



# Accelerator programming with OpenACC

Colaboratorio Nacional de Computación Avanzada

Jorge Castro  
[jcastro@cenat.ac.cr](mailto:jcastro@cenat.ac.cr)

2018

# Agenda

1 Introduction

2 OpenACC life cycle

3 Hands on session

- Profiling and parallelizing
- Optimizing data movement

4 Best practices

# Introduction

# What is OpenACC?

OpenACC is a parallel programming standard that describes a set of compiler directives in C, C++ and Fortran to specify regions of code offloading from a host CPU to an attached accelerator



# What is an Accelerator?

Dedicated piece of hardware that performs specific functions faster than a CPU

- Graphic Processing Unit (GPU): electronic device that runs computer graphic algorithms to render images
- Coprocessor: electronic device to supplement functions of CPU (arithmetic, encryption, error detection)



# Top 500 green

TOP500			Cores	Power		
Rank	Rank	System		Rmax (TFlop/s)	Power (kW)	Efficiency (GFlops/watts)
1	259	Shoubu system B - ZettaScaler-2.2, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 , PEZY Computing / Exascaler Inc. Advanced Center for Computing and Communication, RIKEN Japan	794,400	842.0	50	17.009
2	307	Suiren2 - ZettaScaler-2.2, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 , PEZY Computing / Exascaler Inc. High Energy Accelerator Research Organization /KEK Japan	762,624	788.2	47	16.759
3	276	Sakura - ZettaScaler-2.2, Xeon E5-2618Lv3 8C 2.3GHz, Infiniband EDR, PEZY-SC2 , PEZY Computing / Exascaler Inc. PEZY Computing K.K. Japan	794,400	824.7	50	16.657
4	149	DGX SaturnV Volta - NVIDIA DGX-1 Volta36, Xeon E5-2698v4 20C 2.2GHz, Infiniband EDR, NVIDIA Tesla V100 . Nvidia NVIDIA Corporation United States	22,440	1,070.0	97	15.113
5	4	Oyoukou - ZettaScaler-2.2 HPC system, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 700Mhz , ExaScaler Japan Agency for Marine-Earth Science and Technology Japan	19,860,000	19,135.8	1,350	14.173
6	13	TSUBAME3.0 - SGI ICE XA, IP139-SXM2, Xeon E5-2680v4 14C 2.4GHz, Intel Omni-Path, NVIDIA Tesla P100 SXM2 . HPE GSIC Center, Tokyo Institute of Technology Japan	135,828	8,125.0	792	13.704

Source: [www.top500.org/green500](http://www.top500.org/green500) (November 2017 list)

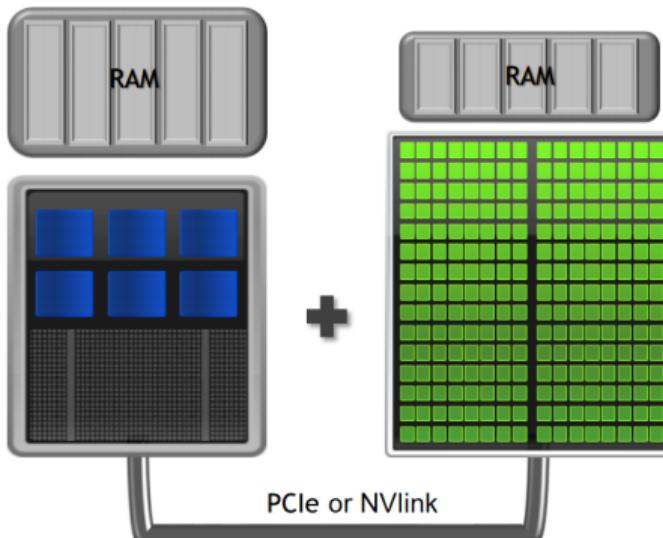
# P100 vs K40

Accelerator	Cores	Boost clock	Memory BW	DP perf	SP perf
Tesla k40	2880	875 MHz	288 GB/s	1.7 TFLOPS	5.0 TFLOPS
Tesla P100	3584	1480 MHz	720 GB/s	5.3 TFLOPS	10.6 TFLOPS
Tesla V100	5120	1530 MHz	900 GB/s	7.8 TFLOPS	15.7 TFLOPS



# Architecture

## Accelerator Nodes



CPU and GPU have distinct memories

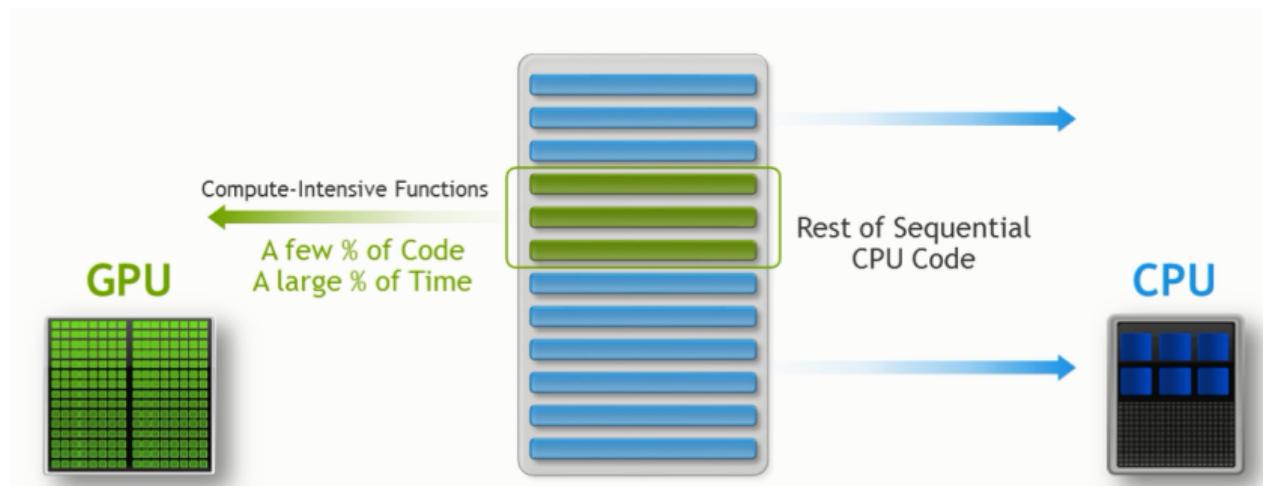
- CPU generally larger and slower
- GPU generally smaller and faster

