Parallel and Distributed Computing: Introduction

Erik Saule
esaule@uncc.edu

Parallel and Distributed Computing

01/08/18

Outline

Format

- 16 weeks + Final
- 2 classes of 1:15 hours each on MW 11am in Woodward 140
- Final on Monday May 7: 11:00am to 1:30pm according to (`https://registrar.uncc.edu/sites/registrar.uncc.edu/files/media/Spring%202018%20FE%20Table.pdf`)
- In-class assignments to complete for the week after (most weeks)
- Mid term
- Final

Getting help

- My office is in Woodward 210D: I am there most of the time.
- You can reach me by email (`esaule@uncc.edu`)
- My office hours are on Wednesday from 2:30pm to 4:30pm
- The TAs are Vivek Soni and Kyle Tibbetts
  - TAs are good and helpful.
  - Use their help!
  - We will organize a poll to decide TA's office hours
- Use the discussion board on Canvas!
  - Often your peers are your best source of insight
  - (No cheating though. Yes, we check!)

Material

Textbook

- I am not quite following a given textbook
- But I'll point you to external material (see last slide)

Class material

- Slides and assignments will be posted online
- Hopefully before the class
- Sometimes right after

Typo hunt

- If you see a typo or something that doesn't make sense in a slide, write a message on the discussion board
- That way I can fix it for the next class and have better slides
- It can also help lifting doubts in case it is actually not a typo

Outline

Survey

What do YOU think this class is about?
What do YOU expect from it?

What do YOU think this class is about? What do YOU expect from it?

- Who took 2214? 2215? 3181? 3146?
- Who can write C or C++?
- Who can use pthreads?
- Who can use OpenMP?
- Who used git before?
- Who is familiar with UNIX environments?
- Who can use CUDA or OpenCL?
- Who can use MPI?
- Other distributed memory programming models?
- Anything else that is relevant?

Survey

What do YOU think this class is about? What do YOU expect from it?

- Who took 2214? 2215? 3181? 3146?
- Who can write C or C++?
- Who can use pthreads?
- Who can use OpenMP?
- Who used git before?
- Who is familiar with UNIX environments?
- Who can use CUDA or OpenCL?
- Who can use MPI?
- Other distributed memory programming models?
- Anything else that is relevant?

Access to laptop in class?

What this class is actually about

Theory

- Basics (this week)
- Formal Representation
- Extracting Parallelism

Shared Memory

- pthreads
- concurrent data structure
- OpenMP - loops
- OpenMP - D&C and tasking

Distributed Memory

- Theory
- MPI
- MapReduce MPI

Tentative calendar on Canvas.

Expectation of Incoming Knowledge

- Programming in C or in C++
- Surviving in a UNIX environment
- Using git
- Basic Algorithm
  - Complexity (Big O, Big Theta)
  - Sorting (Mostly Merge Sort)
  - Some graphs

TA and I are here if you don't know something.

Also there is a discussion board on Canvas.

This is how we learn things! By figuring them out!

Class evaluation

Graded Items

- Assignments
  - about every week
  - programming
  - non programming
- Midterm
- Final (cumulative)
- Projects (as extra credit)

Grading Scale

- Thresholds are set per graded item for each letter grade
- Average per category
- Weighted average across categories

Assignment

Submission

All submissions have to be either a PDF or an archive with code and report on canvas.
Submissions should all be on Canvas. assignments submitted through email may not be graded.

Deadline

We start in class on day D.
The assignment is due one week later, on D+7 before the class.

Late Policy

You can submit assignments late for up to 2 weeks after the deadline.
Your grade is capped at 100-3*L, where L is the numbers of day late.
1 minute late is one day late.

Grading

All programming assignments have points for code clarity

- make sure the code is indented correctly
- make sure obscure features are commented
- make sure variable and functions names are meaningful

Always write a report

- does not need to be long
- make sure all that is marked a question in the assignment is answered
- in plain text or pdf format

Try all questions

- Sometimes, you can not do questions 1 and 2
- But that does not mean you can not do question 3

Academic Integrity

The letter of the law

- available at `http://legal.uncc.edu/policies/up-407`

Mostly:

- Cheating
- Plagiarism
- Complicity in academic dishonesty

The spirit of the law

- This is not a class in Googling
- Nor a class in copy pasting
- The point is that you understand what is going on
- And so that you can do it on your own

The penalty is up to me (or not, check the letter). But essentially, not submitting an assignment has a much better outcome than being caught. And in case you are wondering, I will check. And I will get help from automatic tools.

