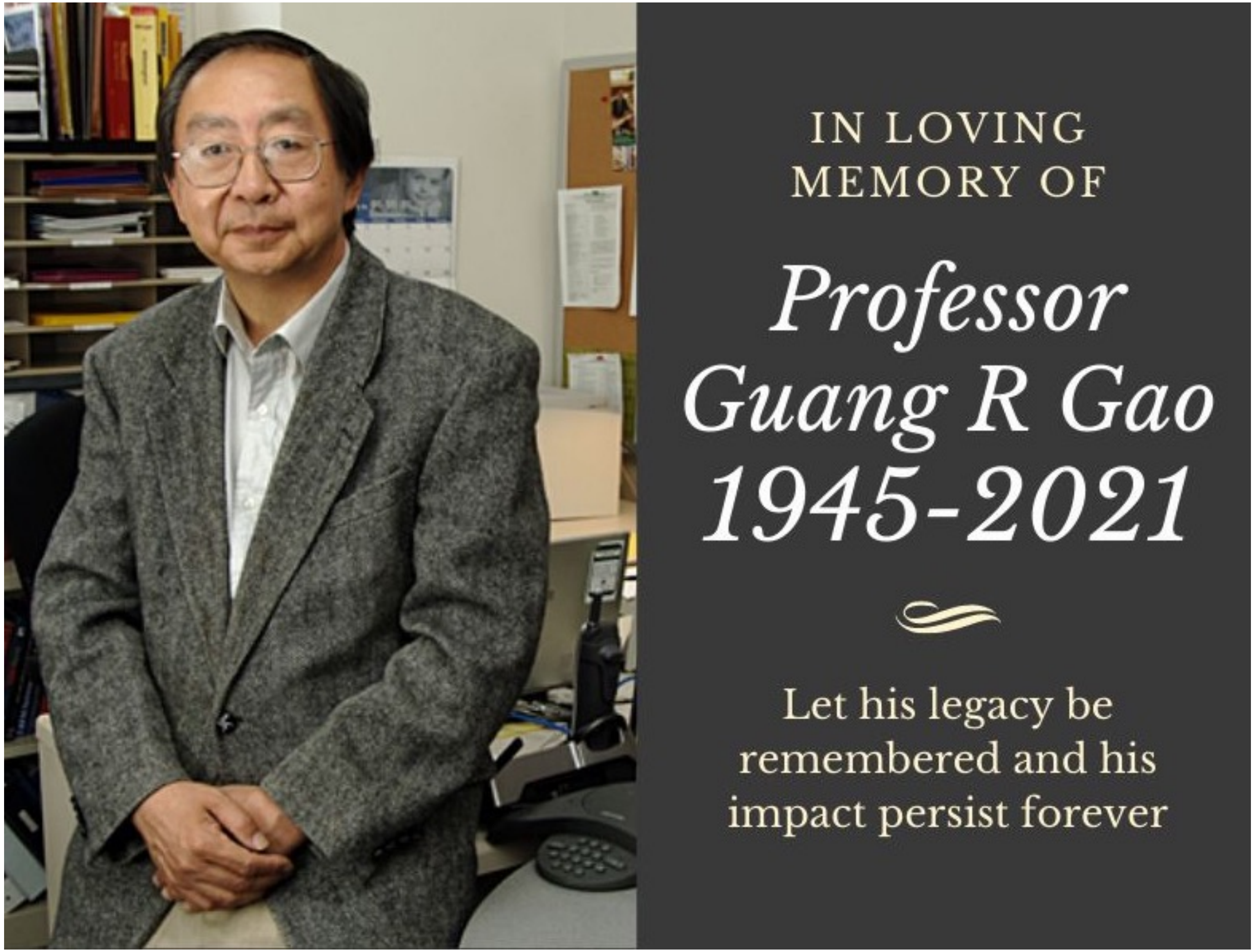
Hewlett Packard Enterprise

DIRECTIVES ROCK!

OPENACC SUMMIT

September 15, 2021

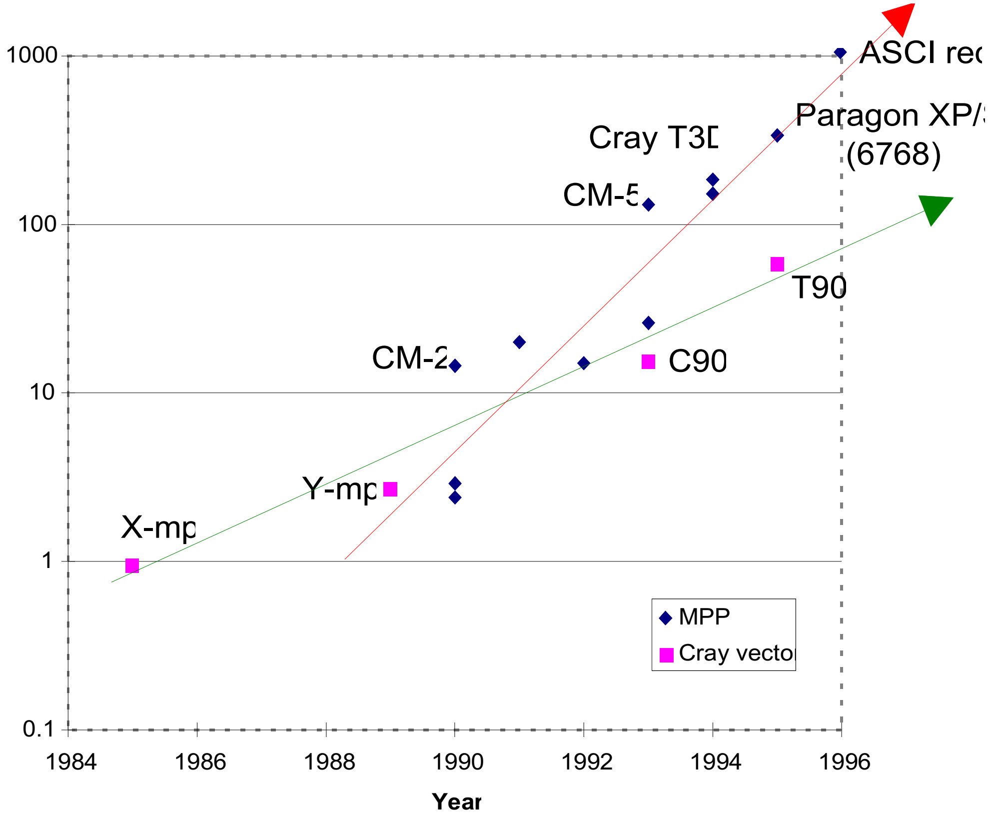
IN REMEMBRANCE



LONG LIVE DIRECTIVES!