CPU and GPU communicate via PCIe

- Data must be copied between these memories over PCIe
- PCIe Bandwidth is much lower than either memories

# Heterogeneous computing

Heterogeneous programming combines the use of more than one type of processors



# CPU vs GPU

Features	CPU	GPU
Main memory	large	small
Memory bandwidth	low	high
Clock Frequency	high	low
Performance per watt	low	high
Throughput <sup>1</sup>	low	high

---

<sup>1</sup>number of operations per unit of time

# Why use OpenACC?

- Simple
- Portable (Nvidia, Intel, AMD)
- Inter-operable (CUDA, MPI, OPENMP)
- Powerful (90% CUDA)

```
main()
{
    <serial code>
    #pragma acc kernels
    //automatically runs on GPU
    {
        <parallel code>
    }
}
```

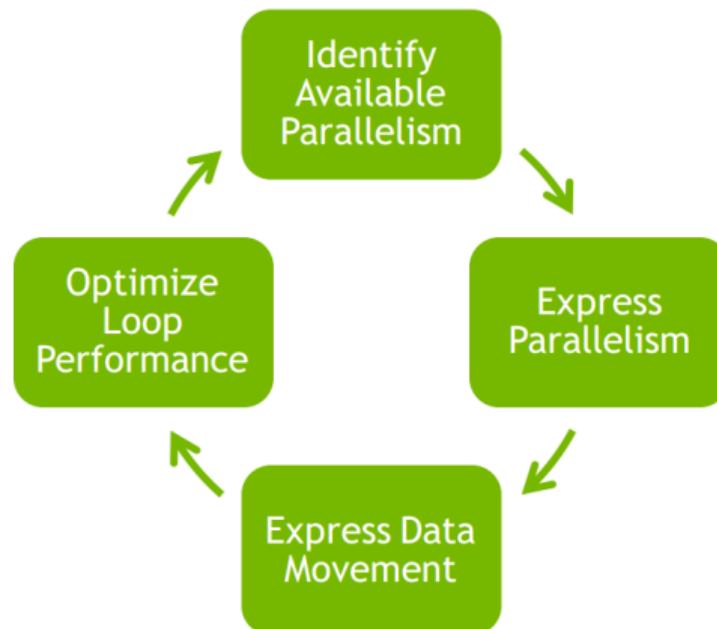


# Why use OpenACC? (2)

	<u>Shared-memory Programming</u>	<u>Accelerator Programming</u>	<u>Distributed-memory Programming</u>
<u>High-level</u>	OpenMP	OpenACC	Global Arrays, Charm++
<u>Low-level</u>	PThreads, OpenThreads	CUDA, OpenCL	MPI

# OpenACC life cycle

# OpenACC life cycle

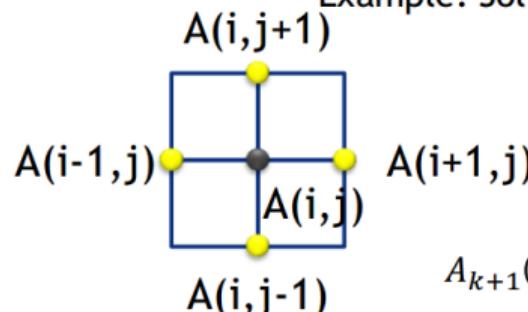


# Jacobi iteration

## Example: Jacobi Iteration

Stencil operation

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 (2)

## Jacobi Iteration: C Code

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

    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]));
        }
    }

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

    iter++;
}
```

Iterate until converged

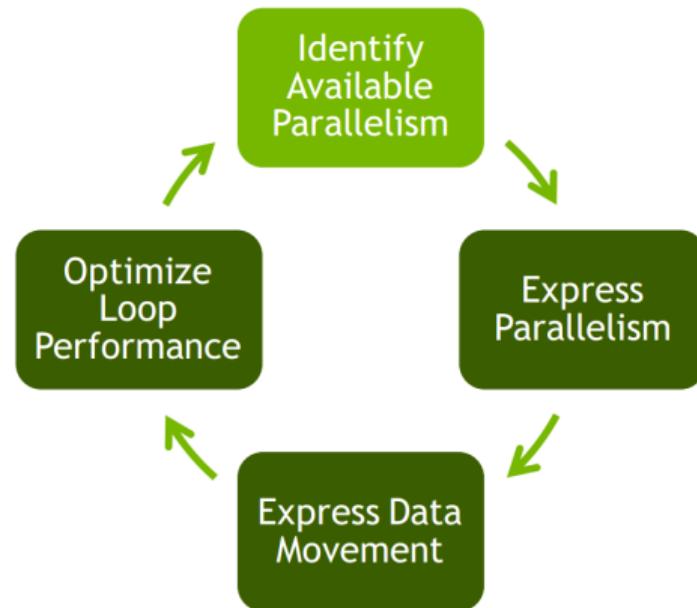
Iterate across matrix elements

Calculate new value from neighbors

Compute max error for convergence

Swap input/output arrays

# OpenACC life cycle



# Identify parallelism

## Identify Parallelism

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

    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]));
        }
    }

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

    iter++;
}
```

