HPES
High Performance Embedded Systems

EEE4120F

University of Cape Town • iYunivesithi yaseKapa • Universiteit van Kaapstad

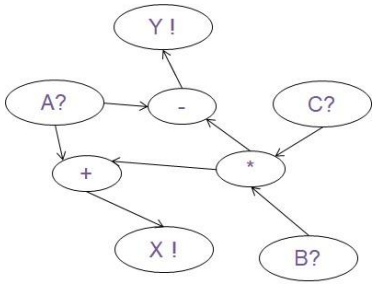
SPES BONA

High Performance Embedded Systems

Lecture 7:

Parallel Computing Fundamentals

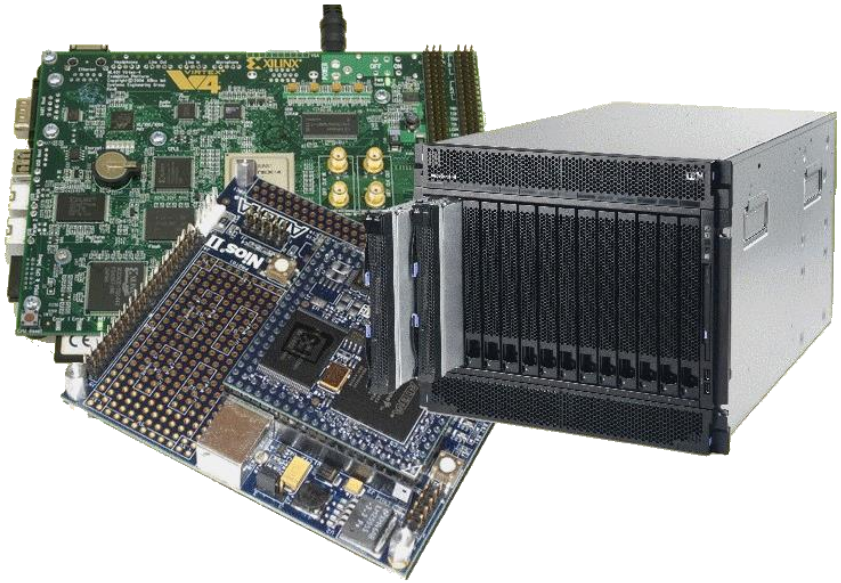
Presented by
Simon Winberg



Planned to be double period lecture

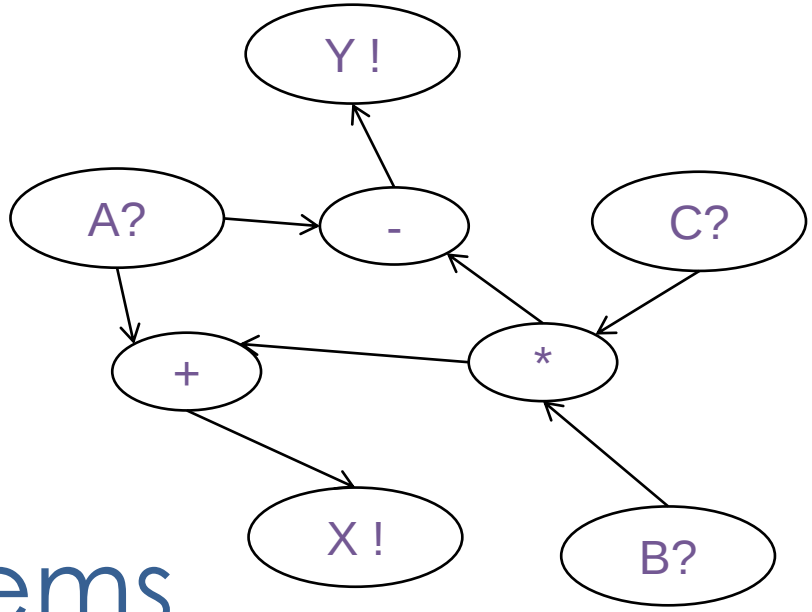
Lecture Overview

- Parallel Computing Fundamentals
- Large Scale Parallelism
- Mainstream parallel
- Classic Parallel approaches
- Flynn's Taxonomy
- Some Calculations
  - Effective parallelism
  - Parallel Efficiency
  - Gustafson's Law



Parallel Systems

EEE4120F: Digital Systems



Remember the spatial computing paradigm??

Computation Methods

Reminder ... this term

Hardware

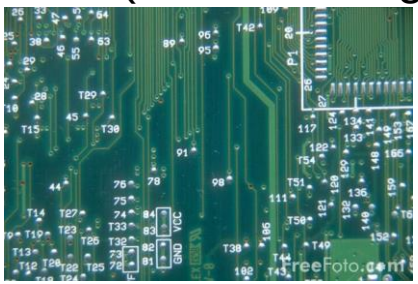
e.g. PCBs, ASICs

Advantages:

- High speed & performance
- Efficient (possibly lower power than idle processors)
- Parallelizable

Drawbacks:

- Expensive
- Static (cannot change)



Reconfigurable Computer

e.g. IBM Blade, FPGA-based computing platform

Advantages:

- Faster than software alone
- More flexible than software
- More flexible than hardware
- Parallelizable

Drawbacks:

- Expensive
- Complex (both s/w & h/w)

Software Processor

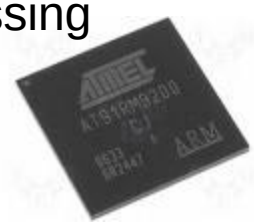
e.g. PC, embedded software on microcontroller

Advantages:

- Flexible
- Adaptable
- Can be much cheaper

Drawbacks:

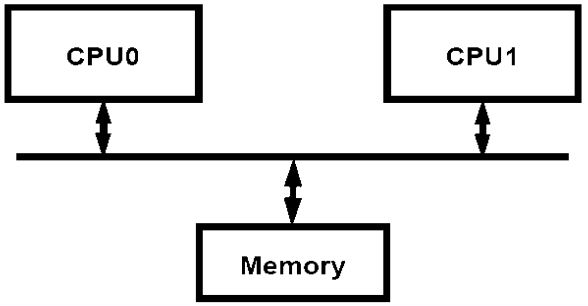
- The hardware is static
- Limit of clock speed
- Sequential processing



