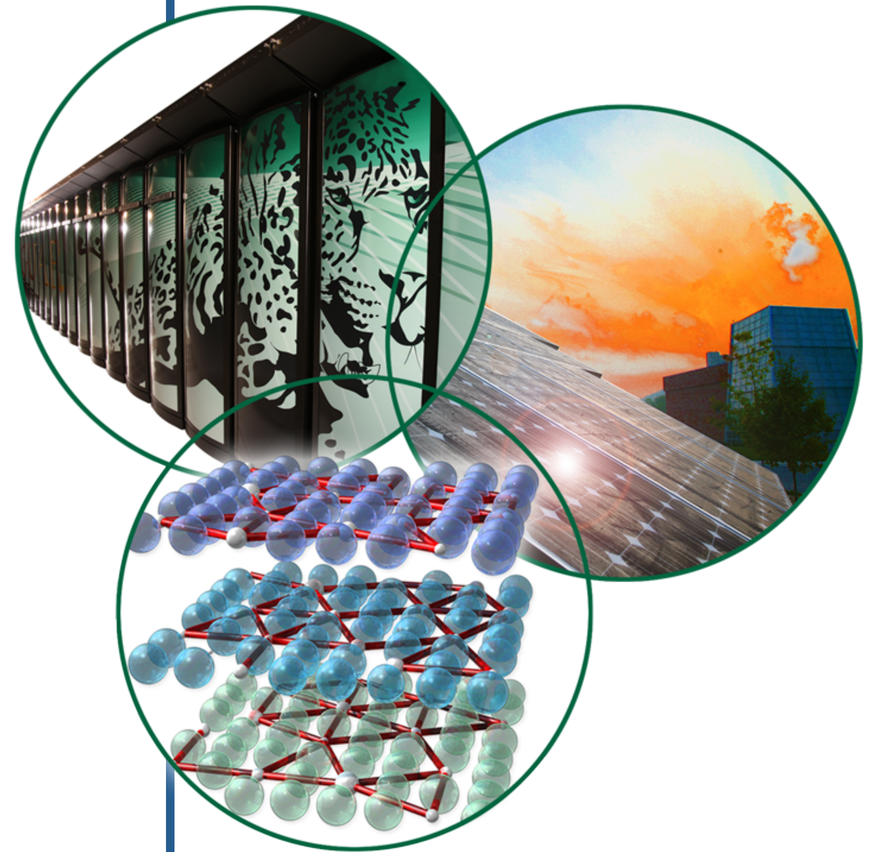
A Programming Environment for Heterogeneous Multi-Core Computer



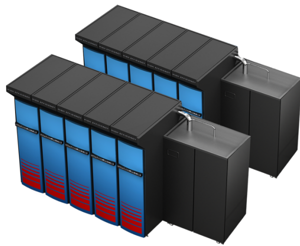
Richard Graham, Oscar Hernandez,
Thomas Ilsche, Christos Kartsaklis,
Tiffany Mintz, Pavel Shamis
Application Performance Tools Group
Oak Ridge National Laboratory

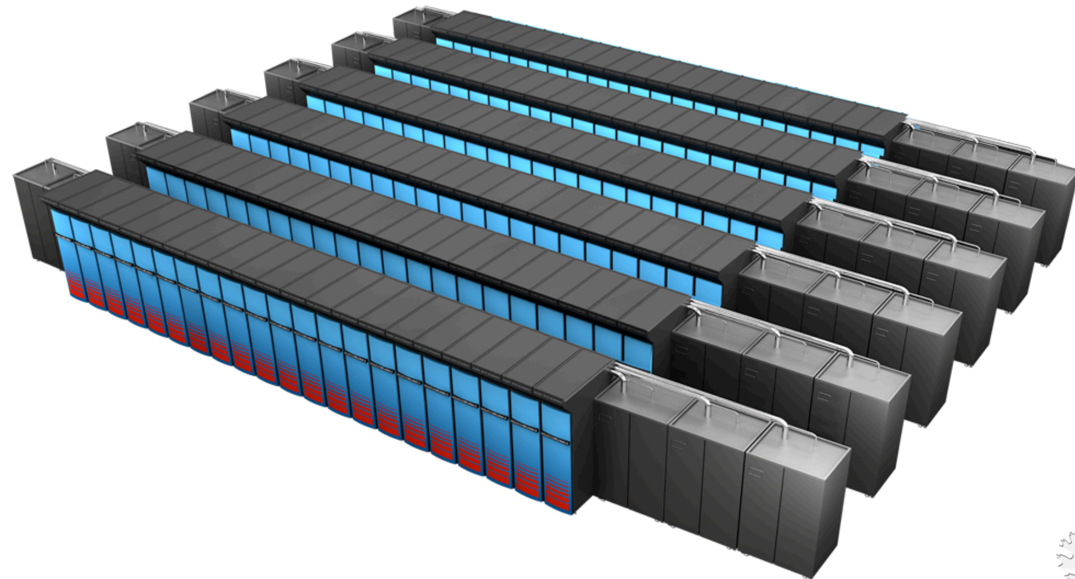
Application Performance Tools Group

- Compiler based code transformations
  - Exposing parallelism: Hercules
    - User directed pattern detection
    - Code transformations
  - Transforming large code bases
    - Klonos: Similarity detection
- High performance communication libraries
  - Hierarchical collectives
  - Scalable run-time support
- Fault-tolerant MPI/runtime

OLCF-3 hardware plan maximizes science output

Initial Delivery System (IDS)	Final System	Scalable File System
• 2nd half of 2011 • 900 TF peak • 10 cabinets • 920 compute nodes	• 2nd half of 2012 • Incorporates upgraded IDS • 16-20 PF peak	• Expansion of Spider • Adds 400-700 GB/s of bandwidth • Adds 10-30 PB





Goals

- Provide OLCF-3 users with high productivity programming tools & compilers.
- Provide a programming environment with tools to support the porting of codes.
- Work with vendors to provide compiler, performance, and debugger capabilities needed to port applications with GPUs :
  - CAPS enterprise (HMPP Directives)
  - The Portland Group (Accelerator Directives)
  - Cray (OpenMP for Accelerators)
  - NVIDIA
  - TU-Dresden (Vampir)
  - Allinea (DDT)
- Join Standardization Efforts: OpenMP ARB