# Identify parallelism (2)

## Identify Parallelism

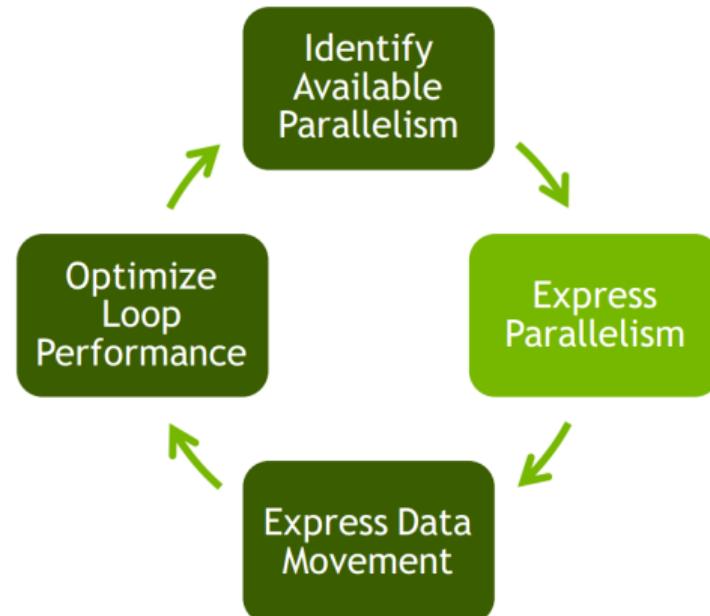
```
while ( err > tol && iter < iter_max ) {  
    err=0.0;  
  
    for( int j = 1; j < n-1; j++) {  
        for(int i = 1; i < m-1; i++) {  
  
            Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +  
                A[j-1][i] + A[j+1][i]);  
  
            err = max(err, abs(Anew[j][i] - A[j][i]));  
        }  
    }  
  
    for( int j = 1; j < n-1; j++) {  
        for( int i = 1; i < m-1; i++ ) {  
            A[j][i] = Anew[j][i];  
        }  
    }  
  
    iter++;  
}
```

Data dependency  
between iterations.

Independent loop  
iterations

Independent loop  
iterations

# OpenACC life cycle



# Express parallelism

## OpenACC kernels Directive

The kernels directive identifies a region that may contain *loops* that the compiler can turn into parallel *kernels*.

```
#pragma acc kernels
{
    for(int i=0; i<N; i++)
    {
        x[i] = 1.0;
        y[i] = 2.0;
    }

    for(int i=0; i<N; i++)
    {
        y[i] = a*x[i] + y[i];
    }
}
```

# Express parallelism

## OpenACC kernels Directive

The kernels directive identifies a region that may contain *loops* that the compiler can turn into parallel *kernels*.

```
#pragma acc kernels
{
    for(int i=0; i<N; i++)
    {
        x[i] = 1.0;
        y[i] = 2.0;
    }

    for(int i=0; i<N; i++)
    {
        y[i] = a*x[i] + y[i];
    }
}
```



The code snippet shows two nested loops. The first loop is enclosed in curly braces and labeled 'kernel 1'. The second loop is also enclosed in curly braces and labeled 'kernel 2', indicating they are separate parallel regions.

The compiler identifies 2 parallel loops and generates 2 kernels.

## Express parallelism (2)

### OpenACC parallel loop Directive

**parallel** - Programmer identifies a block of code containing parallelism. Compiler generates a *kernel*.

**loop** - Programmer identifies a loop that can be parallelized within the kernel.

NOTE: parallel & loop are often placed together

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



Generates a  
Parallel  
Kernel

# Express parallelism (3)

## Parallelize with OpenACC kernels

```
while ( err > tol && iter < iter_max ) {  
    err=0.0;  
  
    #pragma acc kernels  
    {  
        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]));  
            }  
        }  
  
        for( int j = 1; j < n-1; j++) {  
            for( int i = 1; i < m-1; i++ ) {  
                A[j][i] = Anew[j][i];  
            }  
        }  
    }  
    iter++;  
}
```



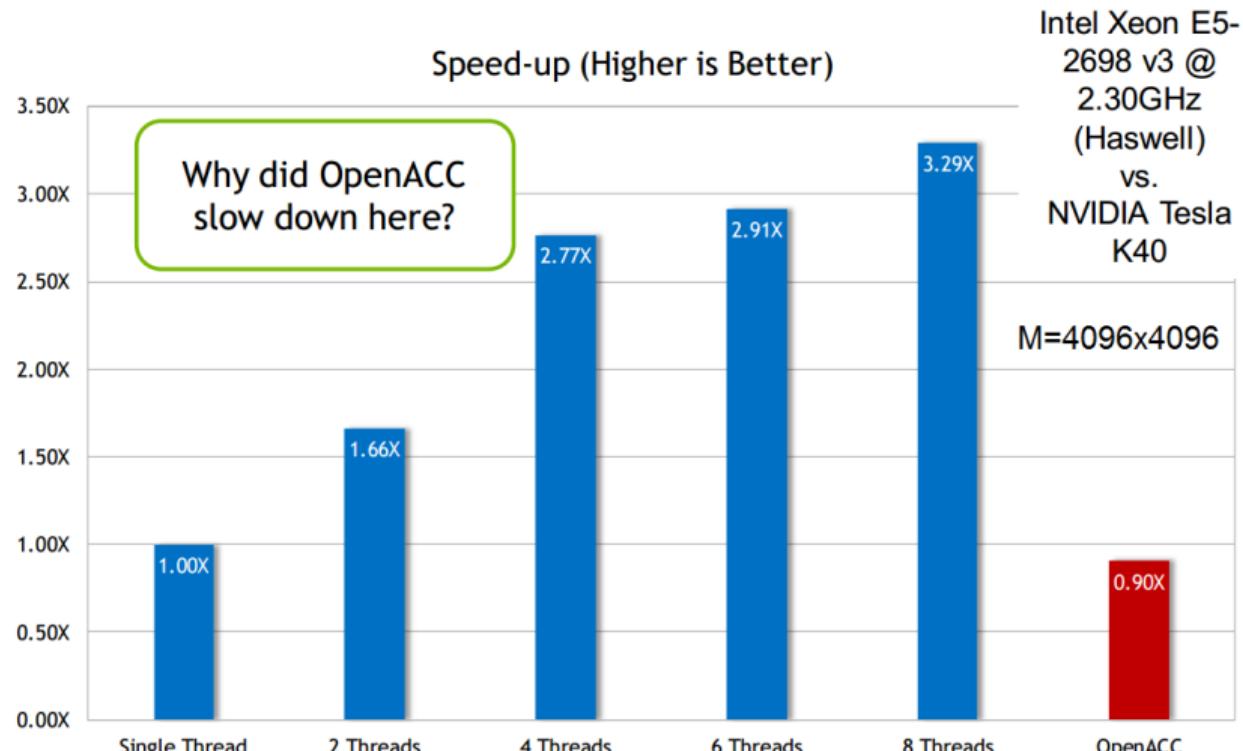
Look for parallelism  
within this region.

