MODULE ONE: INTRODUCTION

Dr. Volker Weinberg | LRZ

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

OpenACC
More Science, Less Programming

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

$$A + B + C + D$$

Sequential

Parallel

3 Steps

2 Steps

OpenACC
More Science, Less Programming
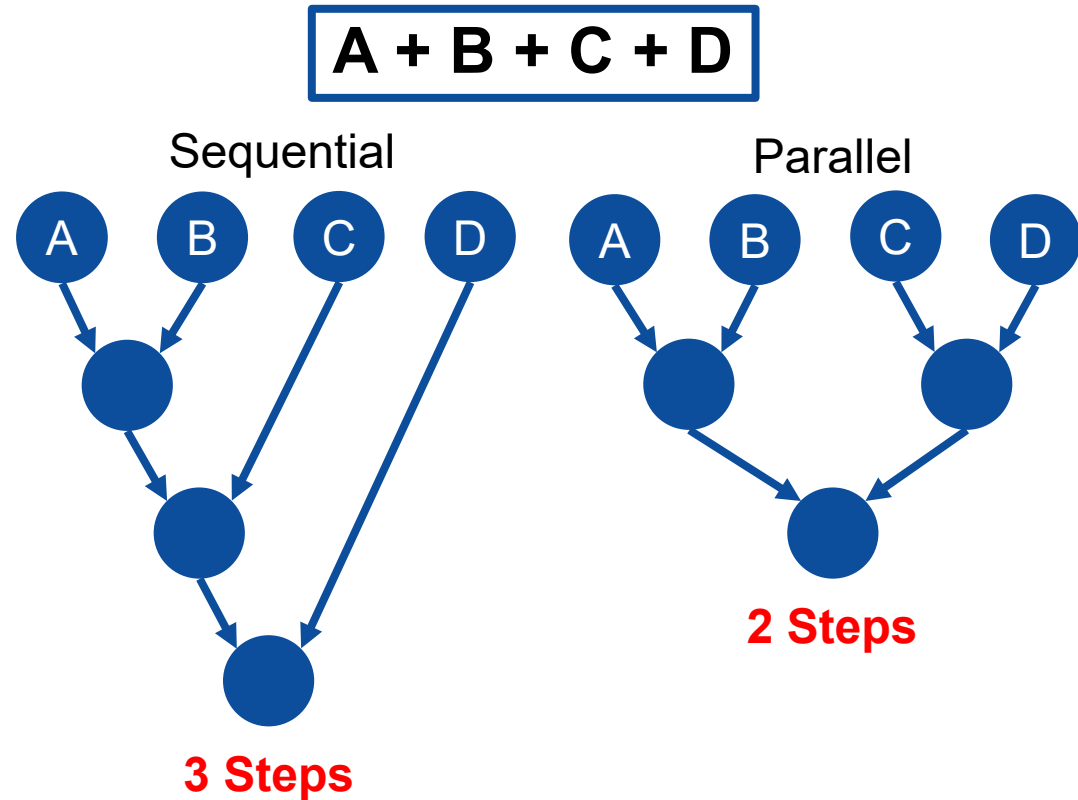
AMDAHL'S LAW

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.

OpenACC
More Science, Less Programming
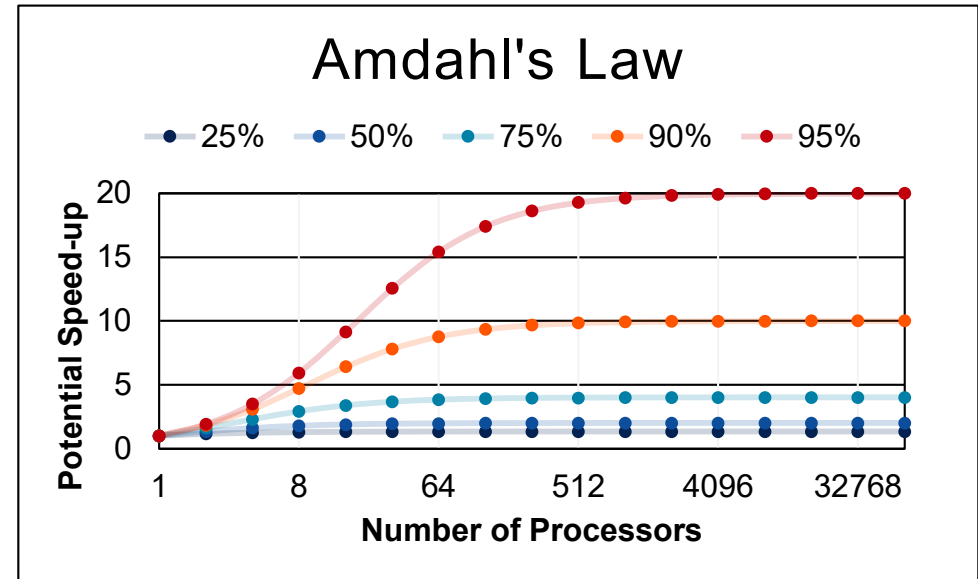
APPLYING AMDAHL'S LAW