Compilers

Improve Productivity and Portability

- The Directive based approach provides:
  - Incremental porting/development
  - Fast Prototyping
    - The programmer can quickly produce code that runs in the accelerator.
  - Increases Productivity
    - Few code modifications to produce accelerated code.
  - Retargability to different architectures (CPU, GPUs, FPGAs)
  - Tools can assist the user generate the directives, debug them, and do performance analysis.
- Leading technologies with accelerator directives:
  - CAPS HMPP directives, [Vendor]
  - PGI accelerator directives [Vendor]
  - Cray OpenMP accelerator directives [Vendor]
  - HiCUDA [Academic, University of Toronto]

Compilers Available for the OLCF-3 Effort

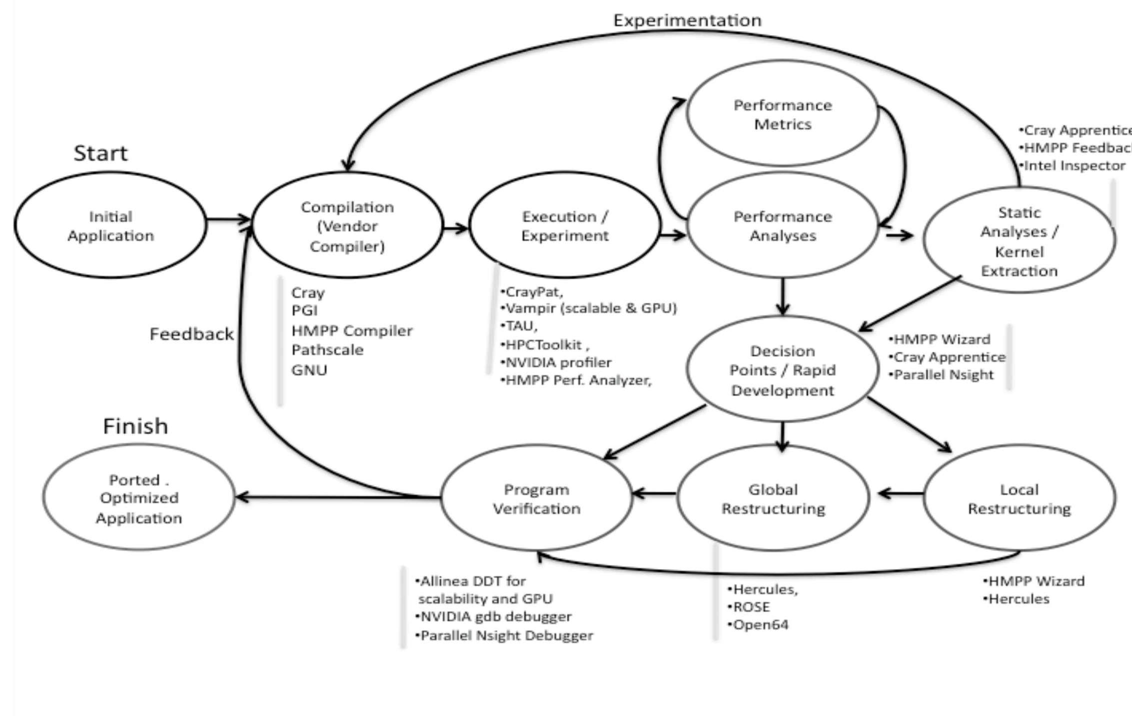
Compiler	C/C++	Fortran	CUDA C / OpenCL	CUDA Fortran	HMPP Acc Dir	PGI Acc Dir	OpenMP Acc Dir	OpenMP CPU
Cray	X	X					P	X
PGI	X	X	P	X		X		X
CAPS	X	X			X			X
NVIDIA			X					
Pathscal	X	X			P			X
Intel	X	X						X
GNU	X	X						X
LLVM	X	X	X					X

X = Supported
P = In Progress

- Cray, CAPS, and NVIDIA are directly involved with the OLCF-3 Effort

Current Work

- We are currently working with Vendors to provide a set of tools that target the application needs for OLCF-3
  - Benchmarks and Applications
- We are also building a tool environment to support the applications:



OAK RIDGE National Laboratory

HMPP Programming Environment

- System supports: C/C++/Fortran, OpenMP, MPI, SHMEM
- In addition, Identified features needed for programmability and performance improvements
  - C++ support [Feature]
  - Parsing issues (Fortran and C) [bugs]
  - Fortran support (Modules) [Feature]
  - Inlining support [Feature]
  - Need to allocate data directly in the GPU [Performance]
  - 3D scheduling support for thread blocks [Performance]
  - Support for libraries [Feature]
  - Codelet Functions [Feature]
  - Concurrent kernel execution in Fermi [Performance]
  - Worksharing between devices in nodes. [Feature]

Example: HMPP C++ Directives Support

- Initial Implementation of C++ directives using MADNESS

HMPP++ Codelet Declaration

```
class SmallComputation : public hmpp::HMPP
{
 public:
 SmallComputation(void) : HMPP() {}

 #pragma hmpp mapbyname, datain, dataout
 hmpp::Argument datain, dataout;

 #pragma hmpp ope codelet, args[dataout].io=out, target=CUDA
 void operation(int n, float *datain, float *dataout,
                const floatfactor)
 {
 #pragma hmppcg parallel
   for (int i = 0; i<n ; i++) {
     dataout[i] = cos(datain[i]) * factor;
   }
  }
};
```

User class inherits from HMPP

Map buffer with codelet parameter

Declare buffer objects

Declare codelet

Use code generation directives

HMPP++ in MADNESS

- Defined Baseline Benchmark
  - Accelerate hotspot:
    - madness::mTxmq (81% time spent)
    - /src/lib/tensor/mtxmq.cc
- Parsing Capabilities for C++
- Implementation of HMPP++
- Hybrid Execution
  - Small matrices executed in CPU
    - MLK Library
  - Large Matrices executed in GPU

```
#pragma hmppcc classlet target=CUDA
  class HybridMatrixMulImpl : public hmppcc::Classlet
  {
  private :
    long m_dimi, m_dimj, m_dimk;
    bool m_gpu;
    bool m_trace;
    bool m_wasAllocated;
```

