SF2568: Parallel Computations for Large-Scale Problems

Lecture 1: Introduction

January 16, 2024

Acknowledgements

These slides are an extension of slides by Michael Hanke and Niclas Jansson.

What is Parallel Computing?

The traditional and basic way software has been written is for *serial* computation

- To be run on a single computer having a single Central Processing Unit (CPU)
- Problem is broken into a discrete series of instructions
- Instructions are executed sequentially and one after another
- Limited as only one instruction is executed at any moment in time

What is Parallel Computing?

The traditional and basic way software has been written is for *serial* computation

- To be run on a single computer having a single Central Processing Unit (CPU)
- Problem is broken into a discrete series of instructions
- Instructions are executed sequentially and one after another
- Limited as only one instruction is executed at any moment in time

What is Parallel Computing?

The traditional and basic way software has been written is for *serial* computation

- To be run on a single computer having a single Central Processing Unit (CPU)
- Problem is broken into a discrete series of instructions
- Instructions are executed sequentially and one after another
- Limited as only one instruction is executed at any moment in time

What is Parallel Computing?

The traditional and basic way software has been written is for *serial* computation

- To be run on a single computer having a single Central Processing Unit (CPU)
- Problem is broken into a discrete series of instructions
- Instructions are executed sequentially and one after another
- Limited as only one instruction is executed at any moment in time

In the most basic form, *parallel computing* is the simultaneous use of multiple compute resources to solve a computational problem

- To be run on multiple CPU cores
- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different CPUs / cores

In the most basic form, *parallel computing* is the simultaneous use of multiple compute resources to solve a computational problem

- To be run on multiple CPU cores
- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different CPUs / cores

In the most basic form, *parallel computing* is the simultaneous use of multiple compute resources to solve a computational problem

- To be run on multiple CPU cores
- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different CPUs / cores

In the most basic form, *parallel computing* is the simultaneous use of multiple compute resources to solve a computational problem

- To be run on multiple CPU cores
- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different CPUs / cores

The compute resources can include

- A single computer with multiple processors
- An arbitrary number of computers connected by a network
- A combination of both

The computational problem usually demonstrates characteristics such as the ability to

- Be broken apart into discrete pieces of work that can be solved simultaneously
- Execute multiple program instructions at any moment in time
- Be solved in less time with multiple compute resources than with a single compute resource

The compute resources can include

- A single computer with multiple processors
- An arbitrary number of computers connected by a network
- A combination of both

The computational problem usually demonstrates characteristics such as the ability to

- Be broken apart into discrete pieces of work that can be solved simultaneously
- Execute multiple program instructions at any moment in time
- Be solved in less time with multiple compute resources than with a single compute resource

The compute resources can include

- A single computer with multiple processors
- An arbitrary number of computers connected by a network
- A combination of both

The computational problem usually demonstrates characteristics such as the ability to

- Be broken apart into discrete pieces of work that can be solved simultaneously
- Execute multiple program instructions at any moment in time
- Be solved in less time with multiple compute resources than with a single compute resource

The compute resources can include

- A single computer with multiple processors
- An arbitrary number of computers connected by a network
- A combination of both

The computational problem usually demonstrates characteristics such as the ability to

- Be broken apart into discrete pieces of work that can be solved simultaneously
- Execute multiple program instructions at any moment in time
- Be solved in less time with multiple compute resources than with a single compute resource

The compute resources can include

- A single computer with multiple processors
- An arbitrary number of computers connected by a network
- A combination of both

The computational problem usually demonstrates characteristics such as the ability to

- Be broken apart into discrete pieces of work that can be solved simultaneously
- Execute multiple program instructions at any moment in time
- Be solved in less time with multiple compute resources than with a single compute resource

The compute resources can include

- A single computer with multiple processors
- An arbitrary number of computers connected by a network
- A combination of both

The computational problem usually demonstrates characteristics such as the ability to

- Be broken apart into discrete pieces of work that can be solved simultaneously
- Execute multiple program instructions at any moment in time
- Be solved in less time with multiple compute resources than with a single compute resource

The compute resources can include

- A single computer with multiple processors
- An arbitrary number of computers connected by a network
- A combination of both

The computational problem usually demonstrates characteristics such as the ability to

- Be broken apart into discrete pieces of work that can be solved simultaneously
- Execute multiple program instructions at any moment in time
- Be solved in less time with multiple compute resources than with a single compute resource

Why Parallel Computations?

The primary reasons for using parallel computing

- Save time - wall clock time
- Solve larger problems

Other reasons might include

- Taking advantage of non-local resources
- Overcoming memory constraints
- Cost saving

Why Parallel Computations?

The primary reasons for using parallel computing

- Save time - wall clock time
- Solve larger problems

Other reasons might include

- Taking advantage of non-local resources
- Overcoming memory constraints
- Cost saving

Why Parallel Computations?

The primary reasons for using parallel computing

- Save time - wall clock time
- Solve larger problems

Other reasons might include

- Taking advantage of non-local resources
- Overcoming memory constraints
- Cost saving

Why Parallel Computations?

The primary reasons for using parallel computing

- Save time - wall clock time
- Solve larger problems

Other reasons might include

- Taking advantage of non-local resources
- Overcoming memory constraints
- Cost saving

Why Parallel Computations?

The primary reasons for using parallel computing

- Save time - wall clock time
- Solve larger problems

Other reasons might include

- Taking advantage of non-local resources
- Overcoming memory constraints
- Cost saving

Why Parallel Computations?

