PRACE Workshop: Deep Learning and GPU programming workshop

7 – 10 September 2020

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

Tentative Agenda Day 2: **Fundamentals of Accelerated Computing with OpenACC**

10:00-12:00 Intro and Profiling

12:00-13:00 Lunch Break

13:00-14:20 OpenACC Directives

14:20-14:30 Coffee Break

14:30-15:45 GPU Programming and Data Management

15:45-16:00 Q&A, Final Remarks

All times are in EEST=CEST+1!

MODULE ONE: INTRODUCTION

Dr. Volker Weinberg | LRZ | 08.09.2020

OpenACC
More Science, Less Programming

lrz

MODULE OVERVIEW

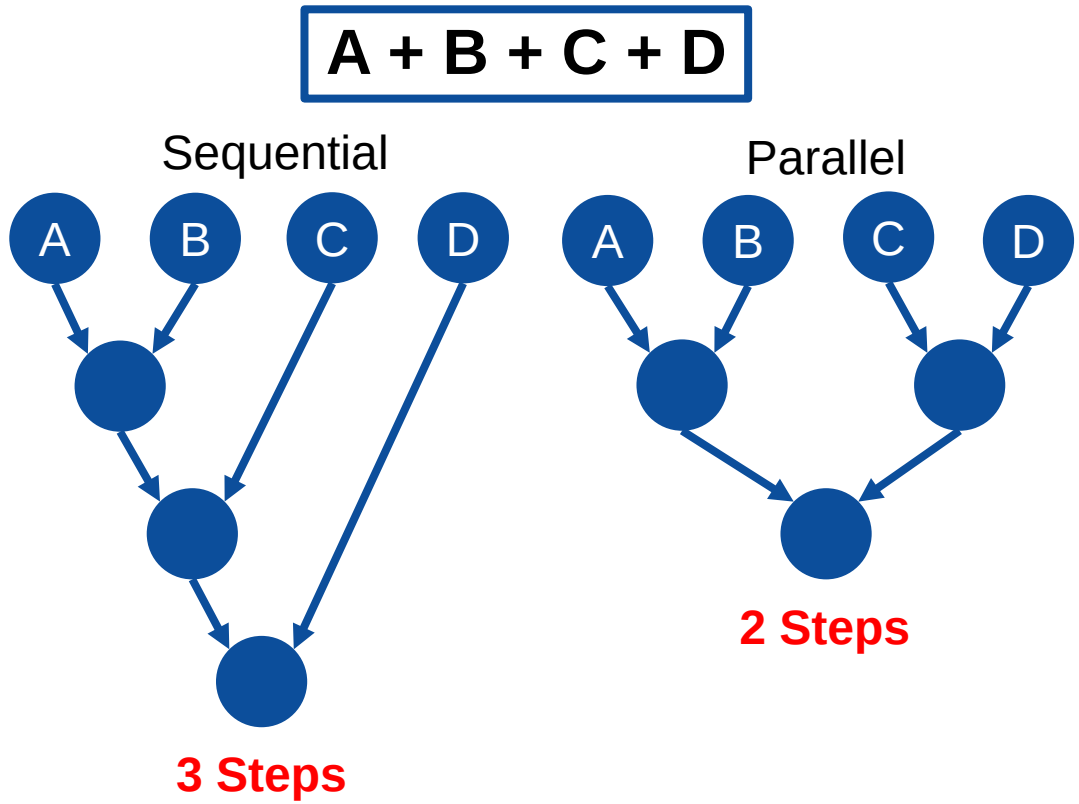
Topics to be covered

- Introduction to parallel programming
- Common difficulties in parallel programming
- Introduction to OpenACC
- Parallel programming in OpenACC

INTRODUCTION TO PARALLEL PROGRAMMING

WHAT IS PARALLEL PROGRAMMING?

- **“Performance Programming”**
- Parallel programming involves exposing an algorithm’s ability to execute in parallel
- This may involve breaking a large operation into smaller tasks (task parallelism)
- Or doing the same operation on multiple data elements (data parallelism)
- Parallel execution enables better performance on modern hardware