- New languages get a lot of attention
  - But few are widely adopted

- Directives have been a key means to program supercomputers from the earliest days
  - Practical
  - Performant
  - Powerful
  - Productive
  - Sometimes portable



HIGH PERFORMANCE FORTRAN (HPF)

- Directives extend Fortran for *distributed* (inter-node) memory parallel programming
  - First definition early 1993, revision 1997
  - Japanese created additional features in JA-HPF
- Main idea is to enable the application developer to achieve data locality
- Main features are **directives** for data mapping and parallel loops
  - Work performed where the data is stored
  - Some library routines
- Broad participation in standards effort

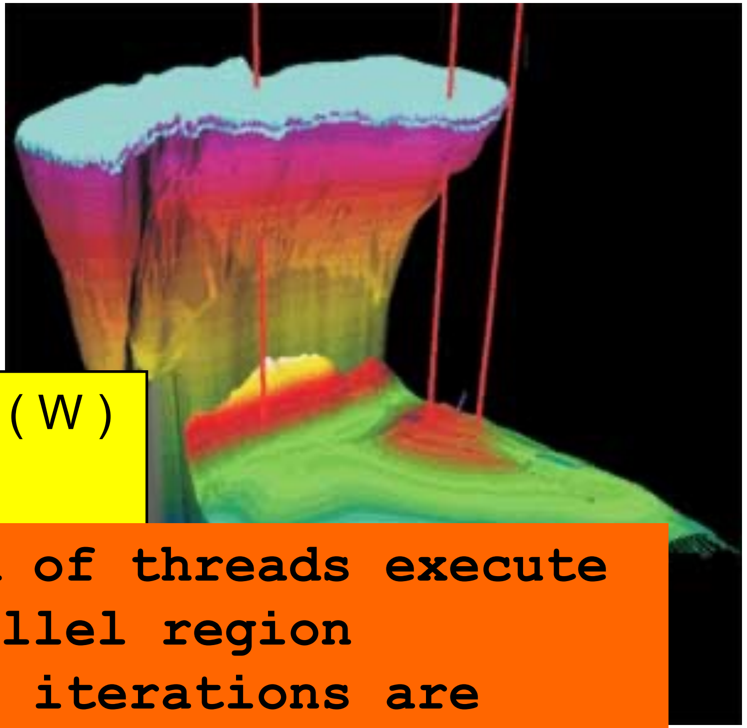
```
!HPF$  DISTRIBUTE W ( BLOCK )
!HPF$  INDEPENDENT, NEW ( X ),  REDUCTION ( SUM )

           DO I = 1, N
               X = W(I) * (I - 0.5)
               SUM = SUM + F ( X )
           END DO
```

```
* Team of processes execute
  entire program
* Loop iterations are
  distributed among processes
  based on data distribution
* Communication at end of loop
  to obtain global value SUM
```

OPENMP

De-facto standard API to write shared memory parallel applications in C, C++, and Fortran

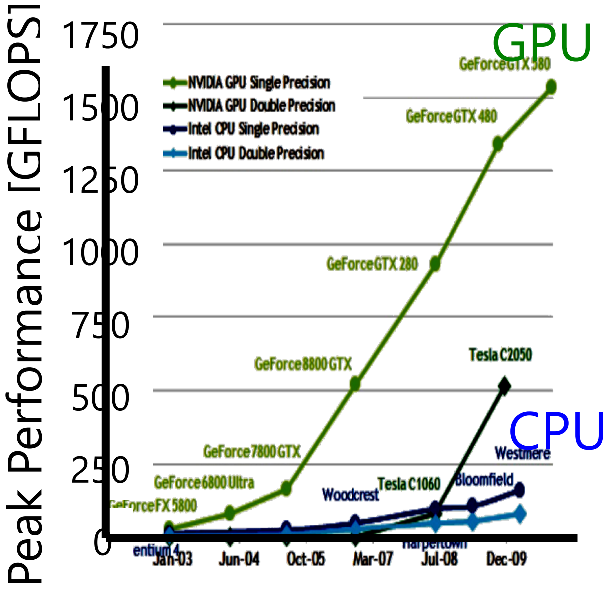


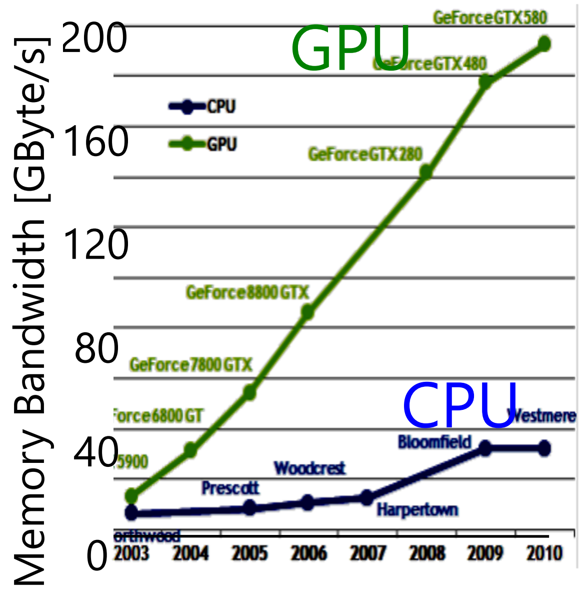
```
!$OMP PARALLEL DO PRIVATE ( X ) , SHARED ( W )
!$OMP& REDUCTION ( +: SUM )
        DO I = 1, N
           X = W(I) * (I - 0.5)
           SUM = SUM + F ( X )
        END DO
!$OMP END PARALLEL
```

* Team of threads execute parallel region
* Loop iterations are distributed among threads
* Implicit synchronization at end of region

* All threads access same W
* Each executing thread has its own copy of variable X
* Each thread creates and initializes a private copy of shared variable SUM.
* SUM is updated at next synchronization point

