Zero to GPU Hero with OpenACC

Aaron Jarmusch
PhD Student
University of Delaware
jarmusch@udel.edu

OpenACC
More Science, Less Programming

OPEN HACKATHONS

UNIVERSITY OF DELAWARE

NVIDIA

Who Am I?

- Aaron Jarmusch, University of Delaware
- PhD Student @ UD
- crpl.cis.udel.edu/

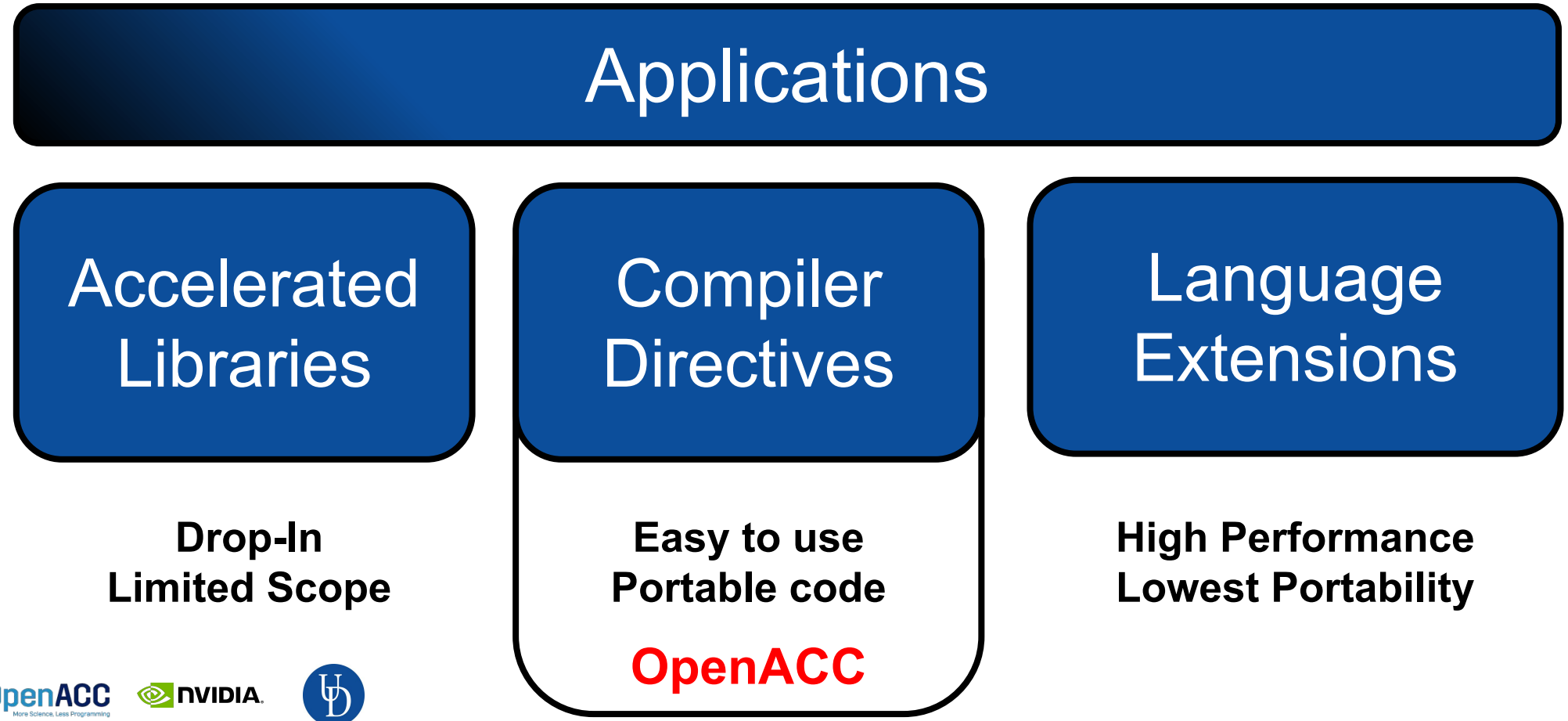
- OpenACC Validation & Verification Testsuite since 2020
  https://crpl.cis.udel.edu/oaccvv/
- LLM4VV - LLMJ

OpenACC More Science, Less Programming NVIDIA

Tutorial Outline

- What is OpenACC and Who uses it?
- First Steps with OpenACC
- Example
- Profile-Driven Development
- Build, Run, Optimize
- Where can I learn more?

Why Use OpenACC?



What is OpenACC?

OpenACC is a directive-based parallel programming model designed for productivity, performance, and portability

APPLICATIONS	PLATFORMS SUPPORTED	COMMUNITY
250+ 3 out of Top 5	NVIDIA GPU X86 CPU POWER CPU Sunway ARM CPU AMD GPU FPGA	**~3000** Slack Members

OpenACC More Science, Less Programming NVIDIA

OpenACC Directives

Manage Data Movement

Initiate Parallel Execution

Optimize Loop Mappings

```
#pragma acc data copyin(a,b) copyout(c)
{
  ...
  #pragma acc parallel
  {
  #pragma acc loop gang vector
      for (i = 0; i < n; ++i) {
          c[i] = a[i] + b[i];
          ...
      }
  }
  ...
}
```

OpenACC
Directives for Accelerators

- Incremental
- Single source
- Interoperable
- Performance portable
- CPU, GPU, & More

OpenACC More Science, Less Programming | NVIDIA | UD

OPENACC

Incremental

- Maintain existing sequential code
- Add annotations to expose parallelism
- After verifying correctness, annotate more of the code

Single Source

- Rebuild the same code on multiple architectures
- Compiler determines how to parallelize for the desired machine
- Sequential code is maintained

Low Learning Curve

- OpenACC is meant to be easy to use, and easy to learn
- Programmer remains in familiar C, C++, or Fortran
- No reason to learn low-level details of the hardware.

OpenACC More Science, Less Programming NVIDIA

OPENACC

Incremental

- Maintain existing sequential code
- Add annotations to expose parallelism
- After verifying correctness, annotate more of the code

Enhance Sequential Code

```
#pragma acc parallel loop
for( i = 0; i < N; i++ )
{
   < loop code >
}

#pragma acc parallel loop
for( i = 0; i < N; i++ )
{
   < loop code >
}
```

Begin with a working sequential code.

Parallelize it with OpenACC.

Rerun the code to verify correctness and performance

OpenACC More Science, Less Programming

NVIDIA

OPENACC

Supported Platforms

POWER

Sunway

x86 CPU

ARM CPU

AMD GPU

NVIDIA GPU

PEZY-SC

Single Source

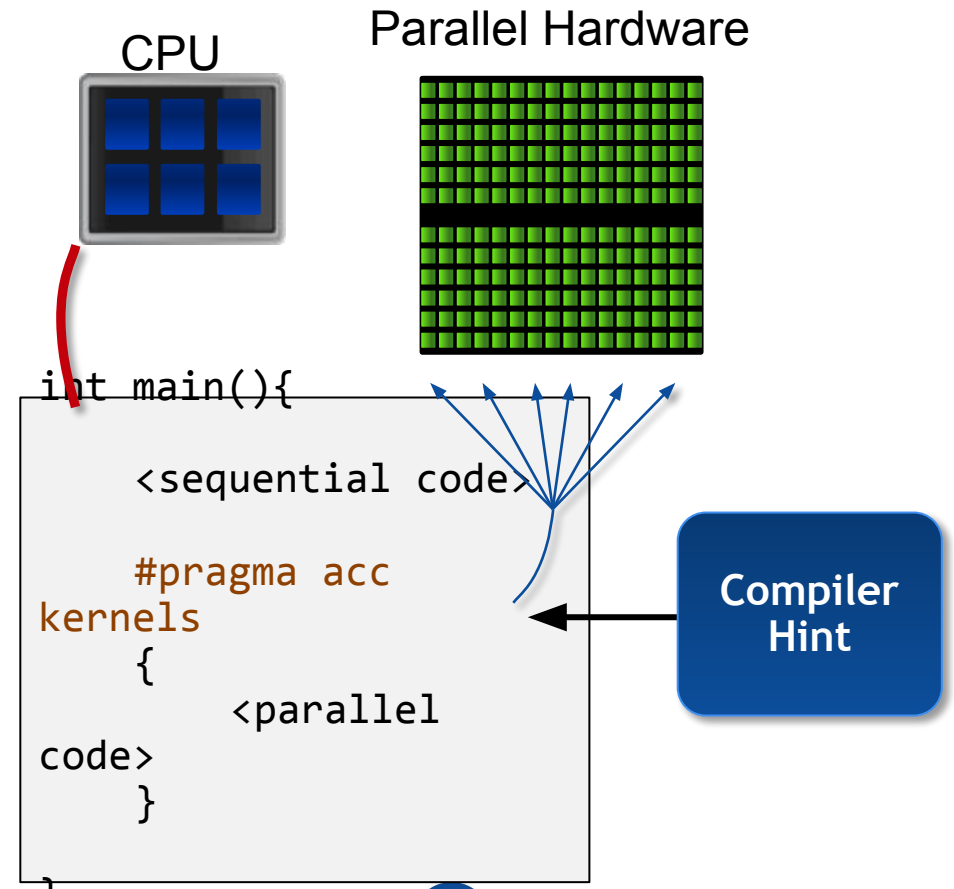
- Rebuild the same code on multiple architectures
- Compiler determines how to parallelize for the desired machine
- Sequential code is maintained

The compiler can **ignore** your OpenACC code additions, so the same code can be used for **parallel** or **sequential** execution.

```
int main(){

...


     for(int i = 0; i < N;
i++)
          < loop code >

}
```

OpenACC More Science, Less Programming NVIDIA

OPENACC



The programmer will give hints to the compiler.

The compiler parallelizes the code.

Low Learning Curve

- OpenACC is meant to be easy to use, and easy to learn
- Programmer remains in familiar C, C++, or Fortran
- No reason to learn low-level details of the hardware.

OpenACC More Science, Less Programming NVIDIA

OPENACC

Incremental

- Maintain existing sequential code
- Add annotations to expose parallelism
- After verifying correctness, annotate more of the code

Single Source

- Rebuild the same code on multiple architectures
- Compiler determines how to parallelize for the desired machine
- Sequential code is maintained

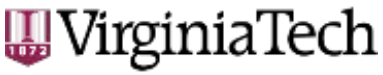
Low Learning Curve

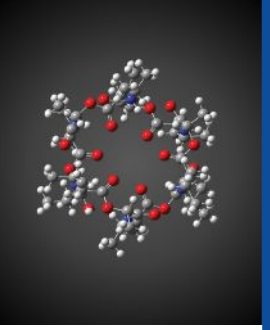
- OpenACC is meant to be easy to use, and easy to learn
- Programmer remains in familiar C, C++, or Fortran
- No reason to learn low-level details of the hardware.

OpenACC More Science, Less Programming NVIDIA

Who's Using OpenACC

OpenACC Members

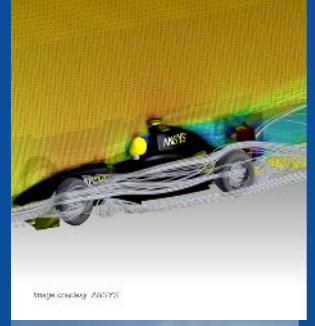




GAUSSIAN 16

Mike Frisch, Ph.D.
President and
CEO
Gaussian, Inc.