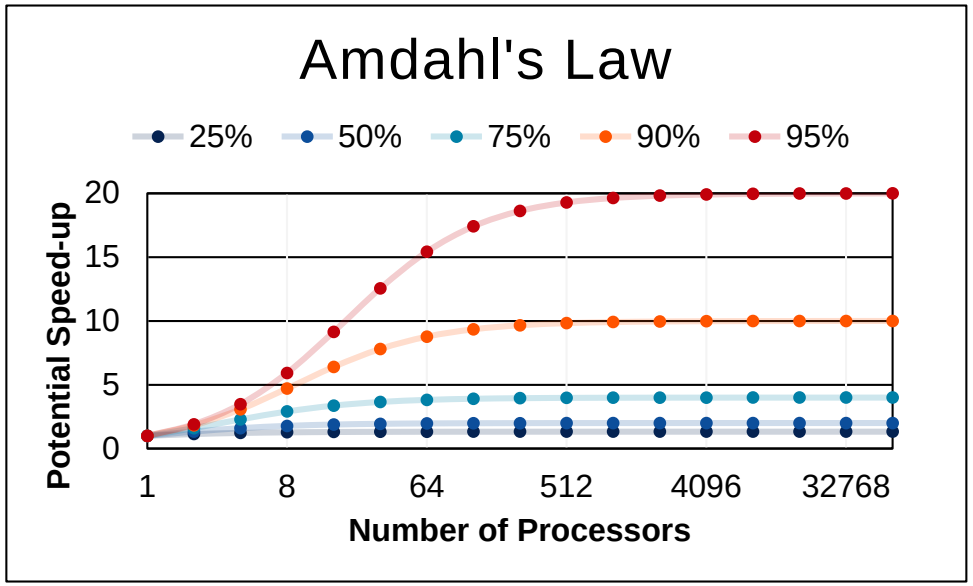


AMDAHL'S LAW

AMDAHL'S LAW

Serialization Limits Performance

- Amdahl's law is an observation that how much speed-up you get from parallelizing the code is limited by the remaining serial part.
- Any remaining serial code will reduce the possible speed-up
- This is why it's important to focus on parallelizing the most time consuming parts, not just the easiest.



APPLYING AMDAHL'S LAW

Estimating Potential Speed-up

- What's the maximum speed-up that can be obtained by parallelizing 50% of the code?

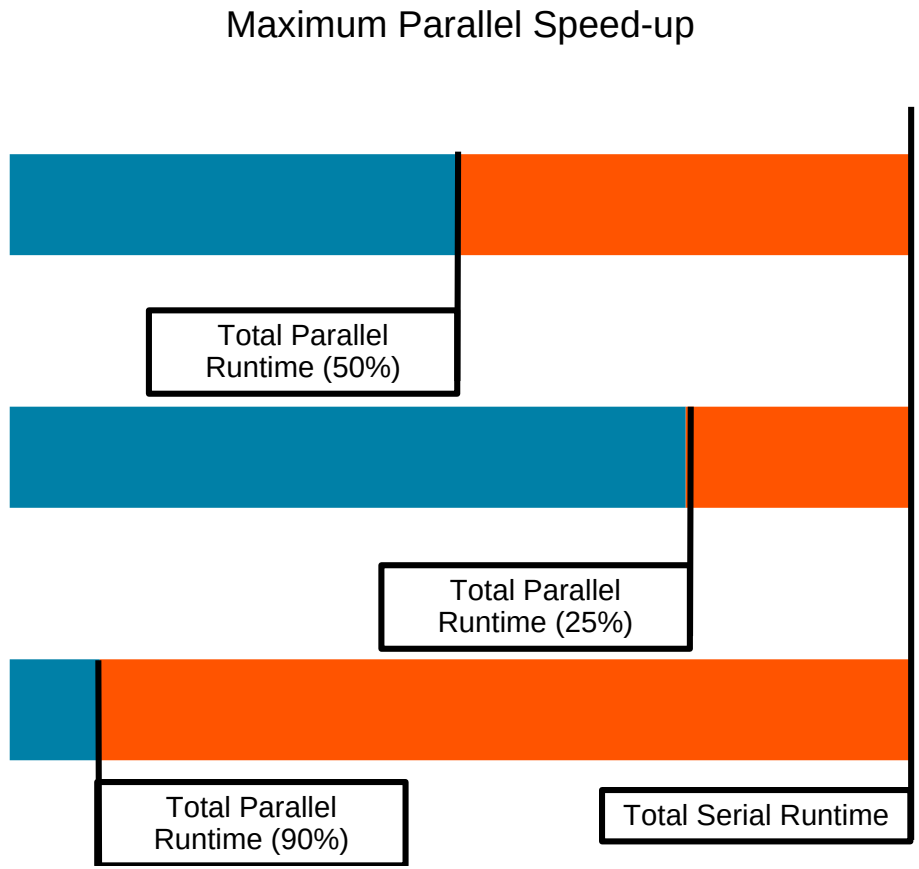
1 / (100% - 50%) = 1 / (1.0 - 0.50) = 2.0X

- What's the maximum speed-up that can be obtained by parallelizing 25% of the code?

1 / (100% - 25%) = 1 / (1.0 - 0.25) = 1.3X

- What's the maximum speed-up that can be obtained by parallelizing 90% of the code?

1 / (100% - 90%) = 1 / (1.0 - 0.90) = 10.0X



INTRODUCTION TO OPENACC

OpenACC is a directives-based programming approach to **parallel computing** designed for **performance** and **portability** on CPUs and GPUs for HPC.

Add Simple Compiler Directive

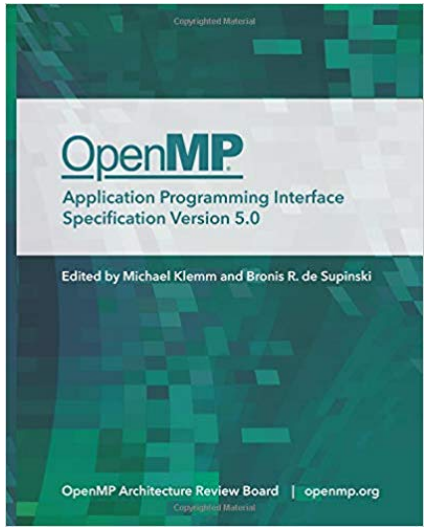
```
main()
 {
  <serial code>
  #pragma acc kernels
  {
    <parallel code>
  }
}
```