Outline

Cluster Environment

All time measurements will be done on our educational cluster `mamba.urc.uncc.edu` hosted by URC.

Specifications

- 12 nodes + headnode
- Two Xeon E5-2667 v3 per node
- (So 16 cores at 3.2Ghz)
- Four NVIDIA K80 in total
- (so 8 programmable GPUs spread across the machine)

Do not run code on the head node!

Cluster Environment

All time measurements will be done on our educational cluster `mamba.urc.uncc.edu` hosted by URC.

Specifications

- 12 nodes + headnode
- Two Xeon E5-2667 v3 per node
- (So 16 cores at 3.2Ghz)
- Four NVIDIA K80 in total
- (so 8 programmable GPUs spread across the machine)

Do not run code on the head node!

Access: `ssh mamba.urc.uncc.edu`
Submit: `qsub -d `pwd` -l nodes=1:ppn=16 ./script.sh -q mamba -l walltime=1:00:00`
You get a jobID.
Eventually, you will get an output and error file.
Check status: `qstat` and `showq`
More on that later!

Cluster Environment

All time measurements will be done on our educational cluster `mamba.urc.uncc.edu` hosted by URC.

Specifications

- 12 nodes + headnode
- Two Xeon E5-2667 v3 per node
- (So 16 cores at 3.2Ghz)
- Four NVIDIA K80 in total
- (so 8 programmable GPUs spread across the machine)

Do not run code on the head node!

Access: `ssh mamba.urc.uncc.edu`
Submit: `qsub -d `pwd` -l nodes=1:ppn=16 ./script.sh -q mamba -l walltime=1:00:00`
You get a jobID.
Eventually, you will get an output and error file.
Check status: `qstat` and `showq`
More on that later!

There are about 70 students and 12 computing nodes.
You can not afford to wait for the last minute!

Using Linux System on your machine

Why?

- To remove the load from the cluster
- To allow your to work from your laptop

What?

- Last Ubuntu LTS VM image provided (Preconfigured)
- Native Ubuntu install work. Some support for MacOS.
- Container available in Windows 10.

Using Linux System on your machine

Why?

- To remove the load from the cluster
- To allow your to work from your laptop

What?

- Last Ubuntu LTS VM image provided (Preconfigured)
- Native Ubuntu install work. Some support for MacOS.
- Container available in Windows 10.

Recommended Workflow

- Develop & Debug on a Linux system
- Upload to git
- Log on cluster
- Download from git
- Recompile & Submit job

Git

Versionning System

- Server at `https://cci-git.uncc.edu`. Though we will use a local repo on mamba.

In 2 minutes

- Initial checkout with: `git clone` *`https://something`*
- Declare new file: `git add` *`somefile`*
- Commit (locally): `git commit -am "message"`
- Upload to server: `git push`
- Download from server: `git pull`

Structure

- Each student makes one repository
- Make one directory per programming assignment
- Give TA and me access to the repository (to share code)

Outline

Concurrency Vs Parallelism

Concurrency

Processes are concurrent when they can happen in different order relatively to one another.
It often needs to concurrency control to make sure that the execution make sense.
The problem with concurrency is to obtain a correct execution of the application.

Concurrency Vs Parallelism

Concurrency

Processes are concurrent when they can happen in different order relatively to one another.
It often needs to concurrency control to make sure that the execution make sense.
The problem with concurrency is to obtain a correct execution of the application.

Example

- The US postal system
- Laundry in a household

Concurrency Vs Parallelism

Concurrency

Processes are concurrent when they can happen in different order relatively to one another.
It often needs to concurrency control to make sure that the execution make sense.
The problem with concurrency is to obtain a correct execution of the application.

Parallelism

Processes are parallel when they run at the same time on different execution units.
It uses techniques that exposes computation that can be executed simultaneously and organize execution units in doing them as fast as possible.
The purpose is to extract more *performance* out of the execution.

Example

- The US postal system
- Laundry in a household

Concurrency Vs Parallelism

Concurrency

Processes are concurrent when they can happen in different order relatively to one another.
It often needs to concurrency control to make sure that the execution make sense.
The problem with concurrency is to obtain a correct execution of the application.