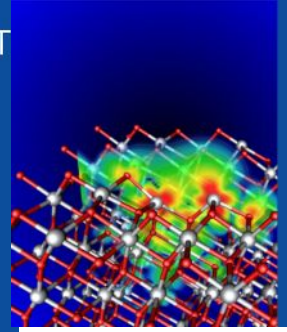
Using OpenACC allowed us to continue development of our fundamental algorithms and software capabilities simultaneously with the GPU-related work. In the end, we could use the same code base for SMP, cluster/network and GPU parallelism. PGI's compilers were essential to the success of our efforts.



ANSYS FLUENT

Sunil Sathe
Lead Software Developer
ANSYS Fluent

We've effectively used OpenACC for heterogeneous computing in ANSYS Fluent with impressive performance. We're now applying this work to more of our models and new platforms.



VASP

Prof. Georg Kresse
Computational Materials Physics
University of Vienna

For VASP, OpenACC is *the* way forward for GPU acceleration. Performance is similar and in some cases better than CUDA C, and OpenACC dramatically decreases GPU development and maintenance efforts. We're excited to collaborate with NVIDIA and PGI as an early adopter of CUDA Unified Memory.

COSMO

Dr. Oliver Fuhrer
Senior Scientist
Meteoswiss

OpenACC made it practical to develop for GPU-based hardware while retaining a single source for almost all the COSMO physics code.

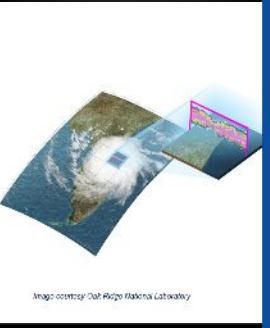


Image courtesy Oak Ridge National Laboratory

E3SM

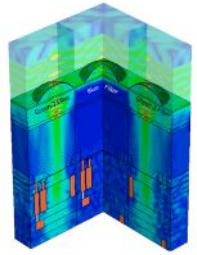
Mark A. Taylor
Multiphysics Applications
Sandia

The CAAR project provided us with early access to Summit hardware and access to PGI compiler experts. Both of these were critical to our success. PGI's OpenACC support remains the best available and is competitive with much more intrusive programming model approaches.

NUMECA FINE/Open

Davic Gutzwiller
Lead Software Developer
NUMECA

Porting our unstructured C++ CFD solver FINE/Open to GPUs using OpenACC would have been impossible two or three years ago, but OpenACC has developed enough that we're now getting some really good results.



SYNOPSYS

Dr. Lutz Schneider
Senior R&D Engineer
Synopsys Inc.

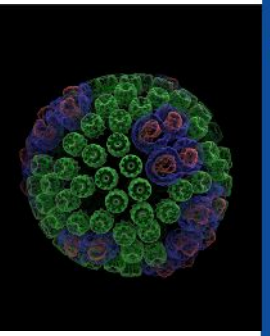
Using OpenACC, we've GPU-accelerated the Synopsys TCAD Sentaurus Device EMW simulator to speed up optical simulations of image sensors. GPUs are key to improving simulation throughput in the design of advanced image sensors.

Image courtesy: NCAR

MPAS-A

Richard Loft
Director, Technology Development
NCAR

Our team has been evaluating OpenACC as a pathway to performance portability for the Model for Prediction (MPAS) atmospheric model. Using this approach on the MPAS dynamical core, we have achieved performance on a single P100 GPU equivalent to 2.7 dual socketed Intel Xeon nodes on our new Cheyenne supercomputer.



VMD

John Stone
Senior Research Programmer
Beckham Institute
University of Illinois

Due to Amdahl's law, we need to port more parts of our code to the GPU if we're going to speed it up. But the sheer number of routines poses a challenge. OpenACC directives give us a low-cost approach to getting at least some speed-up out of these second-tier routines. In many cases it's completely sufficient because with the current algorithms, GPU performance is bandwidth-bound.

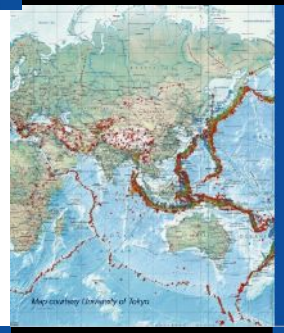
GTC

Zhihong Lin
Professor and Principal Investigator
UC Irvine

Using OpenACC our scientists were able to achieve the acceleration needed for integrated fusion simulation with a minimum investment of time and effort in learning to program GPUs.

OpenACC

More Science, Less Programming



Map courtesy University of Tokyo

GAMERA

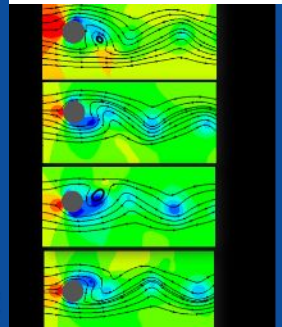
Takuma Yamaguchi, Kohei Fujita, Tsuyoshi Ichimura, Muneo Hori, Lalith Wijerathne
The University of Tokyo

With OpenACC and a compute node based on NVIDIA's Tesla P100 GPU, we achieved more than a 14X speed up over a K Computer node running our earthquake disaster simulation code

SANJEEVINI

Abhilash Jayaraj
Project Scientist
Indian Institute of Technology
New Delhi

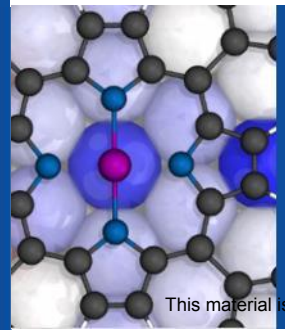
In an academic environment maintenance and speedup of existing codes is a tedious task. OpenACC provides a great platform for computational scientists to accomplish both tasks without involving a lot of efforts or manpower in speeding up the entire computational task.



IBM-CFD

Somnath Roy
Assistant Professor
Mechanical Engineering Department
Indian Institute of Technology Kharagpur

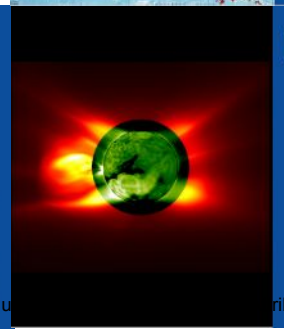
OpenACC can prove to be a handy tool for computational engineers and researchers to obtain fast solution of non-linear dynamics problem. In immersed boundary incompressible CFD, we have obtained order of magnitude reduction in computing time by porting several components of our legacy codes to GPU. Especially the routines involving search algorithm and matrix solvers have been well-accelerated to improve the overall scalability of the code.



PWscf (Quantum ESPRESSO)

Filippo Spiga
Senior Contributor
Quantum ESPRESSO group

CUDA Fortran gives us the full performance potential of the CUDA programming model and NVIDIA GPUs. While leveraging the potential of explicit data movement, !$CUF KERNELS directives give us productivity and source code maintainability. It's the best of both worlds.



MAS

Ronald M. Caplan
Computational Scientist
Predictive Science Inc.

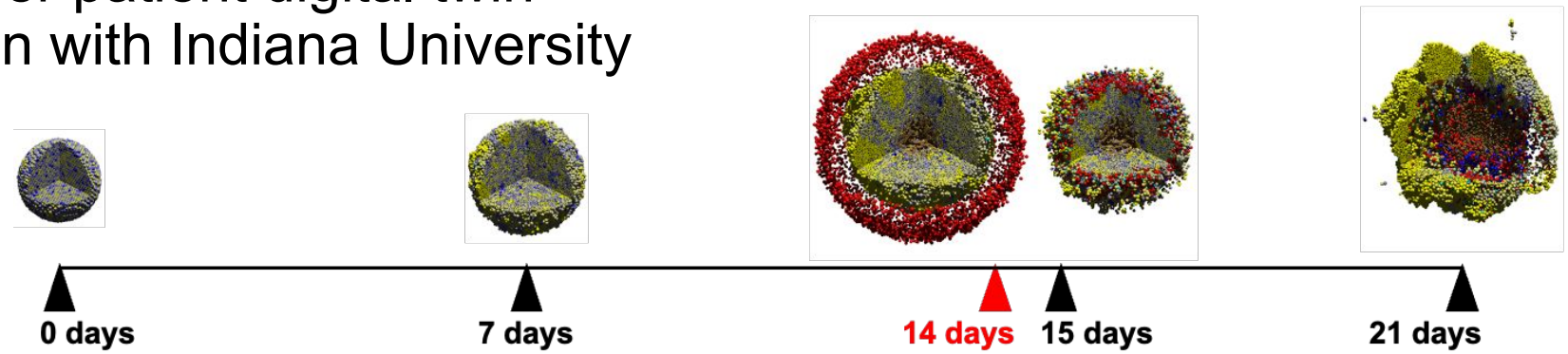
Adding OpenACC into MAS has given us the ability to migrate medium-sized simulations from a multi node CPU cluster to a single multi-GPU server. The implementation yielded a portable single-source code for both CPU and GPU runs. Future work will add OpenACC to the remaining model features, enabling GPU-accelerated realistic solar storm modeling.

PhysiCell: OpenACC Acceleration of an Agent-Based Biological Simulation Framework

- Accelerating an agent-based biological simulation framework using OpenACC
- Reduce the total wall time
- Explore new biological dynamics
- Maintain a single source code base across CPUs & GPUs
- Allow 3D multiscale simulations of cancer and diseases
- Advance cancer patient digital twin
- In collaboration with Indiana University

CPU-only: 9 hours 30 min

GPU: 1 hour 42 min

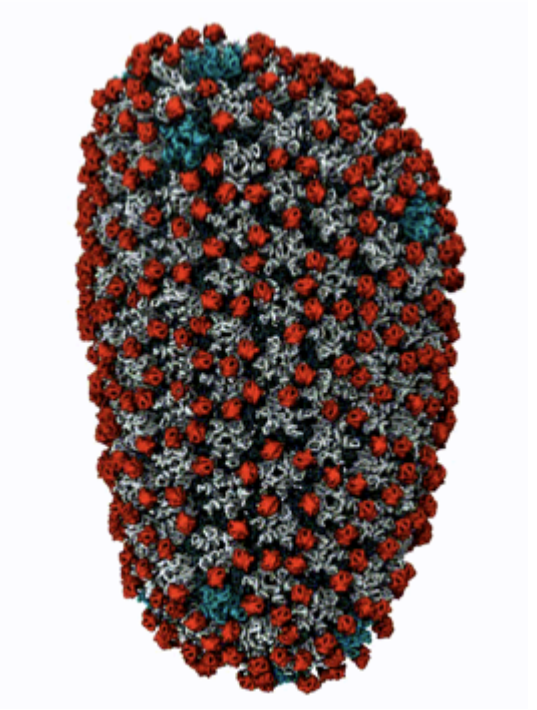


Matt Stack, Paul Macklin, Robert Searles, Sunita Chandrasekaran. ACM Computing in Science & Engineering, 24(5), 53-63
https://dl.acm.org/doi/abs/10.1109/MCSE.2022.3226602

OpenACC More Science, Less Programming NVIDIA

A Biophysics case study: Accelerating Chemical Shift Prediction for Large-scale Biomolecular Modeling