TSUBAME2.0 GPU RATIONALIZATION





- ~3000 CPUs at 200+ Teraflops, ~4000 GPUs at 2.2 Petaflops
- Realistic best case: x5~6 perf gain per socket
  - Machine equivalent to 25,000~30,000 CPUs
- Alternative: CPU only, same $$$ and Power, how big a system?
  - Answer: at best 5~6000 CPUs (Tsubame 1.0) at 400+ Teraflops
- **CPU equivalency = 1.4 x utilization x perf gain > 1.0 then we win!**
- No religious war but simple economics

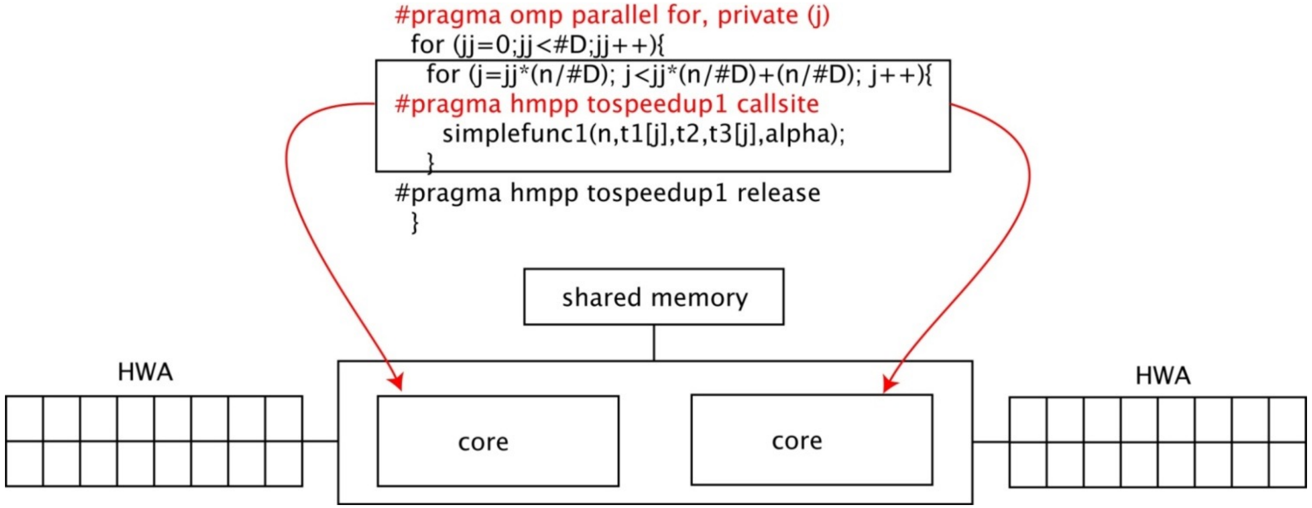
Satoshi Matsuoka, TiTech, 2010

CAPS HMPP: ALREADY THINKING ABOUT ACCELERATOR DEVICES

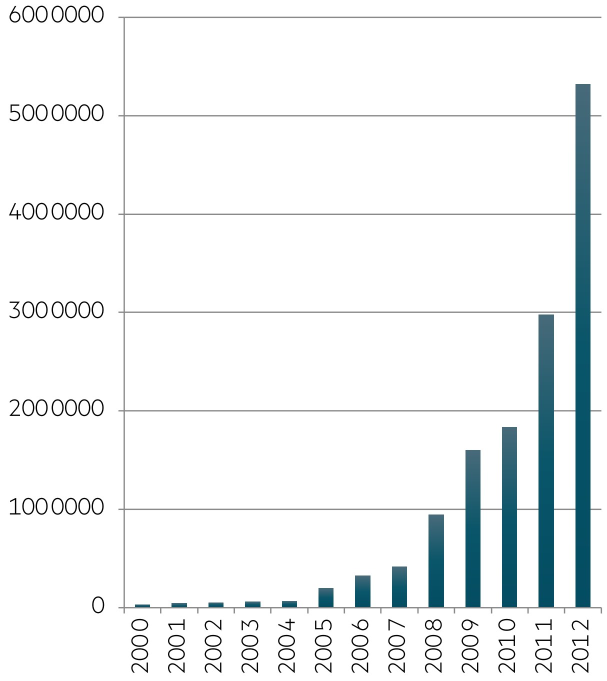
- Declare hardware specific implementations of functions (HMPP codelets)
  - Can be specialized to the execution context (data size, …)
- Codelet calls (RPC)
  - Synchronous, asynchronous properties
- Data transfers
  - Data prefetching
- Synchronization barriers
  - Host CPU will wait until remote computation is complete

- Use #D accelerators in parallel

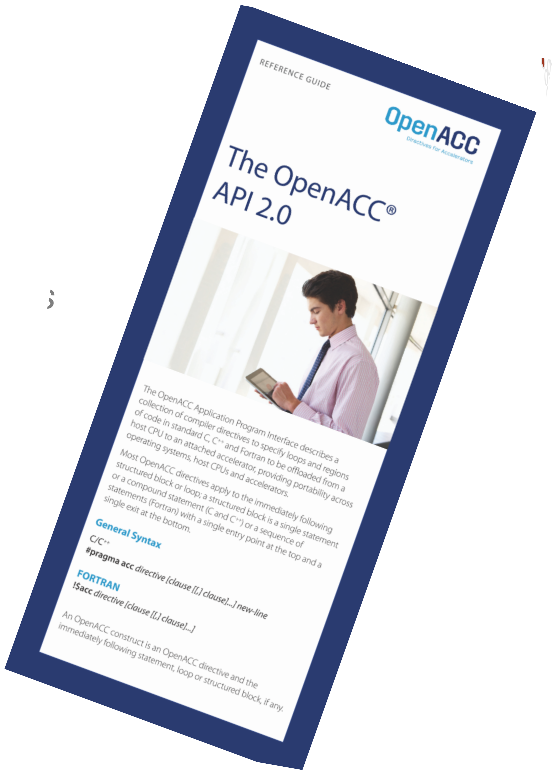
```
#pragma omp parallel for, private (j)
  for (jj=0;jj<#D;jj++){
    for (j=jj*(n/#D); j<jj*(n/#D)+(n/#D); j++){
#pragma hmpp tospeedup1 callsite
      simplefunc1(n,t1[j],t2,t3[j],alpha);
    }
#pragma hmpp tospeedup1 release
  }
```