Mainstream parallel computing

- *Most* server class machines today are:
  - PC class SMP's (Symmetric Multi-Processors *)
  - 2, 4, 8 processors - cheap
  - Run Windows & Linux
- Delux SMP's
  - 8 to 64 processors
  - Expensive:

    16-way SMP costs ≈ 4 x 4-way SMPs



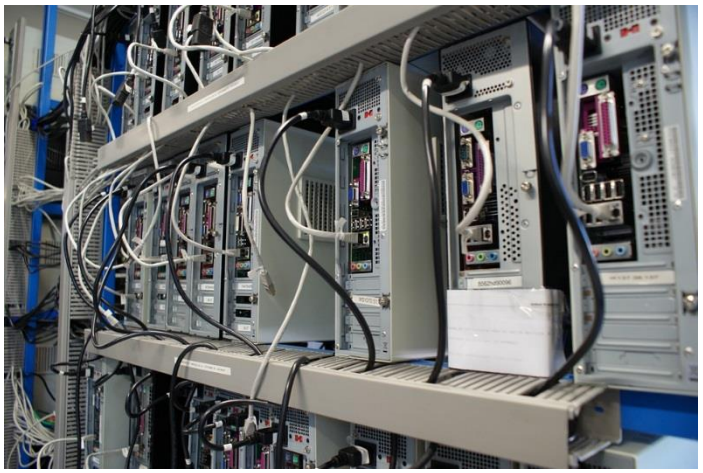
SMP offers all processors and memory on a common front side bus (FSB –bus that connects the CPU and motherboard subsystems).

- Applications: Databases, web servers, internet commerce / OLTP (online transaction processing)
- Newer applications: technical computing, threat analysis, credit card fraud...

* Also termed "Shared Memory Processor" (but you might get 0 for giving this alternate term in a test)

Large scale parallel computing systems

- Hundreds of processors, typically as SMPs clusters
- Traditionally
  - Often custom built with government funding (costly! 10 to 100 million USD)
    - National / international resource
    - Total sales tiny fraction of *PC server* sales
  - Few independent software developers
  - Programmed by small set of majorly smart people
- Later trends
  - Cloud systems
  - Users from all over
  - E.g. Amazon EC, Microsoft Azure



Large scale parallel computing systems

- Some application examples
  - Code breaking (CIA, FBI)
  - Weather and climate modeling / prediction
  - Pharmaceutical – drug simulation, DNA modeling and drug design
  - Scientific research (e.g., astronomy, SETI)
  - Defense and weapons development
- Large-scale parallel systems are often used for modelling

Digital Accelerators: New Practice

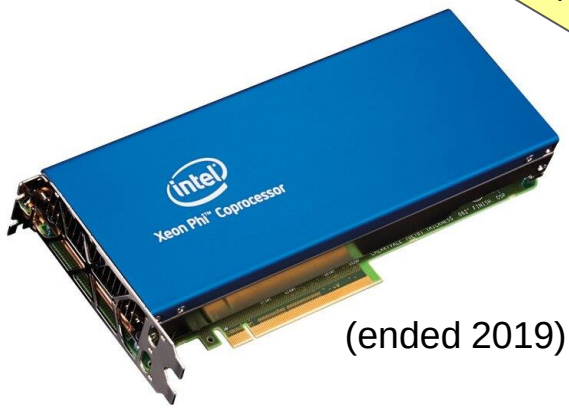
Digital accelerator cards (including GPGPUs) are increasing in popularity, including in data centres.

Term 2: RC / Digital Accelerators

(launched end 2018)

Xilinx Alevo: Adaptable Accelerator Cards for Data Centres
https://www.xilinx.com/products/boards-and-kits/alveo.html

Program with OpenCL or Xilinx's owns accelerator design suite.

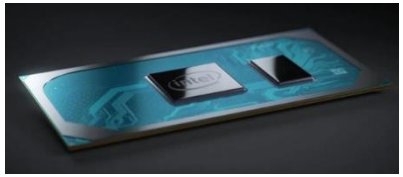


(ended 2019)

HP Intel 5110P Xeon Phi Coprocessor Kit
Intel Xeon Phi
Where and why it's no longer being made, replaced by Xeon Scalable Platform

Replaced by (from 2020)

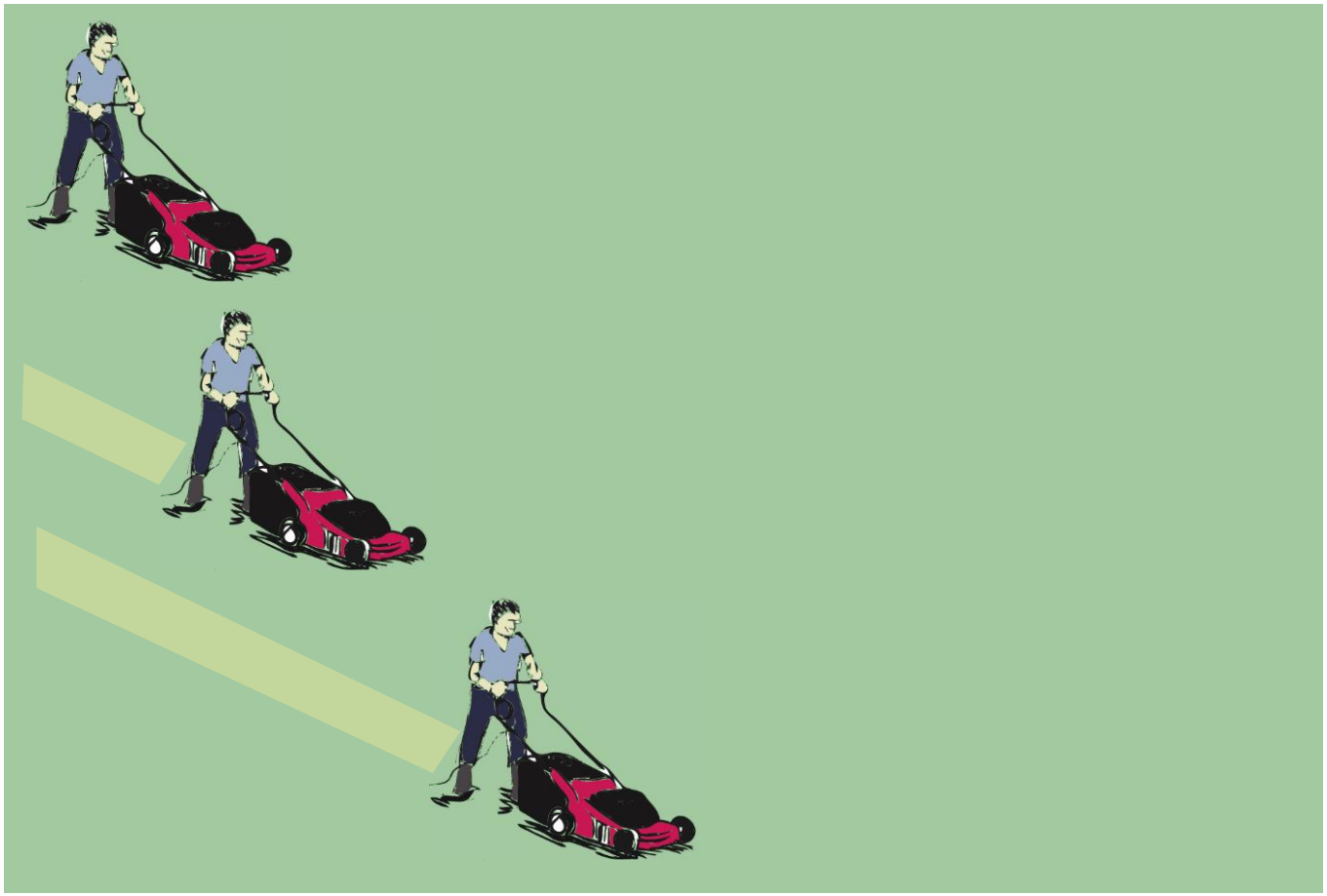
Intel Ice Lake (10 nm process) 10th generation core, successor for Xeon Phis.