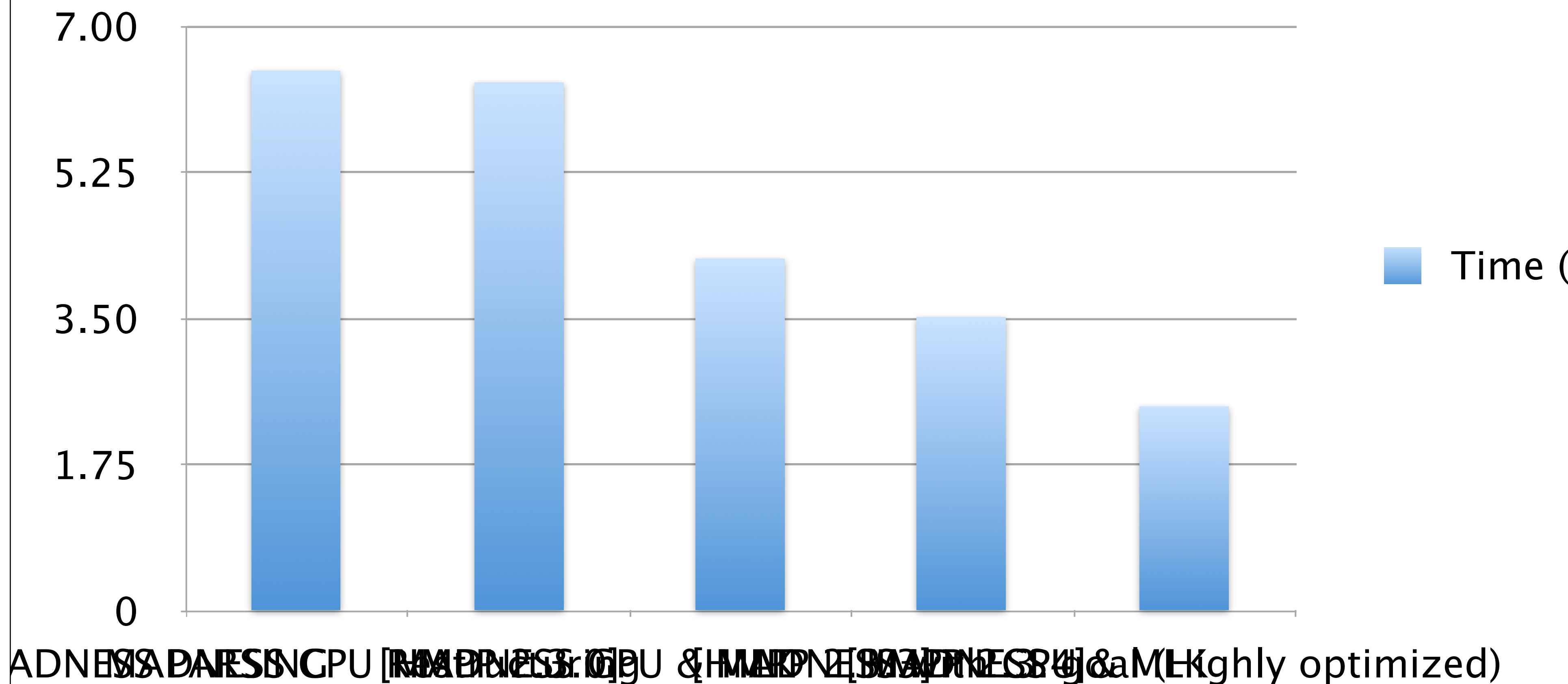
```
#pragma hmppcc codelet,args[c].io=out,args[a;b].io=in, &
#pragma hmppcc & args[dimi].map="dimi",args[dimj].map="dimj", &
#pragma hmppcc & args[dimk].map="dimk",args[c].map="c",  &
#pragma hmppcc & args[b].map="b",args[a].map="a"
    void mTxmq_double_double_gpu(long dimi, long dimj, long dimk,
                                 double* restrict c,
                                 const double* a, const double* b)
    {
      // Pragmas here are optional, loops are automatically detected as
(not)/parallel
#pragma hmppcg parallel
      for(long i=0; i<dimi; i++)
      {
#pragma hmppcg parallel
        for(long j=0; j<dimj; j++)
        {
          double tmp = 0.0f;

#pragma hmppcg noparallel
          for(long k=0; k<dimk; k++)
          {
            tmp = tmp + a[k*dimi+i]*b[k*dimj+j];
          }
          c[i*dimj+j] = tmp;
        }
      }
    }
```


Oak Ridge National Laboratory

Acceleration of MADNESS::MTXMQ

Improvements over HMPP++



HMPP with CAM – SE (Climate)