STANDARDS-BASED PARALLELISM

MPI standard

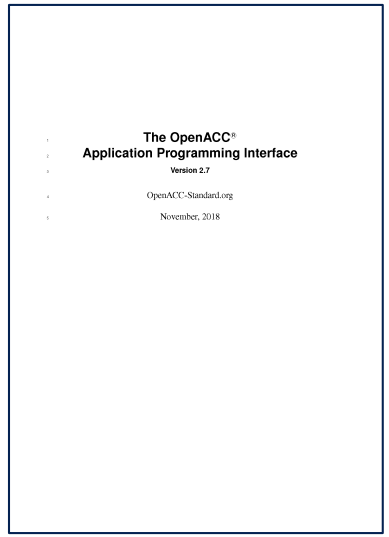
https://www.mpi-forum.org/docs/

OpenMP standard



https://www.openmp.org/specifications/

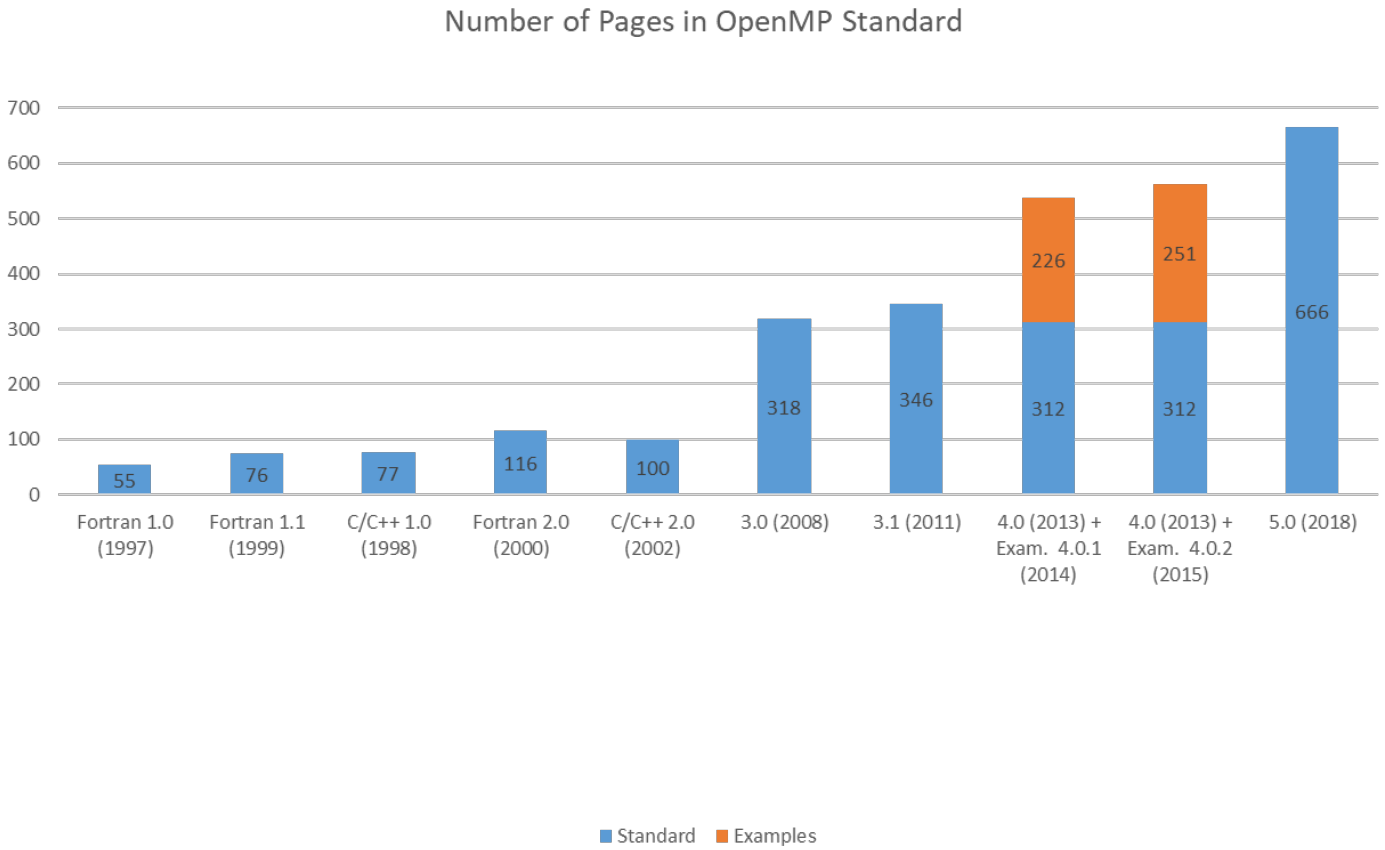
OpenACC standard



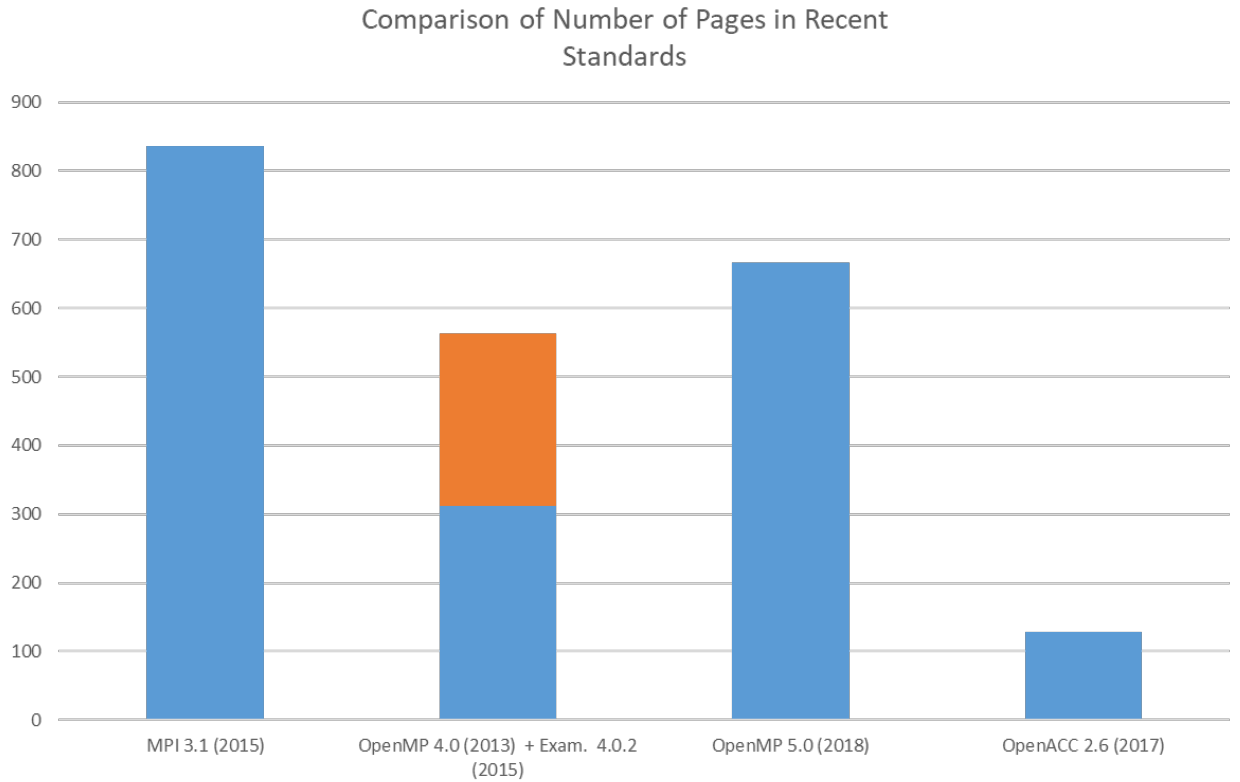
https://www.openacc.org/specification

OpenACC
More Science, Less Programming

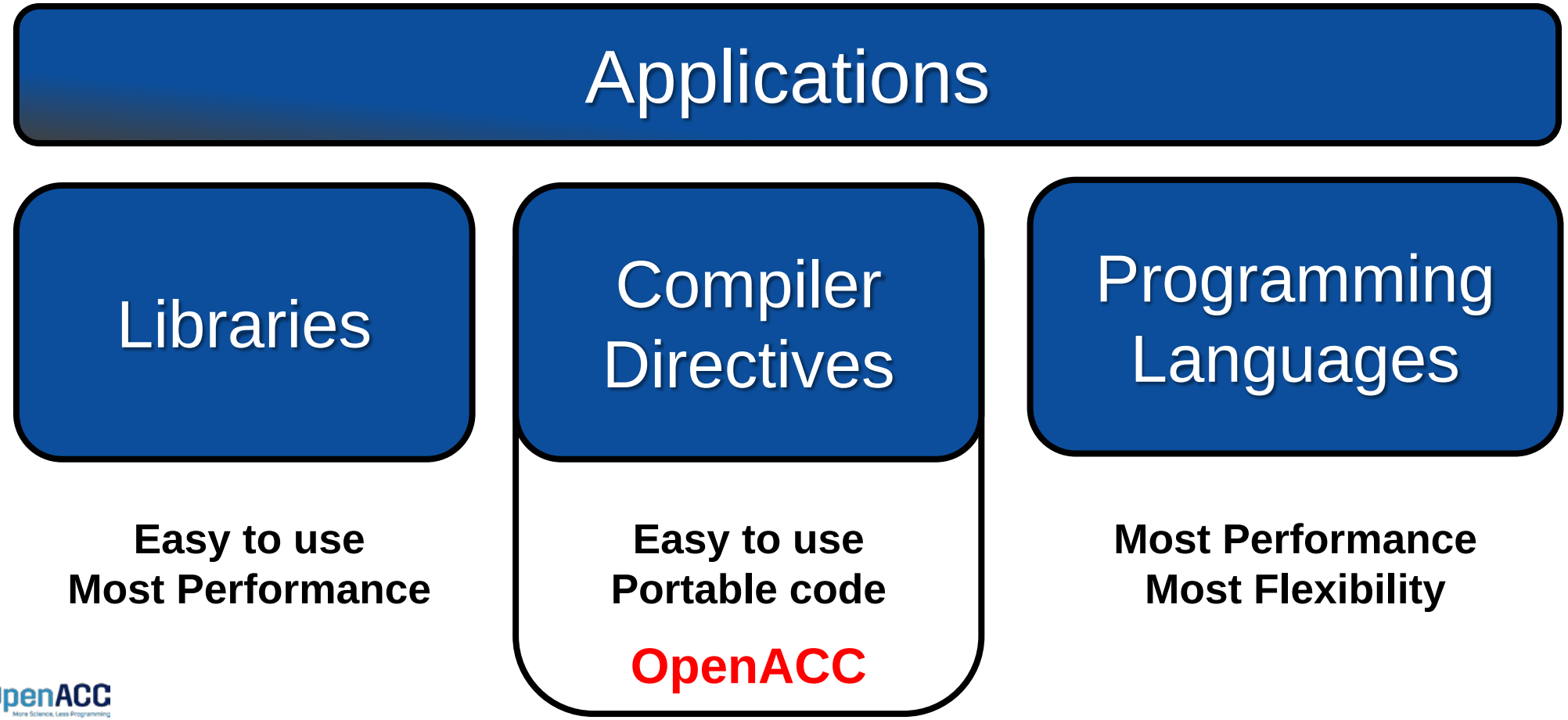
DEVELOPMENT OF OPENMP STANDARD



COMPLEXITY OF RECENT STANDARDS



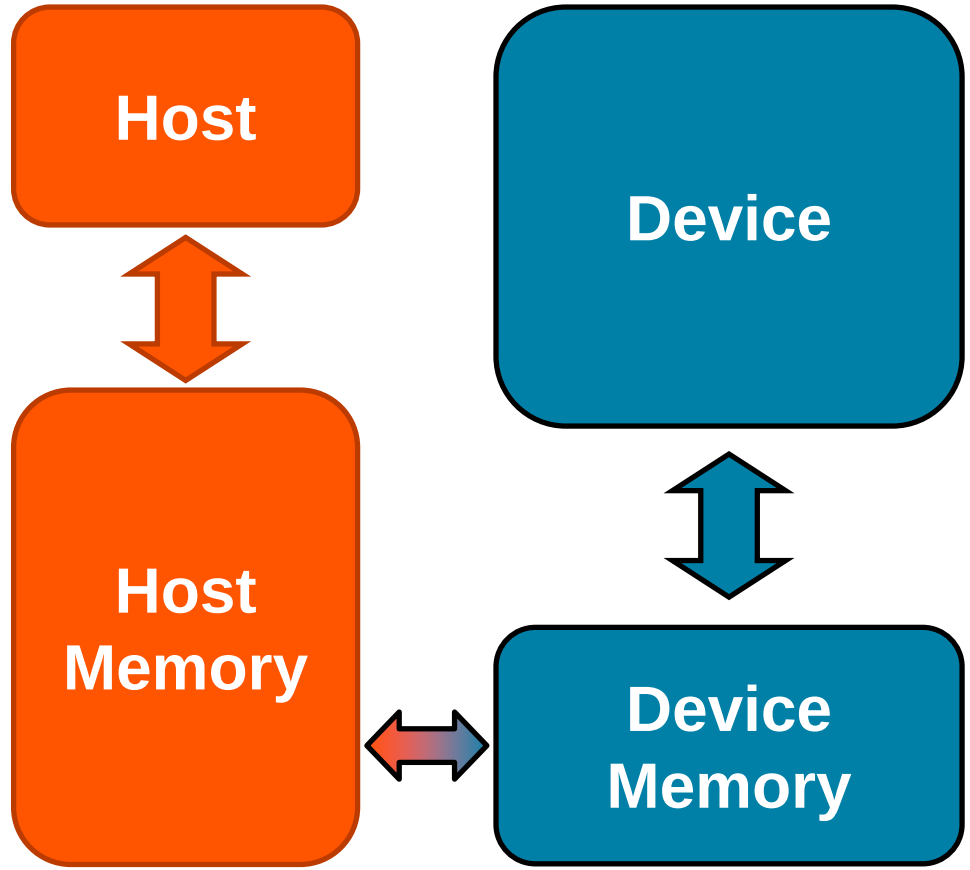
3 WAYS TO ACCELERATE APPLICATIONS



OPENACC PORTABILITY

Describing a generic parallel machine

- OpenACC is designed to be portable to many existing and future parallel platforms
- The programmer need not think about specific hardware details, but rather express the parallelism in generic terms
- An OpenACC program runs on a *host* (typically a CPU) that manages one or more parallel *devices* (GPUs, etc.). The host and device(s) are logically thought of as having separate memories.



OPENACC

Three major strengths

Incremental	Single Source	Low Learning Curve

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 correct behavior, remove/alter OpenACC code as needed.

OpenACC
More Science, Less Programming

OPENACC

Incremental

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

Single Source

Low Learning Curve

OpenACC
More Science, Less Programming

OPENACC

Supported Platforms

POWER

Sunway

x86 CPU

x86 Xeon Phi

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(){

...