- Chemical shift gives insight into the physical structure of the protein
- Predicting chemical shift has important uses in scientific areas such as drug discovery
- Accelerated the code using OpenACC
- Maintained a single source code between CPUs and GPUs
- 37.6x speedup using a V100 GPU compared to the serial unoptimized version
- In collaboration with UDelaware Chemistry & BioChemistry

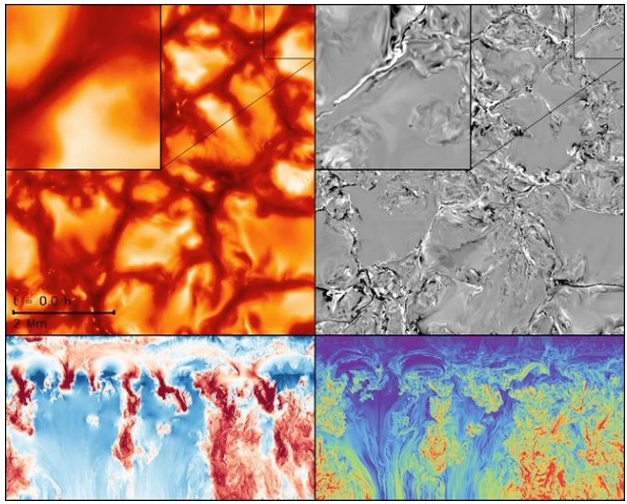


HIV capsid

Eric Wright, Mauricio Ferrato, Alexander J. Bryer, Robert Searles, Juan R. Perilla and Sunita Chandrasekaran. Accelerating prediction of chemical shift of protein structures on GPUs: Using OpenACC. PLOS Computational Biology. May 2020 DOI: https://doi.org/10.1371/journal.pcbi.1007877

MURaM (Max Planck University of Chicago Radiative MHD) - A solar physics project

- Primary solar model for simulations of the different layers of the sun - upper convection zone, photosphere and corona
- Port MURaM to GPUs using the OpenACC programming model
- Create a fast code to be able to create a platform that can enable faster analysis of incoming data from the telescope installed
- Maintained architectural portability between CPUs and GPUs
- Jointly developed by HAO, the Max Planck Institute for Solar System Research (MPS) and the Lockheed Martin Solar and Astrophysics Laboratory (LMSAL)



Wright, E., Przybylski, D., Rempel, M., Miller, C., Suresh, S., Su, S., ... & Chandrasekaran, S. (2021, July). Refactoring the MPS/University of Chicago Radiative MHD (MURaM) model for GPU/CPU performance portability using OpenACC directives. In *Proceedings of the Platform for Advanced Scientific Computing Conference (PASC)* (pp. 1-12).

Wright, Eric, et al. "Analysis of MURaM, a Solar Physics Application, for Scalability, Performance and Portability." Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis. 2023.

OPENACC SYNTAX

OPENACC SYNTAX

Syntax for using OpenACC directives in code

C/C++

```
#pragma acc directive clauses
<code>
```

Fortran

```
!$acc directive clauses
<code>
```

- A ***pragma*** in C/C++ gives instructions to the compiler on how to compile the code. Compilers that do not understand a particular pragma can freely ignore it.
- A ***directive*** in Fortran is a specially formatted comment that likewise instructions the compiler in it compilation of the code and can be freely ignored.
- "***acc"*** informs the compiler that what will come is an OpenACC directive
- ***Directives*** are commands in OpenACC for altering our code.
- ***Clauses*** are specifiers or additions to directives.

OpenACC More Science, Less Programming NVIDIA

EXAMPLE CODE

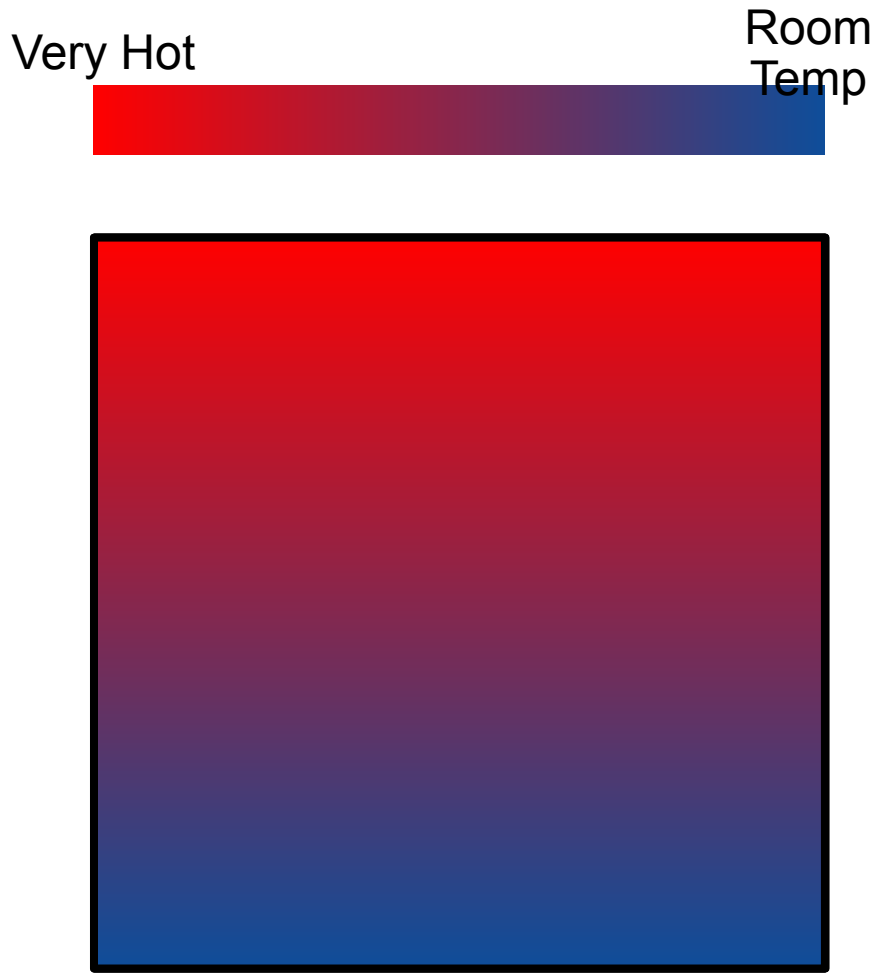
LAPLACE HEAT TRANSFER

Introduction to lab code - visual

We will observe a simple simulation of heat distributing across a metal plate.

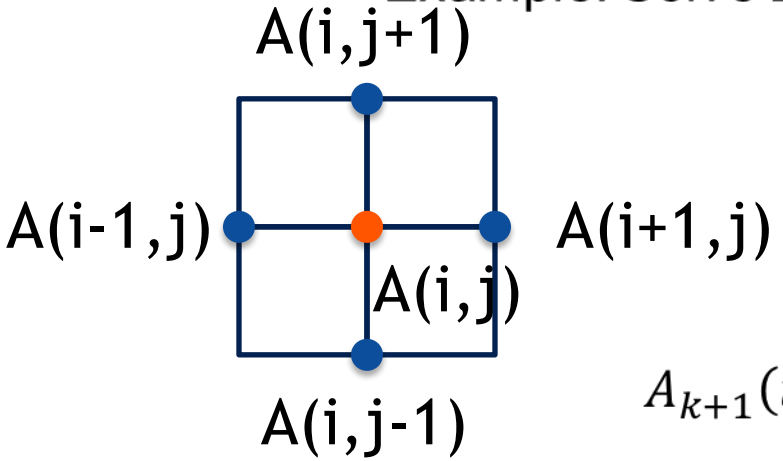
We will apply a consistent heat to the top of the plate.

Then, we will simulate the heat distributing across the plate.



EXAMPLE: JACOBI ITERATION

- Iteratively converges to correct value (e.g. Temperature), by computing new values at each point from the average of neighboring points.
- Common, useful algorithm
- Example: Solve Laplace equation in 2D: $\nabla^2 f(x,y) = 0$



$$A_{k+1}(i,j) = \frac{A_k(i-1,j) + A_k(i+1,j) + A_k(i,j-1) + A_k(i,j+1)}{4}$$

JACOBI ITERATION: C/C++ CODE

```
while ( error > tol && iter < iter_max )
{
  double error = 0.0;
  for (int j = 1; j < n - 1; j++)
  {
    for (int i = 1; i < m - 1; i++)
    {
      Anew[OFFSET(j, i, m)] = 0.25 * \
                (A[OFFSET(j, i + 1, m)] + A[OFFSET(j, i - 1, m)] + \
                 A[OFFSET(j - 1, i, m)] + A[OFFSET(j + 1, i, m)]);


      error = fmax(error, fabs(Anew[OFFSET(j, i, m)] - A[OFFSET(j, i, m)]));
    }
  }

  for (int j = 1; j < n - 1; j++)
  {
    for (int i = 1; i < m - 1; i++)
    {
      A[OFFSET(j, i, m)] = Anew[OFFSET(j, i, m)];
    }
  }

  if (iter % 100 == 0)
    printf("%5d, %0.6f\n", iter, error);

  iter++;
}
```

- Iterate until converged
- Iterate across matrix elements
- Calculate new value from neighbors
- Compute max error for convergence
- Swap input/output arrays

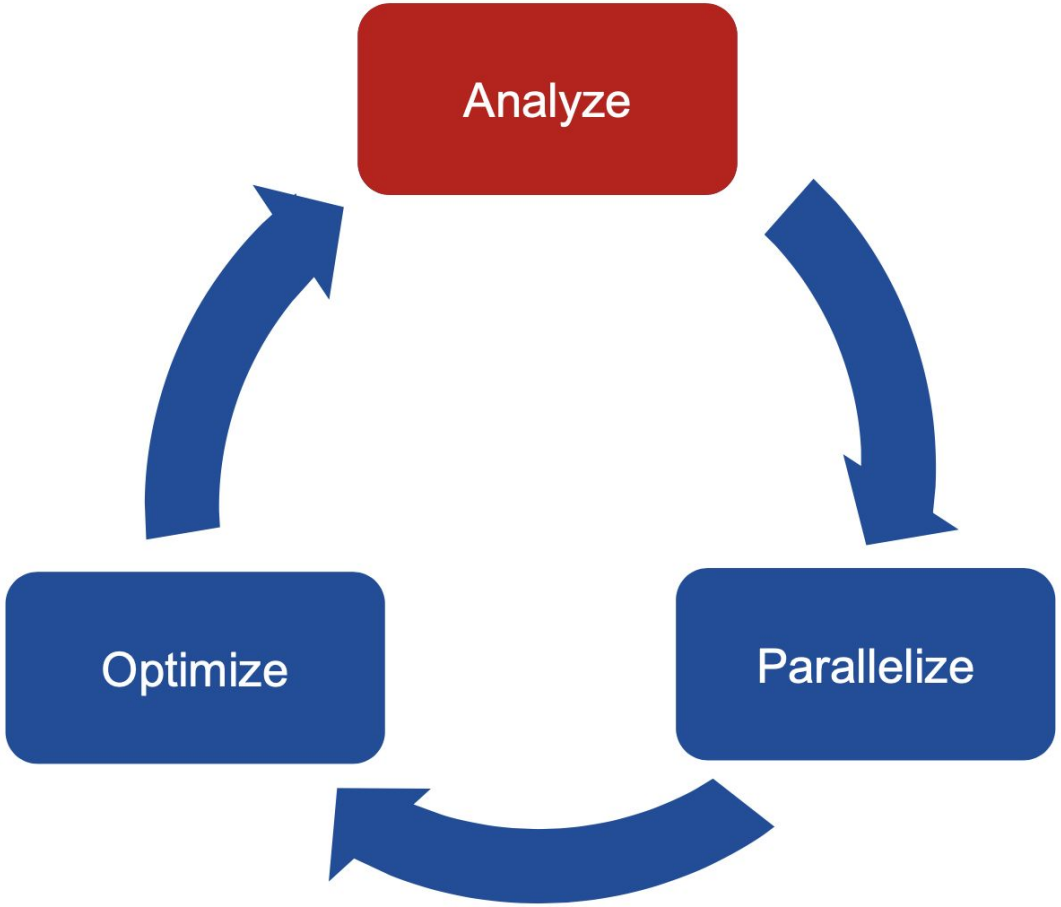
OpenACC More Science, Less Programming

NVIDIA

PROFILE-DRIVEN DEVELOPMENT

GETTING A BASELINE

- **Analyze** your code to determine most likely places needing parallelization or optimization.
- **Parallelize** your code by starting with the most time consuming parts and check for correctness.
- **Optimize** your code to improve observed speed-up from parallelization.