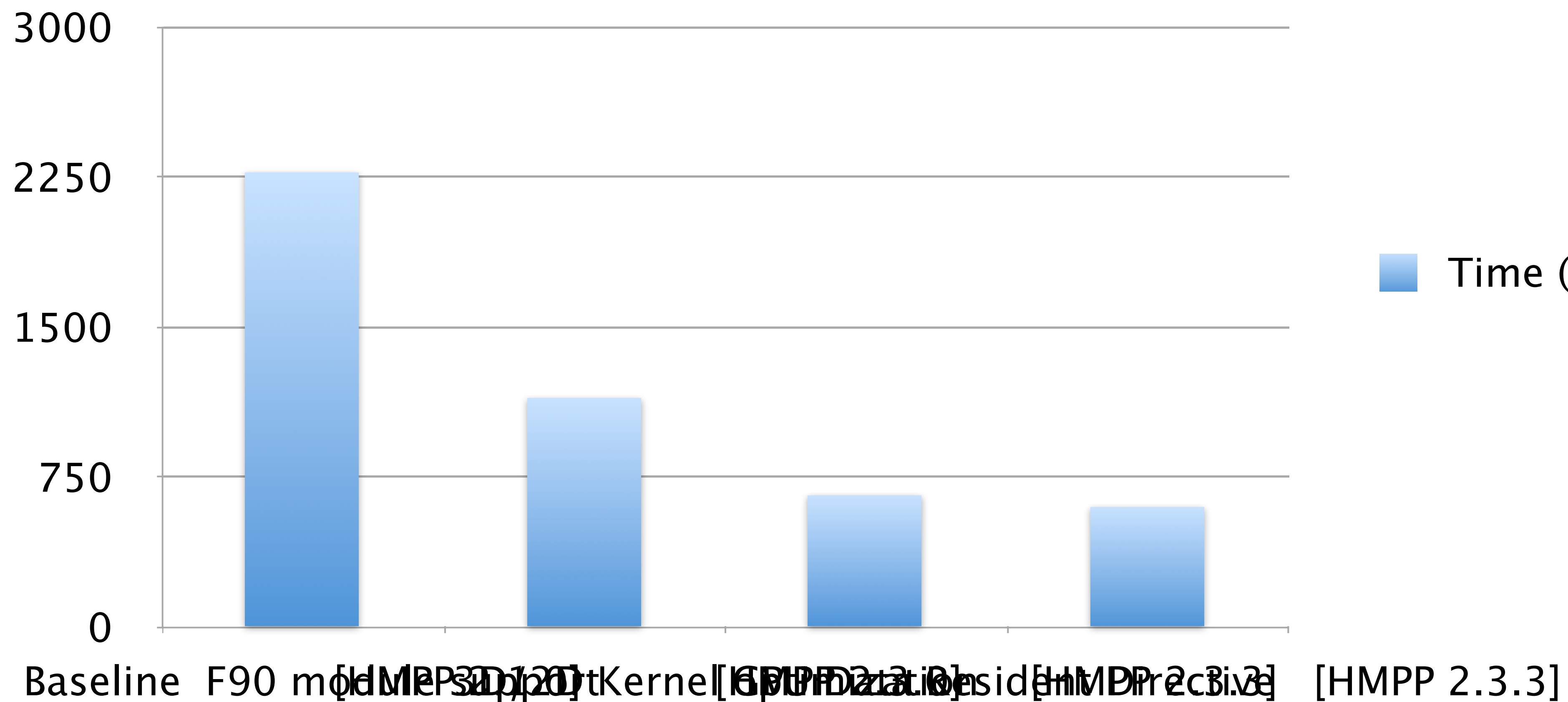
- Defined a baseline benchmark
  - Accelerate divervenge_sphere()
- Support for F90 Modules
- Support for 2D/3D scheduling
- Data Resident Directive
  - Allocate Data in the GPU
- Need for Data Distribution
  - Distribute elements in GPU/CPU
  - Share work among GPU/CPU cores

```
1  subroutine driver
2         ...
3    !$hmpp <cudagroup> group, target=CUDA
4     rdx=2.0D0/(elem(ie)%dx*rearth)
5     rdy=2.0D0/(elem(ie)%dy*rearth)
6           ...
7     metdet = elem(ie)%metdet
8     !$hmpp <cudagroup> kernel callsite
9     call divergence_sphere5d_hmpp_tuned_dw(ie,qsize,...,rmetdetp,divdp4d_omp)
10    ret2(1,ie) = sum(divdp4d_omp(:,:,:,:))
11        ...
12 end subroutine
13
14 !$hmpp <cudagroup> kernel codelet,args[qsize_d;...;metdet;rmetdetp].io=in,args[divdp4d_omp].io=out
15 subroutine divergence_sphere5d_hmpp_tuned_dw (qsize_d, ... , divdp4d_omp)
16   implicit none
17   integer, intent(in) :: qsize_d, nlev_d, nv_d
18   real(kind=8) rdx, rdy
19   real(kind=8), intent(in) :: Dvv(nv_d,nv_d)
20   real(kind=8), intent(in) :: gradQ5d(nv_d,nv_d,nlev_d,qsize_d,2)
21          ...
22   real(kind=8), intent(out) :: divdp4d_omp(nv_d,nv_d,nlev_d,qsize_d)
23          ...
24   do q=1,qsize_d
25     do k=1,nlev_d
26       do j=1,nv_d
27         do l=1,nv_d
28           dudx00=0.0d0
29           dvdy00=0.0d0
30           do i=1,nv_d
31             dudx00 = dudx00 + Dvv(i,l  )*(metdet(i,j)*(Dinv11(i,j)*gradQ5d(i,j,k,q,1) + &
32                                           Dinv(1,2,i,j)*gradQ5d(i,j,k,q,2)))
33             dvdy00 = dvdy00 + Dvv(i,j  ) * (metdet(l,i)*(Dinv(2,1,l,i)*gradQ5d(l,i,k,q,1) + &
34                                           Dinv(2,2,l,i)*gradQ5d(l,i,k,q,2)))
35           end do
36           divdp4d_omp(l,j,k,q)= rmetdetp(l,j) * (rdx*dudx00+rdy*dvdy00)
37         end do
38       end do
39     end do
40   end do
41 end subroutine divergence_sphere5d_hmpp_tuned_dw
```


OAK RIDGE National Laboratory

Acceleration of CAM - SE

Improvements of HMPP divergence_sphere()

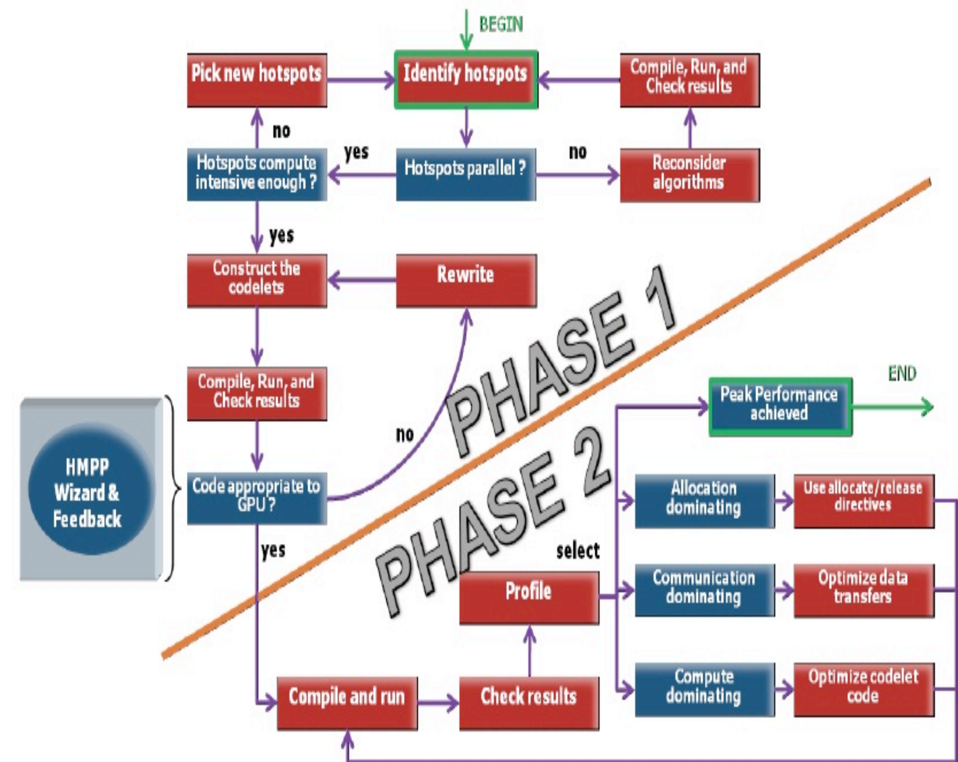


Additional Features in HMPP 2.3.3/2.3.4

- HMPP Dynamic Management of CPU/GPU data coherency
  - Data that were copied from host to device with the intent to remain resident on the HWA and which the host may modify, HMPP automatically manages the necessary updates.
- CUDA shared memory direct copy
  - Stage data that originate from host memory directly in device shared memory (this needs to be programmed explicitly in CUDA).
- User Kernel Integration
  - Allows the user to override HMPP codelets with own, optimized versions.