Estimating Potential Speed-up

Maximum Parallel Speed-up

- What's the maximum speed-up that can be obtained by parallelizing 50% of the code?
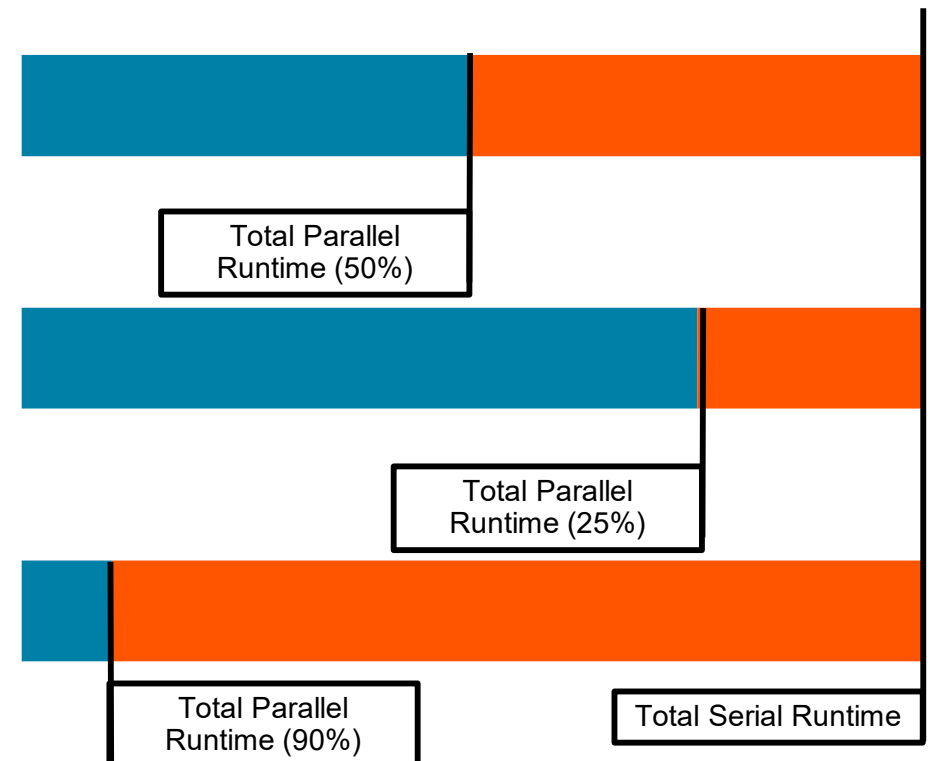
  1 / (100% - 50%) = 1 / (1.0 - 0.50) = 2.0X

- 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

Total Parallel Runtime (50%)

Total Parallel Runtime (25%)

Total Parallel Runtime (90%)