GNU GPROF profiler

- How to use gprof
  - Using the gprof tool is not at all complex
  - Have profiling enabled while compiling the code
  - Execute the program code to produce the profiling data
  - Run the gprof tool on the profiling data file

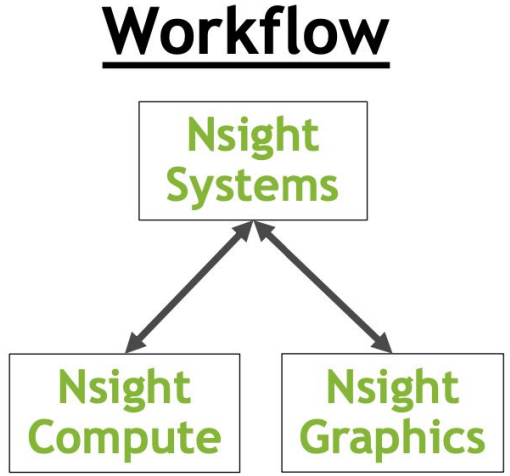
NSIGHT PRODUCT FAMILY

Standalone Performance Tools

Nsight Systems - System-wide application algorithm tuning

Nsight Compute - Debug/optimize specific CUDA kernel

Nsight Graphics - Debug/optimize specific graphics shader



IDE Plugins

Nsight Eclipse Edition/Visual Studio – editor, debugger, some perf analysis

GETTING A BASELINE

Profile Your Code

Profiling your code to obtain detailed information about how does the code run:

- Total runtime
- runtime of routines
- hardware counters

Identify portions that took longer to execute. These are the portions that you will want to parallelize.

to use gprof add -pg to compile the application

LLVM

```
$ clang -Ofast -fopenmp -fno-inline -pg -o jacobi-serial jacobi.c
Jacobi relaxation Calculation: 4096 x 4096 mesh
    0, 0.250000
  100, 0.002397
  200, 0.001204
  300, 0.000804
  400, 0.000603
  500, 0.000483
  600, 0.000403
  700, 0.000345
  800, 0.000302
  900, 0.000269
 total: 25.557923 s
```

GETTING A BASELINE

Profile Your Code

NVIDIA NVHPC

```
$  nvc -O3 jacobi.c -o jacobi_serial.o

Jacobi relaxation Calculation: 4096 x 4096 mesh
   0, 0.250000
 100, 0.002397
 200, 0.001204
 300, 0.000804
 400, 0.000603
 500, 0.000483
 600, 0.000403
 700, 0.000345
 800, 0.000302
 900, 0.000269
total: 19.627867 s
```

OPENACC PARALLEL LOOP DIRECTIVE

OPENACC PARALLEL LOOP DIRECTIVE

Parallelizing a single loop

- Use a **parallel** directive to mark a region of code where you want parallel execution to occur
- The **loop** directive is used to instruct the compiler to parallelize the iterations of the next loop to run across the parallel gangs

C/C++

```
#pragma acc parallel
{
  #pragma acc loop
  for(int i = 0; j < N; i++)
    a[i] = 0;
}
```

Fortran

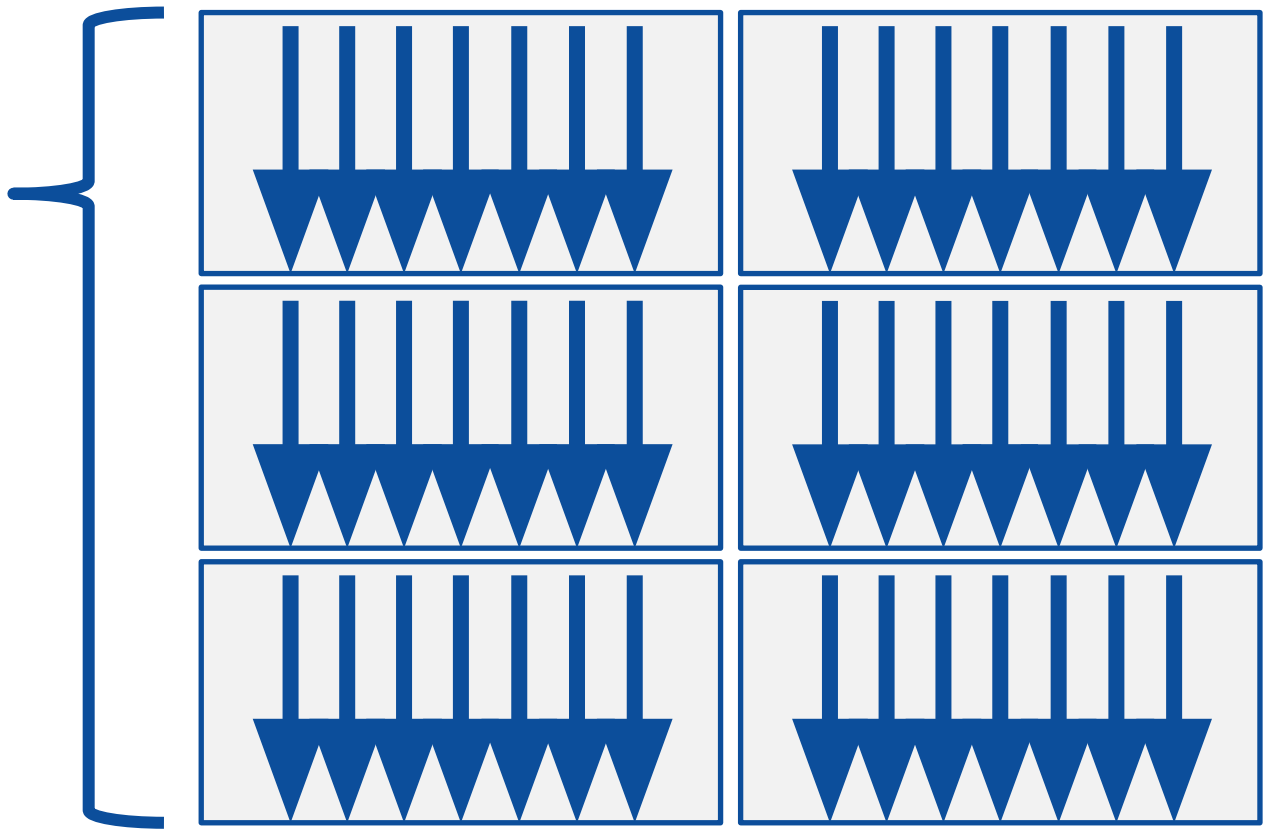
```
!$acc parallel
  !$acc loop
  do i = 1, N
    a(i) = 0
  end do
!$acc end parallel
```

OPENACC PARALLEL LOOP DIRECTIVE

Expressing parallelism

```
#pragma acc parallel loop
{


    for(int i = 0; i < N; i++)
    {
        // Do Something
    }
}
```

Generate parallelism and parallelize the next loop nest



OpenACC More Science, Less Programming NVIDIA

OPENACC PARALLEL LOOP DIRECTIVE

Parallelizing multiple loops

- To parallelize multiple loops, each loop should be accompanied by a parallel directive
- Each parallel loop can have different loop boundaries and loop optimizations
- Each parallel loop can be parallelized in a different way
- This is the recommended way to parallelize multiple loops. Attempting to parallelize multiple loops within the same parallel region may give performance issues or unexpected results

```
#pragma acc parallel loop
for(int i = 0; i < N; i++)
  a[i] = 0;

#pragma acc parallel loop
for(int j = 0; j < M; j++)
  b[j] = 0;
```

THREE LEVELS OF PARALLELISM

Expressing parallelism

- Gang
  - Like work *crews* they are completely independent of each other and may operate in parallel or even at different times
- Worker
  - Individual *painters* they can operate on their own but may also share resources with other workers in the same gang
- Vector
  - *Paint roller* is the vector
  - where the *width* of the roller represents the vector length.

PARALLELIZE WITH OPENACC PARALLEL LOOP

```
    while ( error > tol && iter < iter_max )
    {
      double error = 0.0;
#pragma acc parallel loop reduction(max:error)
      for (int j = 1; j < n - 1; j++)
      {
        for (int i = 1; i < m - 1; i++)
        {
          Anew[OFFSET(j, i, m)] = 0.25 * \
                    (A[OFFSET(j, i + 1, m)] + A[OFFSET(j, i - 1, m)] + \
                     A[OFFSET(j - 1, i, m)] + A[OFFSET(j + 1, i, m)]);
          error = fmax(error, fabs(Anew[OFFSET(j, i, m)] - A[OFFSET(j, i, m)]));
        }
      }

#pragma acc parallel loop
      for (int j = 1; j < n - 1; j++)
      {
        for (int i = 1; i < m - 1; i++)
        {
          A[OFFSET(j, i, m)] = Anew[OFFSET(j, i, m)];
        }
      }

      if (iter % 100 == 0)
        printf("%5d, %0.6f\n", iter, error);

      iter++;
    }
```

Parallelize first loop nest, max *reduction* required.

Parallelize second loop.

We didn't detail *how* to parallelize the loops, just *which* loops to parallelize.