Core i7 1068G7 (2020 debut) 4 CPU cores + 64 Iris+ GPCPU cores

And… some of this it may involve designing specialized compute architectures for the need (using a combination of languages and tools e.g. OpenCL / Verilog, C, R, etc.)

* What was it said in the "Berkeley Landscape" … "Small is beautiful" and "manycore is the future of Computing".



Classic Parallel

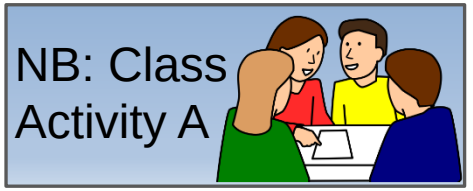
EEE4120F

Classic techniques for parallel programming*

- Single Program Multiple Data (SPMD)
  - Consider it as running the same program, on different data inputs, on different computers (possibly) at the same time
- Multiple Program Multiple Data (MPMD)
  - Consider this one as running the same program with different parameters settings, or recompiling the same code with different sections of code included (e.g., uisng #ifdef and #endif to do this)
- Following this approach performance statistics can be gathered (without necessarily any parallel code being written) and then evaluated after the effect to deem the feasibility of implementing a parallel (e.g. actual pthreads version) of the program.

Can consider the SPSD (i.e. Single Program Single Data) as the case where the whole program is run once on all the data.

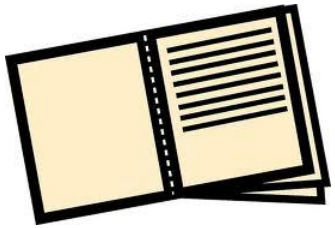
*Informally know as the lazy parallel programming model.



Terms

EEE4120F

Terms (reminders)



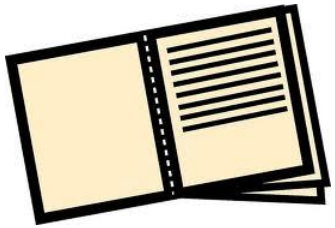
- Observed speedup =

$$\frac{\text{Wallclock time initial version}}{\text{Wallclock time refined version}} = \frac{\text{Wallclock time sequential (or } \textit{gold}\text{)}}{\text{Wallclock time parallel version}}$$

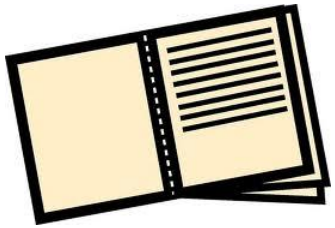
- Parallel overhead:
  - Amount of time to coordinate parallel tasks (excludes time doing useful work).
  - Parallel overhead includes operations such as: Task/co-processor start-up time, Synchronizations, communications, parallelization libraries (e.g., OpenMP, Pthreads.so), tools, operating system, task termination and clean-up time

The parallel overhead of the lazy parallel model could clearly be extreme, considering that is would rely on manual intervention to (e.g.) partition and prepare the data before the program runs.