# Express parallelism (4)

## Building the code

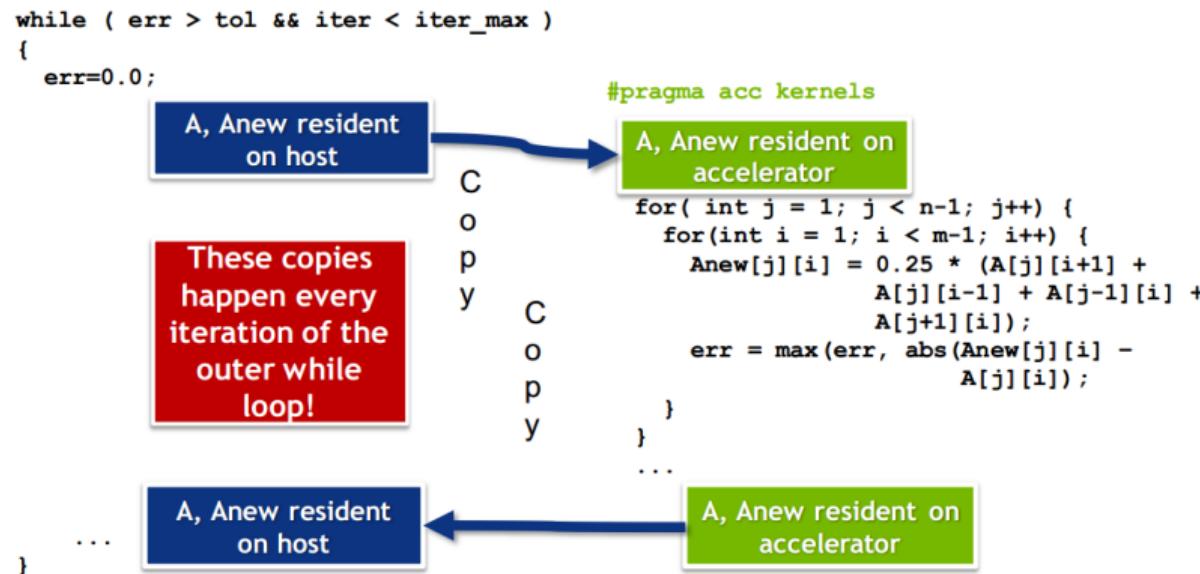
```
$ pgcc -fast -ta=tesla -Minfo=all laplace2d.c
main:
    40, Loop not fused: function call before adjacent loop
        Generated vector sse code for the loop
    51, Loop not vectorized/parallelized: potential early exits
    55, Generating copyout(Anew[1:4094][1:4094])
        Generating copyin(A[:, :])
        Generating copyout(A[1:4094][1:4094])
        Generating Tesla code
    57, Loop is parallelizable
    59, Loop is parallelizable
        Accelerator kernel generated
        57, #pragma acc loop gang /* blockIdx.y */
        59, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
    63, Max reduction generated for error
    67, Loop is parallelizable
    69, Loop is parallelizable
        Accelerator kernel generated
        67, #pragma acc loop gang /* blockIdx.y */
        69, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
```

# Express parallelism (5)



# Express parallelism (6)

## Excessive Data Transfers



# Express parallelism (7)

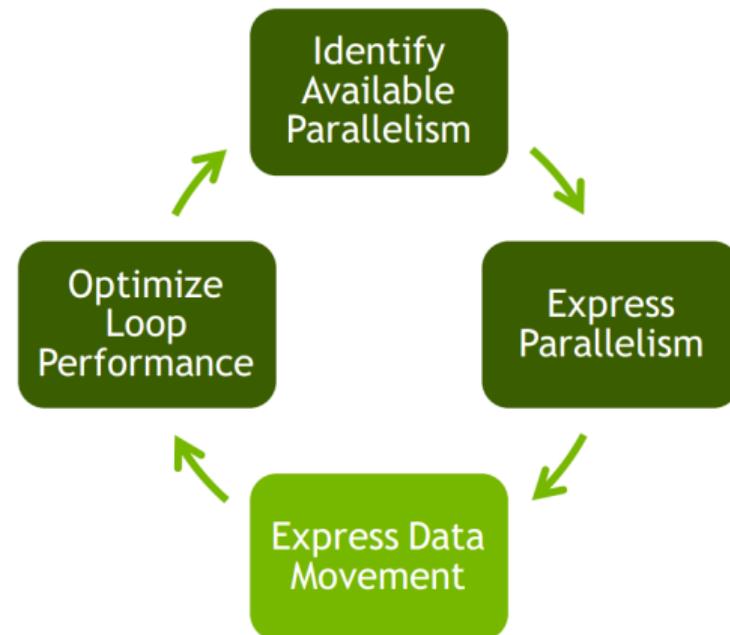
## Identifying Data Locality

```
while ( err > tol && iter < iter_max ) {  
    err=0.0;  
  
    #pragma acc kernels  
    {  
        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]));  
            }  
        }  
  
        for( int j = 1; j < n-1; j++) {  
            for( int i = 1; i < m-1; i++ ) {  
                A[j][i] = Anew[j][i];  
            }  
        }  
    }  
  
    iter++;  
}
```

Does the CPU need the data between these loop nests?

Does the CPU need the data between iterations of the convergence loop?

# OpenACC life cycle



# Express data movement

## Structured Data Regions

The **data** directive defines a region of code in which GPU arrays remain on the GPU and are shared among all kernels in that region.

```
#pragma acc data
{
    #pragma acc parallel loop
    ...
    #pragma acc parallel loop
    ...
}
```

} Data Region

Arrays used within the data region will remain on the GPU until the end of the data region.