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

}
```

OpenACC
More Science, Less Programming

OPENACC

Incremental

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

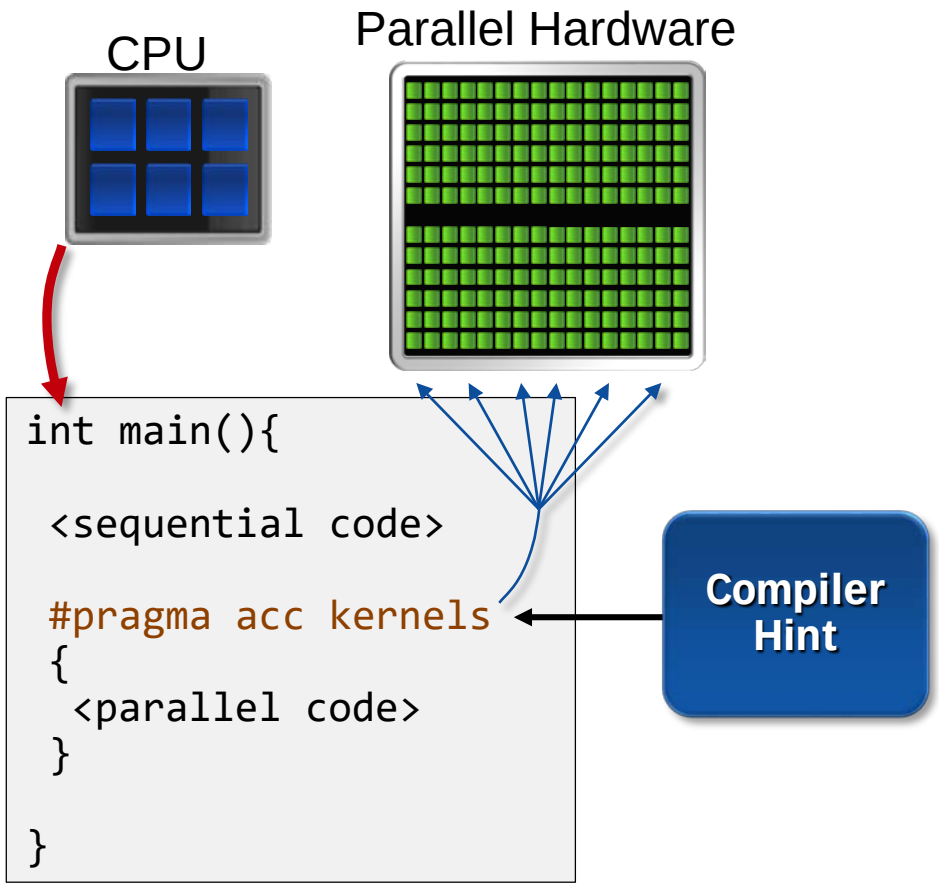
Single Source

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

Low Learning Curve

OpenACC
More Science, Less Programming

OPENACC



The programmer will give hints to the compiler about which parts of the code to parallelize.

The compiler will then generate parallelism for the target parallel hardware.

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

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

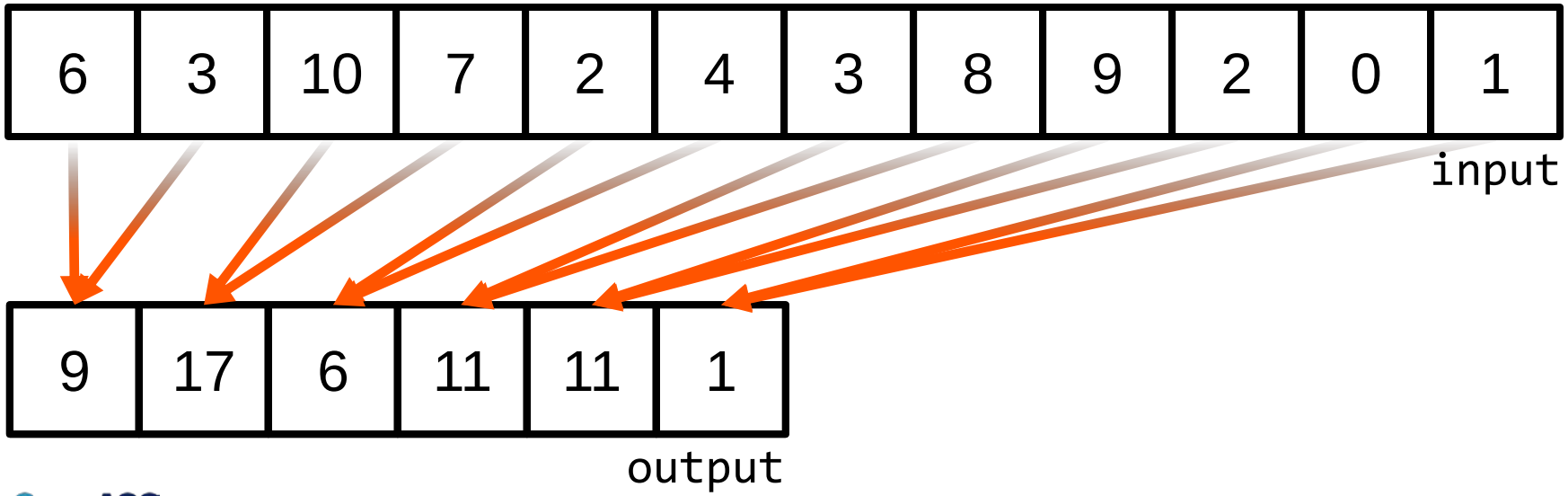
- OpenACC is meant to be easy to use, and easy to learn
- Programmer remains in familiar C, C++, or Fortran
- No reason to learn low-level details of the hardware.

EXPRESSING PARALLELISM WITH OPENACC

CODING WITH OPENACC

Array pairing example- serial

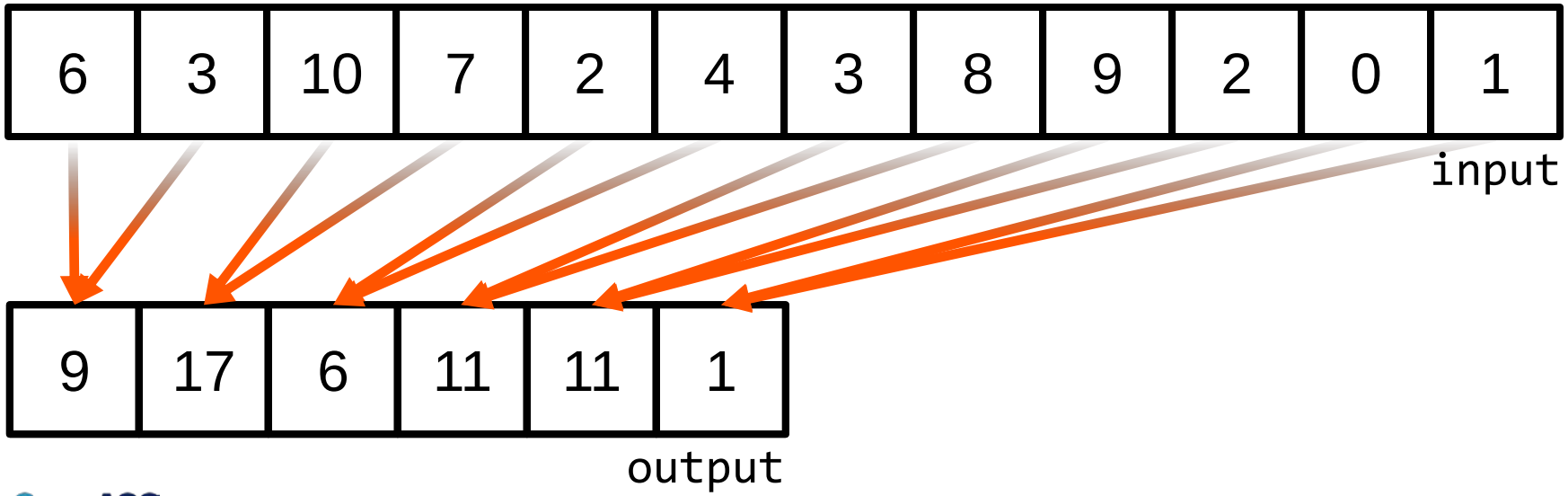
```
void pairing(int *input, int *output, int N){