Total Serial Runtime

OpenACC
More Science, Less Programming

INTRODUCTION TO OPENACC

OpenACC
More Science, Less Programming

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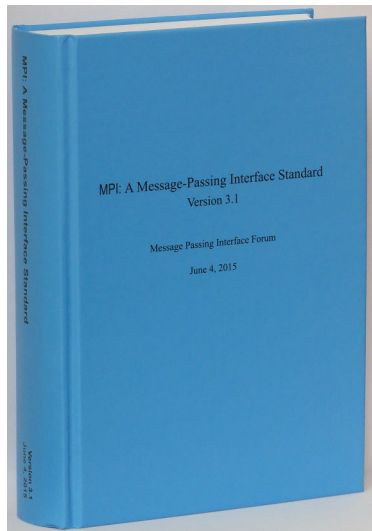
Add Simple Compiler Directive

```
main()
 {
  <serial code>
  #pragma acc kernels
  {
    <parallel code>
  }
 }
```
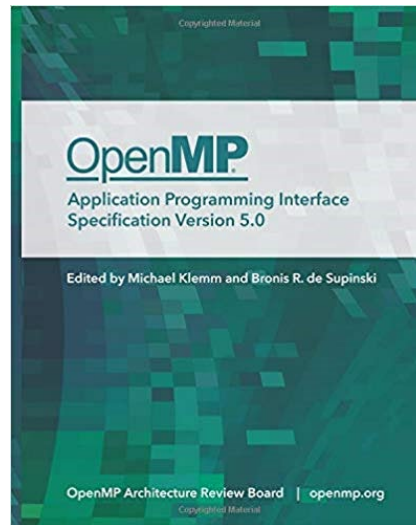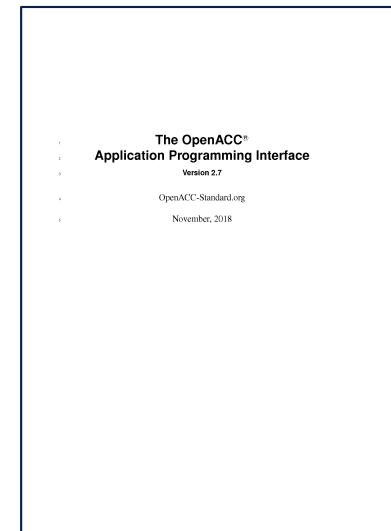
