



# EEE4120F



## High Performance Embedded Systems

### Lecture 2: Terms, Amdahl Law & Dealing with reading assignments

The formula for Amdahl's Law is displayed within a decorative, ornate gold picture frame. The formula is: 
$$\text{Speedup}_{\text{parallel}} = \frac{1}{(1-f) + \frac{f}{n}}$$

Lecturer:  
Simon Winberg

*Notes and  
explanations in the  
slide comments*



# Outline for Lecture

- Terms
- Validation vs. Verification
- Commonly used verification methods
- Amdahl's Law
- Prac prep
- Dealing with reading assignments
- Reminder: Quiz#1 next Tuesday!

**TERMS**

A blue arrow pointing to the right, located on the right side of the 'TERMS' button.A blue arrow pointing to the left, located on the left side of the 'TRENDS' button.

**TRENDS**

# Terms



- Golden measure:
  - A (usually) sequential solution that you develop as the 'yard stick'
  - A solution that may run slowly, isn't optimized, but you *know* it gives (numerically speaking) excellent results
  - E.g., a solution written in OCTAVE or MatLab, verify it is correct using graphs, inspecting values, checking by hand with calculator, etc.

Discussed a bit more later...

Don't confuse the term **Golden Measure** with **Golden Ratio** which is solving for  $g$  in  $g^2 = g + 1$  ...  $g = (1 + \sqrt{5})/2 = 1.61803398875$ .

# Terms: golden measure

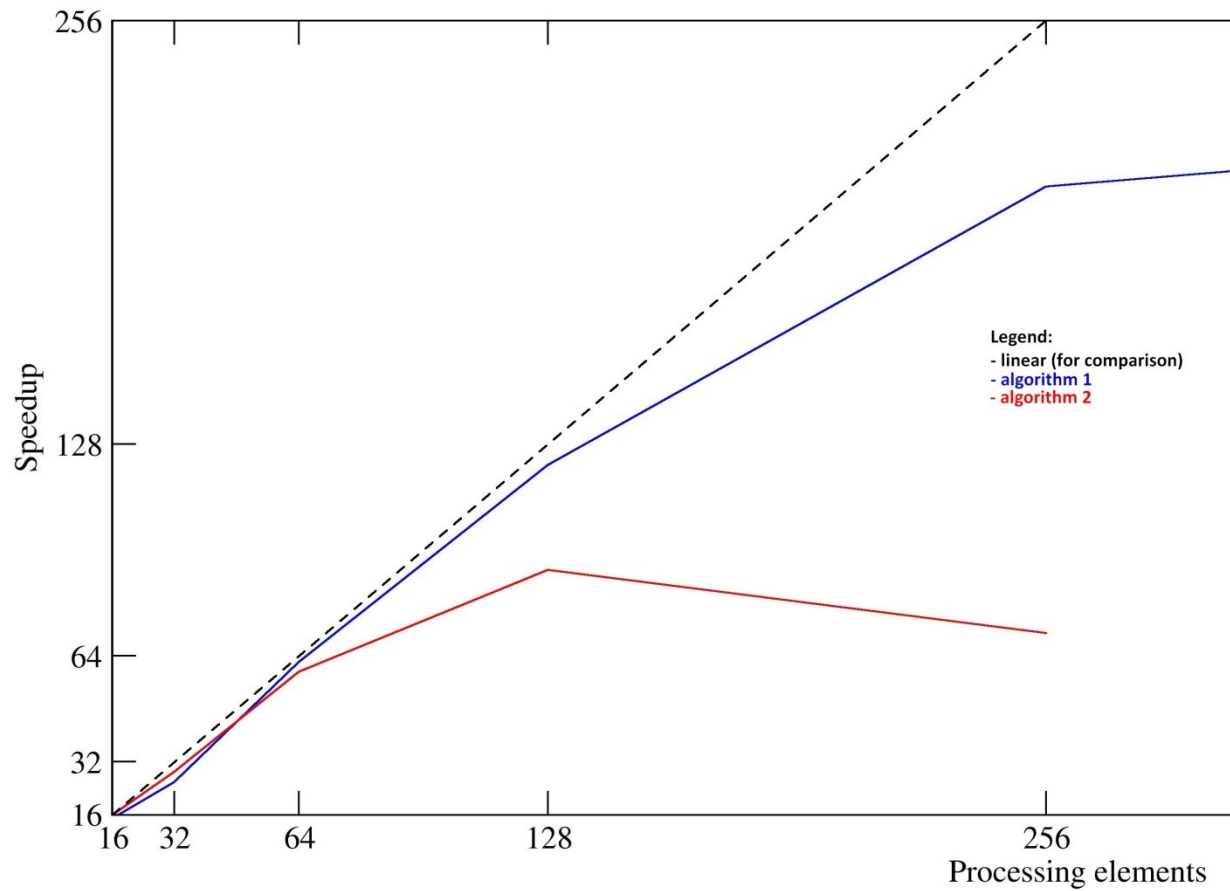
- Sequential / Serial (`serial.c`)
  - A non-parallelized code solution
- Generally, you can call your code solutions `parallel.c` (or `para1.c`, `para2.c` if you have multiple versions)
- You can also include some test data (if it isn't too big,  $<1\text{Mb}$ ), e.g. `gold.csv` or `serial.csv`, and `para1.csv`