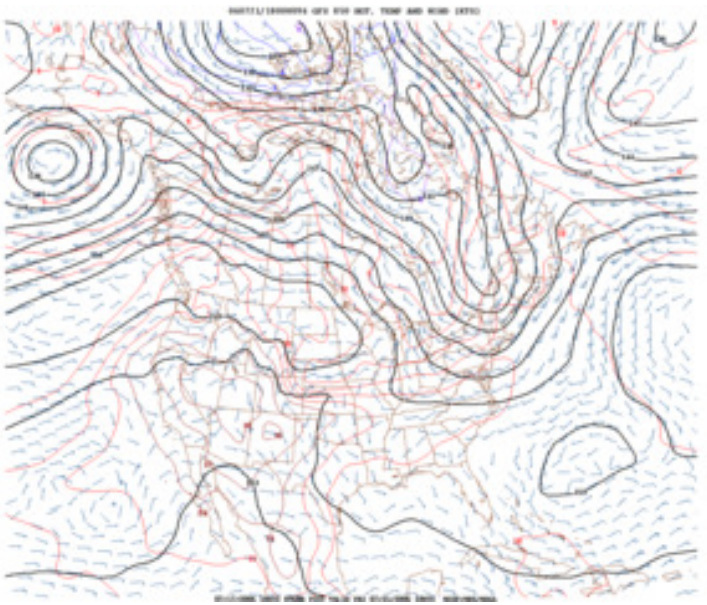
The primary reasons for using parallel computing

- Save time - wall clock time
- Solve larger problems

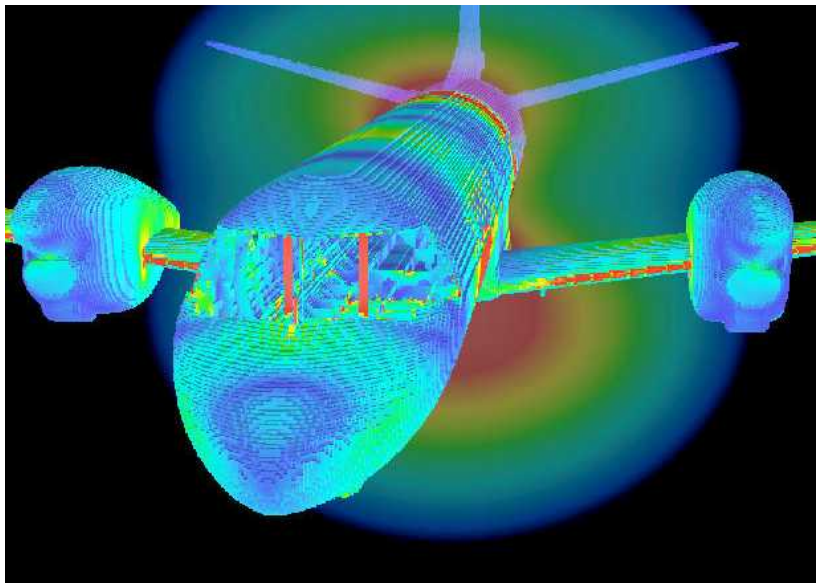
Other reasons might include

- Taking advantage of non-local resources
- Overcoming memory constraints
- Cost saving