Some terms



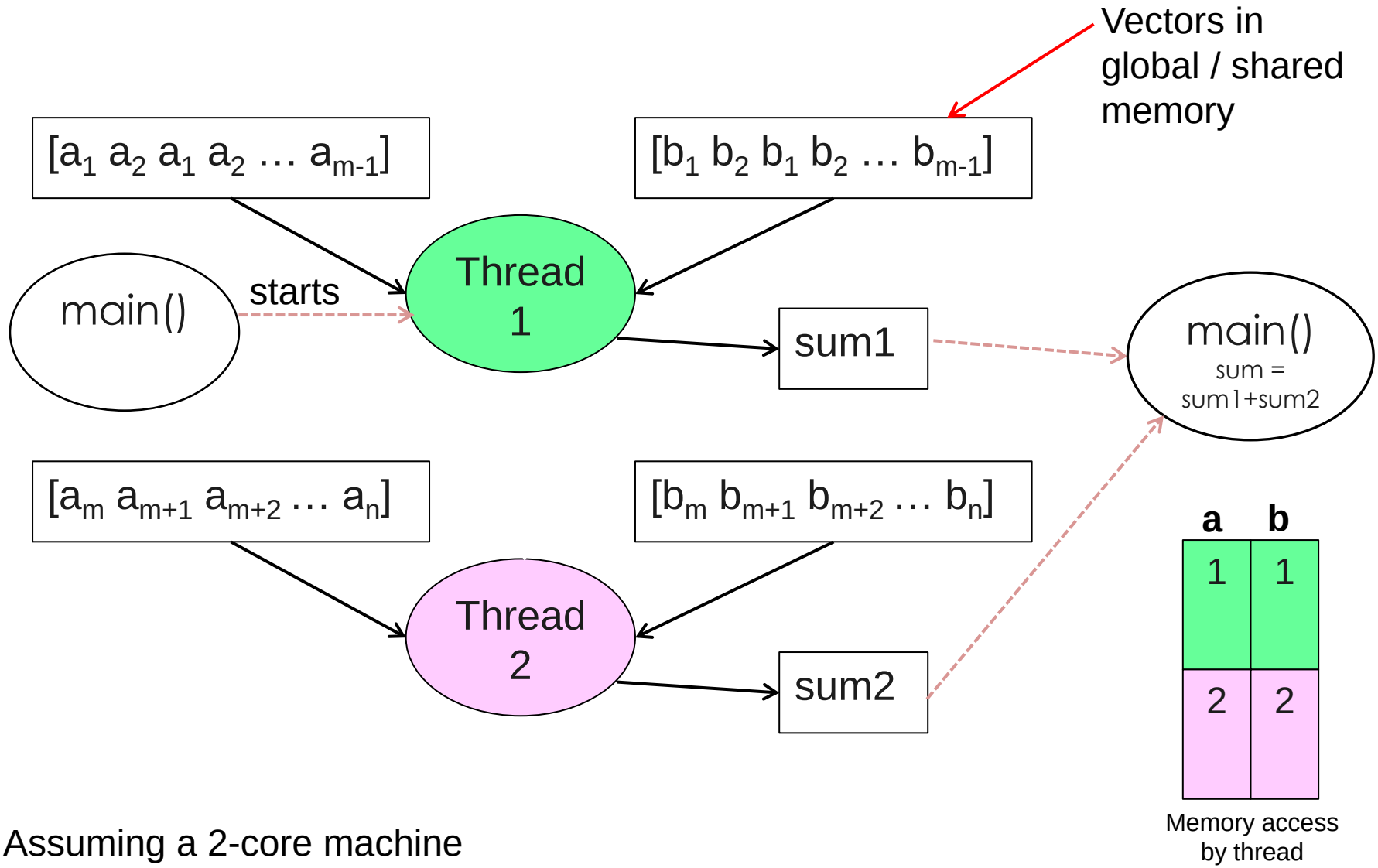
- Embarrassingly Parallel
  - Simultaneously performing many similar, independent tasks, with little to no coordination between tasks.
- Massively Parallel
  - Hardware that has very many processors (execution of parallel tasks). Can consider this classification of 100 000+ parallel tasks.
- { Stupidly Parallel }
  - While this isn't really an official term it typically relates to instances where a big (and possibly very complex) coding effort is put into developing a solution that in practice has negligible savings or worse is a whole lot slower (and possibly more erroneous/buggy) than if it was just left as a simpler sequential implementation.



Introducing to some important terms related to shared memory

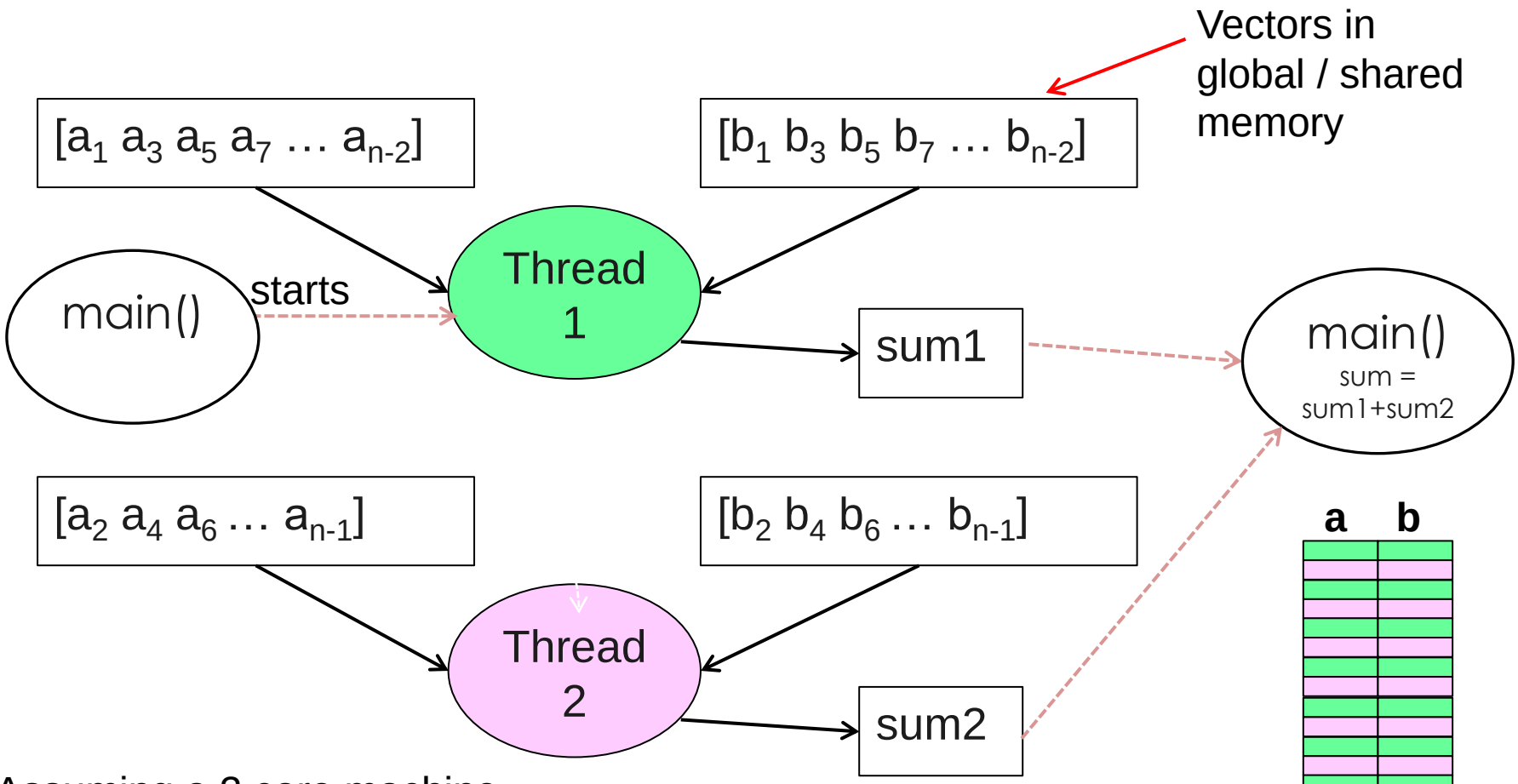
Using scalar product pthreads type implementation for scenarios…