# Speed-up

- $\text{Speed-up} = T_{p1} / T_{p2}$
- Where  $T_{p1}$  = Run-time of original (or non-optimized) program
- $T_{p2}$  = Run-time of optimised program
- Best practice for measuring speedup:
  - **Run the program more than one**, discarding the first result (where the cache, etc. is getting 'warmed up')
  - To be precise should indicate results from **when system wasn't 'warmed up'**\*

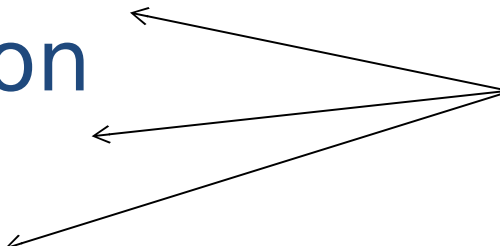
\* (which can be simulated by running a whole lot of other things like CounterStrike, Half-Life, maybe some Solitaire\* for good measure -- but more seriously you could write your own 'cache cleaning program')

# Speed-up graphs




# Other Important Terms

- Verification
- Validation
- Testing
- Correctness proof



These terms are not merely theoretical terms to remember, but relate directly to your project.



Not something done in the project (but if you want to, you can experiment with doing a correctness proof if you are keen)



# Verification and Validation

◦ Two terms you should already know...  
(V&V)

## ◦ Verification

- “Are we building the product right?”
- Have we made what we understood we wanted to make?
- Does the product satisfy its specifications?

## ◦ Validation

- “Are we building the right product?”
- Does the product satisfy the users’ requirements

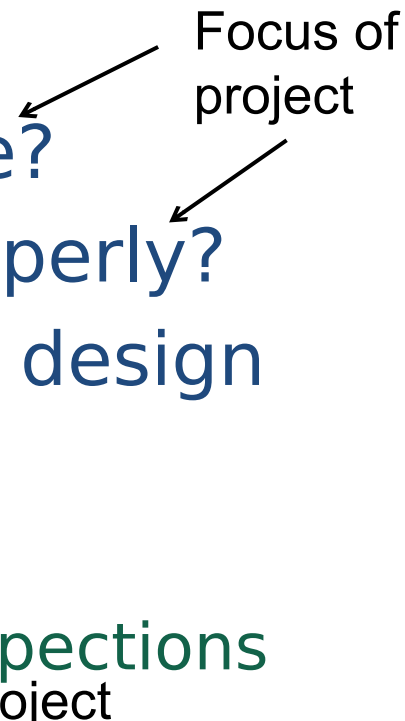
## ◦ Verification before validation (at least in dress)

While it would be nice to validate (seeing that the users are happy) before verifying (checking the specs), doing so would mean your final design might not match the specifications (which could open the door to legal problems). Obviously this often doesn’t happen because in practice you want to make sure the client is happy and there might not be time for proper validation.

# Verification before validation

- The RC engineer (i.e., you) are effectively designing both custom hardware and custom software for the RC platform
- Before attempting to make claims about the *validity* of your system, it's usually best practice to establish your own (or team's) confidence in what your system is doing, i.e. be sure that:
  - The custom hardware working;
  - The software implementation is doing what it was designed to do; and
  - The custom software runs reliably on the custom hardware.

# Verification

- Checking plans, documents, code, requirements and specifications
  - Is everything that you need there?
  - Algorithms/functions working properly?
  - Done during phase interval (e.g., design => implementation)
  - Activities:
    - Review meetings, walkthroughs, inspections
    - Informal demonstrations
- 
- Focus of project
- Focus of project
- The diagram consists of two arrows pointing from the text 'Focus of project' to specific items in the list. The first arrow points to 'Is everything that you need there?'. The second arrow points to 'Review meetings, walkthroughs, inspections'.