Weather Prediction

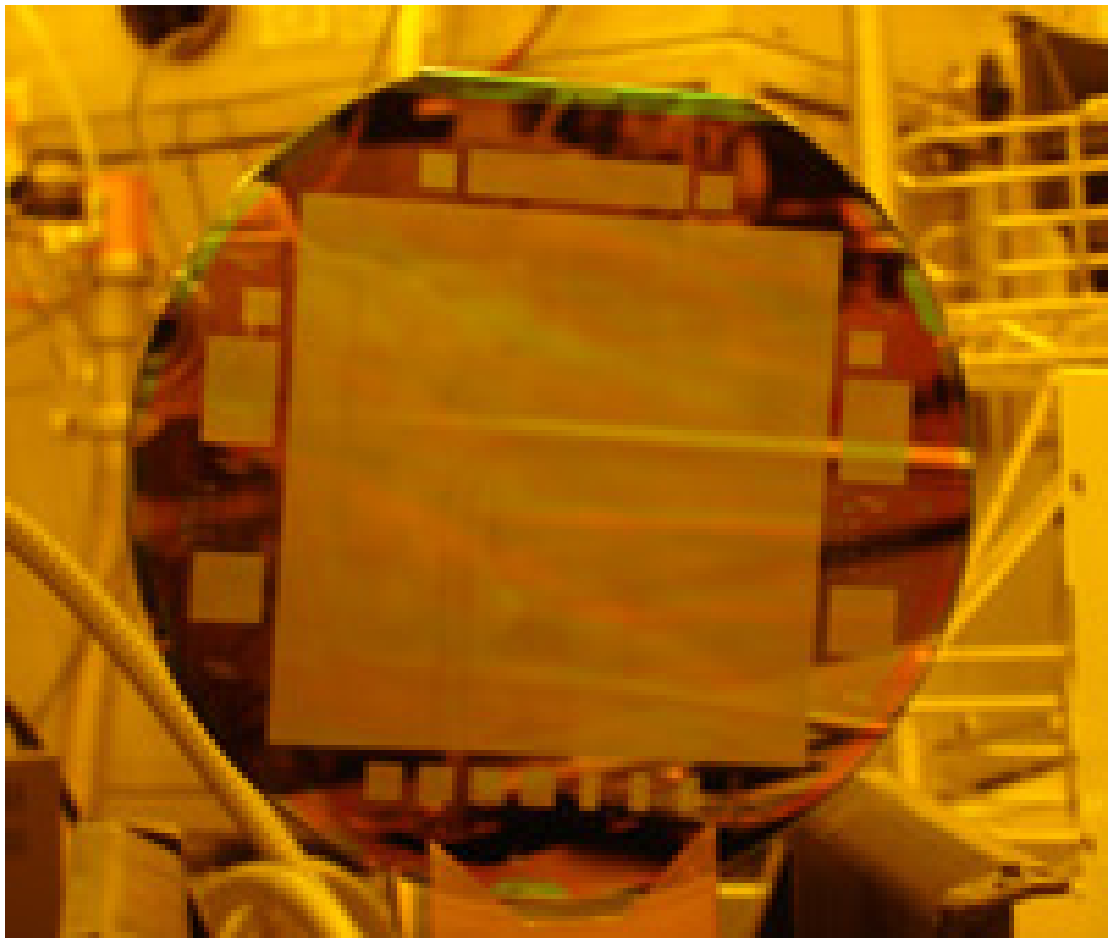


Computational Electromagnetics

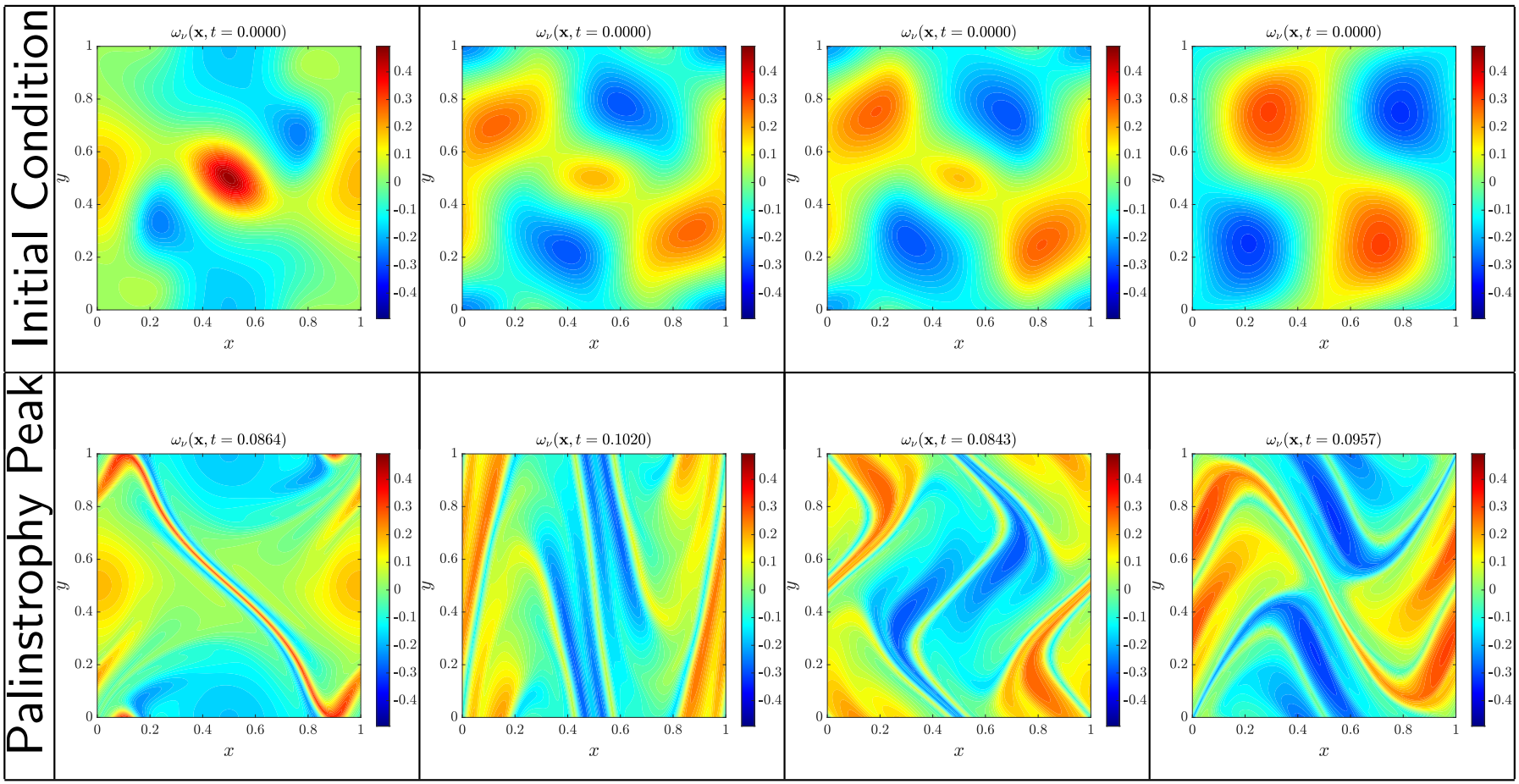


Real-Time Image Processing

Digital Video Broadcasting (HD-TV)