RAPID GROWTH OF ACCELERATED HPC BEGINS

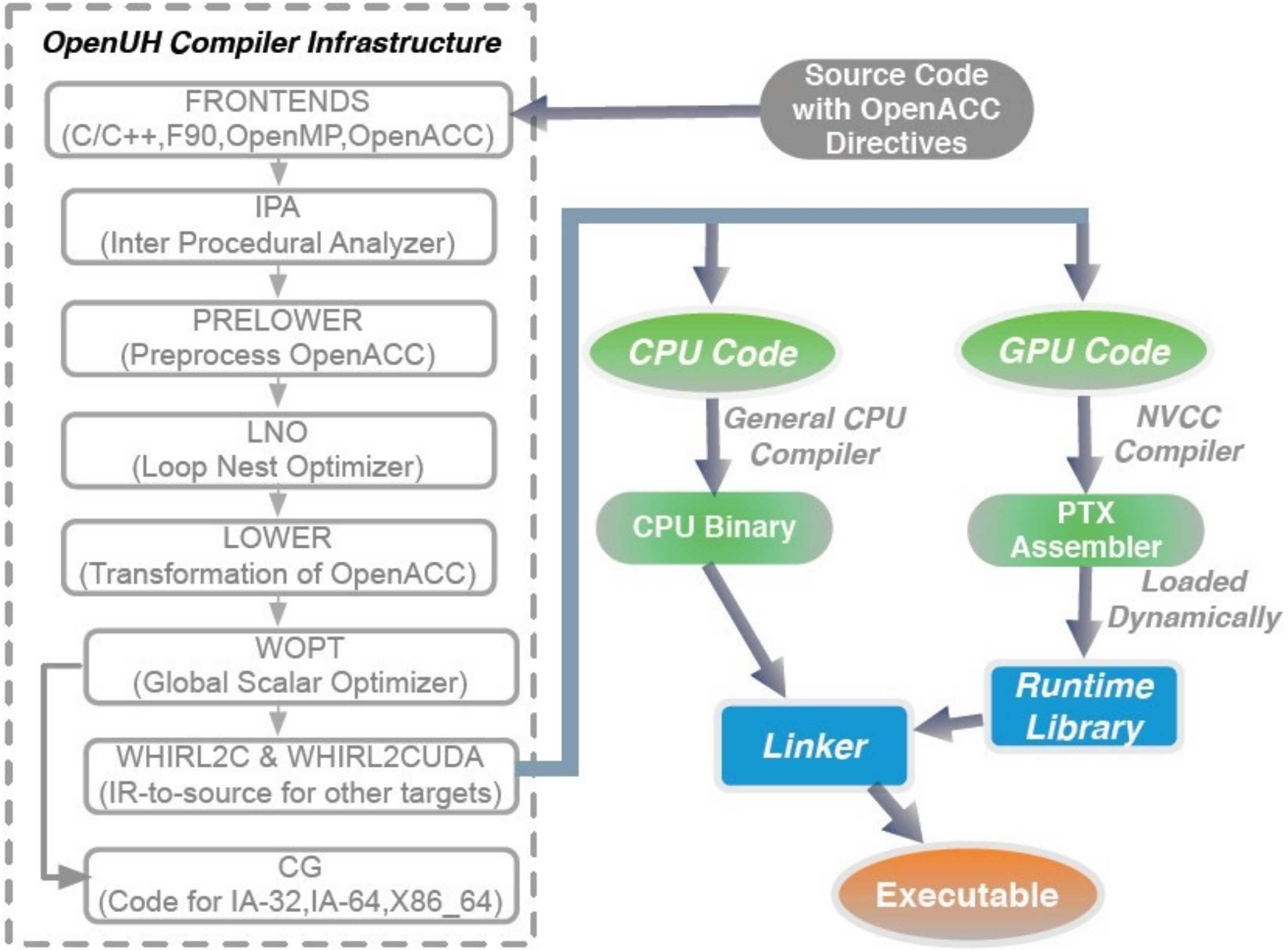


- Announced Supercomputing 2011
  - Initial work by NVIDIA, Cray, PGI, CAPS
- Directive-based programming for accelerators
  - For Fortran, C, C++
  - Loop-based computations
- Compilers: PGI, Cray, CAPS, OpenARC, OpenUH, GCC (4.9)



- Our largest HPC system have $O(10^6)$ cores
- Many of the high end system use hybrid architectures
- Energy usage is a major concern
- Programming has become more of a critical issue
- For many applications performance is measured in a few per cent of peak.

OPENUH – AN OPEN SOURCE OPENACC COMPILER



PERFORMANCE PORTABILITY ACROSS COMPILERS?

- Same OpenACC thread setting does not guarantee best performance for both OpenUH and PGI compilers (PGI 15.7)