  for(int i = 0; i < N; i++)
    output[i] = input[i*2] + input[i*2+1];
}
```



CODING WITH OPENACC

Array pairing example - parallel

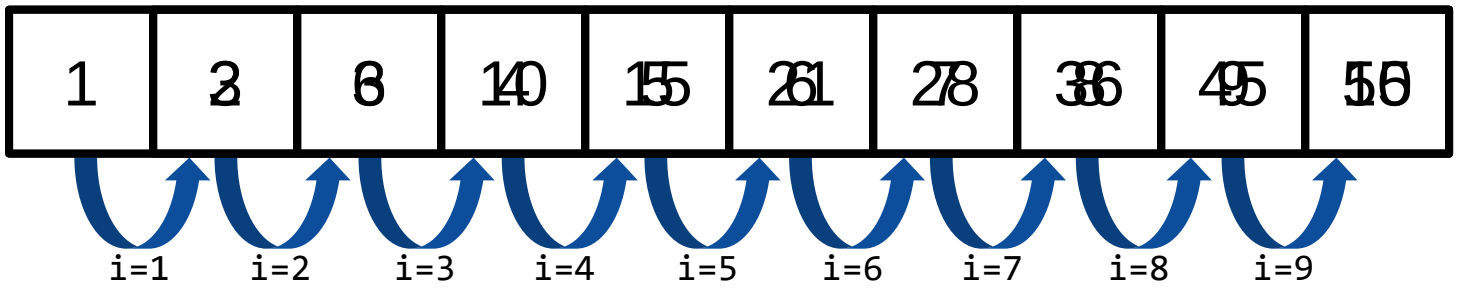
```
void pairing(int *input, int *output, int N){
  #pragma acc parallel loop
  for(int i = 0; i < N; i++)
    output[i] = input[i*2] + input[i*2+1];
}
```



DATA DEPENDENCIES

Not all loops are parallel