Analytic Mathematical Problems



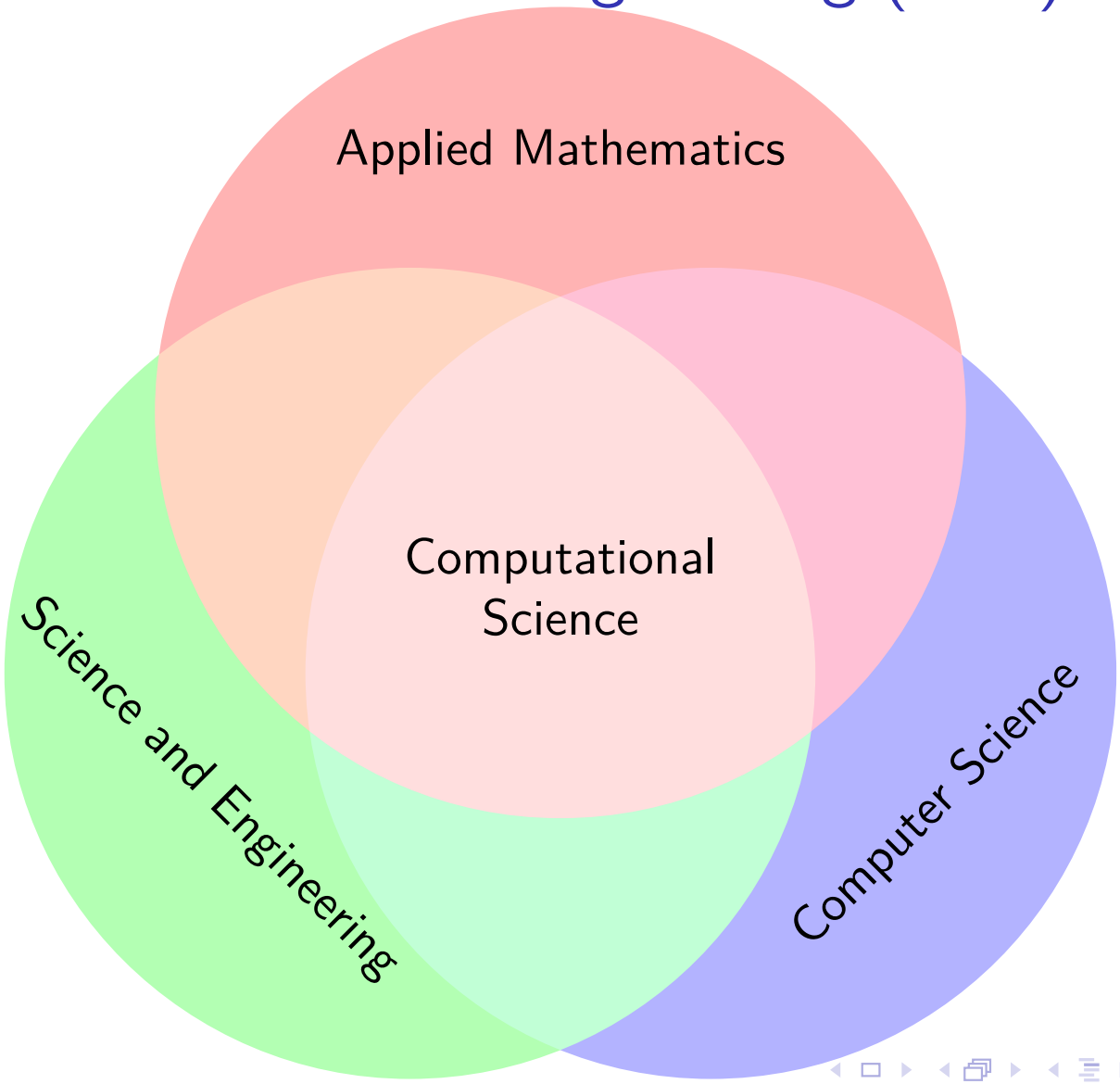
Computational Science and Engineering (CSE)

- Examples from life sciences (molecular biology), material science (phase transition in welding), fluid dynamics (turbulence)
- The era of petascale computing has been a breakthrough for CSE, with more than 1,000,000,000,000,000 floating point operations (FLOPs) per second

Computational Science and Engineering (CSE)

- Examples from life sciences (molecular biology), material science (phase transition in welding), fluid dynamics (turbulence)
- The era of petascale computing has been a breakthrough for CSE, with more than 1,000,000,000,000,000 floating point operations (FLOPs) per second

Computational Science and Engineering (CSE)



Why is Parallelism Necessary

There are limits to serial computing - both physical and practical reasons pose significant constraints to simply building faster serial computers. Parallel computing

- Increases computational power since FLOP rates are limited by the speed of light (30 cm/nanosecond) and molecule sizes, and the transmission limit of copper wire (9cm/nanosecond)
- Increases memory/disk speed, as processor speed on the chip outperform memory/disk subsystem by far
- Avoids communication of distributed data

Why is Parallelism Necessary

There are limits to serial computing - both physical and practical reasons pose significant constraints to simply building faster serial computers. Parallel computing

- Increases computational power since FLOP rates are limited by the speed of light (30 cm/nanosecond) and molecule sizes, and the transmission limit of copper wire (9cm/nanosecond)
- Increases memory/disk speed, as processor speed on the chip outperform memory/disk subsystem by far
- Avoids communication of distributed data

Why is Parallelism Necessary

There are limits to serial computing - both physical and practical reasons pose significant constraints to simply building faster serial computers. Parallel computing

- Increases computational power since FLOP rates are limited by the speed of light (30 cm/nanosecond) and molecule sizes, and the transmission limit of copper wire (9cm/nanosecond)
- Increases memory/disk speed, as processor speed on the chip outperform memory/disk subsystem by far
- Avoids communication of distributed data

Why is Parallelism Programming Interesting

A well behaved single processor algorithm may behave poorly on a parallel computer, and may need to be *reformulated numerically*

There is no magic compiler that can turn a serial program into an efficient program all the time and on all machines

- Performance programming involving low-level details: heavily application dependent
- Irregularity in the computation and its data structures forces us to think even harder. Parallel programs are not naturally deterministic
- Users don't start from scratch — they reuse old code. *Code lives longer than you expect!*

Why is Parallelism Programming Interesting

A well behaved single processor algorithm may behave poorly on a parallel computer, and may need to be *reformulated numerically*

There is no magic compiler that can turn a serial program into an efficient program all the time and on all machines

- Performance programming involving low-level details: heavily application dependent
- Irregularity in the computation and its data structures forces us to think even harder. Parallel programs are not naturally deterministic
- Users don't start from scratch — they reuse old code. *Code lives longer than you expect!*

Why is Parallelism Programming Interesting

A well behaved single processor algorithm may behave poorly on a parallel computer, and may need to be *reformulated numerically*

There is no magic compiler that can turn a serial program into an efficient program all the time and on all machines

- Performance programming involving low-level details: heavily application dependent
- Irregularity in the computation and its data structures forces us to think even harder. Parallel programs are not naturally deterministic
- Users don't start from scratch — they reuse old code. *Code lives longer than you expect!*

Why is Parallelism Programming Interesting

A well behaved single processor algorithm may behave poorly on a parallel computer, and may need to be *reformulated numerically*