Partitioned memory



Assuming a 2-core machine

Interlaced* memory



Assuming a 2 core machine

NB: benefits of interleaving as a means to improve robustness and access of memory
interlacing vs. interleaving : see comments

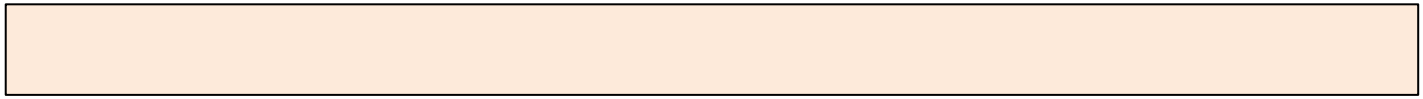
Memory access by thread

* 'Data striping', 'interleaving' and 'interlacing' usually means the same thing but not always.
See further explanation between these on: https://wikidiff.com/interleave/interlace

Memory Partitioning Terms

SUMMARY

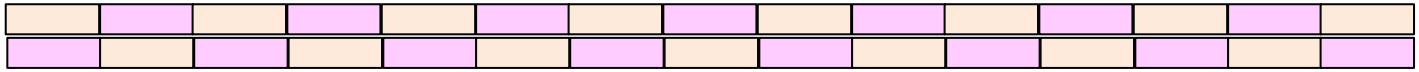
Contiguous



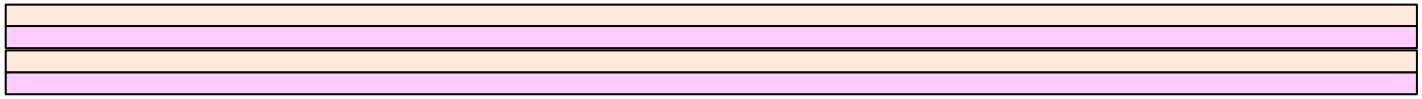
Partitioned (or separated or split)

Interleaved/interlaced (or alternating or data striping)

Interleaved – small stride (e.g. one word stride)



Interleaved – large strides (e.g. row of image pixels at a time)



(the ‘stride’ in data interleaving or data striping refers to the size of the blocks alternated, generally this is fixed but could be changeable, e.g. the last stride might be smaller to avoid padding data.)

EEE4120F

Flynns Taxonomy of Processor Architectures

Flynn's taxonomy

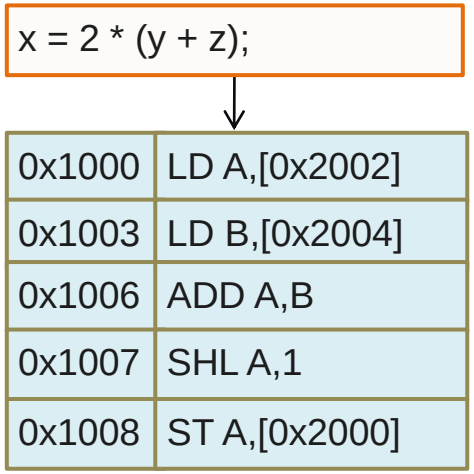
- Flynn's (1966) taxonomy was developed as a means to classify parallel computer architectures
- Computer system can be fit into one of the following four forms:

SISD Single Instruction Single Data	**SIMD** Single Instruction Multiple Data
MISD Multiple Instructions Single Data	**MIMD** Multiple Instructions Multiple Data

Not to be confused with the terms of "Single Program Multiple Data (SPMD)" and "Multiple Program Multiple Data (MPMD)" mentioned earlier.

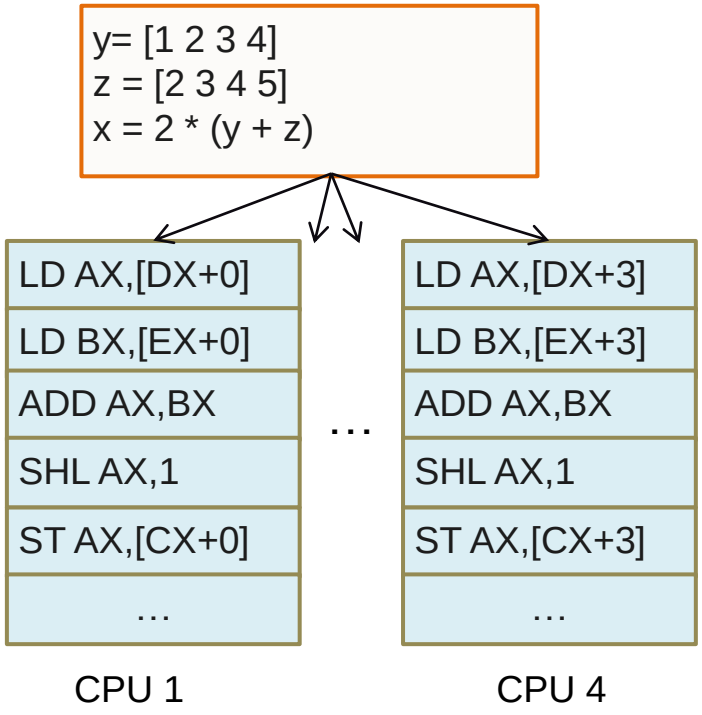
Single Instruction Single Data (SISD)

- This is (obviously) the classic von Neumann Computer: serial (not parallel) computer, e.g.:
  - Old style single core PC CPUs, e.g. i486
- Single instruction →
  - One instruction stream acted on by the CPU during any one clock cycle
- Single data →
  - Only one input data stream for any one clock cycle
- Deterministic execution



Single Instruction Multiple Data (SIMD)