OpenACC More Science, Less Programming

NVIDIA

REDUCTION CLAUSE

- The **reduction** clause takes many values and "reduces" them to a single value, such as in a sum or maximum
- Each thread calculates its part
- The compiler will perform a final reduction to produce a **single global result** using the specified operation

```
for( i = 0; i < size; i++ )
    for( j = 0; j < size; j++ )
        for( k = 0; k < size; k++ )
            c[i][j] += a[i][k] *
b[k][j];
```

```
for( i = 0; i < size; i++ )
    for( j = 0; j < size; j++ )
        double tmp = 0.0f;
        #pragma parallel acc loop \
                    reduction(+:tmp)
        for( k = 0; k < size; k++ )
            tmp += a[i][k] *
b[k][j];
        c[i][j] = tmp;
```

OpenACC More Science, Less Programming NVIDIA

REDUCTION CLAUSE OPERATORS

Operator	Description	Example
`+`	Addition/Summation	`reduction(+:sum)`
`*`	Multiplication/Product	`reduction(*:product)`
`max`	Maximum value	`reduction(max:maximum)`
`min`	Minimum value	`reduction(min:minimum)`
`&`	Bitwise and	`reduction(&:val)`
`|`	Bitwise or	`reduction(|:val)`
`&&`	Logical and	`reduction(&&:val)`
`||`	Logical or	`reduction(||:val)`

OpenACC More Science, Less Programming NVIDIA

BUILD AND RUN THE CODE

GCC, GNU Compiler Collection

C/C++ & Fortran Support for:

AMD GPUs

NVIDIA GPUs

For more information on how to run OpenACC code with GCC see: https://gcc.gnu.org/wiki/OpenACC/Quick%20Reference%20Guide

HPE Cray Compiling Environment

Fortran Support for:

AMD GPUs

NVIDIA GPUs

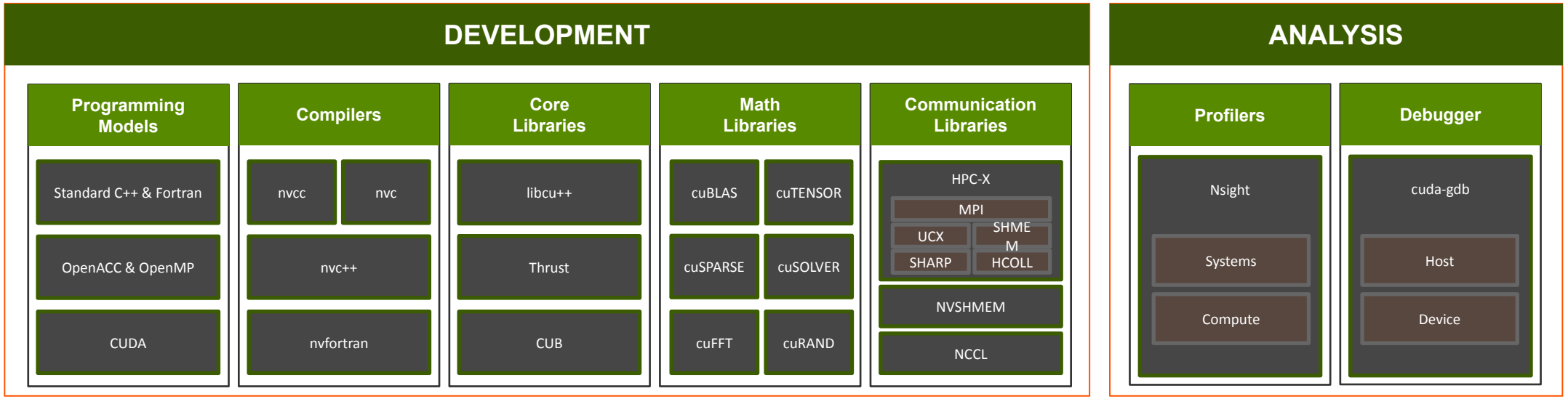
For more information on how to run OpenACC code with Cray ftn see:
https://support.hpe.com/hpesc/public/docDisplay?docId=a00114872en_us&page=OpenACC_Use.html&docLocale=en_US

Hewlett Packard Enterprise

OpenACC More Science, Less Programming

NVIDIA

NVIDIA HPC SDK



Develop for the NVIDIA Platform: GPU, CPU and Interconnect
Libraries | Accelerated C++ and Fortran | Directives | CUDA
7-8 Releases Per Year | Freely Available

OpenACC More Science, Less Programming NVIDIA

NVIDIA HPC SDK

nvc, nvc++ and nvfortran

- Use nvc, nvc++, and nvfortran to compile for C, C++, Fortran
- The -acc flag enables building OpenACC code
- -acc=multicore – Build the code to run across threads on a multicore CPU
- -acc=gpu – Build the code for an NVIDIA GPU
- -Minfo flag will instruct the compiler to print feedback about the compiled code

```
$ nvc –fast –Minfo=accel –acc=multicore main.c
$ nvc++ -fast -Minfo=accel –acc=multicore main.cpp
$ nvfortran –fast –Minfo=accel –acc=multicore main.f90
```

BUILDING THE CODE (MULTICORE)

```
$ nvc++ -fast -acc=multicore -Minfo=accel laplace2d_uvm.cpp
main:
     51, Generating Multicore code
         53, #pragma acc loop gang
     53, Generating implicit reduction(max:error)
     55, Loop is parallelizable
     60, Generating Multicore code
         63, #pragma acc loop gang
     65, Loop is parallelizable
```

OPENACC SPEED-UP

NVIDIA NVHPC

```
$ nvc++ -fast -acc=multicore -Minfo=accel laplace2d_uvm.cpp
```

```
Jacobi relaxation Calculation: 4096 x 4096 mesh
   0, 0.250000
 100, 0.002397
 200, 0.001204
 300, 0.000804
 400, 0.000603
 500, 0.000483
 600, 0.000403
 700, 0.000345
 800, 0.000302
 900, 0.000269
total: 3.831192 s
```

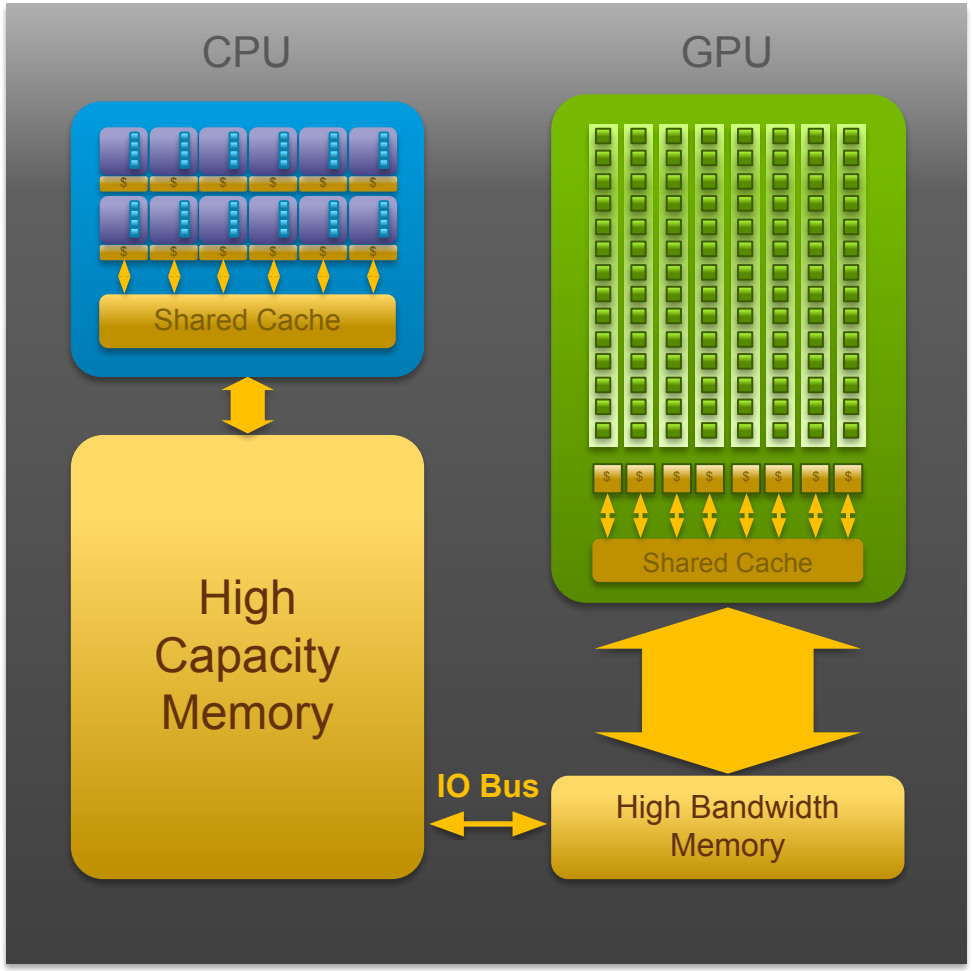
OpenACC More Science, Less Programming NVIDIA

NVIDIA HPC SDK 24.11, NVIDIA Tesla H100, AMD EPYC 7413 CPU

CPU AND GPU MEMORIES

CPU + GPU
Physical Diagram

- CPU memory is larger, GPU memory has more bandwidth
- CPU and GPU memory are usually separate, connected by an I/O bus (PCI-e/ NVLINK)
- Any data transferred between the CPU and GPU will be handled by the I/O Bus
- The I/O Bus is relatively slow compared to memory bandwidth
- The GPU cannot perform computation until the data is within its memory



DATA SHAPING

OPENACC DATA DIRECTIVE

Definition

- The data directive defines a lifetime for data on the device beyond individual loops
- During the region data is essentially “owned by” the accelerator
- Data clauses express shape and data movement for the region

```
#pragma acc data clauses
{
    < Sequential and/or Parallel code >
}
```

```
!$acc data clauses
    < Sequential and/or Parallel code >
!$acc end data
```

DATA CLAUSES

`copy( list )` **Allocates memory on GPU and copies data from host to GPU when entering region and copies data to the host when exiting region.**

Principal use: For many important data structures in your code, this is a logical default to input, modify and return the data.

`copyin( list )` **Allocates memory on GPU and copies data from host to GPU when entering region.**

Principal use: Think of this like an array that you would use as just an input to a subroutine.

`copyout( list )` **Allocates memory on GPU and copies data to the host when exiting region.**

Principal use: A result that isn't overwriting the input data structure.

`create( list )` **Allocates memory on GPU but does not copy.**

Principal use: Temporary arrays.

ARRAY SHAPING

- Sometimes the compiler needs help understanding the *shape* of an array
- The first number is the start index of the array
- In C/C++, the second number is how much data is to be transferred
- In Fortran, the second number is the ending index

```
copy(array[starting_index:length])
```
C/C++

```
copy(array(starting_index:ending_index))
```
Fortran

ARRAY SHAPPING

Mult-dimensional Array

```
copy(array[0:N][0:M])
```

C/C++

Both of these examples copy a 2D array to the device

```
copy(array(1:N, 1:M))
```

Fortran

DATA MOVEMENT

```
while ( err > tol && iter < iter_max ) {
  err=0.0;

#pragma acc parallel loop reduction(max:err) copy(A[:n*m],Anew[:n*m])
  for( int j = 1; j < n-1; j++) {
    for(int i = 1; i < m-1; i++) {

      Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +
                           A[j-1][i] + A[j+1][i]);

      err = max(err, abs(Anew[j][i] - A[j][i]));
    }
  }

#pragma acc parallel loop copy(A[:n*m],Anew[:n*m])
  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      A[j][i] = Anew[j][i];
    }
  }
  iter++;
}
```

Data clauses provide necessary “shape” to the arrays.

BUILD FOR GPU

BUILDING THE CODE (GPU)

```
$ nvc++ -fast -acc=gpu -Minfo=accel laplace2d_uvm.cpp
main:
     52, Generating copyin(A[:m*n]) [if not already present]
         Generating implicit firstprivate(j,n,m)
         Generating NVIDIA GPU code
         54, #pragma acc loop gang /* blockIdx.x */
             Generating reduction(max:error)
         56, #pragma acc loop vector(128) /* threadIdx.x */
     52, Generating implicit copy(error) [if not already present]
         Generating copy(Anew[:m*n]) [if not already present]
     56, Loop is parallelizable
     67, Generating copyout(A[:m*n]) [if not already present]
         Generating copyin(Anew[:m*n]) [if not already present]
         Generating implicit firstprivate(j,n,m)
         Generating NVIDIA GPU code
         69, #pragma acc loop gang /* blockIdx.x */
         71, #pragma acc loop vector(128) /* threadIdx.x */
     71, Loop is parallelizable
```

OPENACC ~~SPEED-UP~~ SLOWDOWN??