Example

- The US postal system
- Laundry in a household

Parallelism

Processes are parallel when they run at the same time on different execution units.
It uses techniques that exposes computation that can be executed simultaneously and organize execution units in doing them as fast as possible.
The purpose is to extract more *performance* out of the execution.

Example

- Team building a house
- Cashiers at a grocery store

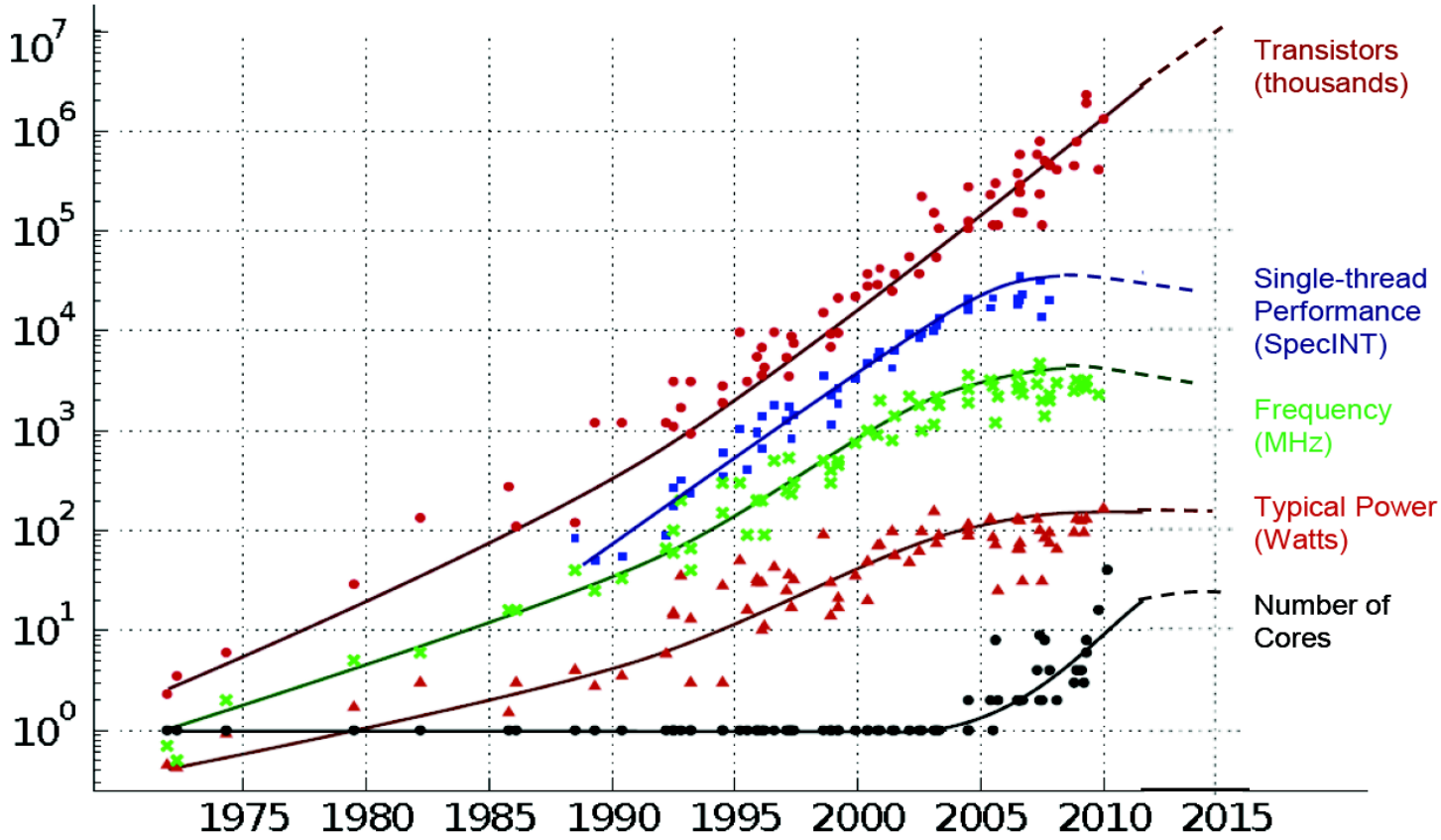
Why Parallelism? The end of Dennard scaling

Why Parallelism? The end of Dennard scaling

Dennard74: as transistors get smaller their power density stays constant.
With shrinking transistor, founder increased frequency.
Computer Scientist in the 80's and 90's: "Why care about code efficiency? Chips will be twice faster in 18 months".