- A form of parallel computer
  - Early supercomputers used this model first
  - Nowadays it has become common – e.g., used in modern computers on GPUs
- Single instruction →
  - All processing units execute the same instruction for any given clock cycle
- Multiple data →
  - Each processing unit can operate on a different data element



Single Instruction Multiple Data (SIMD)

- Runs in lockstep (i.e., all elements synchronized)
- Works well for algorithms with a lot of regularity; e.g. graphics processing.
- Two main types:
  - Processor arrays
  - Vector pipelines
- Still highly deterministic (know the same operation is applied to specific set of data – but more data to keep track of per instruction)

Single Instruction Multiple Data (SIMD) Examples

- Vector pipelines
  - IBM 9000, Cray X-MP, Fujitsu vector processor, NEC SX-2, Hitachi S820, ETA10

←Cray X-MP

- Processor arrays
  - Thinking Machine CM2, MasPar MP-1 & MP-2, ILLIAC IV

←MasPar MP-1

- Graphics processor units usually use SIMD

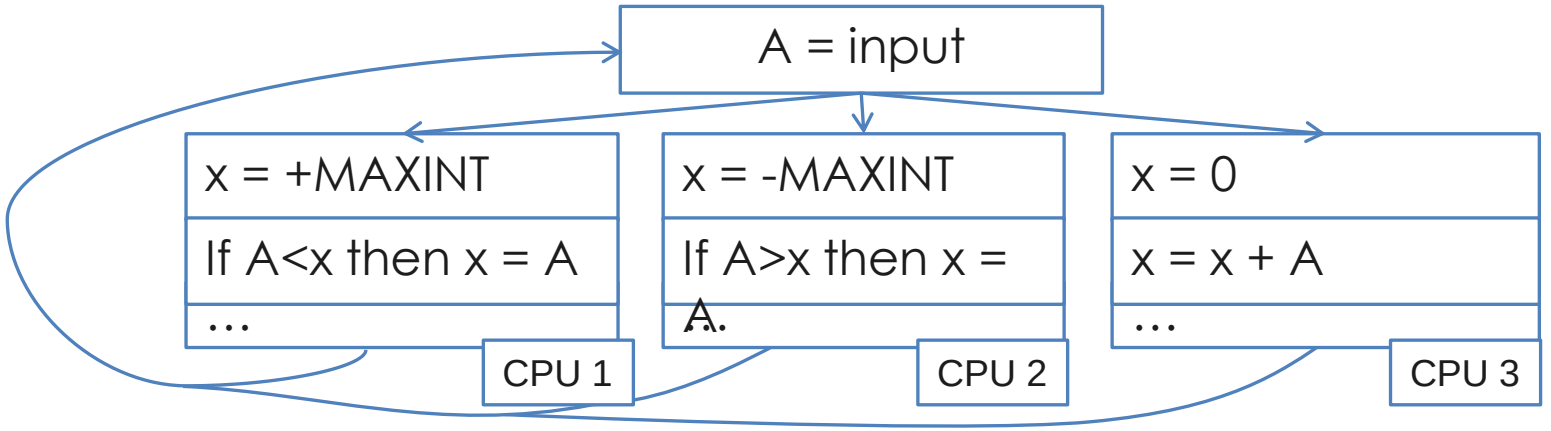
Multiple Instruction Single Data (MISD)

- Single data stream fed into multiple processing units
- Each processing unit works on data independently via independent instruction streams
- Few actual examples of this class of parallel computer have ever existed

Multiple Instruction Single Data (MISD) Example

- Possible uses? Somewhat intellectual?
  - Maybe redundant! (see next slide)
- Possible example application:
  - Different set of signal processing operations working the same signal stream

Example:
Simultaneously find the min and max input, and do a sum of inputs.



Multiple Instruction Multiple Data (MIMD)

- The most common type of parallel computer (most late model computers, e.g. Intel Core Duo, in this category)
- Multiple Instruction →
  - Each processor can be executing a different instruction stream
- Multiple Data →
  - Every processor may be working with a different data stream
- Execution can be asynchronous or synchronous; non-deterministic or deterministic

Multiple Instruction Multiple Data (MIMD)

- Examples
  - Many of the current supercomputers
  - Networked parallel computer clusters
  - SMP computers
  - multi-core PCs
- MIMD architectures could include all the other models. e.g.,
  - SISD – just one CPU active, others running NOP
  - SIMD – all CPUs load the same instruction but apply to different data
  - MISD – all CPUs load different instructions but apply it to the same data

AMD Opteron

IBM BlueGene

EEE4120F

Maximum Effective Parallelism

Significant Tradeoff:

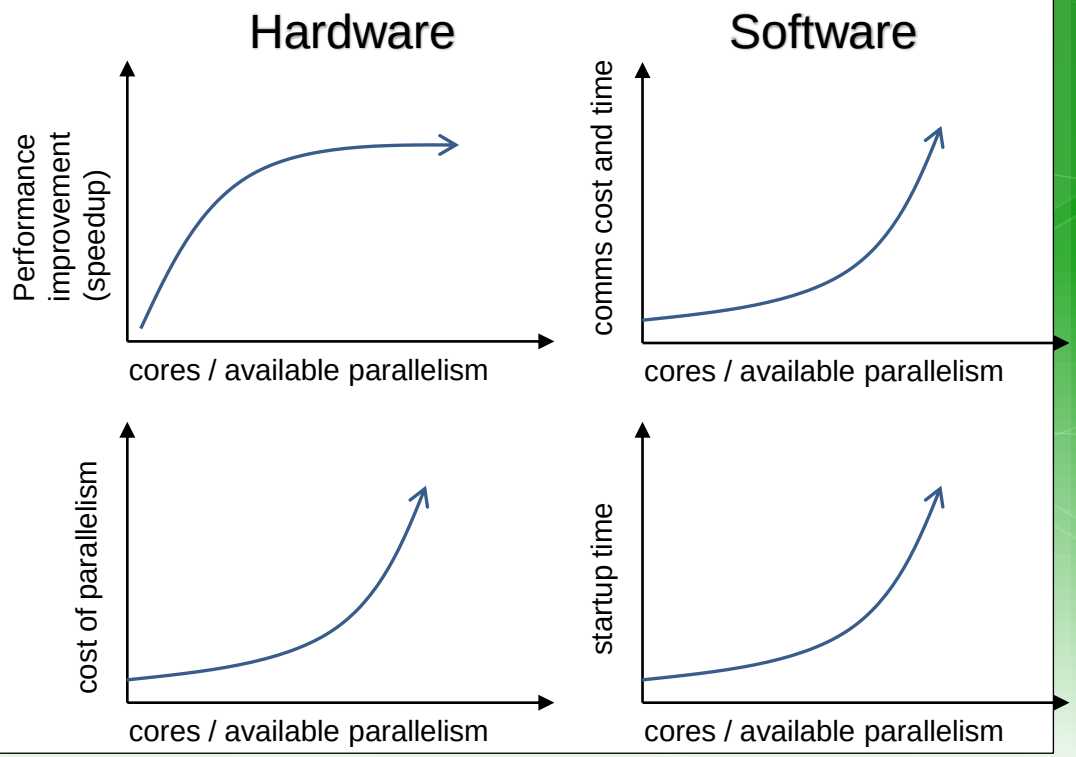
Infrastructure & Setup Cost
vs.
Effective Parallelism