NVIDIA NVHPC

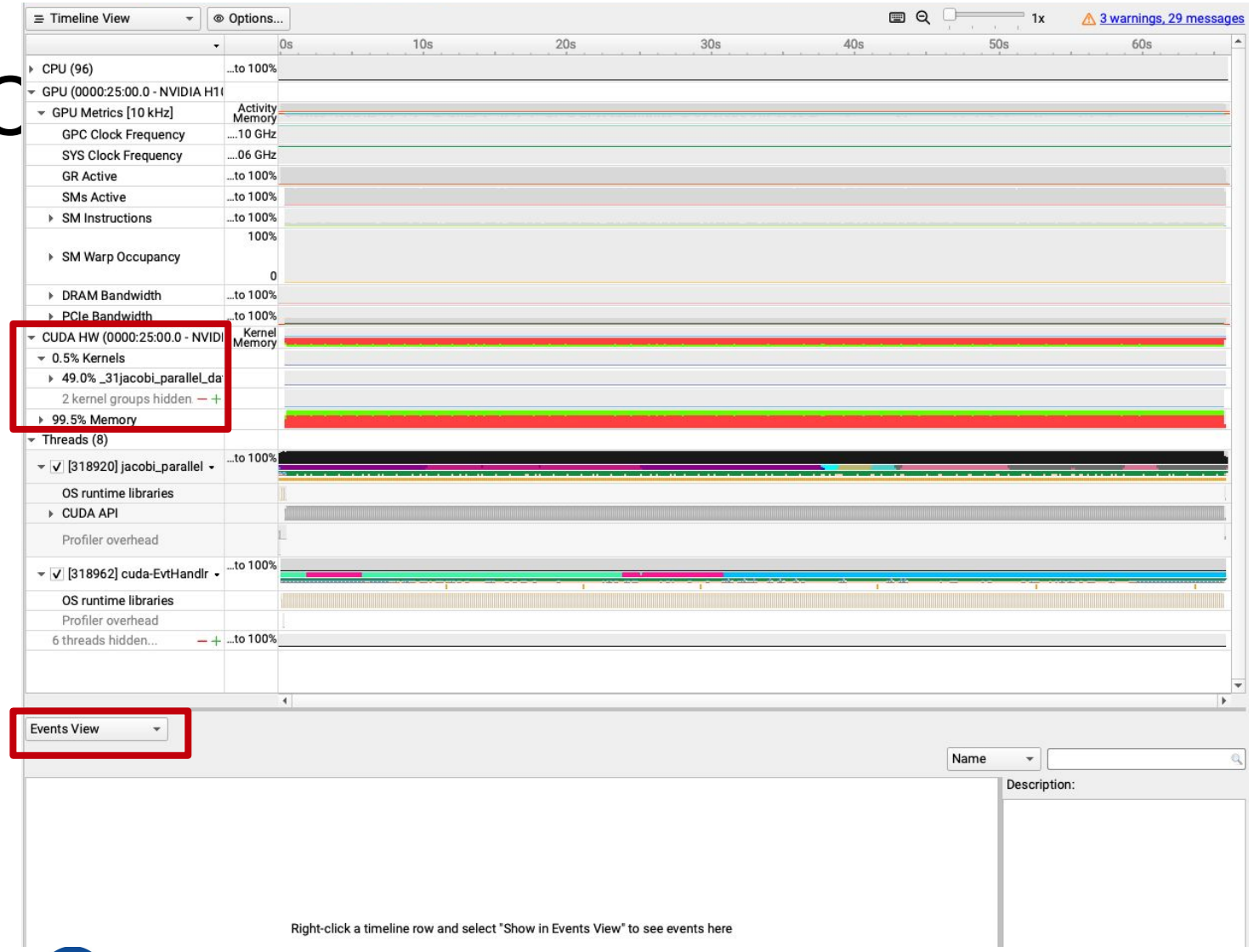
```
$ nvc++ -fast -acc=gpu -Minfo=accel laplace2d_uvm.cpp

Jacobi relaxation Calculation: 4096 x 4096 mesh
   0, 0.250000
 100, 0.002397
 200, 0.001204
 300, 0.000804
 400, 0.000603
 500, 0.000483
 600, 0.000403
 700, 0.000345
 800, 0.000302
 900, 0.000269
total: 63.479911 s
```

NVIDIA HPC SDK 24.11, NVIDIA Tesla H100, AMD EPYC 7413 CPU

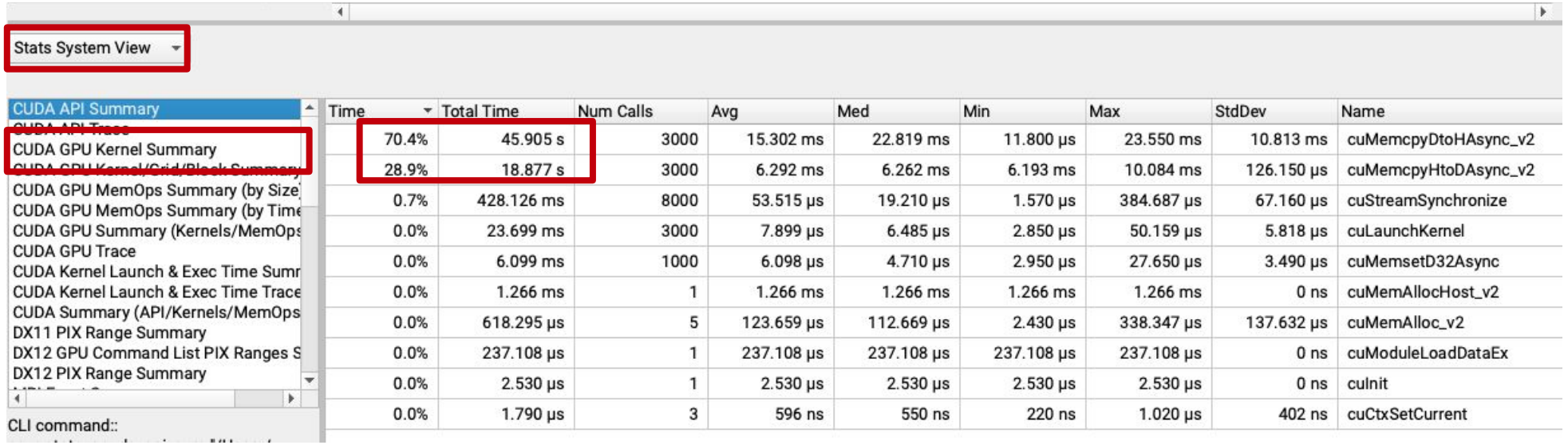
OPENACC SLOWDOWN?? PROFILE

```
$ nsys profile -gpu-metrics-devices=all ./laplace2d_uvm.o
main:
GPU 0: General Metrics for NVIDIA GH100 (any frequency)
Collecting data...
Jacobi relaxation Calculation: 4096 x 4096 mesh
    0, 0.250000
  100, 0.002397
  200, 0.001204
  300, 0.000804
  400, 0.000603
  500, 0.000483
  600, 0.000403
  700, 0.000345
  800, 0.000302
  900, 0.000269
 total: 65.671076 s
Generating '/tmp/nsys-report-3775.qdstrm'
[1/1] [========================100%] report1.nsys-rep
```

OPENAC



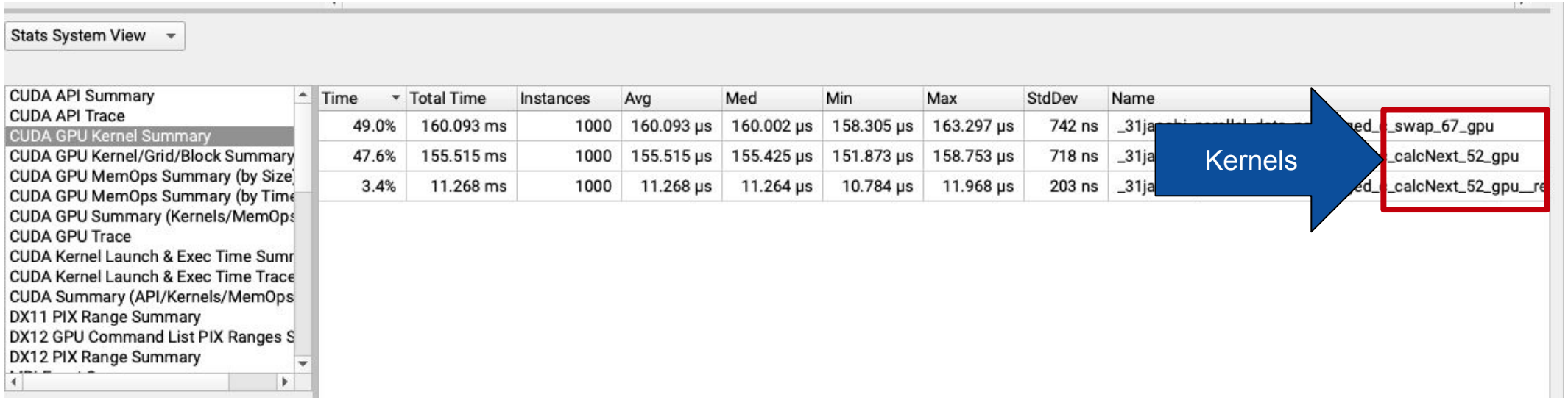
OPENACC SLOWDOWN??



Time	Total Time	Num Calls	Avg	Med	Min	Max	StdDev	Name
70.4%	45.905 s	3000	15.302 ms	22.819 ms	11.800 µs	23.550 ms	10.813 ms	cuMemcpyDtoHAsync_v2
28.9%	18.877 s	3000	6.292 ms	6.262 ms	6.193 ms	10.084 ms	126.150 µs	cuMemcpyHtoDAsync_v2
0.7%	428.126 ms	8000	53.515 µs	19.210 µs	1.570 µs	384.687 µs	67.160 µs	cuStreamSynchronize
0.0%	23.699 ms	3000	7.899 µs	6.485 µs	2.850 µs	50.159 µs	5.818 µs	cuLaunchKernel
0.0%	6.099 ms	1000	6.098 µs	4.710 µs	2.950 µs	27.650 µs	3.490 µs	cuMemsetD32Async
0.0%	1.266 ms	1	1.266 ms	1.266 ms	1.266 ms	1.266 ms	0 ns	cuMemAllocHost_v2
0.0%	618.295 µs	5	123.659 µs	112.669 µs	2.430 µs	338.347 µs	137.632 µs	cuMemAlloc_v2
0.0%	237.108 µs	1	237.108 µs	237.108 µs	237.108 µs	237.108 µs	0 ns	cuModuleLoadDataEx
0.0%	2.530 µs	1	2.530 µs	2.530 µs	2.530 µs	2.530 µs	0 ns	cuInit
0.0%	1.790 µs	3	596 ns	550 ns	220 ns	1.020 µs	402 ns	cuCtxSetCurrent

NVIDIA HPC SDK 24.11, NVIDIA Tesla H100, AMD EPYC 7413 CPU

OPENACC SLOWDOWN??