OpenACC
More Science, Less Programming

STANDARDS-BASED PARALLELISM

MPI standard

OpenMP standard

OpenACC standard

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

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

https://www.openacc.org/specification

DEVELOPMENT OF OPENMP STANDARD

Number of Pages in OpenMP Standard
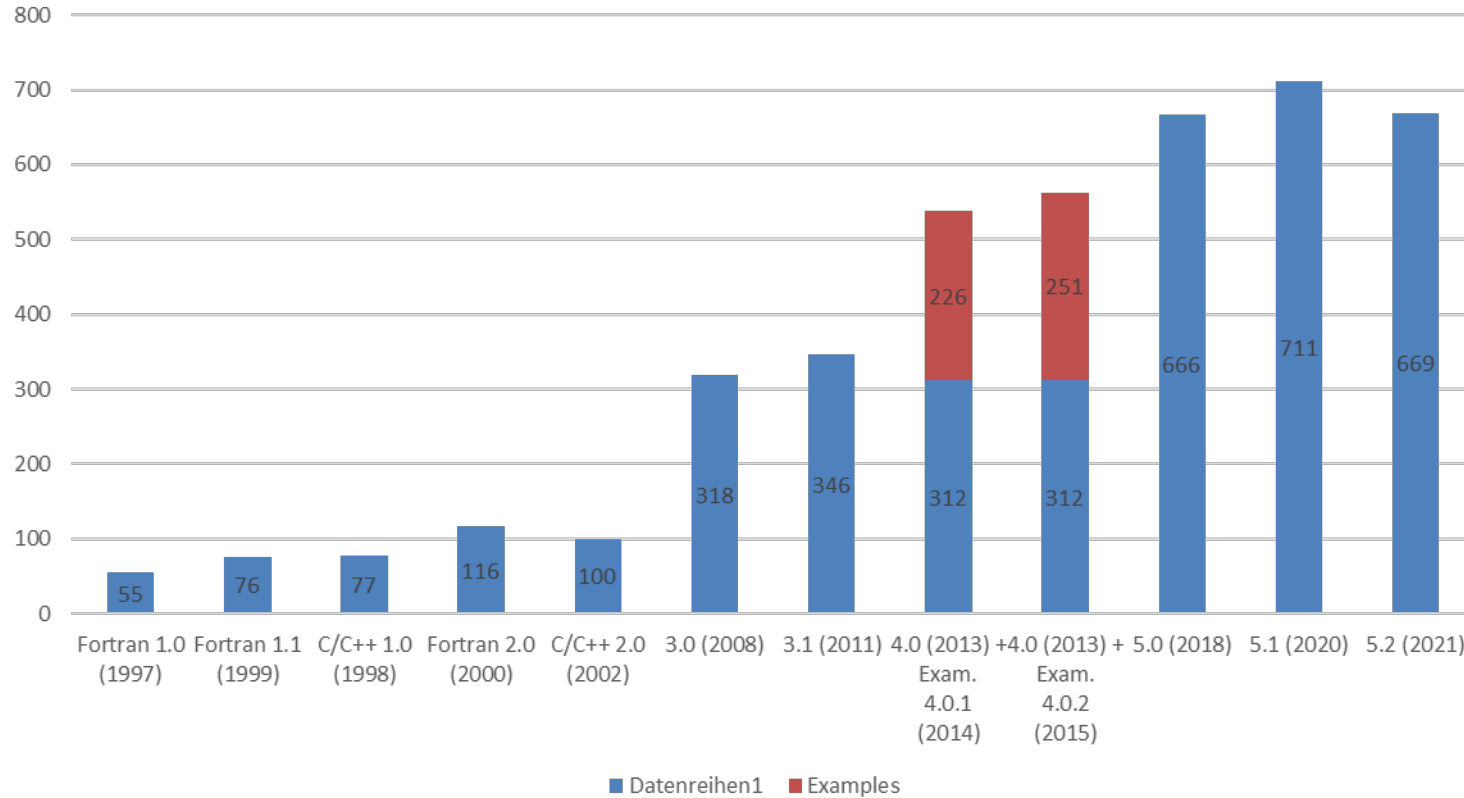
COMPLEXITY OF RECENT STANDARDS