Consideration for
- hardware
- software

Maximum Effective Parallelism:
This refers to the point of the number of cores / amount of parallelism beyond which additional parallelism provides no further benefit or (worse) may reduce performance.

Significant Tradeoff:

Infrastructure & Setup Cost vs. Effective Parallelism



Maximum Effective Parallelism:
This refers to the point of the number of cores / amount of parallelism beyond which additional parallelism provides no further benefit or (worse) may reduce performance.

Calculation Example:
Maximum Effective Parallelism

- Remember from Amdahl:
  - $S = T_S / T_P$ (i.e. sequential time over parallel time)
- You can use these equations to approximate behaviour of modelled systems.
- E.g.: to represent speedup, comms overhead, cost of equipment, etc. and use calculations to estimate optimal selections.
  (simple example to follow)

Learning Activity

- Quick estimation for maximum effective parallelism

EEE4120F High Performance Embedded Systems

Class Activity: Estimating the Maximum Effective Parallelism

A program that involves both processing and comms between threads has been developed that can use from 1 to 64 threads (there are 64 cores on the machine). When run with one thread, the system utilizes 100ms processing time and 0ms comms time, for a total execution time of 100ms. For two threads, the processing time halves taking 50ms but the comms take 10ms, for a total run time of 60ms. Each subsequent time the number of threads is doubled, the processing time also halves but likewise the comms time doubles. What is the *maximum effective parallelism* in number of threads for this program? (You can provide a rough answer as a power of two closest to the optimal number of processors).

Space for your working:

Solution follows shortly!

Please try it for your self, or work with a buddy to think about how to respond to this question.

Learning Activity Answer

N	Proc (ms)	Comms (ms)	Total (ms)
1	100.000	0	100.000
2	50.000	10	60.000
4	25.000	20	45.000
8	12.500	40	52.500
16	6.250	80	86.250
32	3.125	160	163.125
64	1.563	320	321.563

Could do some rough calculations (or use binary search)…

check:

N=2,

N=64,

N=8,

N=4 ✓

Multi processors and sequential setup time

2nd part of Amdahl's law video

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

If you haven't already watched this please do!

Amdahl2.flv

Parallel Efficiency (E_{par})

- We can think of the efficiency of various things, e.g. power systems, motors, heaters etc. We can also think of the efficiency of parallel computing…
- Parallel Efficiency
  - Defined as the:

ratio of speedup (S) to the number of processors (p)

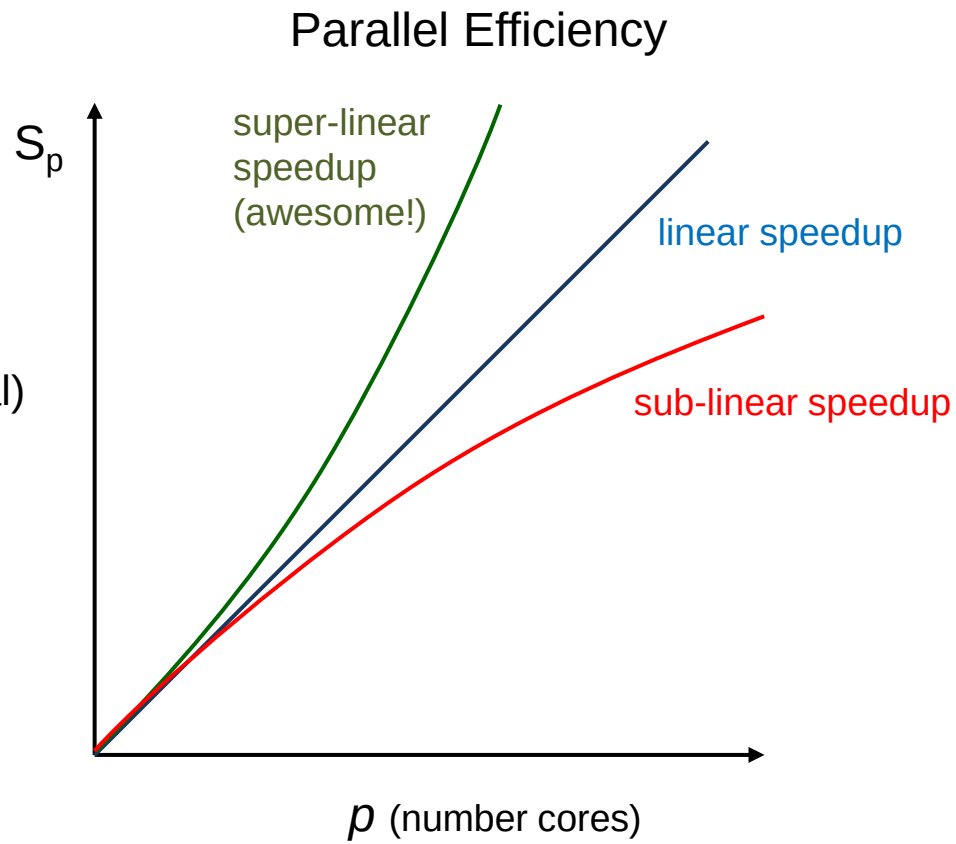
$$E = \frac{S}{p} = \frac{T_s}{pT_p}$$

- Parallel Efficiency measures the fraction of time for which a processor is usefully used.
- An efficiency of 1 is ideal (>1 means you may be harnessing energy from another dimension ☺)

