Welcome to COMP4300/8300 Parallel Systems

Teaching staff, communication, website, and week 1 checklist

- Course Convenor: Prof. Dirk Pattinson
- Lecturers:
  - Dr. Alberto F. Martin (1st half of semester)
  - Prof. John Taylor (2nd half of semester)
- Tutors: Elise Palethorpe, Ryan Stocks, Calum Snowdon
- Communication with teaching staff handled via Ed Discussion
  (do not send course-related e-mails to our personal ANU accounts please!)
- Course webpage at https://comp.anu.edu.au/courses/comp4300
  (we will use Wattle only for mid-sem exam, lecture recordings and marking)
- Week 1 checklist
  - Go to this page and follow the steps there
  - Labs start in week 2 Register before the end of week 1 via MyTimeTable
  - Account at NCI/Gadi needed (account creation instructions)
  - Read carefully course policy on the appropriate usage of allocated computed time available (available here)

Course Assessment

The total workload for this (6 units) course is around 150 hours (12 hours per semester week). For S1 2024, the following assessment scheme:

- **Assignments – 50% of final mark**
  - There will be two assignments, each worth 25% of the final mark (50% in total)
  - These assignments will contain a significant parallel programming component plus a report where you will have to answer practical and theoretical questions relevant to the course content
- **Mid-Semester Exam – 10% of final mark – Redeemable**
  - 1-hour exam held in Tue, wk6 (lecture time) worths 10% of final mark
  - Remote/electronic multiple-choice exam in Wattle
- **Final Exam – 40% of final mark**
  - This will be a 3 hour written exam held at the end of the semester during the normal examination period
  - This exam is worth 40% of the final course mark

Software and minimum IT requirements (I)

You will need satisfactory Internet connection to be able to access to:

- course website (lecture slides etc.)
- GitLab (Labs; Assignments submission)
  - Some familiarity with Git/GitLab (clone, commit, push, pull, etc.) assumed
  - You will need a private/public key installed in your GitLab account to be able to work with Git repos (instructions here)
  - Click here for a set of best practices and instructions on the workflow to be used for the assignments
- Wattle (mid-sem, marking, and lecture recordings)
- Ed Discussion
- NCI Gadi documentation

Software and minimum IT requirements (II)

You will also need:

- An SSH client to access `gadi.nci.org.au` with your NCI account (if it gets as far as asking for a username/password outgoing SSH is not being blocked)
- Later, will need access to a GPU server (`stugpu2.anu.edu.au` via `partch.anu.edu.au`)
- For code development, you may use a remote terminal editor (advanced users) or a graphical editor, in particular VSCode with the Remote Development extension highly recommended for beginners. Click here for instructions.
- For Windows users, Moba XTerm highly recommended to access Gadi

Course recommended texts + extra materials

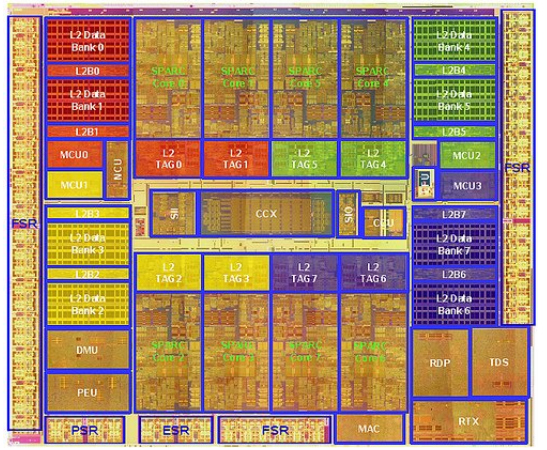
- The course does not strictly follow any particular text book
- However, most of the concepts treated in the course are covered in the materials available at resources section of web page

Health Warning

- it's a 4000/8000-level course, it's supposed to:
  - be more challenging than a 3000-level course
  - you will be exposed to bleeding edge technologies
  - be less well structured
  - have a greater expectation for your self-directed understanding/independence
  - have more student participation
  - and why not ... be fun as well
- it assumes you have done some background in concurrency (e.g. COMP2310); also in computer organization
- it requires strong programming skills – in `C`!
- attendance at lectures/labs strongly recommended (even though not assessed)

Today's lecture main take-away message

To extract all the computational performance from today's (even commodity) microprocessors there is no alternative but to **EXPLICITLY** manage parallelism in our programs!



UltraSPARC T2 (2007) (Niagara-2) multicore chip layout

(courtesy of T. Okazaki, Flickr)

Lecture Overview

- parallel computing concepts and scales
- sample application area: computational science and engineering (CSE)
- the role of Moore's Law and Dennard's Scaling Law in the exponential growth of computing performance
- end of Dennard's scaling Law (mid 2000s) and multicore revolution
- the Top 500 supercomputers and challenges for the future of computing
- why parallel programming is hard

Parallel Computing: Concept and Rationale

The idea:

Split computation into tasks that can be executed simultaneously on different processors

Motivation:

- Speed, Speed, Speed ··· at a cost-effective price
  - if we didn't want it to go faster we wouldn't bother with the hassles of parallel programming!
- reduce the time to solution to acceptable levels (e.g., under real-time constraints)
  - no point in taking 1 week to predict tomorrow's weather!
  - simulations that take months are NOT useful in a design environment
- tackle larger-scale (i.e., with higher computational demands) problems
- keep power consumption and heat dissipation under control

Parallelization

Split program up and run parts simultaneously on different processors

- on p computers, the time to solution should (ideally!) be reduced by $\frac{1}{p}$
- parallel programming: the art of writing the parallel code
- parallel computer: the hardware on which we run our parallel code

This course will discuss both!