Comparison of Number of Pages in Recent Standards
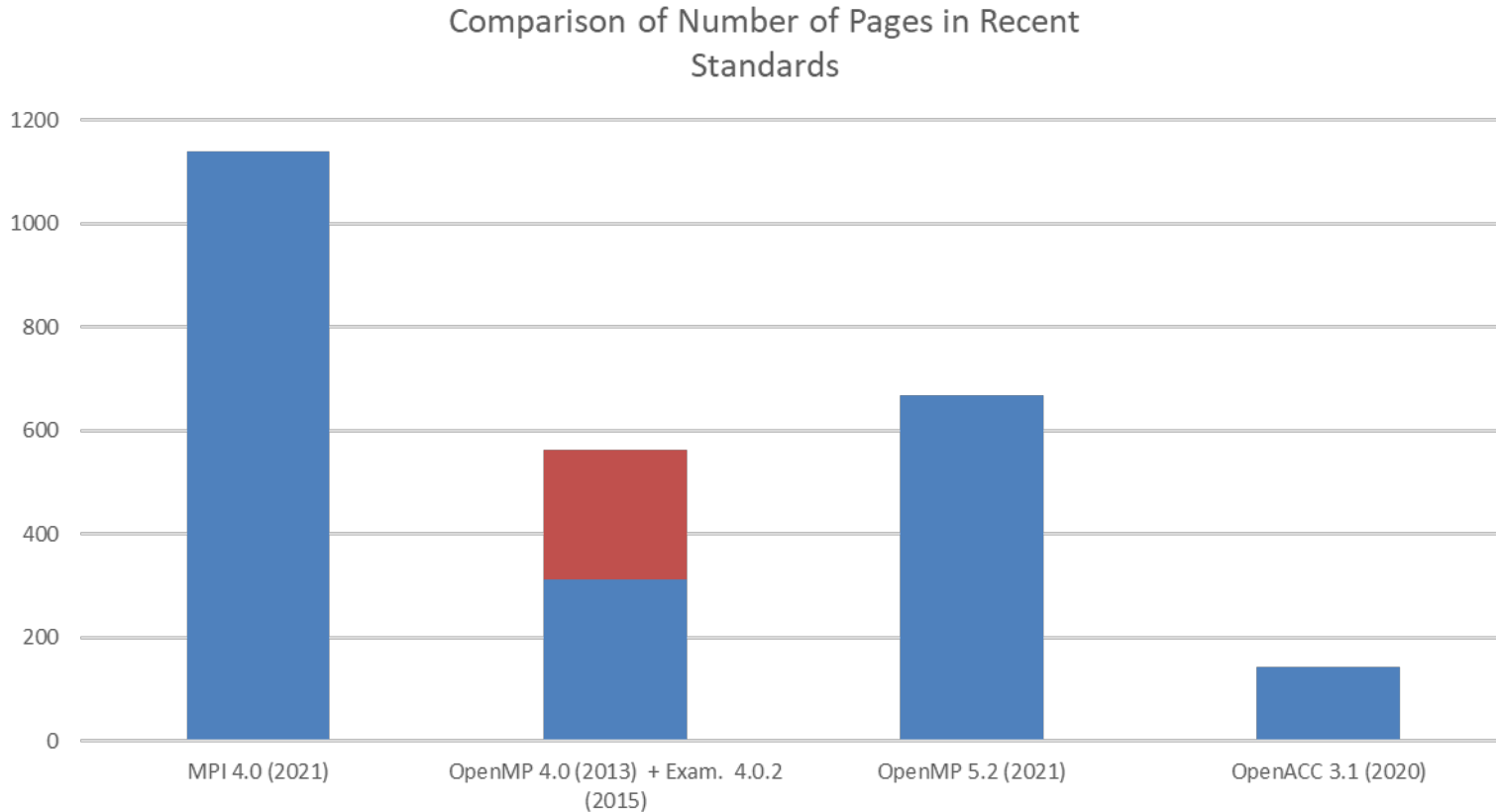
3 WAYS TO ACCELERATE APPLICATIONS