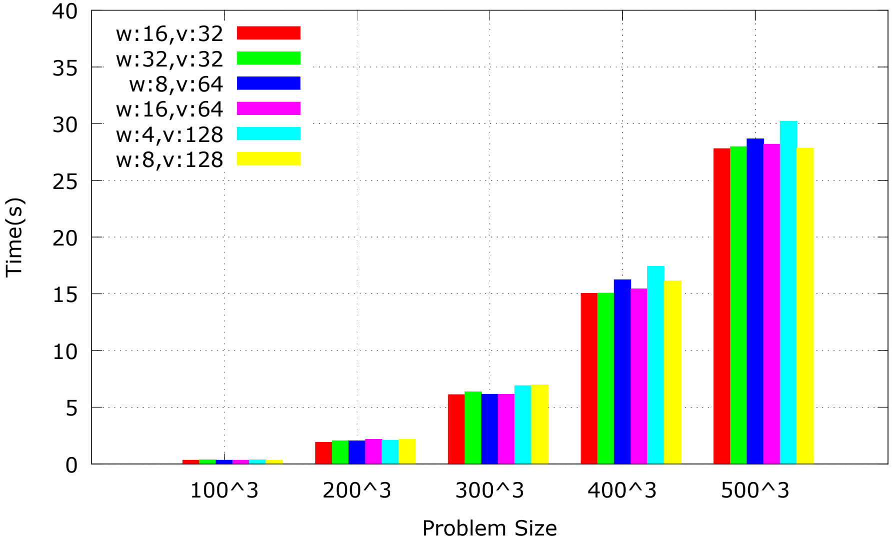


Fig: OpenUH performance for micro-RTM

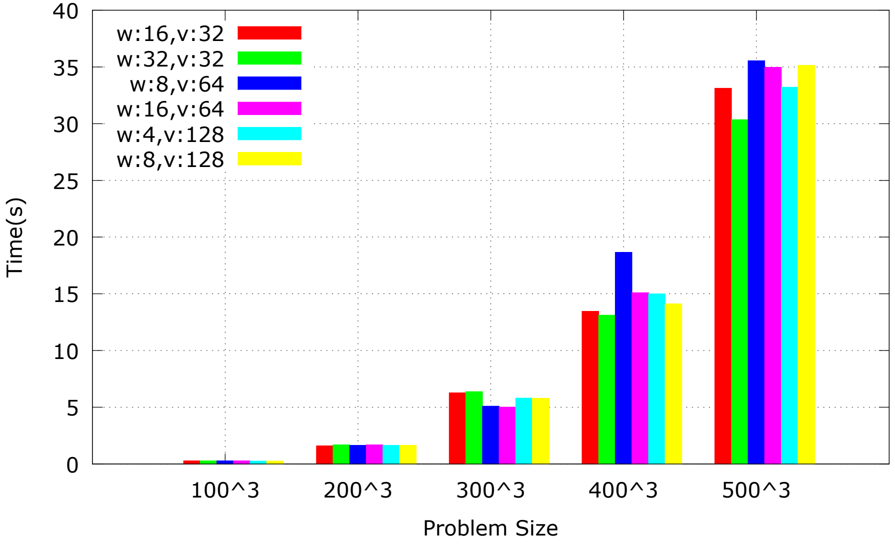
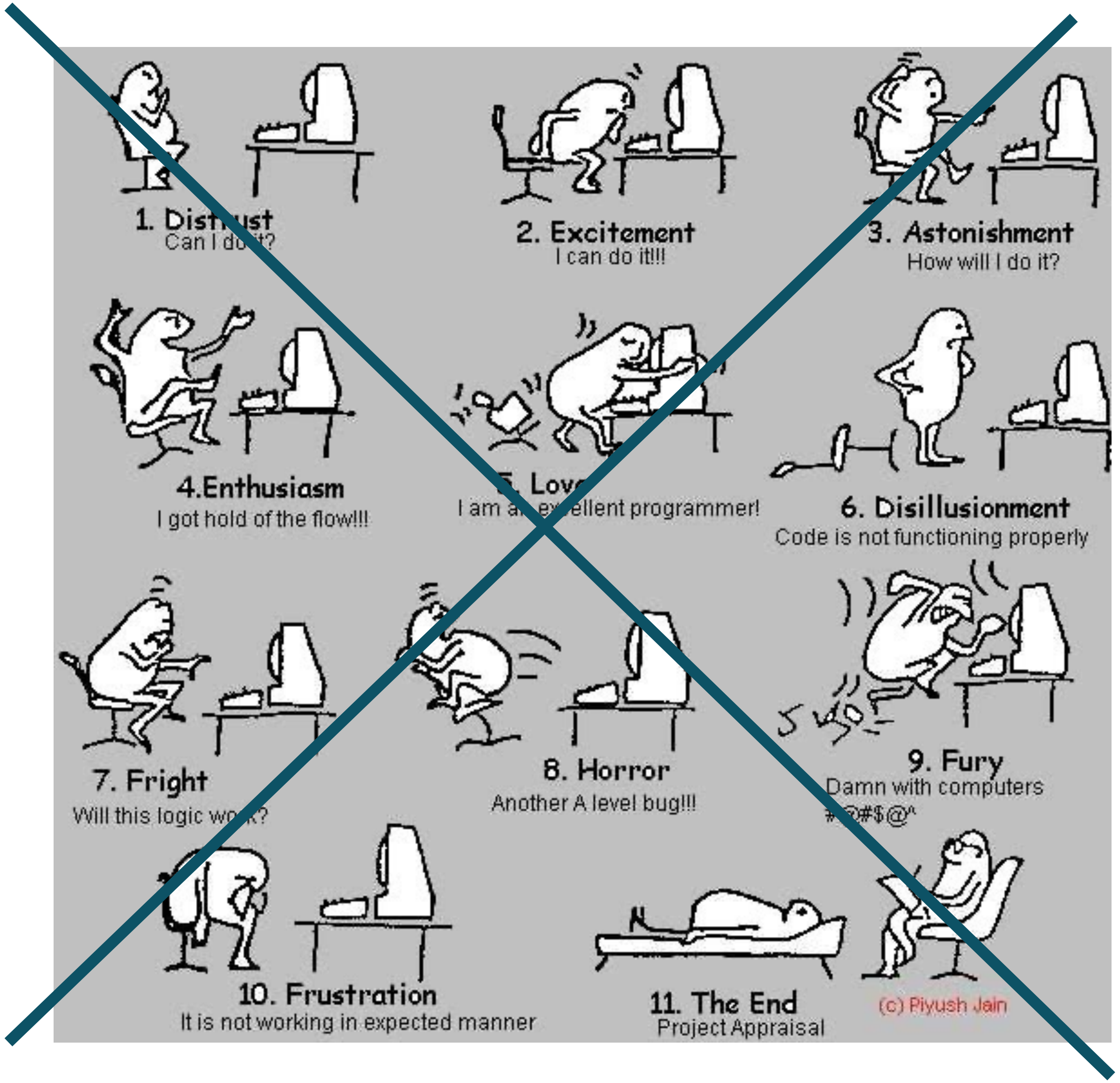


Fig: PGI performance for micro-RTM

THE ROAD AHEAD

- Applications are long-lived
- Multiple GPUs on node
- High rate of innovation in hardware
  - Including accelerators
- What new accelerators will be configured on HPC platforms?
- New classes of users
- Still need scalable performance and productive programming models