Why Parallelism? The end of Dennard scaling

Dennard74: as transistors get smaller their power density stays constant.
With shrinking transistor, founder increased frequency.
Computer Scientist in the 80's and 90's: "Why care about code efficiency? Chips will be twice faster in 18 months".



Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

Why Parallelism? Some problems are just too big

Main memory limits

- ~64GB on a desktop.
- up to 4TB on a server.

So if your problem needs more than that, there are about 4 options:

- Don't compute it
- Shrink memory usage somehow
- Swap using disk
- Use multiple computers

Why Parallelism? Some problems are just too big

Main memory limits

- ∼64GB on a desktop.
- up to 4TB on a server.

So if your problem needs more than that, there are about 4 options:

- Don't compute it
- Shrink memory usage somehow
- Swap using disk
- Use multiple computers

Time limits

There is only so much one core can compute. You might not want to wait 3 months for matching two databases!
Some tasks have a target computation time.

- Weather forecast
- Google search
- Video game

Types of computational parallelism

Circuit level parallelism

Bits of an instructions are decoded in parallel. With SIMD, integers can be added simultaneously.

Instruction level parallelism

Modern processors can execute multiple instruction per cycle.

Shared memory parallelism

Different cores all access the same memory space.
Here often the problem is to make sure that simultaneous execution make sense.

Types of computational parallelism

Circuit level parallelism

Bits of an instructions are decoded in parallel. With SIMD, integers can be added simultaneously.

Instruction level parallelism

Modern processors can execute multiple instruction per cycle.

Shared memory parallelism

Different cores all access the same memory space.
Here often the problem is to make sure that simultaneous execution make sense.

Distributed memory parallelism

With multiple nodes, each has its own memory space.
The problem is often to reduce the amount of communication that the nodes must exchange.

Accelerators

Many systems have devices one can communicate with that provide additional processing power. Often they are parallel themselves.
Modern accelerators: GPU, Xeon Phi, FPGA.

Outline

Measure of Success: speedup

Why not just measure time?

Makes it hard to get insight across different machines.

Definition

$S(N) = \frac{T_1}{T_N}$ where T_1 is sequential time and T_N is time on N processors.
You should use as T_1 the best sequential time you can get.
Ideally, you would want to achieve linear speedup: $S(N) = N$.
If you achieve linear speedup (or close), the code is said to scale well.

Super linear speedup

You can sometimes get $S(N) > N$. It is usually an anomaly that comes from the fact that the parallel algorithm does less computation than the sequential one in that particular case.

Can all computation scale ? (Strong scaling)

Inherently sequential computation

Two ovens won't cook a cake faster.

Can all computation scale ? (Strong scaling)

Inherently sequential computation

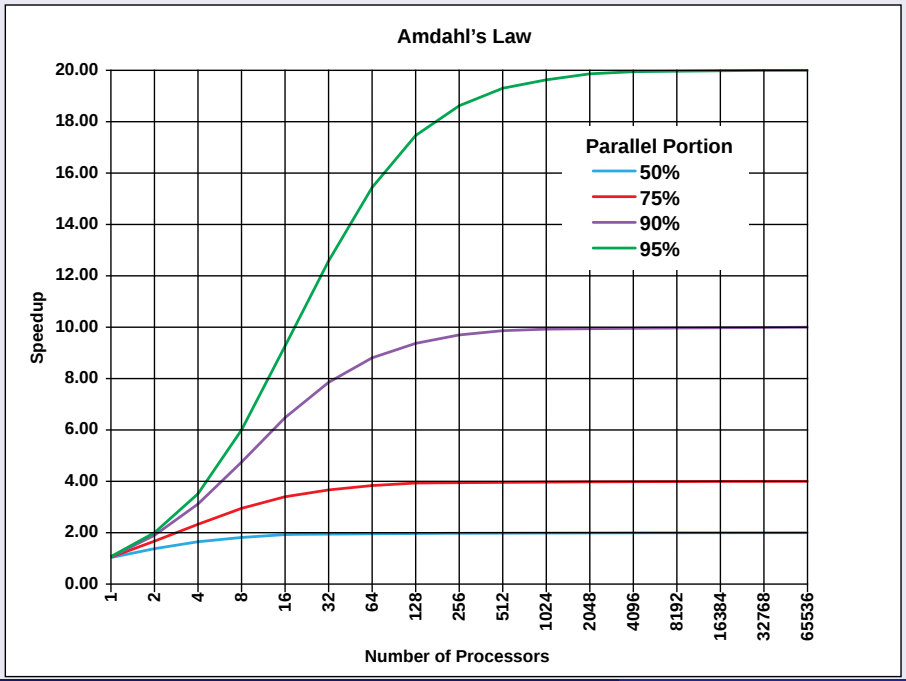
Two ovens won't cook a cake faster.