# Express data movement (2)

## Data Clauses

<code>copy ( list )</code>	Allocates memory on GPU and copies data from host to GPU when entering region and copies data to the host when exiting region.
<code>copyin ( list )</code>	Allocates memory on GPU and copies data from host to GPU when entering region.
<code>copyout ( list )</code>	Allocates memory on GPU and copies data to the host when exiting region.
<code>create ( list )</code>	Allocates memory on GPU but does not copy.
<code>present ( list )</code>	Data is already present on GPU from another containing data region.
<code>deviceptr( list )</code>	The variable is a device pointer (e.g. CUDA) and can be used directly on the device.

# Express data movement (3)

## Array Shaping

Compiler sometimes cannot determine size of arrays

Must specify explicitly using data clauses and array “shape”

### C/C++

```
#pragma acc data copyin(a[0:nelem]) copyout(b[s/4:3*s/4])
```

### Fortran

```
!$acc data copyin(a(1:end)) copyout(b(s/4:3*s/4))
```

Note: data clauses can be used on **data**, **parallel**, or **kernels**

# Express data movement (4)

## Express Data Locality

```
while ( err > tol && iter < iter_max ) {
    err=0.0;
#pragma acc kernels
{
    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]));
        }
    }

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

# Express data movement (4)

## Express Data Locality

```
#pragma acc data copy(A) create(Anew)
while ( err > tol && iter < iter_max ) {
    err=0.0;
#pragma acc kernels
{
    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]));
        }
    }

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

Copy A to/from the accelerator only when needed.

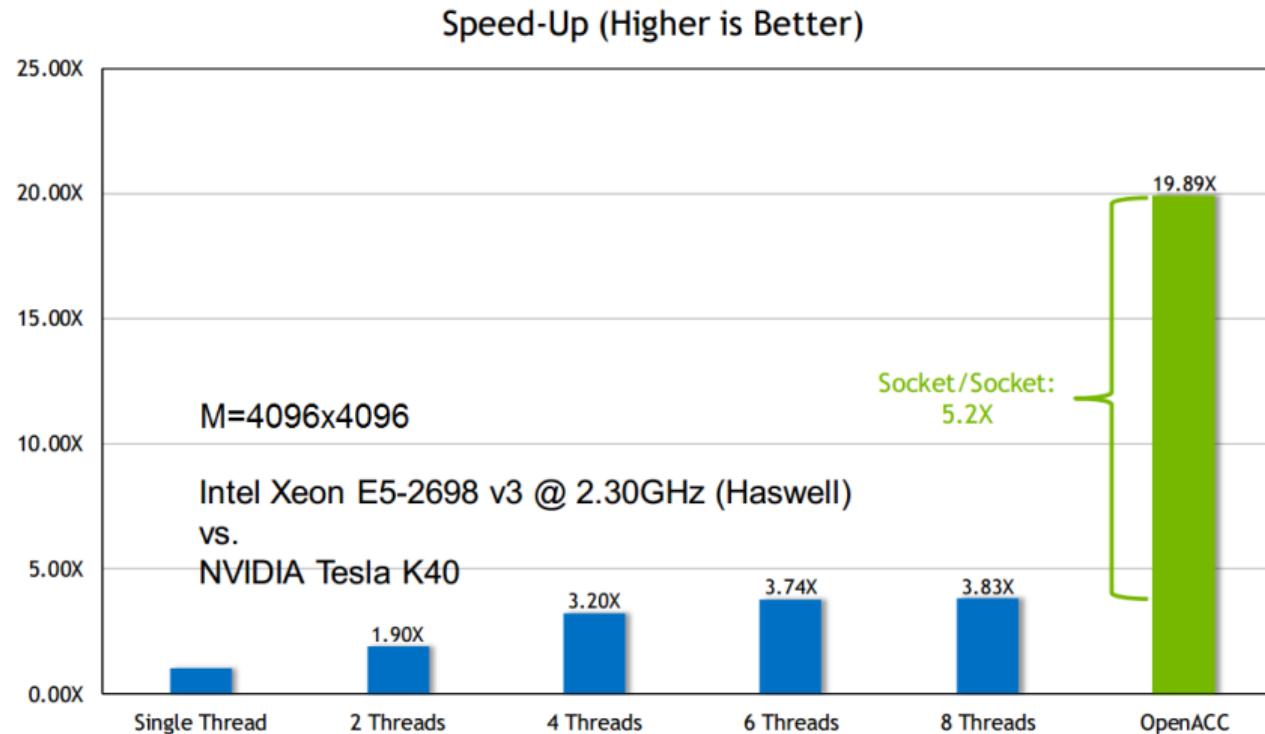
Create Anew as a device temporary.

## Express data movement (5)

# Rebuilding the code

```
$ pgcc -fast -acc -ta=tesla -Minfo=all laplace2d.c
main:
 40, Loop not fused: function call before adjacent loop
     Generated vector sse code for the loop
 51, Generating copy(A[:, :])
     Generating create(Anew[:, :])
     Loop not vectorized/parallelized: potential early exits
 56, Accelerator kernel generated
      56, Max reduction generated for error
      57, #pragma acc loop gang /* blockIdx.x */
      59, #pragma acc loop vector(256) /* threadIdx.x */
 56, Generating Tesla code
 59, Loop is parallelizable
 67, Accelerator kernel generated
      68, #pragma acc loop gang /* blockIdx.x */
      70, #pragma acc loop vector(256) /* threadIdx.x */
 67, Generating Tesla code
 70, Loop is parallelizable
```

# Express data movement (6)



# Express data movement (7)

## Unstructured Data Directives

Used to define data regions when scoping doesn't allow the use of normal data regions (e.g. the constructor/destructor of a class).