Tool support: HMPP Wizard

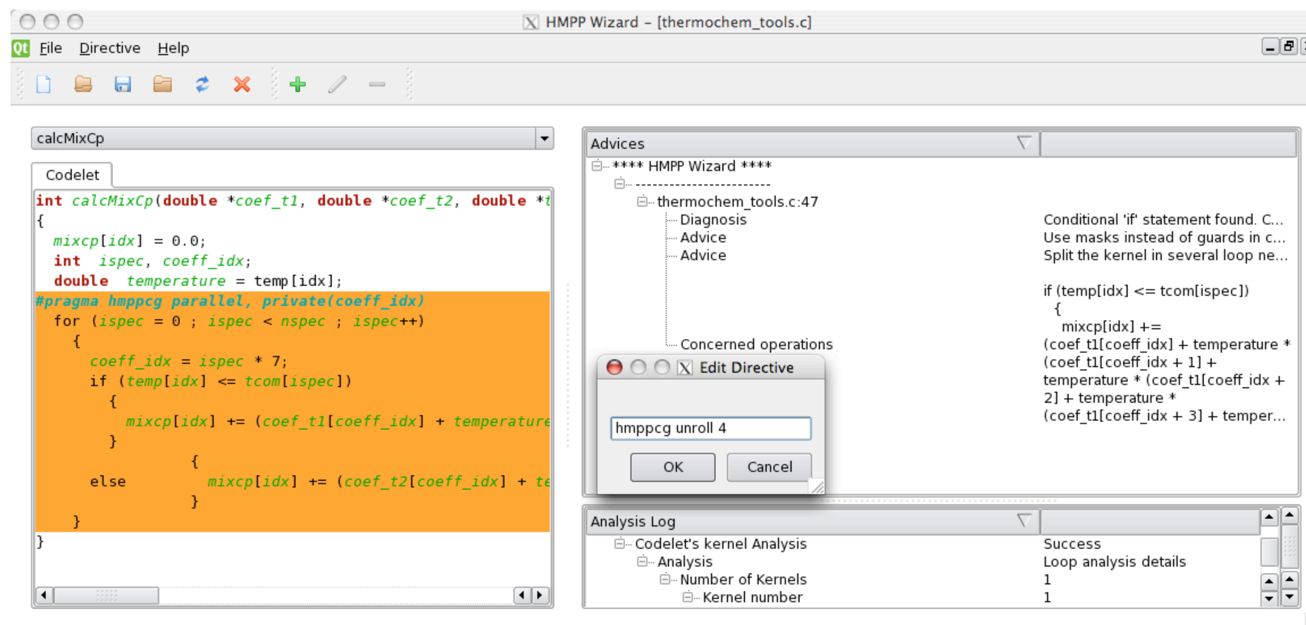
- Objective
  - Designed to give pertinent, efficient optimization advice for inserting directives and kernel optimization
  - Interactive way of automatically applying proposed optimizations
- Approach
  - Very Interactive
  - Step-by-step diagnoses
    - Validation
    - Pattern Matching
  - Phase 1: Codelet Identification
  - Phase 2: Codelet Optimization



OAK RIDGE
National Laboratory

Running HMPP Wizard

- Launching HMPP Wizard
  - Setup wizard environment
    - $ source “HMPP_WIZARD_INSTALLATION_DIRECTORY”/bin/wizard-env.sh
  - Compilation command parameter
    - $ hmppWizard [--help] [-v] [-V] <compiler> [compiler options] [<file>.c]



HMPP Wizard UI with the S3D application

HMPP Wizard Directive Insertion

Codelet Browser

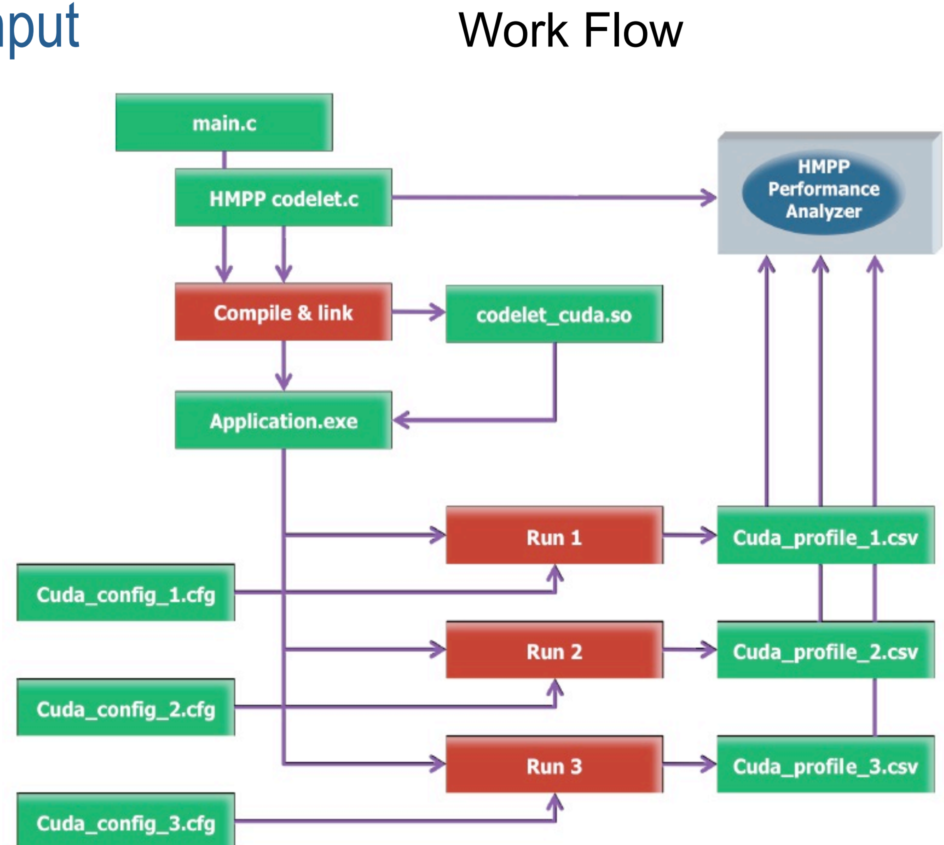
Insertion of HMPP directives

Diagnostics Information

HMPP Performance Analyzer Metrics

- Performance Analysis for HMPP codelets
- Analysis and Metrics for:
  - Average GPU Execution Time
  - Gridification / Grid / Thread Block Size
  - Global Memory Read/Write Throughput
  - Load/Store Execution Density
  - Load/Store Code Density
  - Computation Density
  - Branch Divergence Ratio