OPTIMIZE DATA MOVEMENT

OPENACC DATA DIRECTIVE

Definition

- The data directive defines a lifetime for data on the device beyond individual loops
- During the region data is essentially “owned by” the accelerator
- Data clauses express shape and data movement for the region

```
#pragma acc data clauses
{
    < Sequential and/or Parallel code >
}
```

```
!$acc data clauses
    < Sequential and/or Parallel code >
!$acc end data
```

DATA CLAUSES

`copy( list )` **Allocates memory on GPU and copies data from host to GPU when entering region and copies data to the host when exiting region.**

Principal use: For many important data structures in your code, this is a logical default to input, modify and return the data.

`copyin( list )` **Allocates memory on GPU and copies data from host to GPU when entering region.**

Principal use: Think of this like an array that you would use as just an input to a subroutine.

`copyout( list )` **Allocates memory on GPU and copies data to the host when exiting region.**

Principal use: A result that isn't overwriting the input data structure.

`create( list )` **Allocates memory on GPU but does not copy.**

Principal use: Temporary arrays.

ARRAY SHAPING

- Sometimes the compiler needs help understanding the *shape* of an array
- The first number is the start index of the array
- In C/C++, the second number is how much data is to be transferred
- In Fortran, the second number is the ending index

```
copy(array[starting_index:length])
```

C/C++

```
copy(array(starting_index:ending_index))
```

Fortran

OPTIMIZED DATA MOVEMENT

```
    while ( error > tol && iter < iter_max )
    {
      double error = 0.0;
#pragma acc parallel loop reduction(max:error) copy(A[:n*m],Anew[:n*m])

      for (int j = 1; j < n - 1; j++)
      {
        for (int i = 1; i < m - 1; i++)
        {
          Anew[OFFSET(j, i, m)] = 0.25 * \
                    (A[OFFSET(j, i + 1, m)] + A[OFFSET(j, i - 1, m)] + \
                     A[OFFSET(j - 1, i, m)] + A[OFFSET(j + 1, i, m)]);
          error = fmax(error, fabs(Anew[OFFSET(j, i, m)] - A[OFFSET(j, i, m)]));
        }
      }

#pragma acc parallel loop copy(A[:n*m],Anew[:n*m])

      for (int j = 1; j < n - 1; j++)
      {
        for (int i = 1; i < m - 1; i++)
        {
          A[OFFSET(j, i, m)] = Anew[OFFSET(j, i, m)];
        }
      }

      if (iter % 100 == 0)
        printf("%5d, %0.6f\n", iter, error);

      iter++;
    }
```

Currently we're copying to/from the GPU for each loop, can we reuse it?

OPTIMIZED DATA MOVEMENT

```
#pragma acc data copy(A[:n*m]) create(Anew[:n*m])
      while ( error > tol && iter < iter_max )
    {
      double error = 0.0;
#pragma acc parallel loop reduction(max:error) copy(A[:n*m],Anew[:n*m])
      for (int j = 1; j < n - 1; j++)
      {
        for (int i = 1; i < m - 1; i++)
        {
          Anew[OFFSET(j, i, m)] = 0.25 * \
                    (A[OFFSET(j, i + 1, m)] + A[OFFSET(j, i - 1, m)] + \
                     A[OFFSET(j - 1, i, m)] + A[OFFSET(j + 1, i, m)]);
          error = fmax(error, fabs(Anew[OFFSET(j, i, m)] - A[OFFSET(j, i, m)]));
        }
      }

#pragma acc parallel loop copyin(Anew[:n*m],A[:n*m])
      for (int j = 1; j < n - 1; j++)
      {
        for (int i = 1; i < m - 1; i++)
        {
          A[OFFSET(j, i, m)] = Anew[OFFSET(j, i, m)];
        }
      }

      if (iter % 100 == 0)
        printf("%5d, %0.6f\n", iter, error);

      iter++;
```

Copy A to/from the accelerator only when needed.

Copy initial condition of Anew, but not final value

REBUILD THE CODE

```
$ nvc++ -fast -acc=gpu -Minfo=accel laplace2d_uvm.cpp
main:
     50, Generating copy(A[:16777216]) [if not already p.esen
         Generating copyin(Anew[:16777216]) [if not already p
     52, Generating NVIDIA GPU code
         54, #pragma acc loop gang /* blockIdx.x */
             Generating reduction(max:error)
         56, #pragma acc loop vector(128) /* threadIdx.x */
     52, Generating implicit copy(error) [if not already present]
     56, Loop is parallelizable
     61, Generating NVIDIA GPU code
         64, #pragma acc loop gang /* blockIdx.x */
         66, #pragma acc loop vector(128) /* threadIdx.x */
     66, Loop is parallelizable
```

Now data movement only happens at our data region.

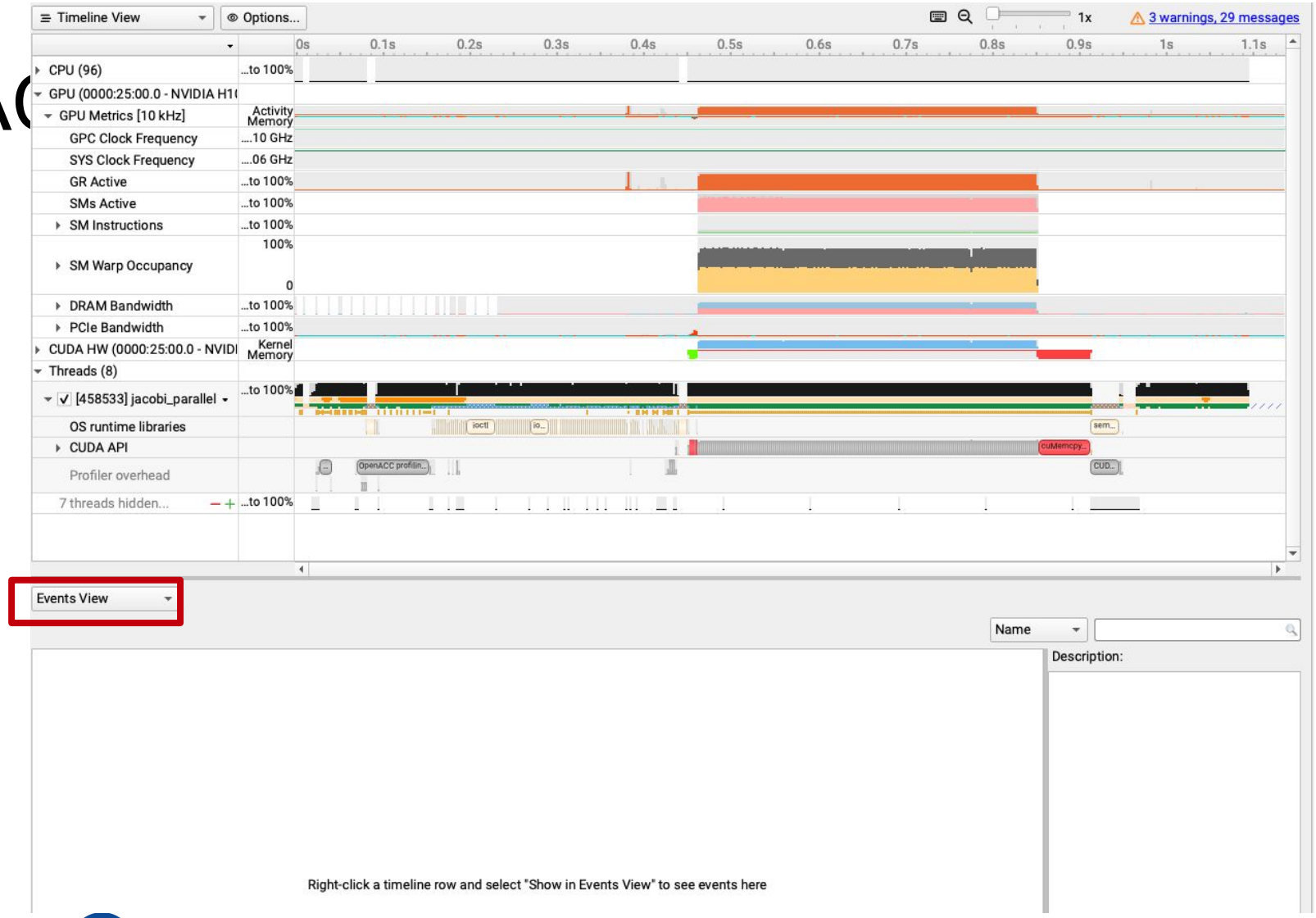
OPENACC SPEED-UP

NVIDIA NVHPC

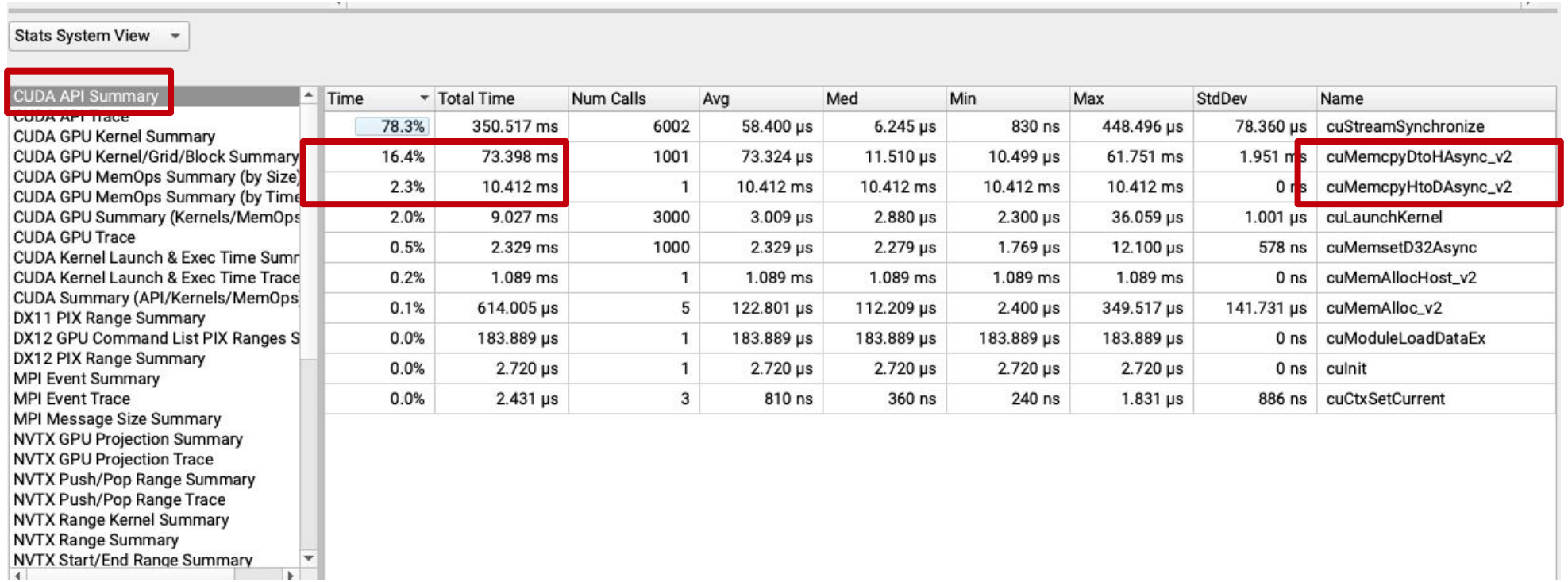
```
$ nvc++ -fast -acc=gpu -Minfo=accel laplace2d_uvm.cpp
```

```
Jacobi relaxation Calculation: 4096 x 4096 mesh
    0, 0.250000
  100, 0.002397
  200, 0.001204
  300, 0.000804
  400, 0.000603
  500, 0.000483
  600, 0.000403
  700, 0.000345
  800, 0.000302
  900, 0.000269
 total: 0.693270 s
```