**enter data** Defines the start of an unstructured data lifetime

- clauses: `copyin(list)` , `create(list)`

**exit data** Defines the end of an unstructured data lifetime

- clauses: `copyout(list)` , `delete(list)`

```
#pragma acc enter data copyin(a)
...
#pragma acc exit data delete(a)
```

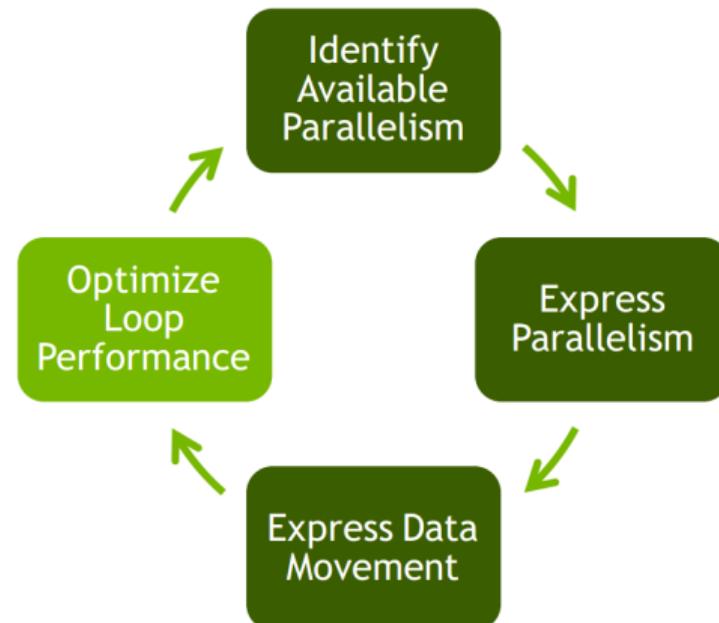
# Express data movement (8)

## Unstructured Data: C++ Classes

- ▶ Unstructured Data Regions enable OpenACC to be used in C++ classes
- ▶ Unstructured data regions can be used whenever data is allocated and initialized in a different scope than where it is freed (e.g. Fortran modules).

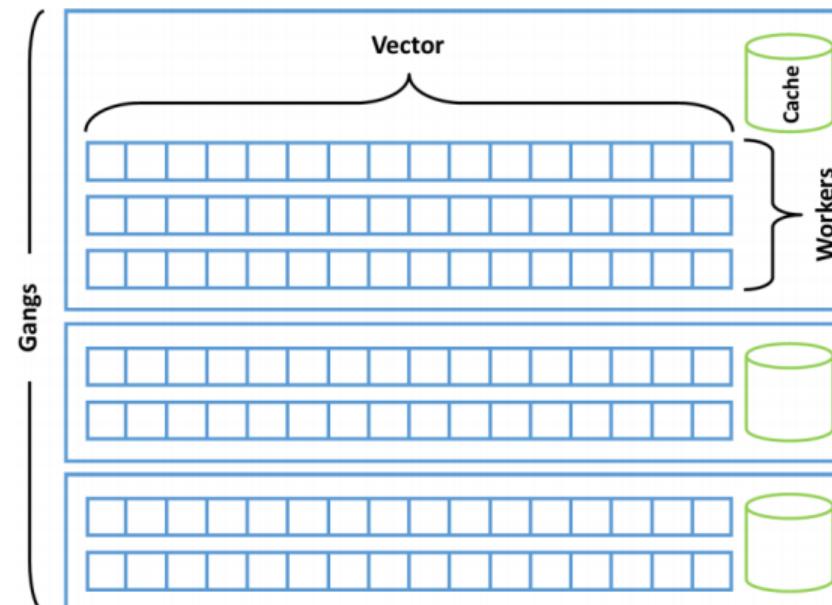
```
class Matrix {  
    Matrix(int n) {  
        len = n;  
        v = new double[len];  
#pragma acc enter data  
        create(v[0:len])  
    }  
    ~Matrix() {  
#pragma acc exit data  
        delete(v[0:len]);  
        delete[] v;  
    }  
  
    private:  
        double* v;  
        int len;  
};
```

# OpenACC life cycle



# Optimize loop performance

## Levels of Parallelism



# Optimize loop performance (2)

OpenACC	GPU
Vector	Thread
Worker	Warp
Gang	Thread block

# Optimize loop performance (3)

## The loop Directive

The **loop** directive gives the compiler additional information about the *next* loop in the source code through several clauses.

- **independent** - all iterations of the loop are independent
- **collapse (N)** - turn the next N loops into one, flattened loop
- **tile (N[ ,M,...])** - break the next 1 or more loops into *tiles* based on the provided dimensions.

# Optimize loop performance (4)

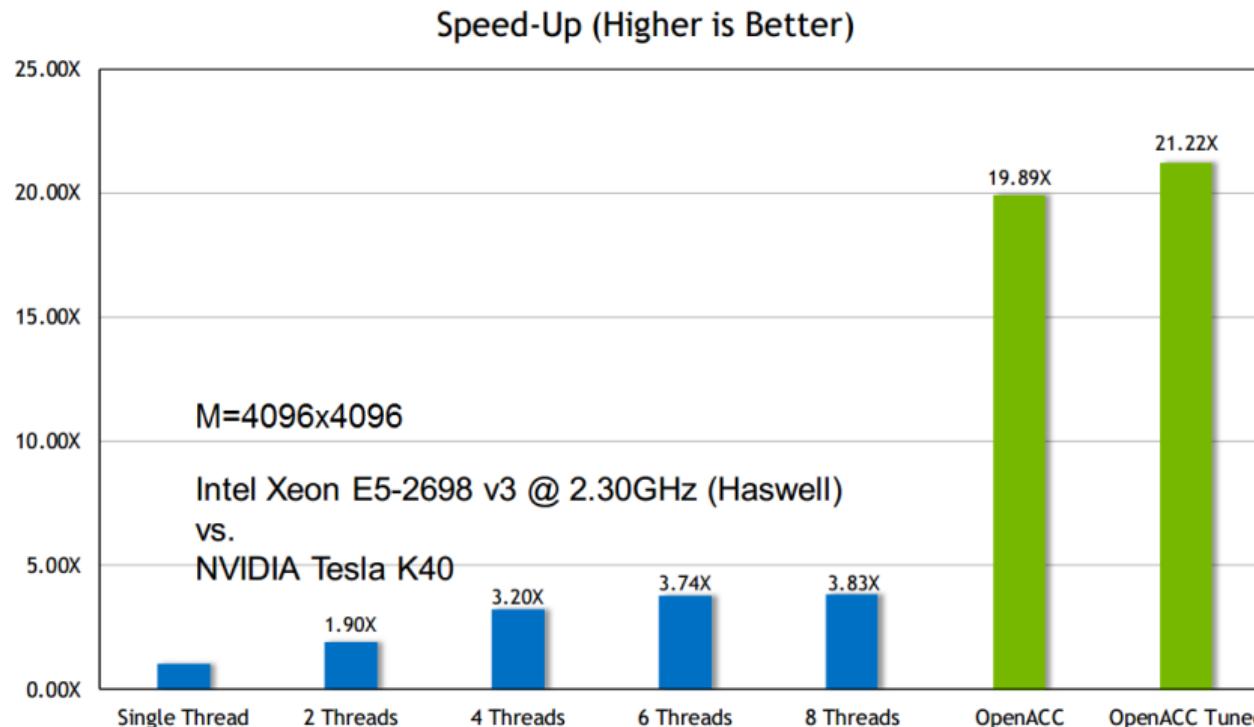
## Optimize Loop Performance

```
#pragma acc data copy(A) create(Anew)
while ( err > tol && iter < iter_max ) {
    err=0.0;
#pragma acc kernels
{
#pragma acc loop device_type(nvidia) tile(32,4)
    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 loop device_type(nvidia) tile(32,4)
    for( int j = 1; j < n-1; j++) {
        for( int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i];
        }
    }
}
iter++;
}
```