Parallel Efficiency (E_{par})

$$E = \frac{S}{p} = \frac{T_S}{pT_p}$$

- p = number processors
- T_s = exe time of the seq. alg.
- T_p = exe time of the parallel alg. with p processors used
- S_p= speedup (linear being the realistic ideal)



Gustafson's Law *

- Gustafson's law =
  - The theoretical speedup in latency of the execution of a task at fixed execution time that can be expected of a system whose resources are improved.
  - A follow-up to Amdahl's Law

formulation: Speedup *S* gained by *N* processors (instead of just one) for a task with a serial fraction *s* (not benefiting from parallelism) is as follows:

$$S = N + (1 - N)s$$

S = speedup, *N* = cores, *s* = serial portion

Using different variables, can be formulated as:

$$S_{\text{latency}}(s) = 1 - p + sp$$

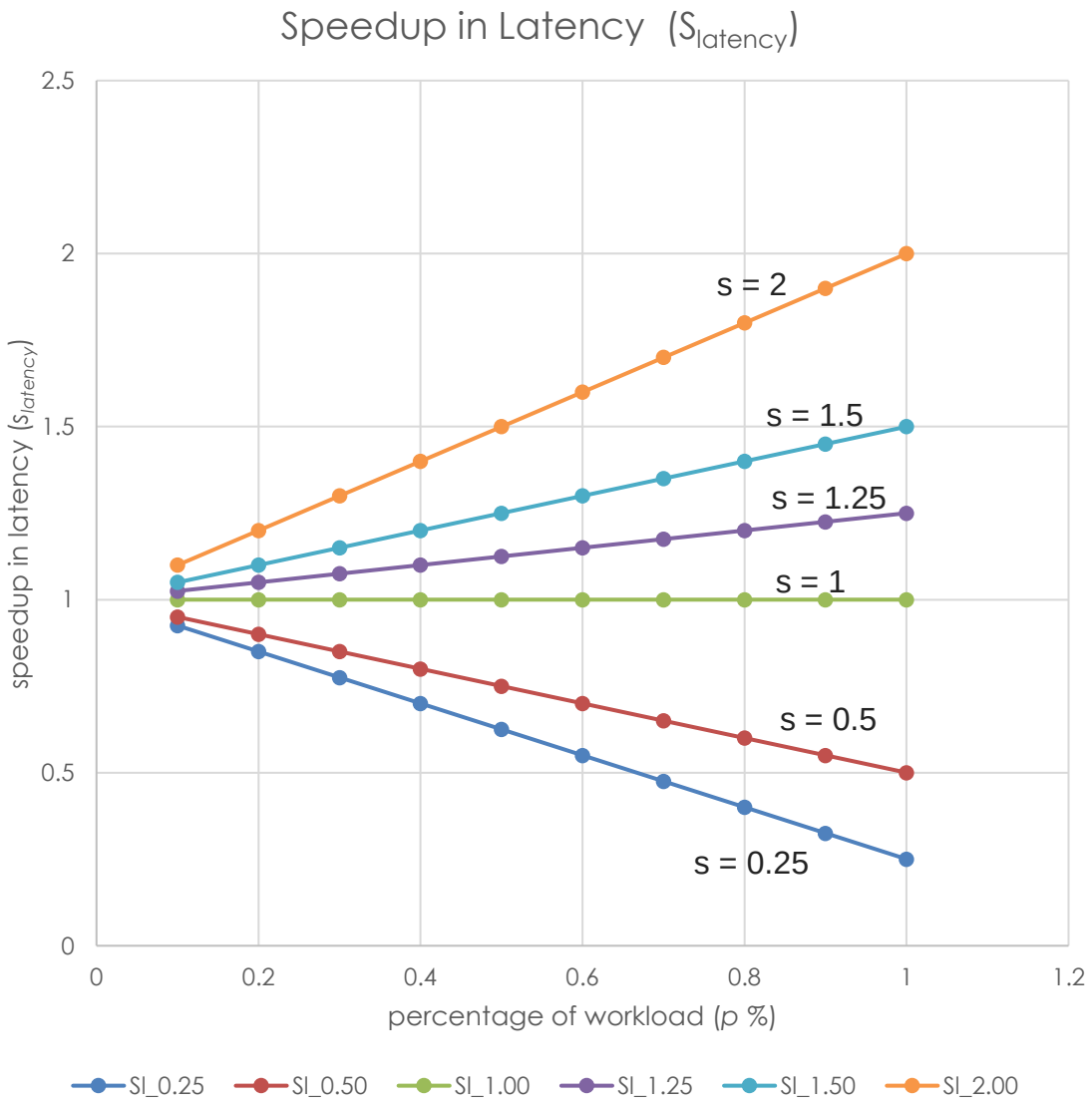
S_{latecy} = theoretical speedup in latency of the execution of the whole task

s = speedup in latency of part of the task benefiting from improved parallelism

p = % of workload before the improvement that will be speeded up.

* also referred to, more fairly, as the Gustafson–Barsis's law

Gustafson's Law



This is a plot of

$$S_{\text{latency}}(s) = 1 - p + sp$$

s going from 0.5 to 2

p going from 10% to 100%

As you can see, the latency is reduced (i.e. $S_{latency}$ increased) as *s* increases and *p* increases. But for e.g. a real speedup of 2 you actually need *p*=100% (i.e. the entire workload improved) to achieve it.

s = speedup in latency of part of the task benefiting from improved parallelism

p = % of workload before the improvement that will be speeded up.

Gustafson's law can be more useful to real-time embedded systems because it look more towards improving the response time of a system.

Glimpse into Next Lecture

Amdahl's Law Revisited:

Applying refinements for the heterogeneous, multicore era of computing….

The next lecture helps you understand the prescribed reading, a highly influential paper, that brings Amdahl's Law more accurately into the 21st century.

End of Lecture

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:

Clipart sources – public domain CC0 (http://pixabay.com/)
PxFuel – CC0 (https://www.pxfuel.com/)
FreeSVG – CC0 public dimain images (https://freesvg.org/)
Pixabay
commons.wikimedia.org
Images from flickr