OPENAC



OPENACC PROFILE



Time	Total Time	Num Calls	Avg	Med	Min	Max	StdDev	Name
78.3%	350.517 ms	6002	58.400 µs	6.245 µs	830 ns	448.496 µs	78.360 µs	cuStreamSynchronize
16.4%	73.398 ms	1001	73.324 µs	11.510 µs	10.499 µs	61.751 ms	1.951 ms	cuMemcpyDtoHAsync_v2
2.3%	10.412 ms	1	10.412 ms	10.412 ms	10.412 ms	10.412 ms	0 ns	cuMemcpyHtoDAsync_v2
2.0%	9.027 ms	3000	3.009 µs	2.880 µs	2.300 µs	36.059 µs	1.001 µs	cuLaunchKernel
0.5%	2.329 ms	1000	2.329 µs	2.279 µs	1.769 µs	12.100 µs	578 ns	cuMemsetD32Async
0.2%	1.089 ms	1	1.089 ms	1.089 ms	1.089 ms	1.089 ms	0 ns	cuMemAllocHost_v2
0.1%	614.005 µs	5	122.801 µs	112.209 µs	2.400 µs	349.517 µs	141.731 µs	cuMemAlloc_v2
0.0%	183.889 µs	1	183.889 µs	183.889 µs	183.889 µs	183.889 µs	0 ns	cuModuleLoadDataEx
0.0%	2.720 µs	1	2.720 µs	2.720 µs	2.720 µs	2.720 µs	0 ns	cuInit
0.0%	2.431 µs	3	810 ns	360 ns	240 ns	1.831 µs	886 ns	cuCtxSetCurrent

OPENACC PROFILE

Before Optimization

Time	Total Time	Num Calls	Avg	Med	Min	Max	StdDev	Name
70.4%	45.905 s	3000	15.302 ms	22.819 ms	11.800 µs	23.550 ms	10.813 ms	cuMemcpyDtoHAsync_v2
28.9%	18.877 s	3000	6.292 ms	6.262 ms	6.193 ms	10.084 ms	126.150 µs	cuMemcpyHtoDAsync_v2
0.7%	428.126 ms	8000	53.515 µs	19.210 µs	1.570 µs	384.687 µs	67.160 µs	cuStreamSynchronize
0.0%	23.699 ms	3000	7.899 µs	6.485 µs	2.850 µs	50.159 µs	5.818 µs	cuLaunchKernel

After Optimization

Time	Total Time	Num Calls	Avg	Med	Min	Max	StdDev	Name
78.3%	350.517 ms	6002	58.400 µs	6.245 µs	830 ns	448.496 µs	78.360 µs	cuStreamSynchronize
16.4%	73.398 ms	1001	73.324 µs	11.510 µs	10.499 µs	61.751 ms	1.951 m	cuMemcpyDtoHAsync_v2
2.3%	10.412 ms	1	10.412 ms	10.412 ms	10.412 ms	10.412 ms	0 n	cuMemcpyHtoDAsync_v2
2.0%	9.027 ms	3000	3.009 µs	2.880 µs	2.300 µs	36.059 µs	1.001 µs	cuLaunchKernel

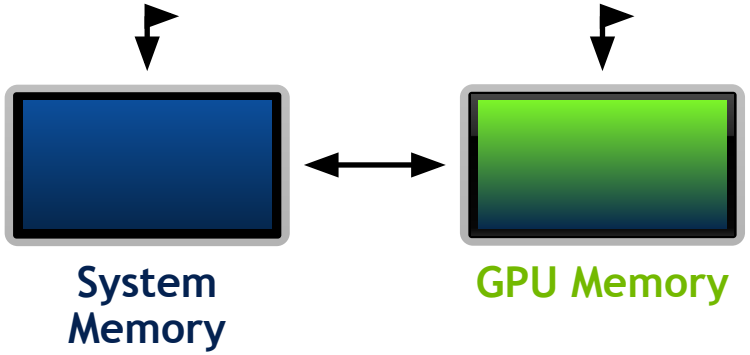
CUDA UNIFIED MEMORY

CUDA UNIFIED MEMORY

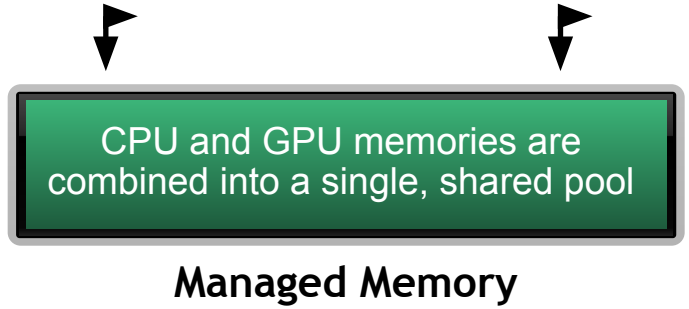
Simplified Developer Effort

Commonly referred to as
"managed memory."

Without Managed Memory



With Managed Memory



OpenACC More Science, Less Programming

NVIDIA

BUILDING THE CODE (GPU)

```
$ nvc++ -fast -acc=gpu -gpu=mem:managed -Minfo=accel laplace2d_uvm.cpp
main:
     51, Generating NVIDIA GPU code
         53, #pragma acc loop gang /* blockIdx.x */
             Generating implicit reduction(max:error)
         55, #pragma acc loop vector(128) /* threadIdx.x */
     51, Generating implicit copy(error) [if not already present]
     55, Loop is parallelizable
     60, Generating NVIDIA GPU code
         63, #pragma acc loop gang /* blockIdx.x */
         65, #pragma acc loop vector(128) /* threadIdx.x */
     65, Loop is parallelizable
```

OPENACC SPEED-UP

NVIDIA NVHPC

```
$ nvc++ -fast -acc=gpu -Minfo=accel laplace2d_uvm.cpp
```

```
Jacobi relaxation Calculation: 4096 x 4096 mesh
   0, 0.250000
 100, 0.002397
 200, 0.001204
 300, 0.000804
 400, 0.000603
 500, 0.000483
 600, 0.000403
 700, 0.000345
 800, 0.000302
 900, 0.000269
total: 0.426259 s
```

OpenACC More Science, Less Programming NVIDIA

NVIDIA HPC SDK 22.3, NVIDIA Tesla A100, AMD EPYC 7742 CPU

LOOP OPTIMIZATIONS

OPENACC LOOP DIRECTIVE

Expressing parallelism

- Mark a single for loop for parallelization
- Allows the programmer to give additional information and/or optimizations about the loop
- Provides many different ways to describe the type of parallelism to apply to the loop
- Must be contained within an OpenACC compute region (either a kernels or a parallel region) to parallelize loops

C/C++

```
#pragma acc loop
for(int i = 0; i < N; i++)
    // Do something
```

Fortran

```
!$acc loop
do i = 1, N
    ! Do something
```

COLLAPSE CLAUSE

- **`collapse( N )`**
- Combine the next N tightly nested loops
- Can turn a multidimensional loop nest into a single-dimension loop
- This can be extremely useful for increasing memory locality, as well as creating larger loops to expose more parallelism

```
#pragma acc parallel loop collapse(2)
for( i = 0; i < size; i++ )
  for( j = 0; j < size; j++ )
    double tmp = 0.0f;
    #pragma acc loop reduction(+:tmp)
    for( k = 0; k < size; k++ )
      tmp += a[i][k] * b[k][j];
      c[i][j] = tmp;
```

CAN YOU DO BETTER?

CONCLUSION

OpenACC is a mature, directive-based programming model that is available for GPUs, multicore CPUs, and more and is in use by more than 250 scientific applications.

This talk only scratches the surface of OpenACC capabilities.

Please refer to the resources on the next slide for more information.

OPENACC RESOURCES

Guides • Talks • Tutorials • Videos • Books • Spec • Code Samples • Teaching Materials • Events • Success Stories • Courses • Slack • Stack Overflow

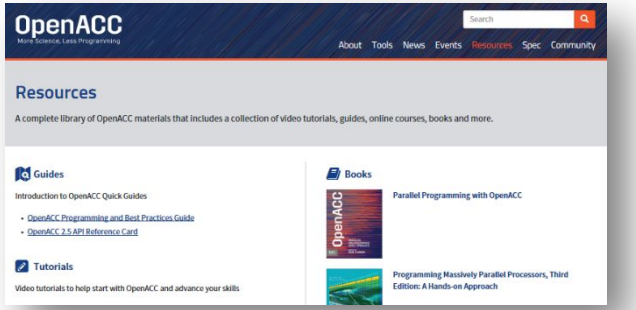
FREE Compilers

NVIDIA

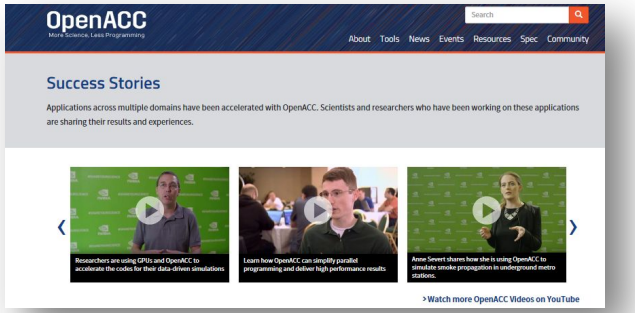
slack

https://www.openacc.org/community

Resources

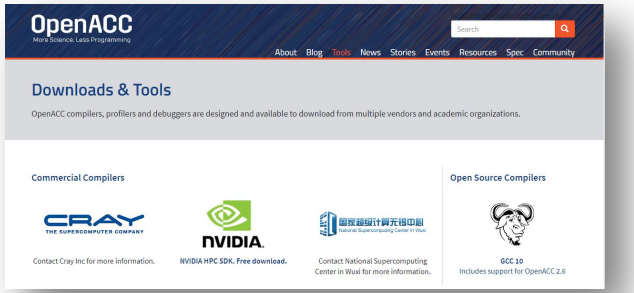


Success Stories



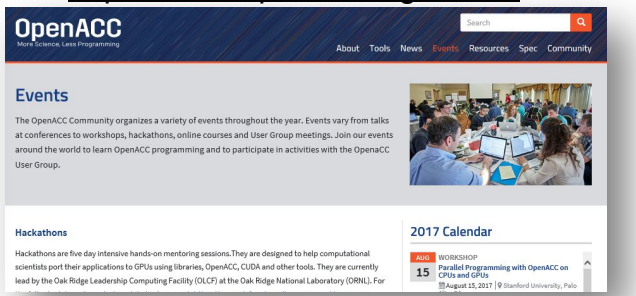
Compilers and Tools

https://www.openacc.org/tools



Events

https://www.openacc.org/events



OpenACC More Science, Less Programming NVIDIA