## Commonly used verification methods

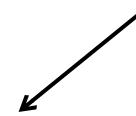
1. Dual processing, producing two result sets
  1. One version using PC & simulation only;
  2. Other version including RC platform
2. Assume the PC version is the correct one (i.e., the gold measure)
3. Correlate the results to establish correlation coefficients

(complex systems may have many different sets of possibly multidimensional data that need to be compared)

The correlation coefficients can be used as a kind of 'confidence factor'

# Validation

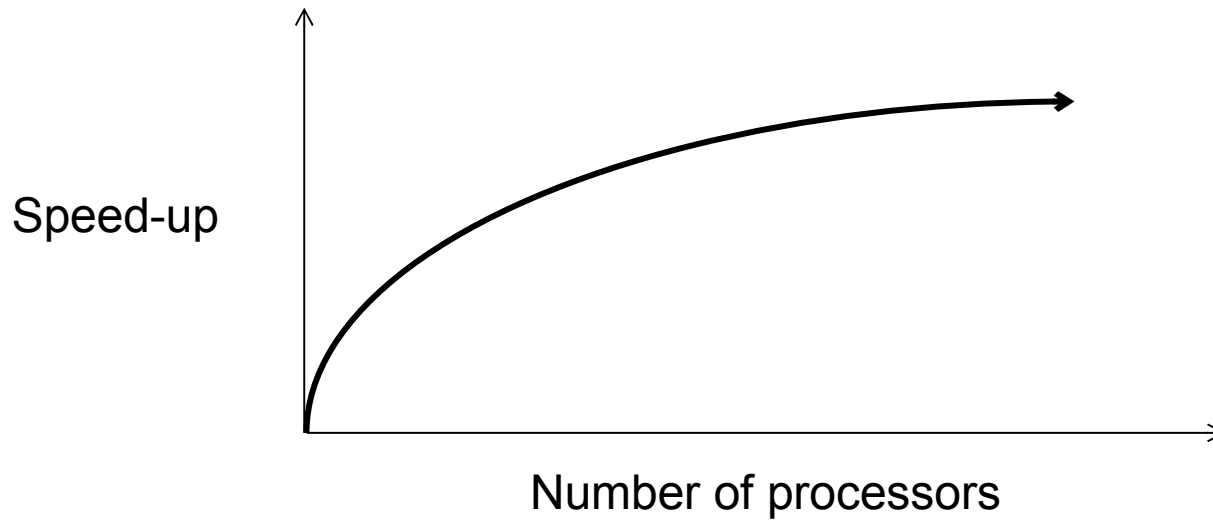
Focus of  
project



- Testing of the whole product / system
- Input: checklist of things to test or list of issues that need to have been provided/fixed
- Towards end of project
- Activities:
  - Formal demonstrations
  - Factory Acceptance Test

# Testing and Correctness proofs

- Testing
  - Generally refers to aspects of *dynamic validation* in which a program is executed and the results analysed
- Correctness proofs / formal verification
  - More a mathematical approach
  - Exhaustive test => specification guaranteed correct
  - Formal verification of hardware is especially relevant to RC. Formal methods include:
    - Model checking / state space exploration
    - Use of linear temporal logic and computational tree logic
    - Mathematical proof (e.g. proof by induction)



# Amdahl's Law

EEE4084F

# Amdahl's Law: **History**



- The guy: Gene Amdahl
  - Was chief architect for IBM's first mainframe series of computers
  - Founder of Amdahl Corporation
- Amdahl found stringent restrictions on the speedup possible for given parallelized tasks.
- These observations packaged as:  
*Amdahl's Law*



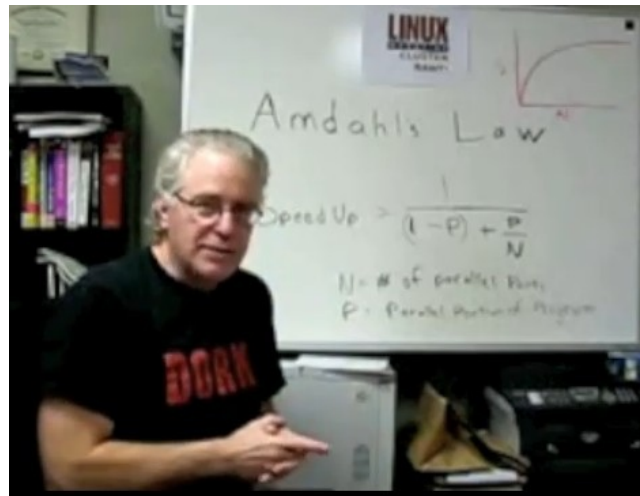
# Essentials of Amdahl's Law

- Be aware that a computer program to run on a parallel computer \* pretty much always has a part that is sequential, which can run on only one core, and a part that is parallel, that can be split between available cores
- But, let's make things more fun (and hope you then understand Amdahl's better) by proceeding to video linked on next slide.
- Comments on slide 19 elaborates further.