Applications

Libraries

Compiler Directives

Programming Languages

**Easy to use
Most Performance**

**Easy to use
Portable code**

OpenACC

**Most Performance
Most Flexibility**

OpenACC
More Science, Less Programming

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.

Host

Device

Host Memory

Device Memory
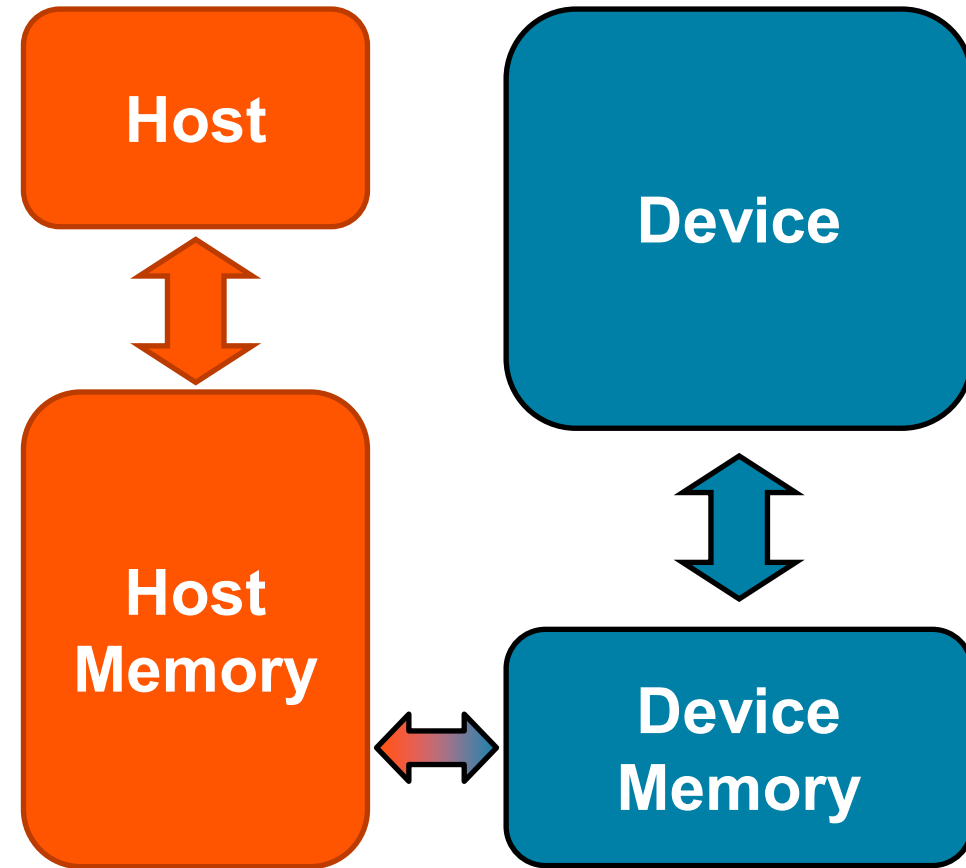
OPENACC

Three major strengths

Incremental	Single Source	Low Learning Curve

OpenACC
More Science, Less Programming

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 >

}
```
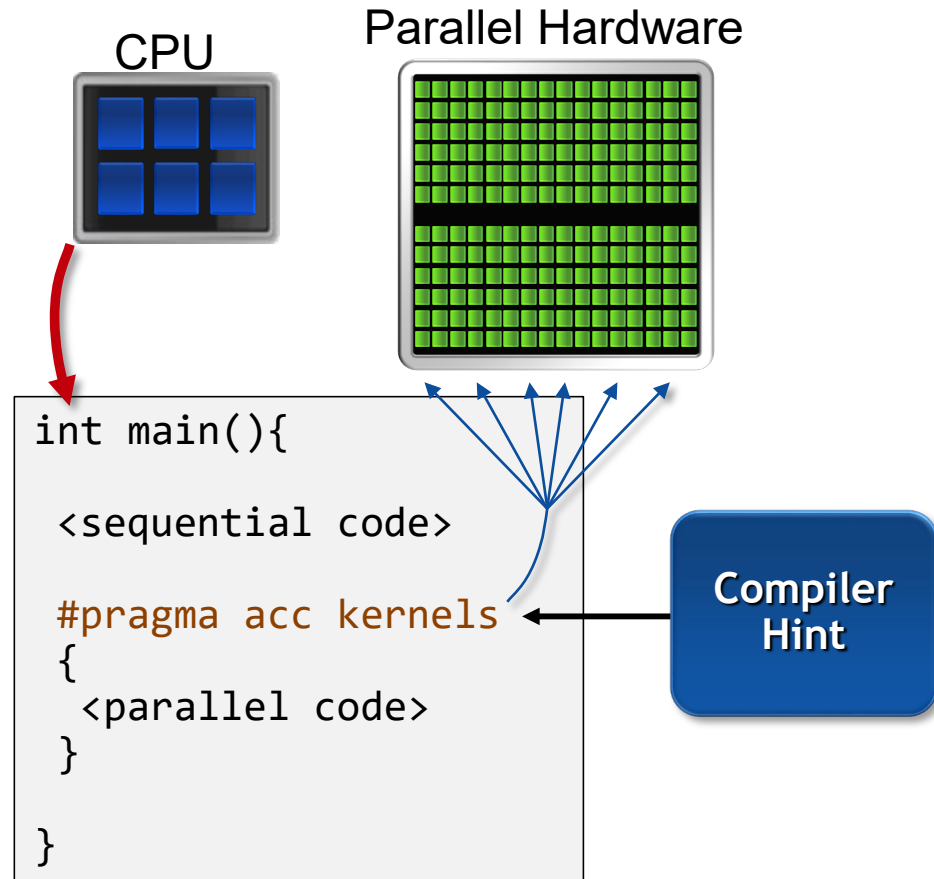
OpenACC
More Science, Less Programming

OPENACC

Incremental	Single Source	Low Learning Curve
▪ Maintain existing sequential code ▪ Add annotations to expose parallelism ▪ After verifying correctness, annotate more of the code	▪ Rebuild the same code on multiple architectures ▪ Compiler determines how to parallelize for the desired machine ▪ Sequential code is maintained	

OpenACC
More Science, Less Programming

OPENACC

CPU

Parallel Hardware