Work Flow



Performance Analyzer Graphical Interface

Kernel Performance Information

Codelet

```
#pragma hmpp <Group> group target=CUDA
#pragma hmpp <Group> mapbyname m, n, p, alpha, vin1, vin2, beta, vout
#pragma hmpp <Group> dgemm codelet, args[vout].io=inout
void dgemm(int m, int n, int p, real alpha, const real vin1[n][m], const r
{
  int i, j, k;
  for (j = 0 ; j < p ; j++)
    {
      for (i = 0 ; i < m ; i++)
        {
          double prod = 0.0;
          double v1a, v2a;
          k = 0;
          v1a = vin1[k][i];
          v2a = vin2[j][k];
          for (k = 1 ; k < n ; k++)
            {
              prod += v1a * v2a;
              v1a = vin1[k][i];
              v2a = vin2[j][k];
            }
          prod += v1a * v2a;
          vout[j][i] = alpha * prod + beta * vout[j][i];
        }
    }
  for (j = 0 ; j < p ; j++)
    {
      for (i = 0 ; i < m ; i++)
        {
          if (i % 2)
            vout[j][i] *= 3.14;
          else
            vout[j][i] *= -6.28;
        }
    }
}
```

HMPP Codelet

Kernel 1 gradient

Kernel 2 gradient

Kernel selected

Codelet <Group> 'dgemm' Kernel #1 99%

Kernel #2 0%

HMPP PerfAnalyzer - [dgemm-codelet.c] <@amrod>

File Directive Help

dgemm

Advices

**** HMPP PerfAnalyzer ****
**** Advices ****
dgemm-codelet.c:57
dgemm-codelet.c:74
dgemm-codelet.c:74
**** Profile ****
Kernel number
Kernel Name:
Average gpu execut
Gridification:
Global memory reac
Global memory writ
Global memory thro
Load/Store Executic
Branch divergence
Computation densit
Load/Store Code De
Detailled metrics
Kernel number
Kernel Name:
Average gpu execut
Gridification:
Global memory reac
Global memory writ
Global memory thro
Load/Store Executic
Branch divergence
Computation densit
Load/Store Code De
Detailled metrics

Analysis Log

**** Analysis Logs ****
Codelet Validation
Profile Analysis
Analysis Report
Profile Files
Profile Loaded
Profile Loaded
Profile Loaded

Performance Analysis

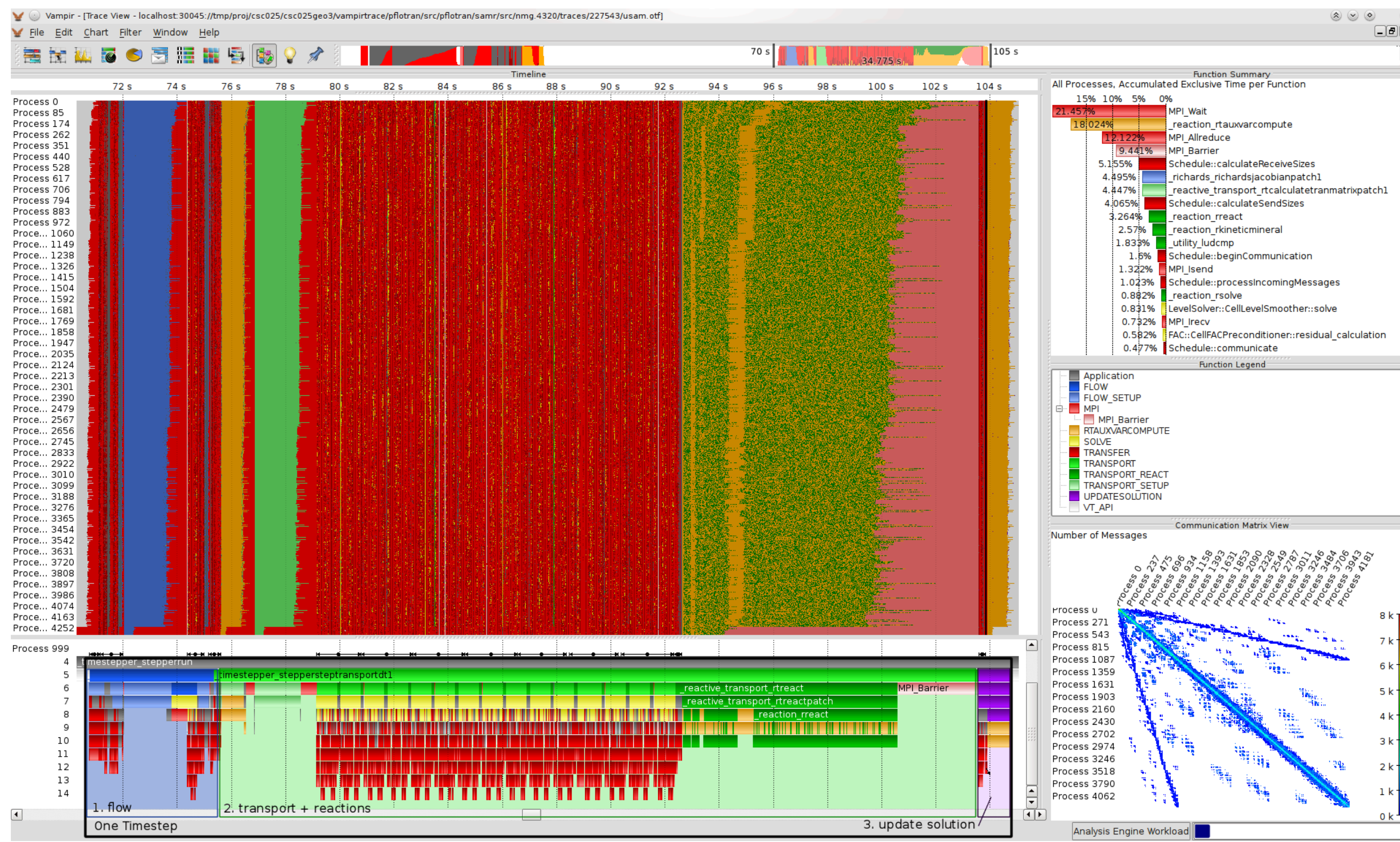
Oak Ridge National Laboratory

Vampir Performance Analysis Tools

- Trace-based performance analysis tool set
- Custom improvements for the OLCF-3 system
- Focused on two main areas
  1. Scaling the Vampir tool set to higher processor counts
  2. Integrating GPU support for a comprehensive analysis of heterogeneous systems
  - Additional usability enhancements