- We need good compilers and mature programming ecosystems

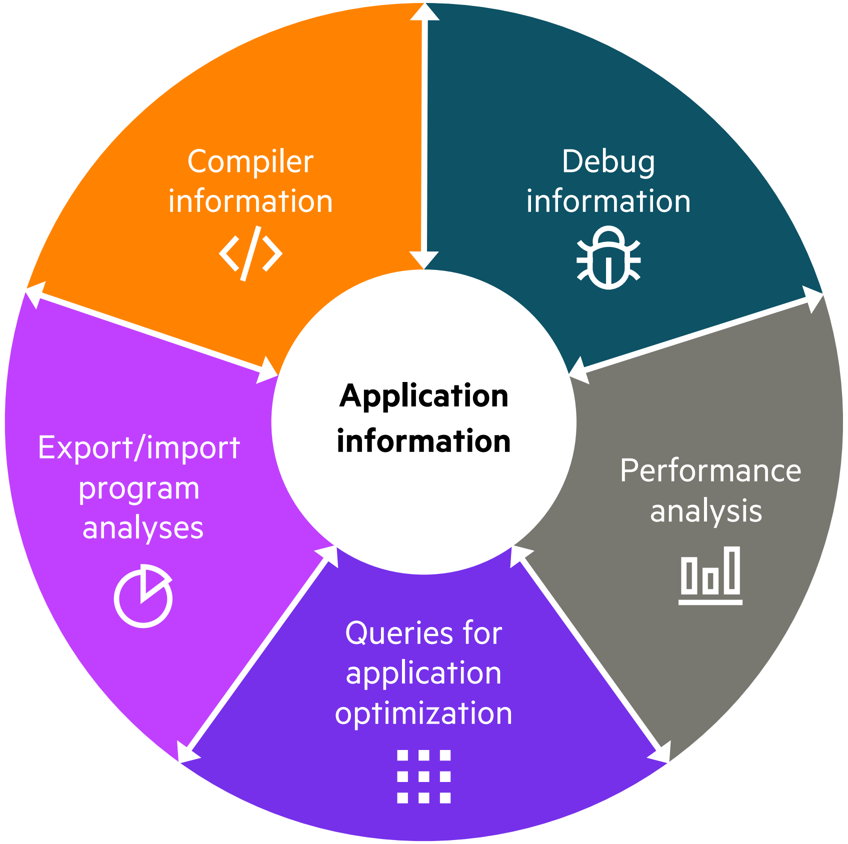


HPE CRAY PROGRAMMING ENVIRONMENT

Essential toolset for HPC organizations developing HPC code in-house.

Fully integrated software suite with compilers, tools, and libraries designed to increase programmer productivity, application scalability, and performance.

Complete toolchain For the whole application development process.	**Holistic solution** Unlike processor-specific tools, the suite enables software development for the full system (including CPUs, GPUs and interconnect) for the best performance.	**Programmability** Offering users intuitive behaviour, automation of tasks and best performance for their applications with little effort.
Scalability Improving performance of applications on systems of any size—up to Exascale deployments.	**Complete Support** HPE Pointnext Services support the whole suite, not just the tools we developed.	**From HPC experts for HPC experts** Developed for over 30 years in close interaction and contributions from our users.



HPE CRAY PROGRAMMING ENVIRONMENT

Comprehensive set of tools for developing, porting, debugging, and tuning of HPC applications on HPE & HPE Cray systems

Development

Programming Models

- HPE Cray MPI
- SHMEM
- OpenMP | OpenACC
- AMD ROCm HIP | NVIDIA CUDA
- UPC | Fortran co-arrays
- Global Arrays

Programming Environments

- Compiling Environment
- GNU
- Intel Programming Environment
- AMD Programming Environment
- NVIDIA HPC SDK

Programming Languages

- C
- C++
- Fortran
- Python
- R

Optimized Libraries

- LibSci (BLAS)
- LAPACK & ScaLAPACK
- LibSci_ACC
- IRT
- FFTW

I/O Libraries

- NetCDF
- HDF5

DL / AI Tools

- Deep Learning Plug-in

Debugging

Comparative Debugger
Compare two versions of an application

GDB for HPC
Parallelized gdb for HPC

Valgrind for HPC
Memory debugging at scale

Stack Trace Analysis Tool
Stack tracing at scale

Tool for Abnormal Termination Processing
Manage core files at scale

TotalView

DDT

Performance Analysis & Optimization

Performance Analysis Tool (PAT)
Whole program performance analysis, exposing wide set of indicators, identifying bottlenecks and automatically generating suggestions to improve performance.

Visualization Tool
Complements text reports with summary of performance data in graphs and charts, allowing users to drill down and resolve issues

Code Parallelization Assistant
Reveal hidden potential of an application via code restructuring

Setup & Runtime

Environment Setup

Tool Enablement
(for Spack, CMake, EasyBuild, etc)

Modules / Lmod

Supported systems:

- HPE Cray supercomputers
- HPE Apollo 2000
- HPE ProLiant DL systems
- Legacy Cray systems

HPE –authored | HPE Added-value to 3rd party | 3rd party

HPE CRAY PROGRAMMING ENVIRONMENT

Comprehensive set of tools for developing, porting, debugging, and tuning of HPC applications on HPE & HPE Cray systems

Development

Programming Models

- HPE Cray MPI
- SHMEM
- OpenMP | OpenACC
- AMD ROCm HIP | NVIDIA CUDA
- UPC | Fortran co-arrays
- Global Arrays

Programming Environments

- Compiling Environment
- GNU
- Intel Programming Environment
- AMD Programming Environment
- NVIDIA HPC SDK

Programming Languages

- C
- C++
- Fortran
- Python
- R

Optimized Libraries

- LibSci (BLAS)
- LAPACK & ScaLAPACK
- LibSci_ACC
- IRT
- FFTW

I/O Libraries

- NetCDF
- HDF5

DL / AI Tools

- Deep Learning Plug-in

Debugging

- **Comparative Debugger** – Compare two versions of an application
- **GDB for HPC** – Parallelized gdb for HPC
- **Valgrind for HPC** – Memory debugging at scale
- **Stack Trace Analysis Tool** – Stack tracing at scale
- **Tool for Abnormal Termination Processing** – Manage core files at scale
- TotalView
- DDT

Performance Analysis & Optimization

- **Performance Analysis Tool (PAT)** – Whole program performance analysis, exposing wide set of indicators, identifying bottlenecks and automatically generating suggestions to improve performance.
- **Visualization Tool** – Complements text reports with summary of performance data in graphs and charts, allowing users to drill down and resolve issues
- **Code Parallelization Assistant** – Reveal hidden potential of an application via code restructuring

Setup & Runtime

Environment Setup

- Tool Enablement (for Spack, CMake, EasyBuild, etc)
- Modules / Lmod

Supported systems:

- HPE Cray supercomputers
- HPE Apollo 2000
- HPE ProLiant DL systems
- Legacy Cray systems