Amdahl's Law

With a fraction x is serial.

$Time(n) \leq T(1)(x + \frac{1-x}{n})$

$S(N) \leq \frac{1}{x}$



Can all computation scale ? (Strong scaling)

Inherently sequential computation

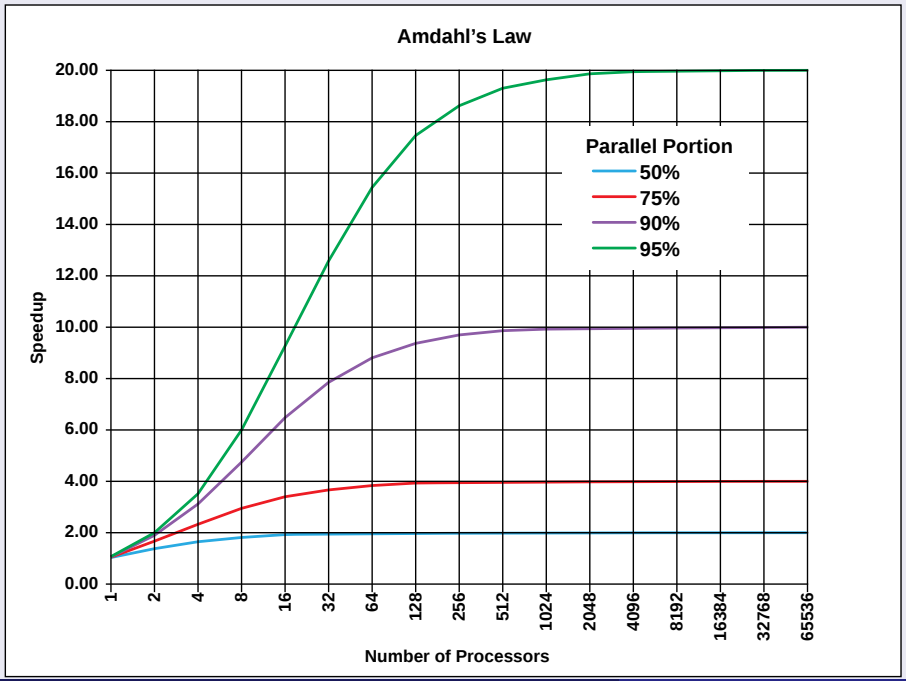
Two ovens won't cook a cake faster.