Vampir - Example Trace

- Application: PFLOTRAN-AMR
- Main timeline shows application behavior for all processes over time



Vampir - Example trace

- Application: PFLOTRAN-AMR
- Zoom into specific MPI behavior

Vampir - scalability improvements

- Optimization of communication patterns in parallel analysis server
  - Analysis server runs well using > 10,000 processes
- Parallel versions of additional post-processing tools
  - Filtering and merging of > 30,000 process traces now feasible
  - Parallel generation of PDF profiles from the trace

Vampir - Scalability Challenges

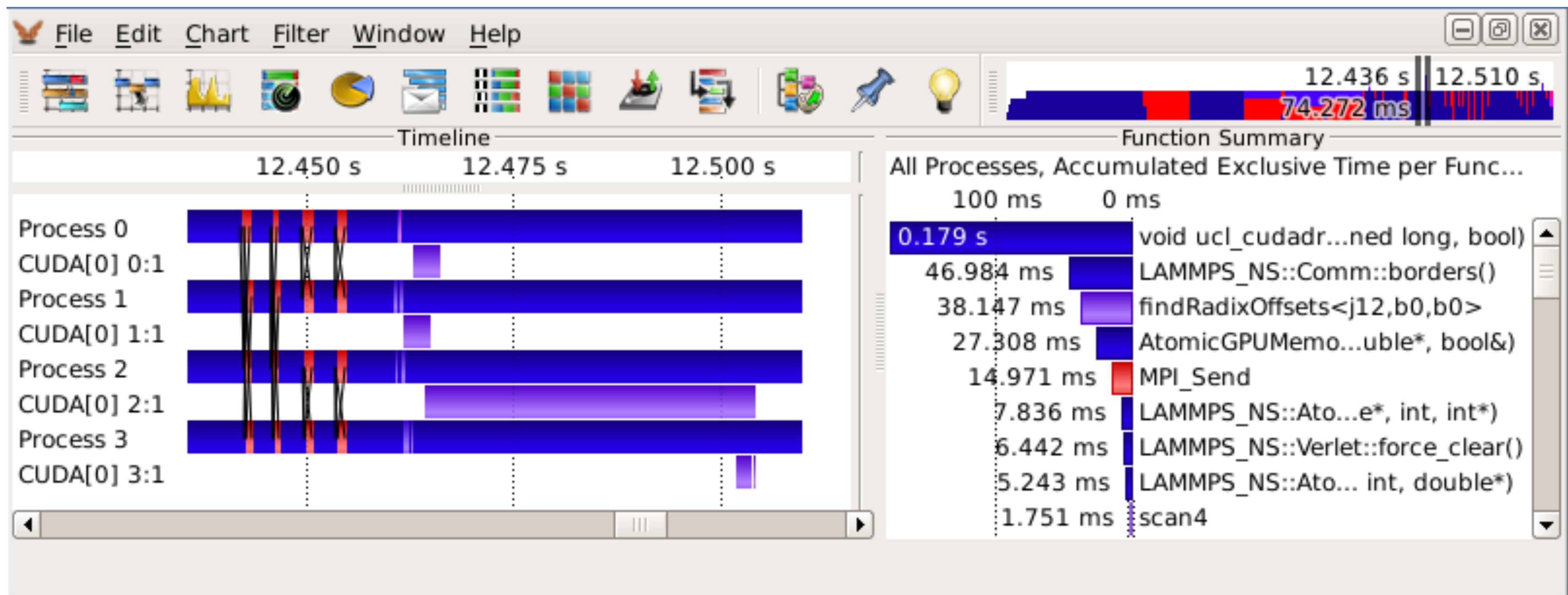
- Better handling of files during the trace generation
- Pattern matching to handle increasing number of trace events generated to 100,000s of threads
- User interface improvements to make huge traces accessible

Vampir - CUDA Support

- Wrapping of the CUDA runtime library provides basic information about CUDA events
  - GPU kernel execution (like functions in a CPU program)
  - Memory copies between GPU and Host memory (like MPI communication)
  - Works with asynchronous operations
  - Embedded CUPTI performance counters
- All GPU information is embedded into the trace containing MPI, OpenMP, CPU, PAPI etc.
  - Allows to analyze GPU code in the context of the real application rather than looking at an individual kernel