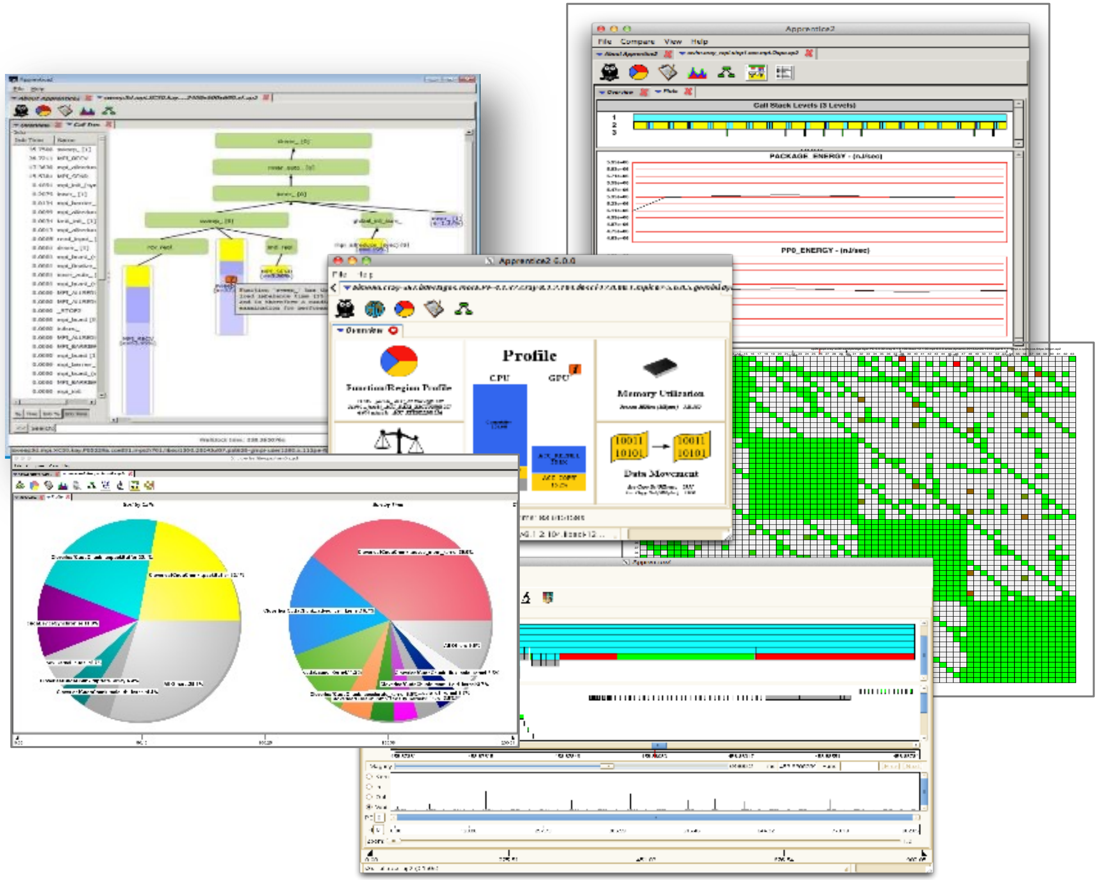
HPE –authored | HPE Added-value to 3rd party | 3rd party

HPE PERFORMANCE AND OPTIMIZATION TOOLS

Reduce time and effort associated with porting and tuning of applications on Cray/HPE systems

Highlights:

- Different tools to fit different developer needs—from quick visual analysis to variety of different experiments, integration with compilers and more…
- Target **scalability** issues in all areas of tool development—designed to improve performance on the largest of systems
- Provide **whole program performance analysis** across many nodes to identify critical performance bottlenecks within a program
- Help to uncover issues but also **suggestions to improve performance**
- Unique and valuable **load imbalance analysis**
- Target **ease of use** with simple and advanced user interfaces



Our performance tools profiled production applications with over 256,000 ranks.

PERFORMANCE ANALYSIS TOOL (PAT)

Gain valuable insight into performance of your application

Main Features:

- Collects and present computation, communication, I/O and memory statistics, including automatic:
  - Identification & display of program's top time-consumers for future analysis
  - Identification of and bottlenecks, for example, load imbalance = "Where are the slow paths in the code?"
- Automatically generates observations and suggestions based on analysis of collected data.
- Enable developers to perform sampling, profile, and trace experiments on single- or multi-processor executables., including API for fine-grained instrumentation
- Detects communication grids and presents rank re-ordering analysis and suggestions
  - Improve application performance by maximizing on-node communication.
- Supports programs written in Fortran, C or C++ with MPI, SHMEM, UPC, OpenMP or OpenACC, CUDA or HIP, and their combinations.

Lightweight version:

- Provides performance analysis information automatically, with a minimum of user interaction.
- Starting point for users who wish to explore a program's behavior further using the full toolset.

PROFILE INFORMATION

First column is percent of inclusive time
Second column is inclusive time
Third column is number of times executed
Fourth column is average loop iteration count
Last column is name of program unit, LOOP or Routine, line number in source

Let's start by putting triple nested loop on the GPU

```
Table 1:  Calltree with Loop Inclusive Time
   Incl |  Incl | Loop Exec |    Loop | Calltree
  Time% |  Time |           |   Trips |
        |       |           |     Avg |
 100.0% | 54.61 |        -- |      -- | Total
|-----------------------------------------------------------------
| 100.0% | 54.61 |        -- |      -- | hackakernel_
||----------------------------------------------------------------
||  95.1% | 51.92 |         1 |    96.0 | hackakernel_.LOOP.04.li.121
|||---------------------------------------------------------------
3||  92.2% | 50.34 |        96 |   240.0 | hackakernel_.LOOP.08.li.166
||||--------------------------------------------------------------
4|||  92.2% | 50.34 |    23,040 |   100.0 | hackakernel_.LOOP.09.li.170
|||||-------------------------------------------------------------
5||||  92.1% | 50.27 | 2,304,000 | 8,000.0 | hackakernel_.LOOP.10.li.177
||||==============================================================
3||   2.8% |  1.50 |        96 |   240.0 | hackakernel_.LOOP.11.li.192
||||--------------------------------------------------------------
4|||   1.6% |  0.88 |    22,944 | 6,000.0 | hackakernel_.LOOP.12.li.215
|=================================================================
```