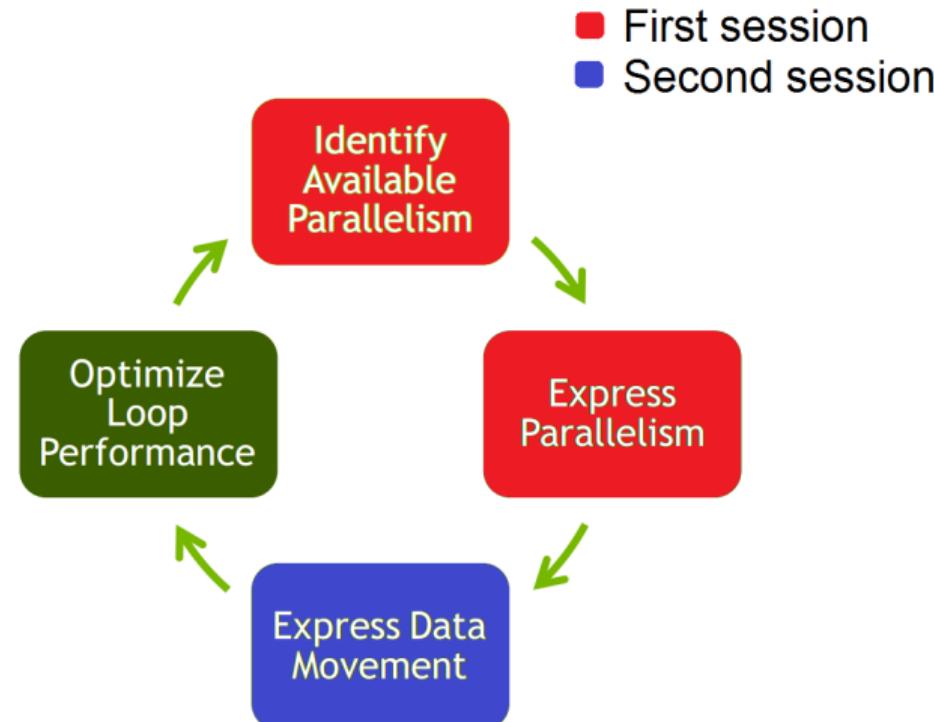
“Tile” the next two loops  
into 32x4 blocks, but  
only on NVIDIA GPUs.

# Optimize loop performance (5)



# Hands on session

# OpenACC life cycle



# Profiling tools

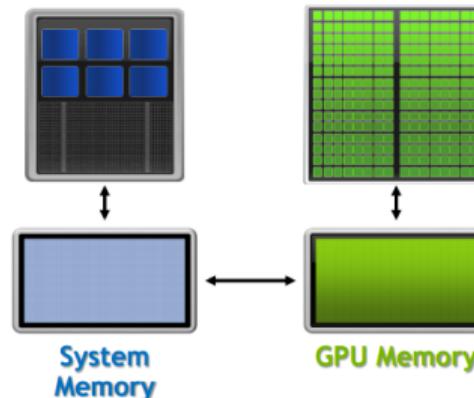
- A profiler allows to analyze the behaviour of a program
  - Duration of function calls
  - Performance Optimization
- Graphic profiling tools
  - Nvvp, pgprof, vampir, etc
- Command-line profiling tools
  - **nvprof**, gprof, etc

# CUDA Unified Memory

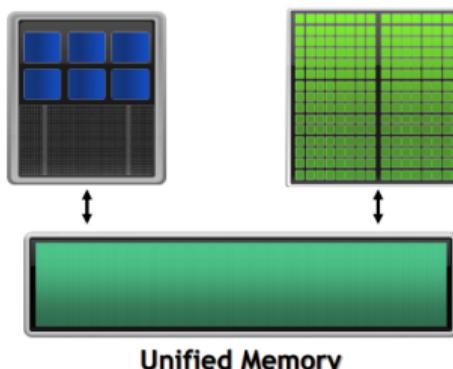
## CUDA Unified Memory Simplified Developer Effort

Sometimes referred to as  
“managed memory.”