```
int main(){

 <sequential code>

 #pragma acc kernels
 {
  <parallel code>
 }

}
```

Compiler Hint

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
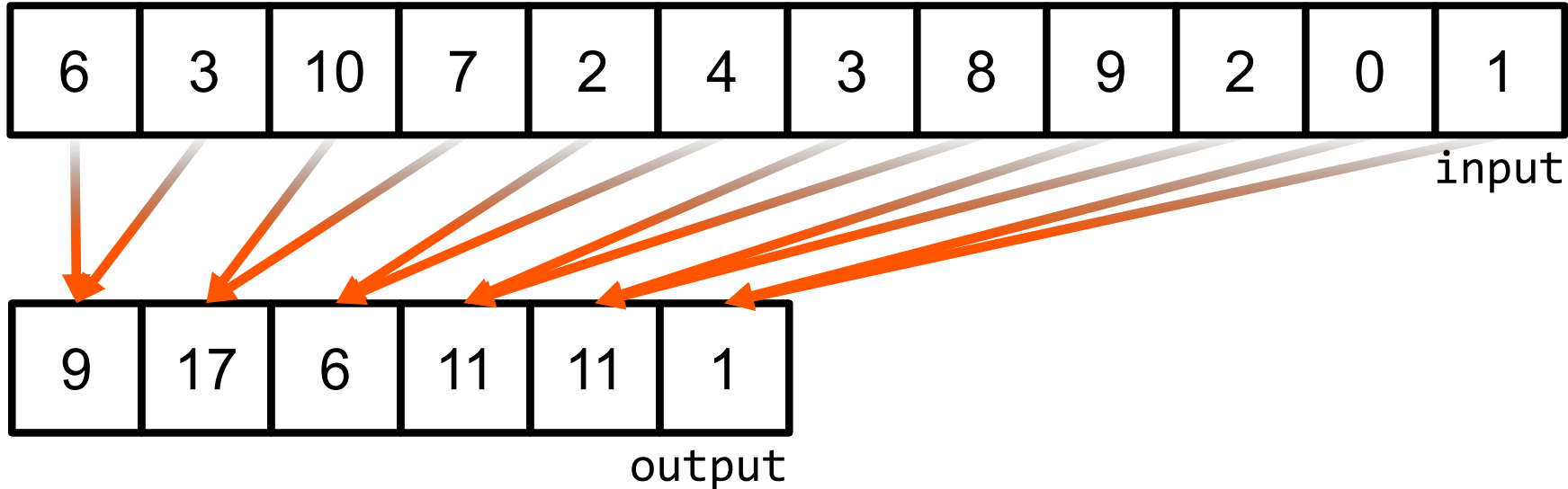- Sequential code is maintained

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.

OpenACC
More Science, Less Programming
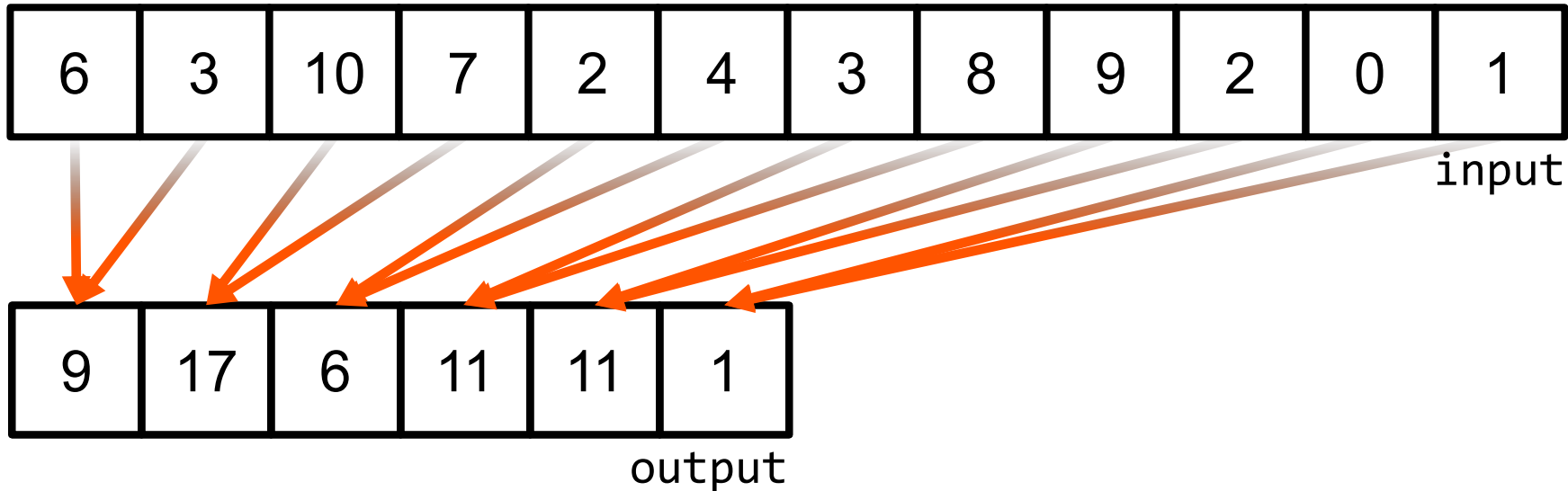
CODING WITH OPENACC

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];
}
```