Amdahl's Law

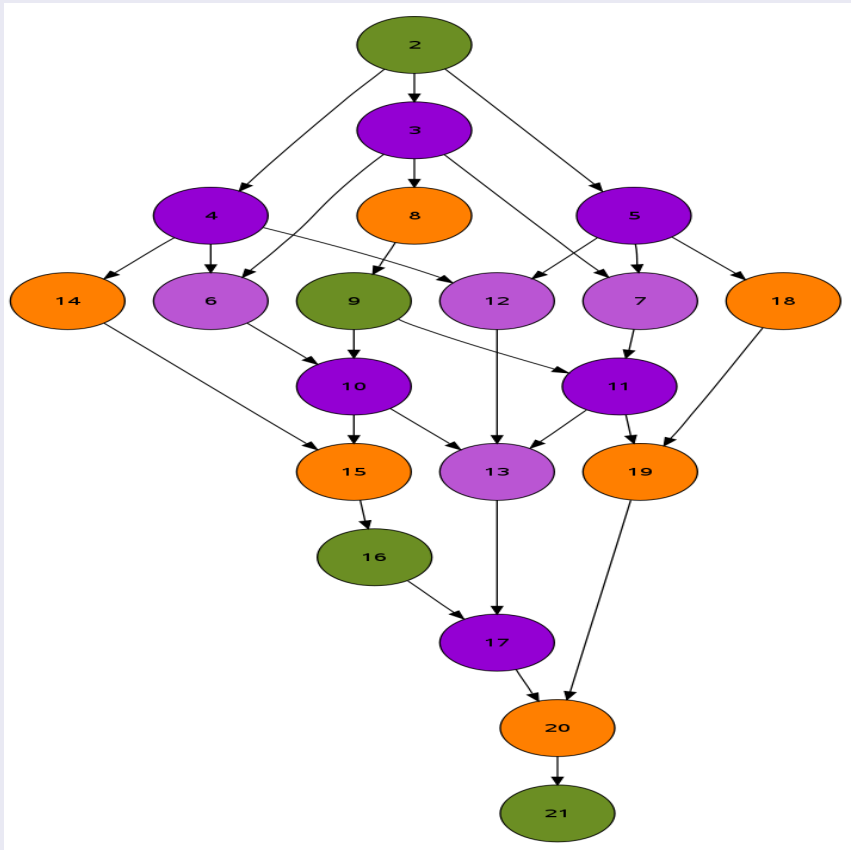
With a fraction x is serial.