Without Unified Memory



With Unified Memory



# Nvidia OpenACC course repository

- Log into cluster Kabré
- Pull repository CRHPCS-2018

```
1 cd CRHPCS-2018
2 git pull
```

- Load CUDA toolkit

```
1 module load cuda/9.0.176
```

- Load pgi compiler

```
1 module load pgi
```

# Profiling and parallelizing

- Access laboratory #2

```
1 cd openacc/lab2/c99/
```

- Open README file in browser

```
1 https://github.com/CNCA-CeNAT/CRHPCS-2018/blob/master/openacc/lab2/README.md
```

- Complete steps 0-3 (Send jobs to queue: k40)

Action	Command
Queue system	-qsub [jobName.pbs]
Check job status	watch -n 5 qstat -u USERNAME
Check GPU info	nvidia-smi

# Compiler flags

- PGI C compiler: pgcc
- PGI C++ compiler: pgc++

Flag	Action
-acc	Enable OpenACC directives
-fast	Choose optimal flags for target platform
-ta=[ <b>tesla:managed,multicore,etc</b> ]	Specify accelerator type
-Minfo=[ <b>accel,all,etc</b> ]	Show compilation information

# Optimizing data movement

- Access laboratory #3

```
1 cd openacc/lab3/c99/
```

- Open README file in browser

```
1 https://github.com/CNCA-CeNAT/CRHPCS-2018/blob/master/openacc/lab3/README.md
```

- Complete steps 0,1,2 and 4 (Send jobs to queue: k40)

## Best practices

# Optimization tips

- Use restrict keyword to avoid false loop dependencies (pointer aliasing)
- **collapse(N)**, useful when:
  - Many nested loops
  - Very small loops
- **tile(N[,M,...])**, useful when
  - high data locality
- Efficient **loop ordering**
  - Innermost loop iterates on fastest varying array dimension
  - Improve cache efficiency (access consecutive memory addresses)
- On **NVIDIA** devices:
  - vector lengths must be multiples of 32 (up to 1024)
  - (workers X vector) must be less than 1024

# Current limitations

- Shallow copy vs Deep copy

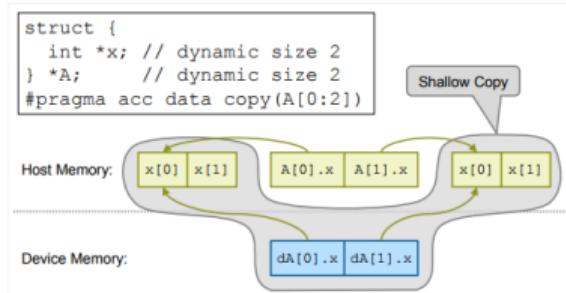


Figure 1. Shallow copy

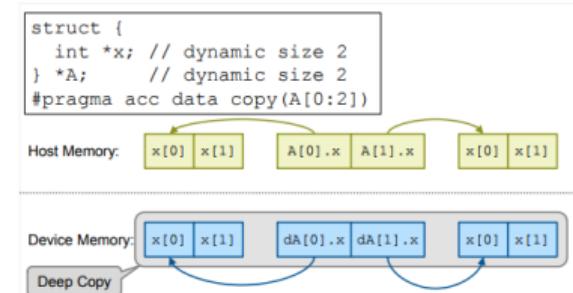


Figure 2. Deep copy

2

<sup>2</sup>Beyer, James, David Oehmke, and Jeff Sandoval. "Transferring user-defined types in OpenACC." Proceedings of Cray User Group (2014).

# Current limitations (2)

- Debugging is complicated
  - Unsupported use of print functions
- Limited use of dynamic memory in accelerated regions
- Some math library functions are still unsupported
- OpenACC still under development (Compiler Bugs)

The screenshot shows a web browser displaying the PGI User Forum. The URL in the address bar is [www.pgroup.com/userforum/viewforum.php?f=12&sid=f8319ae16f546d6eccddaa34621c9ee12](http://www.pgroup.com/userforum/viewforum.php?f=12&sid=f8319ae16f546d6eccddaa34621c9ee12). The page title is "PGI User Forum". The main navigation menu includes "WHAT'S NEW", "FEATURES", "DOWNLOADS", "SUPPORT", "PURCHASE", "ABOUT", and a green "GET PGI FOR FREE" button. Below the menu, there's a banner for "FREE OPENACC COURSE" with the tagline "More Science, Less Programming". The main content area is titled "Accelerator Programming". It features a "newtopic" button and a link to "PGI User Forum Forum Index > Accelerator Programming". A table lists several forum topics with columns for "Topics", "Replies", "Author", "Views", and "Last Post".

Topics	Replies	Author	Views	Last Post
create a static library for kernel. [ D Go to page: 1, 2 ]	8	eggbab123456	145	Sat Jan 20, 2018 1:01 am eggbab123456 ►D
Parallelizing stationary wavelet transform using OpenACC [ D Go to page: 1 ... 5, 6, 7 ]	32	jcastro9999	1753	Fri Jan 19, 2018 5:31 pm mkosog ►D
fortran module save attribute arrays and OpenACC	1	dshawul	27	Wed Jan 17, 2018 2:25 pm mkosog ►D
Serial execution on GPU	3	LO_UZH	2544	Wed Jan 17, 2018 12:51 pm mkosog ►D
Calling functions/methods in OpenAcc programme environment	1	kspan	58	Tue Jan 16, 2018 8:57 am mkosog ►D
Interpreting output generated by setting PGI_ACC_TIME=1	7	Nagevdyananathan	410	Mon Jan 08, 2018 9:47 am mkosog ►D

# Summary

- Minimize data movement
- Maximize compute intensity
- More explicit mapping of parallelism, less portable code
  - Use **device type** clause for architecture-specific optimizations
  - nvidia, radeon, xeonphi
- When using OpenACC:
  - Measure sequential performance
  - Understand program structure and data movement
  - Find hot-spots (profiler: gprof, nvvp)
  - Ensure safe parallelism

# OpenACC material

## Where to find help

- OpenACC Course Recordings - <https://developer.nvidia.com/openacc-course>
- OpenACC on StackOverflow - <http://stackoverflow.com/questions/tagged/openacc>
- OpenACC Toolkit - <http://developer.nvidia.com/openacc>

### Additional Resources:

- Parallel Forall Blog - <http://devblogs.nvidia.com/parallelforall/>
- GPU Technology Conference - <http://www.gputechconf.com/>
- OpenACC Website - <http://openacc.org/>

# Acknowledgements



Thank you!

- Lecture notes by Jeff Larkin, NVIDIA Developer Technologies
- Lecture notes by Esteban Meneses, CNCA