There is no magic compiler that can turn a serial program into an efficient program all the time and on all machines

- Performance programming involving low-level details: heavily application dependent
- Irregularity in the computation and its data structures forces us to think even harder. Parallel programs are not naturally deterministic
- Users don't start from scratch — they reuse old code. *Code lives longer than you expect!*

Aim of the course

From the students' handbook

The goal of the course is to provide a basic understanding of how to develop algorithms and how to implement them in distributed memory computers using the message-passing paradigm

Aim of the course

This understanding means that after the course you are able to

- explain *parallelization strategies*
- *select and/or develop an algorithm* for solving a given problem which has the potential for an efficient parallelization
- *select and/or develop data structures* for implementing parallel computations
- *theoretically analyze* a given parallel algorithm with respect to efficiency
- *implement* a given algorithm on a distributed-memory computer using the *message passing library MPI*
- understand the message flow and avoid unwanted situations (e.g. deadlock, synchronization delays)
- modify and adapt a set of basic routines to special situations
- *experimentally evaluate* the performance of a parallel program
- explain differences between the theoretically expected performance and the practically observed performance
- understand the challenges of *Green Computing*

(Preliminary!) Overview of the Course

- Basic ideas of parallel computations
- An Introduction To MPI
- Image Reconstruction And Poisson's Equation
- Matrix-Vector Multiplication And Collective Communication
- Linear Systems of Equations
- Parallel Sorting
- Algorithms On Graphs
- Advanced Topics

Practical Details

- Prerequisites
  - Basic course in numerical analysis and programming (C, C++ or Fortran). A short introduction to Fortran (or C) will be provided
- Grading
  - Homework assignments (3 assignments, in a groupd of at most 2 students)
  - A midterm quiz
  - Larger project, topic chosen yourself
- Computer assignments on Dardel at PDC
  - Fastest supercomputer in Sweden
  - Apply for a PDC account **here**
    - "Time allocation" should be answered by `edu24.sf2568`
    - Complete this ASAP!

Practical Details

- Prerequisites
  - Basic course in numerical analysis and programming (C, C++ or Fortran). A short introduction to Fortran (or C) will be provided
- Grading
  - Homework assignments (3 assignments, in a groupd of at most 2 students)
  - A midterm quiz
  - Larger project, topic chosen yourself
- Computer assignments on Dardel at PDC
  - Fastest supercomputer in Sweden
  - Apply for a PDC account **here**
    - "Time allocation" should be answered by `edu24.sf2568`
    - Complete this ASAP!

Practical Details

- Prerequisites
  - Basic course in numerical analysis and programming (C, C++ or Fortran). A short introduction to Fortran (or C) will be provided
- Grading
  - Homework assignments (3 assignments, in a groupd of at most 2 students)
  - A midterm quiz
  - Larger project, topic chosen yourself
- Computer assignments on Dardel at PDC
  - Fastest supercomputer in Sweden
  - Apply for a PDC account **here**
    - "Time allocation" should be answered by `edu24.sf2568`
    - Complete this ASAP!

Examination

The course consists of two components, HEMA and PROA:

- Assignment evaluation and mid-term quiz (HEMA, 4.5 credits):
  There will be 3 assignments and 1 quiz, consisting of Assignment 1 (15 pts), Assignment 2 (12 pts), Assignment 3 (12 pts), the Quiz (17 pts). Maximum number of points is 56 pts for this portion of the course. In order to pass (grade E), *at least one point from each component is required*

- Project report with an oral defense/ presentation (PROA, 3 credits).
  Projects will be graded based on: Correctness of algorithm, theoretical analysis, implementation, experimental analysis, and the quality of the report.
  In order to pass (grade E), *a correctly running program and report must be submitted*

Examination

The course consists of two components, HEMA and PROA:

- Assignment evaluation and mid-term quiz (HEMA, 4.5 credits):
  There will be 3 assignments and 1 quiz, consisting of Assignment 1 (15 pts), Assignment 2 (12 pts), Assignment 3 (12 pts), the Quiz (17 pts). Maximum number of points is 56 pts for this portion of the course. In order to pass (grade E), *at least one point from each component is required*

- Project report with an oral defense/ presentation (PROA, 3 credits).
  Projects will be graded based on: Correctness of algorithm, theoretical analysis, implementation, experimental analysis, and the quality of the report.
  In order to pass (grade E), *a correctly running program and report must be submitted*

Examination

The course consists of two components, HEMA and PROA:

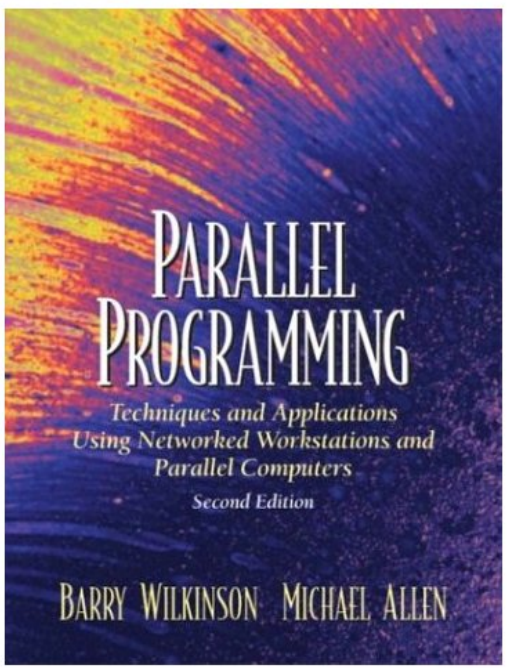
- Assignment evaluation and mid-term quiz (HEMA, 4.5 credits):
  There will be 3 assignments and 1 quiz, consisting of Assignment 1 (15 pts), Assignment 2 (12 pts), Assignment 3 (12 pts), the Quiz (17 pts). Maximum number of points is 56 pts for this portion of the course. In order to pass (grade E), *at least one point from each component is required*