$Time(n) \leq T(1)(x + \frac{1-x}{n})$

$S(N) \leq \frac{1}{x}$



Communications/Synchronizatio

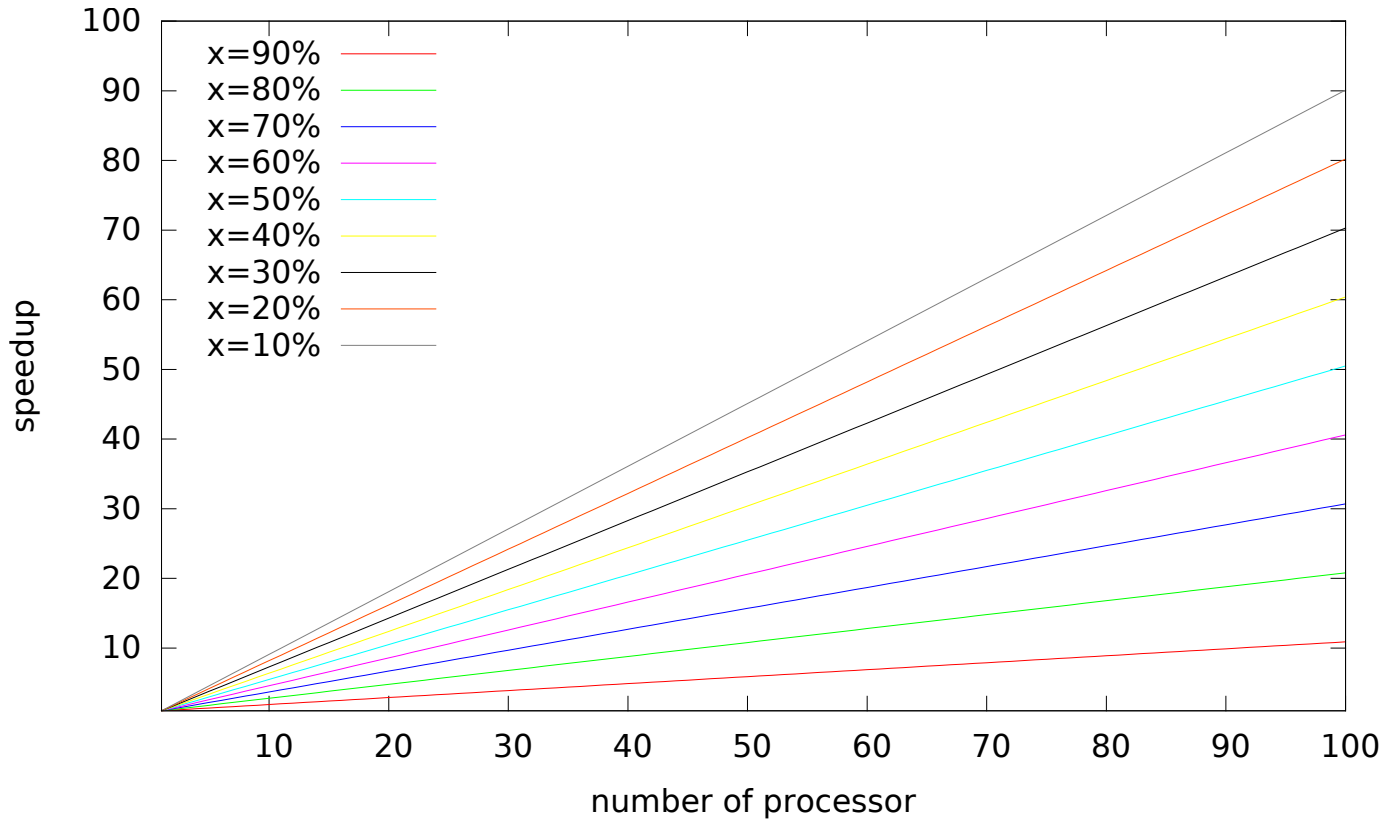


Source: OmpSs manual

Can all computation scale ? (Weak Scaling)

Gustafson's Law

If the sequential part x is a one time fixed overhead, and the rest of the computation scales linearly, then you can process a larger dataset in the same amount of time and achieve a speedup of $S(N) = x + N(1 - x)$.



Two kinds of parallelism experiment

Strong Scaling

In a strong scaling experiment you pick one particular test case that you want to solve. And you see how fast that one can be solved.

+ The best kind of experiment
- Scaling depends on problem size
- How to pick a representative problem?

Weak Scaling

In a weak scaling experiment, you increase the volume of computation (in term of complexity) with the amount of ressource. (Usually report time.)

+ Often easier to achieve scaling
- Not all problems have increasing size

Scaling systems

Metric: Latency

Time it takes to perform one computation.

Metric: Bandwidth

Rate at which one can perform computation

Scaling systems

Metric: Latency

Time it takes to perform one computation.

Metric: Bandwidth

Rate at which one can perform computation

Vertical Scaling

Vertically scaling adds ressources into a single node to increase its computational speed.
The goal is often to improve the latency of the system

Horizontal Scaling

Horizontal scaling adds ressources as additional nodes to increase the computational speed.
The goal is often to improve the bandwidth of the system

Scaling systems

Metric: Latency

Time it takes to perform one computation.

Metric: Bandwidth

Rate at which one can perform computation

Vertical Scaling

Vertically scaling adds ressources into a single node to increase its computational speed.
The goal is often to improve the latency of the system

Horizontal Scaling

Horizontal scaling adds ressources as additional nodes to increase the computational speed.
The goal is often to improve the bandwidth of the system

- Increasing speed limits will decrease commute time.
- Using better servers to decrease query time.

- Adding a lane to the highway will increase flow.
- Adding servers to process more clients.

Outline

External

Git stuff:

- Basic: `https://git-scm.com/docs/gittutorial`
- Complete: `https://www.atlassian.com/git/tutorials/`

C crash course:

- `http://www.mattababy.org/~belmonte/Teaching/CCC/CrashCourseC.html`
- `http://www.cprogramming.com/tutorial/c-tutorial.html`
- `http://www.physics.drexel.edu/courses/Comp_Phys/General/C_basics/c_tutorial.html`

C++ crash course:

- `http://www.cplusplus.com/doc/tutorial/`

Unix:

- `http://www.ee.surrey.ac.uk/Teaching/Unix/`
- Makefiles crashcourse: `http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/`

Cluster:

- URC: `http://urc.uncc.edu/`
- Torque: `http://urc.uncc.edu/faqs/job-scheduling-with-torque/`
- Mamba is based on Copperhead: `http://urc.uncc.edu/faqs/copperhead-user-notes/`
- Status of Mamba:
  `http://urc.uncc.edu/stats/?r=hour&cs=&ce=&m=load_one&c=MAMBA&s=by+name&tab=m&vn=&hide-hf=false`

On parallel computing:

- Wikipedia on parallel computing `https://en.wikipedia.org/wiki/Parallel_computing`
- Amdahl's Law `https://en.wikipedia.org/wiki/Amdahl's_law`
- "Parallel Programming in MPI and OpenMP" by Victor Eijkhout `https://bitbucket.org/VictorEijkhout/parallel-computing-book/raw/94518afb68da8f5af7104c0ce2343a8dc04f3a16/EijkhoutParComp.pdf`