Vampir - CUDA Example

- Application: GPU-LAMMPS
- 4 MPI processes each with a CUDA stream



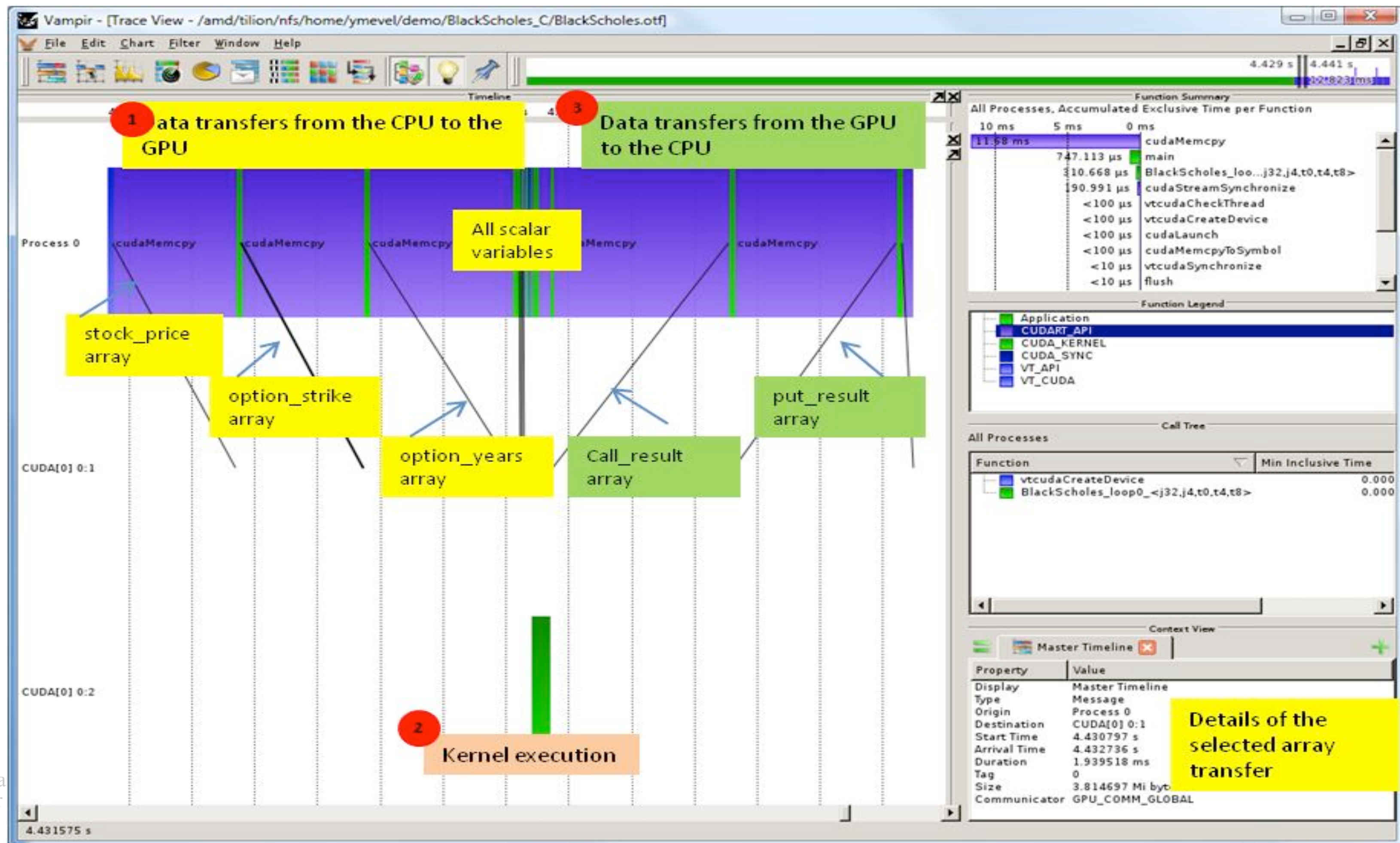
Oak Ridge National Laboratory

HMPP and Vampir Integration

- Integration with Vampir for scalable performance analysis is operational
  - Some operations of HMPP at CUDA level cause runtime errors with VampirTrace → under investigation
- HMPP and VampirTrace can be combined without explicit integration
  - HMPP does source to source transformation
  - VampirTrace wraps CUDA library calls
  - Vampir visualizes CUDA memory copies and kernel execution generated by HMPP
  - Both tools utilize compiler wrappers, handle with care
- Next step focuses on integrating HMPP semantics (e.g. HMPP arguments) into the trace

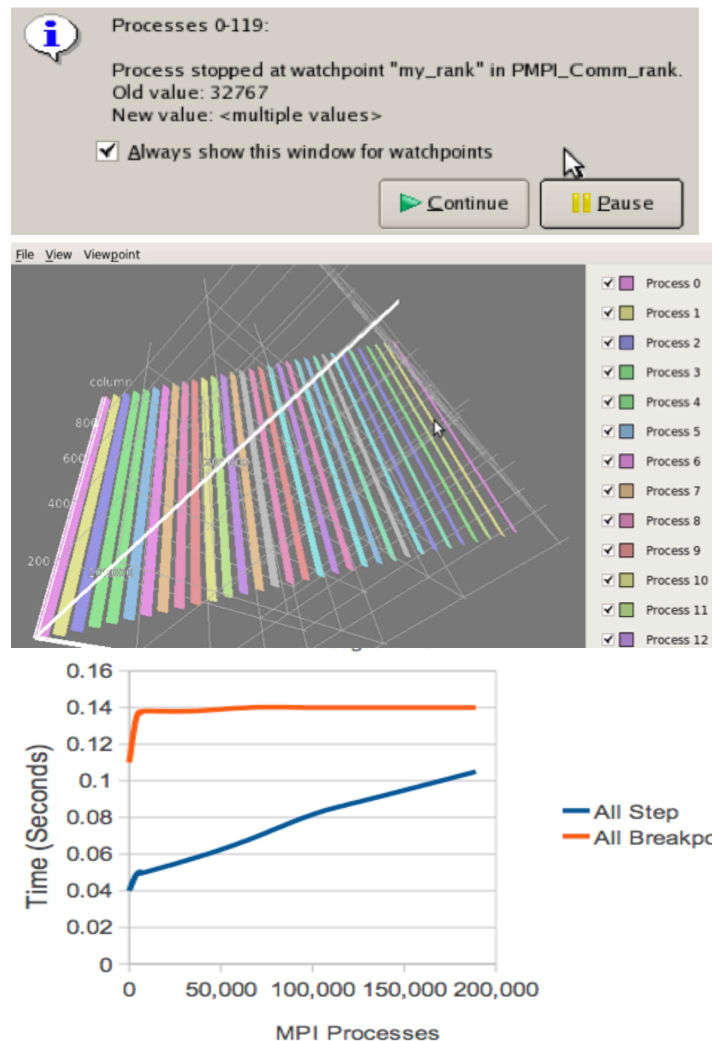
HMPP with Vampir

- VT_INST=compinst VT_CC=gcc-4.3.5 \
  hmpp vtcc -vt:verbose -c -DFLAG \
  -Wno-unknown-pragmas file.c -lcudart \
  -L$CUDA_HOME/lib64/



Debuggers

- Work with Allinea DDT to improve the scalability of the debugger
- Data Analysis
  - Parallel Watchpoints
  - Scalable data analysis
  - Scalable breakpoints, stepping and program stack queries



Debuggers

- Tight integration with Cray PE
  - Support for Abnormal Process Termination (APT), allows to attach DDT to aborted process and review stack

```
Application 1110443 is crashing. ATP analysis proceeding...

Stack walkback for Rank 23 starting:
  _start@start.S:113
  __libc_start_main@libc-start.c:220
  main@atploop.c:48
  __kill@0x4b5be7
Stack walkback for Rank 23 done
Process died with signal 11: 'Segmentation fault'
View application merged backtrace tree file 'atpMergedBT.dot' with
'statview'
You may need to 'module load stat'.

atpFrontend: Waiting 5 minutes for debugger to attach...
```

  - Multiple core file support using xt_setup_core_handler()
  - Open MPI (Cray XT/alps) version support

Debuggers

- Support for CUDA & accelerator directives