```
void pairing(int *a, int N){

  for(int i = 1; i < N; i++)
    a[i] = a[i] + a[i-1];
}
```

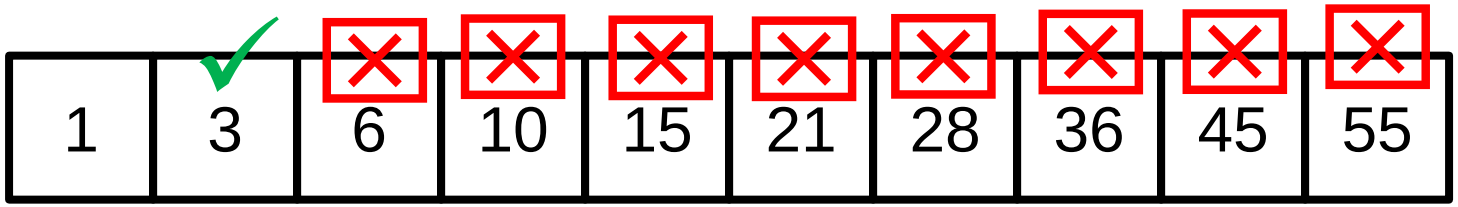


DATA DEPENDENCIES

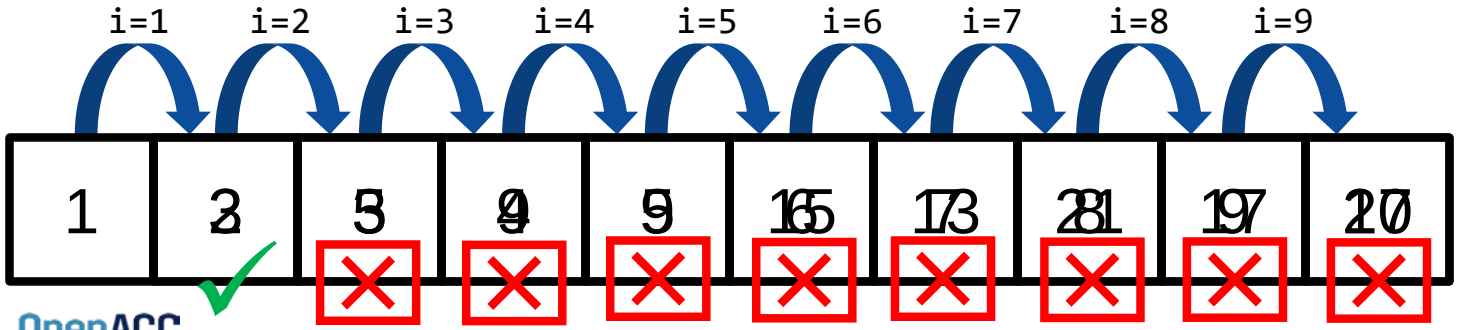
Not all loops are parallel

```
void pairing(int *a, int N){
  #pragma acc parallel loop
  for(int i = 1; i < N; i++)
    a[i] = a[i] + a[i-1];
}
```

If we attempted to parallelize this loop we would get wrong answers due to a *forward dependency.*



Sequential



Parallel

OpenACC
More Science, Less Programming

MODULE 1 REVIEW

CLOSING SUMMARY

Module One: Introduction

- Parallel programming is a technique of utilizing modern hardware to do lots of work all at once.
- Amdahl's law is the gravity of parallel programming, break this law at your own peril.
- Not all loops are parallel, but often can be rewritten to be parallelizable
- OpenACC is a high level model for generating parallel code from serial loops

OPENACC RESOURCES

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

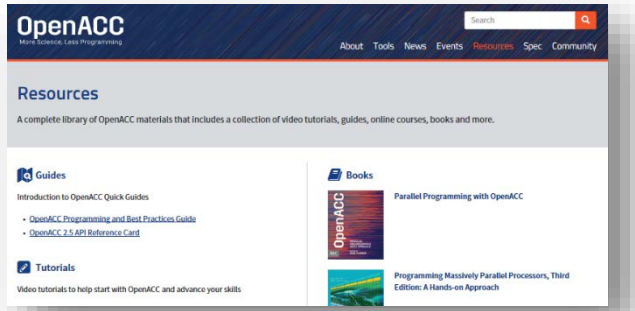
FREE Compilers

slack

https://www.openacc.org/community#slack

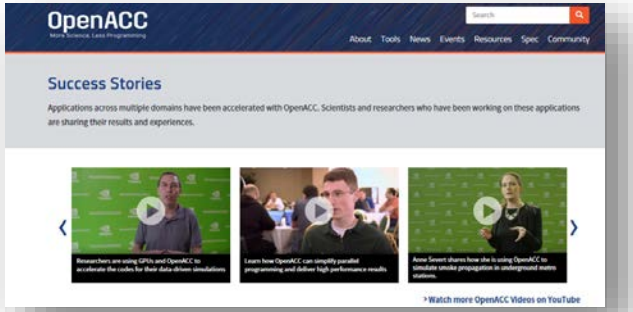
Resources

https://www.openacc.org/resources



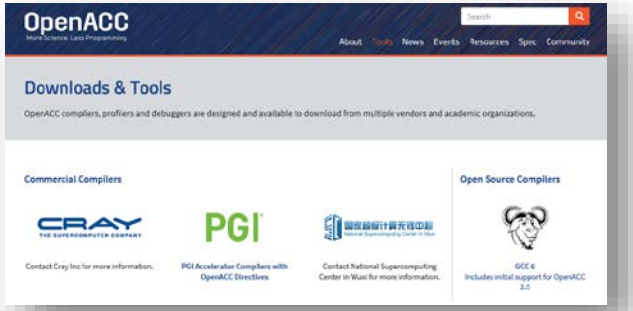
Success Stories

https://www.openacc.org/success-stories



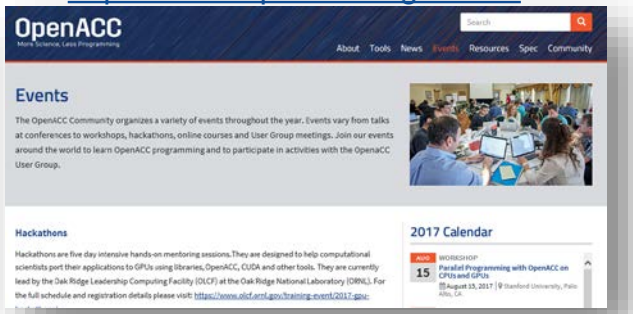
Compilers and Tools

https://www.openacc.org/tools



Events

https://www.openacc.org/events



THANK YOU

OpenACC
More Science, Less Programming