- Project report with an oral defense/ presentation (PROA, 3 credits).
  Projects will be graded based on: Correctness of algorithm, theoretical analysis, implementation, experimental analysis, and the quality of the report.
  In order to pass (grade E), *a correctly running program and report must be submitted*

Literature

- Course book: Barry Wilkinson, Michael Allan: *Parallel Programming, 2nd edition*

- **Lecture Notes**



Instructor

- Instructor: Pritpal ‘Pip‘ Matharu (call me Pip)
  - Researcher in numerical analysis, studying problems in fluids and approximation functions!
- Office: Room 3603, Lindstedtsvägen 25
- Email: `pritpal@kth.se`
- Canvas will be used for announcements, assignments (instructions and submissions), project instructions, and discussion boards
- For general PDC support: `support@pdc.kth.se`

Instructor

- Instructor: Pritpal ‘Pip‘ Matharu (call me Pip)
  - Researcher in numerical analysis, studying problems in fluids and approximation functions!
- Office: Room 3603, Lindstedtsvägen 25
- Email: `pritpal@kth.se`
- Canvas will be used for announcements, assignments (instructions and submissions), project instructions, and discussion boards
- For general PDC support: `support@pdc.kth.se`

Instructor

- Instructor: Pritpal 'Pip' Matharu (call me Pip)
  - Researcher in numerical analysis, studying problems in fluids and approximation functions!
- Office: Room 3603, Lindstedtsvägen 25
- Email: `pritpal@kth.se`
- Canvas will be used for announcements, assignments (instructions and submissions), project instructions, and discussion boards
- For general PDC support: `support@pdc.kth.se`

Instructor

- Instructor: Pritpal 'Pip' Matharu (call me Pip)
  - Researcher in numerical analysis, studying problems in fluids and approximation functions!
- Office: Room 3603, Lindstedtsvägen 25
- Email: `pritpal@kth.se`
- Canvas will be used for announcements, assignments (instructions and submissions), project instructions, and discussion boards
- For general PDC support: `support@pdc.kth.se`

Instructor

- Instructor: Pritpal 'Pip' Matharu (call me Pip)
  - Researcher in numerical analysis, studying problems in fluids and approximation functions!
- Office: Room 3603, Lindstedtsvägen 25
- Email: `pritpal@kth.se`
- Canvas will be used for announcements, assignments (instructions and submissions), project instructions, and discussion boards
- For general PDC support: `support@pdc.kth.se`

Instructor

- Instructor: Pritpal ‘Pip‘ Matharu (call me Pip)
  - Researcher in numerical analysis, studying problems in fluids and approximation functions!
- Office: Room 3603, Lindstedtsvägen 25
- Email: `pritpal@kth.se`
- Canvas will be used for announcements, assignments (instructions and submissions), project instructions, and discussion boards
- For general PDC support: `support@pdc.kth.se`

The Situation

- In 2013, data centres in the US consumed 91 GWh electrical energy amounting to *2.4% of overall consumption*
- Peak power consumption of US data centres *7 GW in 2007*
- In comparison: Overall installed power capacity in Sweden 36 GW in 2013 (Germany: 153 GW)

Source: Center for Sustainable Systems, University of Michigan. 2017. "Green IT Factsheet." Pub. No. CSS09-07 and CIA World Factbook

The Situation

- In 2013, data centres in the US consumed 91 GWh electrical energy amounting to *2.4% of overall consumption*
- Peak power consumption of US data centres *7 GW in 2007*
- In comparison: Overall installed power capacity in Sweden 36 GW in 2013 (Germany: 153 GW)

Source: Center for Sustainable Systems, University of Michigan. 2017. "Green IT Factsheet." Pub. No. CSS09-07 and CIA World Factbook

The Situation

- In 2013, data centres in the US consumed 91 GWh electrical energy amounting to *2.4% of overall consumption*
- Peak power consumption of US data centres *7 GW in 2007*
- In comparison: Overall installed power capacity in Sweden 36 GW in 2013 (Germany: 153 GW)

Source: Center for Sustainable Systems, University of Michigan. 2017. "Green IT Factsheet." Pub. No. CSS09-07 and CIA World Factbook

New Challenges: Green Computing and HPC

System	Rmax [TFlops/s]	Power [kW]	[GFlops/W]	[Ext (MW)]
Sweden's 2nd fastest (#110) Berzelius (LiU)	2253	???		
Sweden's fastest (#68) Dardel (KTH) (#5 Green500)	8260	146	57	18
Sweden's fastest (2018) Beskow (KTH)	1802	842	2	500
#2 Top500 Fugaku (JP)	442,010	29,899	15	67
#1 Green500 (#405 Top500) Henri (US)	2,040	31	66	15

Note: Note. The column "Ext" is the virtual power consumption if a computer using the given technology would provide 1 EFlop/s

Source: top500.org

New Challenges: Green Computing and HPC

Average power consumption of a city with 100,000 inhabitants (2013)

Country	Average power consumption [MW]
US	920
Sweden	683
Germany	515
Japan	475
China	296
India	81

Source: Wikipedia, CIA World Factbook

A Pilot Project at KTH

- Most of the consumed energy is converted to heat
- It is expensive to get rid of the waste heat (cooling)

What if one could reuse this waste heat?

- Problem: The heat appears as warm air and has, therefore, low energy content
- Under these conditions, efficiency of heat exchangers is heavily restricted
- For more information, see PDC's webpage

A Pilot Project at KTH

- Most of the consumed energy is converted to heat
- It is expensive to get rid of the waste heat (cooling)

What if one could reuse this waste heat?