\*well, we're thinking here computers with one or more CPUs for their processing



# Video Clip...



Linux Magazine Video: Understanding Parallel Computing: Amdahl's Law

<https://www.youtube.com/watch?v=WdRiZEwBhsM>

Amdahl.flv

# Amdahl's Law

- Define  $f$  as: fraction of computation that can be parallelized (ignoring scheduling overhead)
- Then  $(1 - f)$  is the fraction that is sequential
- Define  $n$  = no. processors for parallel case
- The maximum speed-up achievable is:  
$$\text{Speedup}_{\text{parallel}} = \frac{1}{(1 - f) + \frac{f}{n}}$$

## Amdahl's Law: Alternate Representation

$P$  = expected performance improvement

$E^u$  = Execution time on a uniprocessor (serial)

$E^p$  = Execution time on a number of processors (parallel)

$n$  = number of processors

$S$  = fraction of time spent in the sequential time

$$P = \frac{E^u}{E^p} = \frac{E^u}{SE^u + \frac{(1-s)}{N} E^u} = \frac{1}{s + \frac{1-s}{N}}$$

# Homework task

Watch 2<sup>nd</sup> part of Amdahl's law video



[Understanding Parallel Computing \(Part 2\): The Lawn Mower Law](#)  
[LinuxMagazine](#)

Amdahl2.flv

*Prelude towards YODA project with is more a Term 2 activity*

# YODA Project Topics

Current projects listing:

***New link added soon***

*The focus is around a Verilog module you would implement, but that module needs to be hooked up to the carrying system*



Call for  
**PROPOSALS**  
now open!

To submit your own proposal please using this structure and send to me as an email:

Pxx: <acronym> - <proj title>  
<brief overview>

<possibly code snippet to explain the algorithm is relevant>

<any added wishlist/upgrade items>

Inputs: <interface to your module>

Outputs: < interface out of your module >

# An example YODA Project

## Topic: SALG - Selection Address List Generator

- SALG sent starting address of a table in memory.
- Table has n elements.
- Each element of the table is in the form TableElement below.
- The SALG is sent a second address, inds, use to store the addresses (i.e., the starting address of the relevant record field) that matches the selection criteria (which is hardcoded).

```
void SALGA(TableElement* table,                struct TableElement{
    unsigned* inds, unsigned n){               unsigned key;
    unsigned n_inds = 0;                       byte record[rsize];
    for (int i = 0; i < n; i++) {               };
        if (table[i]->key & 1) } ← Choose your own selection criteria
            inds[n_inds++] =
                &table[i]->record[0];
    }
    inds[n_inds] = 0; // set last one to null to indicate end of list
}
```

# Soon over to Prac1!





closing  
remarks & reminders...



## Dealing with reading assignments

- You are suppose to read (at least speed read) the readings assigned as recommended – the others are more for deepening your insights into an area

```
open("L01 Berkeley 2006 - Landscale of  
Parallel Computing Research.pdf") do pg1-8  
... you might need to do more readings than that
```

# Assigned Reading

## For Tuesday next week...

### ***S1 - Landscape of parallel computing research: a view from Berkeley***

Find it in: Abathuba / Readings  
listed in Readings resources

There will be a short quiz, and I will follow that with solutions and a short seminar on the paper to explain its highlights.  
This paper is usually in the final exam syllabus.

# End of Lecture 2

FREE Creative Commons License

JAZZY FRENCHY

Music: <https://www.bensound.com>



## ***Disclaimers and copyright/licensing details***

I have tried to follow the correct practices concerning copyright and licensing of material, particularly image sources that have been used in this presentation. I have put much effort into trying to make this material open access so that it can be of benefit to others in their teaching and learning practice. Any mistakes or omissions with regards to these issues I will correct when notified. To the best of my understanding the material in these slides can be shared according to the Creative Commons “Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)” license, and that is why I selected that license to apply to this presentation (it’s not because I particulate want my slides referenced but more to acknowledge the sources and generosity of others who have provided free material such as the images I have used).

### ***Image sources:***

Wikipedia (open commons)

<http://www.flickr.com>

<http://pixabay.com/>

<https://publicdomainvectors.org>