GET ANNOTATED LISTING BY USING –H LIST=A

```
    + 1 G--------< !$ACC parallel
      1 G          !$ACC loop private(ifreq)
      1 G g------<         do ifreq=1,nFreq
      1 G g
      1 G g                    schDt = (0D0,0D0)
      1 G g
      1 G g        !$ACC loop private(my_igp,igmax,schDtt)
      1 G g g----<             do my_igp = 1, ngpown
      1 G g g
      1 G g g                    if (my_igp .gt. ncouls .or. my_igp .le. 0) cycle
      1 G g g
      1 G g g                    igmax=ncouls
      1 G g g
      1 G g g                    schDtt = (0D0,0D0)
      1 G g g      !$ACC loop vector private(ig,I_epsRggp_int,I_epsAggp_int,schD)reduction(+:schDtt)
      1 G g g g--<               do ig = 1, igmax
      1 G g g g                    I_epsRggp_int = I_epsR_array(ig,my_igp,ifreq)
      1 G g g g                    I_epsAggp_int = I_epsA_array(ig,my_igp,ifreq)
      1 G g g g                    schD=I_epsRggp_int-I_epsAggp_int
      1 G g g g                    schDtt = schDtt + matngmatmgpD(ig,my_igp)*schD
      1 G g g g-->               enddo
      1 G g g                    schdt_array(ifreq) = schdt_array(ifreq) + schDtt
      1 G g g---->             enddo
      1 G g
      1 G g------>         enddo
      1 G--------> !$ACC end parallel
```

G – Kernel

g – parallel levels

PAT_REPORT <STATISTIC DIRECTORY>

Table 5: Time and Bytes Transferred for Accelerator Regions

```
  Host |  Host |    Acc |   Acc |  Acc Copy |  Acc Copy | Events | Calltree
 Time% |  Time |  Time% |  Time |        In |       Out |        |  Thread=HIDE
       |       |        |       | (MiBytes) | (MiBytes) |        |

 100.0% | 58.54 | 100.0% | 58.31 |   563,672 |      0.35 |    480 | Total
|-----------------------------------------------------------------------
| 100.0% | 58.54 | 100.0% | 58.31 |   563,672 |      0.35 |    480 | hackakernel_
|        |       |        |       |           |           |        |  hackakernel_.ACC_REGION@li.166
|||---------------------------------------------------------------------
3||  97.9% | 57.33 |  97.9% | 57.10 |   563,672 |        -- |     96 | hackakernel_.ACC_COPY@li.166
3||   2.1% |  1.20 |     -- |    -- |        -- |        -- |     96 | hackakernel_.ACC_SYNC_WAIT@li.191
3||   0.0% |  0.00 |   2.1% |  1.21 |        -- |        -- |     96 | hackakernel_.ACC_ASYNC_KERNEL@li.166
|=======================================================================
```

Bottom line, the kernel runs 45 times faster than one core on the host; however, the data movement takes all the time – need to move data outside outer loop.

Only show part of profile

GET MORE KERNEL AND DATA TRANSFER DATA

```
ACC:
ACC: End transfer (to acc 6156803840 bytes, to host 0 bytes, time 604039 usec)
ACC:
ACC: Start kernel hackakernel_$ck_L166_1 async(auto) from bgw.f90:166
ACC:       flags: CACHE_MOD CACHE_FUNC AUTO_ASYNC FLEX_BLOCKS
ACC:    mod cache:  0x4c4880
ACC: kernel cache:  0x4c4840
ACC:   async info:  0x15554939c4e0
ACC:    arguments: GPU argument info
ACC:          param size:  168
ACC:       param pointer:  0x7fffffff5120
ACC:      blocks:  240
ACC:      threads:  128
ACC:     event id:  3
ACC:  Start tracking event 3 index 0 (total 1) stream 0x1736764d0
ACC:    loading module data
ACC:    getting function
ACC:      stats threads=640 threadblocks per sm=5 shared=2048 total shared=10240
ACC:      prefer L1 cache
ACC:     kernel information
ACC:             num registers :       88
ACC:       max theads per block :      640
ACC:               shared size :     2048 bytes
ACC:                const size :       0 bytes
ACC:                local size :       0 bytes
ACC:
ACC:     launching kernel new
ACC:     caching function
ACC:     caching module
ACC:  End tracking event 3 index 0 (total 1)
ACC: End kernel
```

Setenv CRAY_ACC_DEBUG=3

HPE CRAY PROGRAMMING ENVIRONMENT

Features At-a-Glance

Compiling Environment, Programming Models, and Languages

- **Our Fortran, C and C++ compilers** are designed to extract maximum performance from the systems regardless of the underlying architecture, including :
  - Compiler optimization feedback for app tuning
  - Integration w/performance tools to optimize performance
  - Support for standard programming languages and programs and focus on compliance for investment protection
  - Integration with 3rd party programming environments for more convenience
- **Scalable communication libraries**
  - **HPE Cray MPI**: supports extreme scaling for job startup, memory footprint and collective algorithms including use of HW collectives
  - Performance-optimized **SHMEM**

Scientific, Math & I/O Libraries

Comprehensive collection of **highly tuned linear algebra subroutines** designed to extract maximum possible performance with minimum effort.

- Customized **LibSci** (including BLAS, LAPACK, and ScaLACK), our iterative refinement toolkit, and **LibSci_ACC** (accelerated BLAS, and LAPACK) are designed to take full advantage of the underlying hardware and interconnect.

Debuggers

Traditional debuggers combined with new innovative techniques allowing users to address debugging problems at a broader range and scale:

- **Comparative debugger—**this market-unique tool helps programmers uncover issues easier by running two versions of an application side by side.
- **Valgrind for HPC—**parallel memory analysis tool
- **GDB for HPC—**provides a gdb debugging experience for applications that run at scale across many nodes.
- **Stack Trace Analysis Tool (STAT):** Helps developers identify if an application is hung or still making progress when running.
- **Abnormal Termination Processing (ATP) tool**: When an application crashes, the tool detects a signal and generates a merged-backtrace resulting in a minimal core file set
- Support for 3rd party debuggers (Arm Forge & TotalView)

Deep Learning Plug-in

Helps to optimize scaling and performance across multiple machine learning and deep learning frameworks = streamlining the deep learning training on the HPE HPC systems.

Performance Profiling

Comprehensive collection of tools designed to reduce the time and effort associated with porting and tuning of applications:

- **Performance analysis tool (PAT)** brings valuable insight when analyzing bottlenecks to improve performance of applications that run across the whole system
- A **visualization tool** helps to assess the type and severity of performance issues quickly. Drill down to get to the bottom of issues.
- **Code parallelization assistant** helps developers reveal targets for parallelism and assists with adding OpenMP to an application.

Hewlett Packard Enterprise

THANK YOU

Barbara Chapman, barbara.chapman@hpe.com