- Problem: The heat appears as warm air and has, therefore, low energy content
- Under these conditions, efficiency of heat exchangers is heavily restricted
- For more information, see PDC's webpage

A Pilot Project at KTH

- Most of the consumed energy is converted to heat
- It is expensive to get rid of the waste heat (cooling)

What if one could reuse this waste heat?

- Problem: The heat appears as warm air and has, therefore, low energy content
- Under these conditions, efficiency of heat exchangers is heavily restricted
- For more information, see PDC's webpage

A Pilot Project at KTH

- Most of the consumed energy is converted to heat
- It is expensive to get rid of the waste heat (cooling)

What if one could reuse this waste heat?

- Problem: The heat appears as warm air and has, therefore, low energy content
- Under these conditions, efficiency of heat exchangers is heavily restricted
- For more information, see PDC's webpage

A Pilot Project at KTH

- Most of the consumed energy is converted to heat
- It is expensive to get rid of the waste heat (cooling)

What if one could reuse this waste heat?

- Problem: The heat appears as warm air and has, therefore, low energy content
- Under these conditions, efficiency of heat exchangers is heavily restricted
- For more information, see PDC's webpage

A Pilot Project at KTH

- Most of the consumed energy is converted to heat
- It is expensive to get rid of the waste heat (cooling)

What if one could reuse this waste heat?

- Problem: The heat appears as warm air and has, therefore, low energy content
- Under these conditions, efficiency of heat exchangers is heavily restricted
- For more information, see PDC's webpage

von Neumann Architecture

- For over 40 years, virtually all computers have followed a common machine model known as the von Neumann computer (named after the Hungarian mathematician John von Neumann)

- A von Neumann computer uses the stored-program concept. The CPU executes a stored program that specifies a sequence of read and write operations on the memory

von Neumann Architecture

- For over 40 years, virtually all computers have followed a common machine model known as the von Neumann computer (named after the Hungarian mathematician John von Neumann)

- A von Neumann computer uses the stored-program concept. The CPU executes a stored program that specifies a sequence of read and write operations on the memory

von Neumann Architecture

- Basic design:
  - Memory is used to store both program and data instructions
  - Program instructions are coded data which tell the computer to do something
  - Data is simply information to be used by the program
- A central processing unit (CPU) gets instructions and/or data from memory, decodes the instructions and then **sequentially** performs them

von Neumann Architecture

- Basic design:
  - Memory is used to store both program and data instructions
    - Program instructions are coded data which tell the computer to do something
    - Data is simply information to be used by the program
- A central processing unit (CPU) gets instructions and/or data from memory, decodes the instructions and then **sequentially** performs them

von Neumann Architecture

- Basic design:
  - Memory is used to store both program and data instructions
  - Program instructions are coded data which tell the computer to do something
  - Data is simply information to be used by the program
- A central processing unit (CPU) gets instructions and/or data from memory, decodes the instructions and then **sequentially** performs them

von Neumann Architecture

- Basic design:
  - Memory is used to store both program and data instructions
  - Program instructions are coded data which tell the computer to do something
  - Data is simply information to be used by the program
- A central processing unit (CPU) gets instructions and/or data from memory, decodes the instructions and then **sequentially** performs them

von Neumann Architecture

- Basic design:
  - Memory is used to store both program and data instructions
  - Program instructions are coded data which tell the computer to do something
  - Data is simply information to be used by the program
- A central processing unit (CPU) gets instructions and/or data from memory, decodes the instructions and then **sequentially** performs them

Flynn's Classical Taxonomy

- There are different ways to classify parallel computers. One of the more widely used classifications, in use since 1966, is called Flynn's Taxonomy

- Flynn's taxonomy distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of **Instruction** and **Data**. Each of these dimensions can have only one of two possible states: **Single** or **Multiple**

Flynn's Classical Taxonomy

- There are different ways to classify parallel computers. One of the more widely used classifications, in use since 1966, is called Flynn's Taxonomy

- Flynn's taxonomy distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of **Instruction** and **Data**. Each of these dimensions can have only one of two possible states: **Single** or **Multiple**

Flynn's Classical Taxonomy

There are 4 possible classifications according to Flynn

- SISD: Single Instruction, Single Data
- SIMD: Single Instruction, Multiple Data
- MISD: Multiple Instruction, Single Data
- MIMD: Multiple Instruction, Multiple Data

Flynn's Classical Taxonomy

There are 4 possible classifications according to Flynn

- SISD: Single Instruction, Single Data
- SIMD: Single Instruction, Multiple Data
- MISD: Multiple Instruction, Single Data
- MIMD: Multiple Instruction, Multiple Data

Flynn's Classical Taxonomy

There are 4 possible classifications according to Flynn

- SISD: Single Instruction, Single Data
- SIMD: Single Instruction, Multiple Data
- MISD: Multiple Instruction, Single Data
- MIMD: Multiple Instruction, Multiple Data

Flynn's Classical Taxonomy

There are 4 possible classifications according to Flynn

- SISD: Single Instruction, Single Data
- SIMD: Single Instruction, Multiple Data
- MISD: Multiple Instruction, Single Data
- MIMD: Multiple Instruction, Multiple Data

Single Instruction, Single Data

- A serial (non-parallel) computer
- Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle
- Single data: only one data stream is being used as input during any one clock cycle
- Deterministic execution
- This is the oldest and until recently, the most prevalent form of computer

Single Instruction, Single Data

- A serial (non-parallel) computer
- Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle
- Single data: only one data stream is being used as input during any one clock cycle
- Deterministic execution
- This is the oldest and until recently, the most prevalent form of computer

Single Instruction, Single Data

- A serial (non-parallel) computer
- Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle
- Single data: only one data stream is being used as input during any one clock cycle
- Deterministic execution
- This is the oldest and until recently, the most prevalent form of computer

Single Instruction, Single Data

- A serial (non-parallel) computer
- Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle
- Single data: only one data stream is being used as input during any one clock cycle
- Deterministic execution
- This is the oldest and until recently, the most prevalent form of computer