6	3	10	7	2	4	3	8	9	2	0	1

input

9	17	6	11	11	1

output

OpenACC
More Science, Less Programming

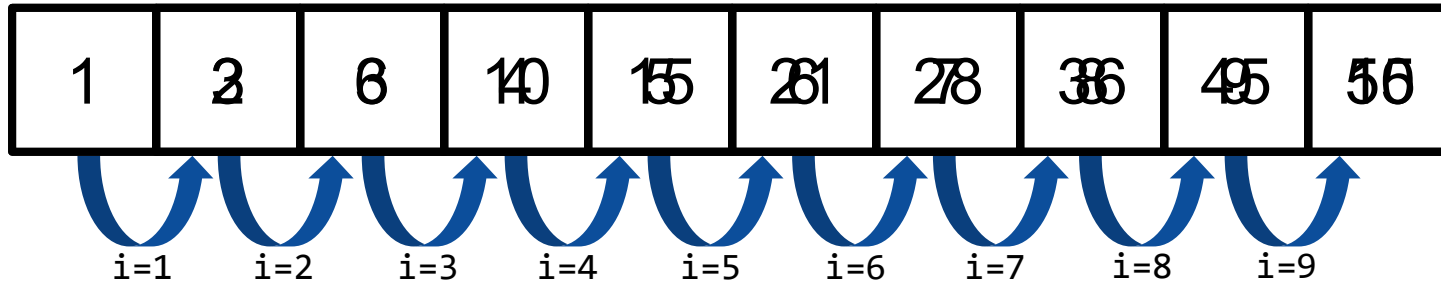# CODING WITH OPENACC

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];
}
```

6	3	10	7	2	4	3	8	9	2	0	1

input

9	17	6	11	11	1

output

DATA DEPENDENCIES

Not all loops are parallel

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

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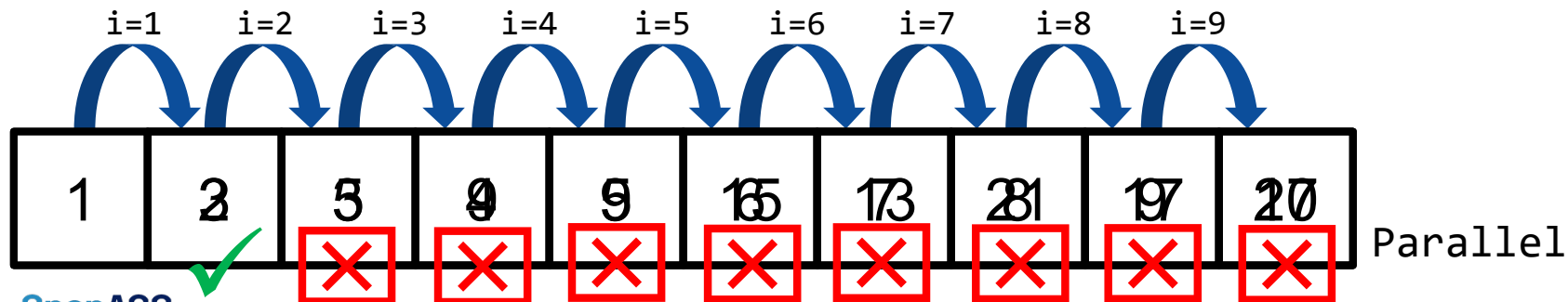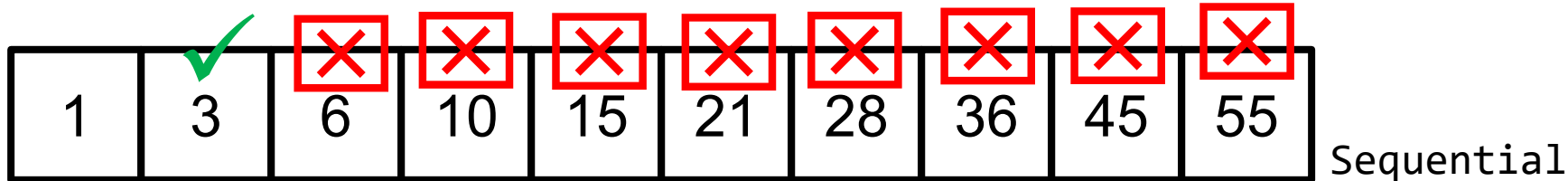
DATA DEPENDENCIES

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*.

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
More Science, Less Programming

OPENACC RESOURCES

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

FREE Compilers

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

OpenACC
More Science, Less Programming

THANK YOU

OpenACC

More Science, Less Programming