Single Instruction, Single Data

- A serial (non-parallel) computer
- Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle
- Single data: only one data stream is being used as input during any one clock cycle
- Deterministic execution
- This is the oldest and until recently, the most prevalent form of computer

Single Instruction, Multiple Data

- A type of parallel computer
- Single instruction: All processing units execute the same instruction at any given clock cycle
- Multiple data: Each processing unit can operate on a different data element
- This type of machine typically has an instruction dispatcher, a very high-bandwidth internal network, and a very large array of very small-capacity instruction units
- Best suited for specialized problems characterized by a high degree of regularity, such as image processing
- Synchronous (lockstep) and deterministic execution

Single Instruction, Multiple Data

- A type of parallel computer
- Single instruction: All processing units execute the same instruction at any given clock cycle
- Multiple data: Each processing unit can operate on a different data element
- This type of machine typically has an instruction dispatcher, a very high-bandwidth internal network, and a very large array of very small-capacity instruction units
- Best suited for specialized problems characterized by a high degree of regularity, such as image processing
- Synchronous (lockstep) and deterministic execution

Single Instruction, Multiple Data

- A type of parallel computer
- Single instruction: All processing units execute the same instruction at any given clock cycle
- Multiple data: Each processing unit can operate on a different data element
- This type of machine typically has an instruction dispatcher, a very high-bandwidth internal network, and a very large array of very small-capacity instruction units
- Best suited for specialized problems characterized by a high degree of regularity, such as image processing
- Synchronous (lockstep) and deterministic execution

Single Instruction, Multiple Data

- A type of parallel computer
- Single instruction: All processing units execute the same instruction at any given clock cycle
- Multiple data: Each processing unit can operate on a different data element
- This type of machine typically has an instruction dispatcher, a very high-bandwidth internal network, and a very large array of very small-capacity instruction units
- Best suited for specialized problems characterized by a high degree of regularity, such as image processing
- Synchronous (lockstep) and deterministic execution

Single Instruction, Multiple Data

- A type of parallel computer
- Single instruction: All processing units execute the same instruction at any given clock cycle
- Multiple data: Each processing unit can operate on a different data element
- This type of machine typically has an instruction dispatcher, a very high-bandwidth internal network, and a very large array of very small-capacity instruction units
- Best suited for specialized problems characterized by a high degree of regularity, such as image processing
- Synchronous (lockstep) and deterministic execution

Single Instruction, Multiple Data

- A type of parallel computer
- Single instruction: All processing units execute the same instruction at any given clock cycle
- Multiple data: Each processing unit can operate on a different data element
- This type of machine typically has an instruction dispatcher, a very high-bandwidth internal network, and a very large array of very small-capacity instruction units
- Best suited for specialized problems characterized by a high degree of regularity, such as image processing
- Synchronous (lockstep) and deterministic execution

Multiple Instructions, Single Data

- A single data stream is fed into multiple processing units
- Each processing unit operates on the data independently via independent instruction streams
- Few actual examples of this class of parallel computer have ever existed

Multiple Instructions, Single Data

- A single data stream is fed into multiple processing units
- Each processing unit operates on the data independently via independent instruction streams
- Few actual examples of this class of parallel computer have ever existed

Multiple Instructions, Single Data

- A single data stream is fed into multiple processing units
- Each processing unit operates on the data independently via independent instruction streams
- Few actual examples of this class of parallel computer have ever existed

Multiple Instructions, Multiple Data

- Currently, the most common type of parallel computer. Most modern computers fall into this category.
- Multiple Instruction: every processor may be executing a different instruction stream
- Multiple Data: every processor may be working with a different data stream
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- Examples: most current supercomputers, networked parallel computer clusters and multi-processor SMP computers (most of the current PCs)

Multiple Instructions, Multiple Data

- Currently, the most common type of parallel computer. Most modern computers fall into this category.
- Multiple Instruction: every processor may be executing a different instruction stream
- Multiple Data: every processor may be working with a different data stream
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- Examples: most current supercomputers, networked parallel computer clusters and multi-processor SMP computers (most of the current PCs)

Multiple Instructions, Multiple Data

- Currently, the most common type of parallel computer. Most modern computers fall into this category.
- Multiple Instruction: every processor may be executing a different instruction stream
- Multiple Data: every processor may be working with a different data stream
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- Examples: most current supercomputers, networked parallel computer clusters and multi-processor SMP computers (most of the current PCs)

Multiple Instructions, Multiple Data

- Currently, the most common type of parallel computer. Most modern computers fall into this category.
- Multiple Instruction: every processor may be executing a different instruction stream
- Multiple Data: every processor may be working with a different data stream
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- Examples: most current supercomputers, networked parallel computer clusters and multi-processor SMP computers (most of the current PCs)

Multiple Instructions, Multiple Data

- Currently, the most common type of parallel computer. Most modern computers fall into this category.
- Multiple Instruction: every processor may be executing a different instruction stream
- Multiple Data: every processor may be working with a different data stream
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- Examples: most current supercomputers, networked parallel computer clusters and multi-processor SMP computers (most of the current PCs)

A First Example: Rank Sort

Problem

Given n (pairwise distinct) numbers. Sort the numbers in increasing order.

Solution

- Let the **rank** r_i of an element a_i of the set M of numbers be the number of elements being less than that element,

$$r_i = \#\{a_j \in M | a_j < a_i\}$$

- In a fully sorted list, the position of element i is just its rank r_i.

A First Example: Rank Sort

Problem

Given n (pairwise distinct) numbers. Sort the numbers in increasing order.

Solution

- Let the **rank** r_i of an element a_i of the set M of numbers be the number of elements being less than that element,

$$r_i = \#\{a_j \in M | a_j < a_i\}$$

- In a fully sorted list, the position of element i is just its rank r_i.