Beyond raw compute performance, other motivations may include

- enabling more accurate simulations in the same time (finer grids)
- providing access to huge aggregate memories
- providing more and/or better input/output capacity
- hiding latency

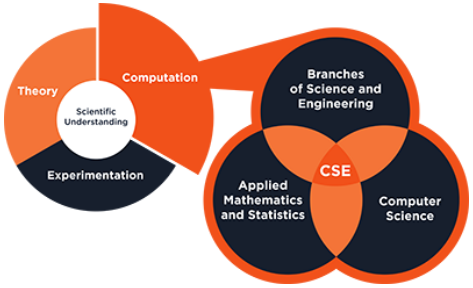
Scales of Parallelism

- within a CPU/core: pipelined instruction execution, multiple instruction issue (superscalar), other forms of instruction level parallelism, SIMD units*
- within a chip: multiple cores*, hardware multithreading*, accelerator units* (with multiple cores), transactional memory*
- within a node: multiple sockets* (CPU chips), interleaved memory access (multiple DRAM chips), disk block striping / RAID (multiple disks)
- within a SAN (system area network): multiple nodes* (clusters, typical supercomputers), parallel filesystems
- within the internet: grid/cloud computing*

*requires significant parallel programming effort

What programming paradigms are typically applied to each feature?

Application Area - Computational Science & Engineering (CSE)



Third pillar of scientific discovery

- Integrates applied mathematics, computer science, and branches of science/engineering in a single discipline (e.g., computational geophysics)
- Leverages computational models, algorithms, data, software and HPC (i.e. **supercomputers**) to tackle grand-challenges in science and engineering

Application Area - R&D in CSE

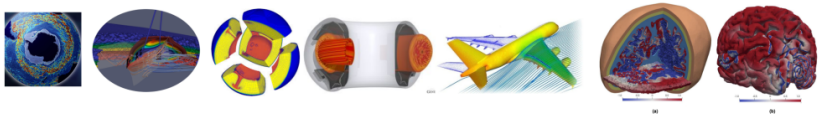
- Objective: improve the state-of-the-art in **computational models**, **algorithms**, and **software** to push the boundaries of what is currently achievable in CSE
- **Strong potential**: simulate out of reach problems, more precise predictive CSE, improved scientific knowledge, revolutionize decision-making across science, technology and society

Main research areas

- Mathematical modelling
- Numerical methods (**discretization**, **solvers**)
- Data assimilation (e.g., **machine learning**)
- **HPC** (parallel software/hardware innovations)

Application areas (examples)

- Geophysics
- Nuclear fusion
- Aeronautics
- Personalized medicine (brain/heart)
- Nanoscience, Smart manufacturing, (large) Etc.



Application Area - Synergy among CSE and HPC

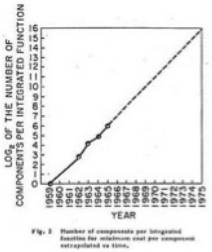
- We already find ourselves in the **Exascale** era ($O(10^{18})$ FLOPs/s peak)
- **Frontier**: 1st Exascale supercomputer (Oak Ridge US National Labs) (∼10M cores, 1.1EFLOPs/s, ranked #1 Jun, 2023 Top500 list)

- Performance boost mostly based on adding hardware parallelism (e.g., higher #cores/CPU) and heterogeneous hardware (CPUs, GPUs, . . .)
- To exploit such vast concurrency is a **formidable task** for CSE (breakthroughs in scalable algorithms and software innovations)

Moore's Law & Dennard Scaling

Two "laws" underpin exponential performance increase of microprocessors

Moore's Law
'Transistor density will double approximately every two years.'



Dennard Scaling
'As MOSFET features shrink, switching time *and* power consumption will fall proportionately'

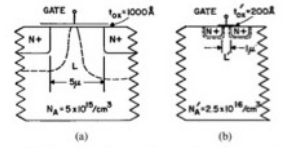


Fig. 1. Illustration of device scaling principles with $\kappa = 5$. (a) Conventional commercially available device structure. (b) Scaled-down device structure.

Moore's law

The number of transistors in a chip doubles every 24 months

Does this automatically imply that performance (per chip) doubles every 24 months?

No! Moore's law is not a performance law! However, there has been a strong correlation between the # of transistors and performance across time

How is that possible?

Dennard's scaling law (until early 2000s)

Device or circuit parameter	Scaling factor
Device dimension *tox, L, W*	1/κ
Doping concentration *Na*	κ
Voltage *V*	1/κ
Current *I*	1/κ
Capacitance *εA/t*	1/κ
Delay time/ circuit *VC/I*	1/κ
Power dissipation/ circuit *VI*	1/κ2
Power density *VI/A*	1

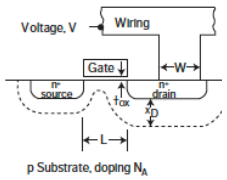


Figure 1. Traditional MOSFET scaling as described by Robert Dennard.

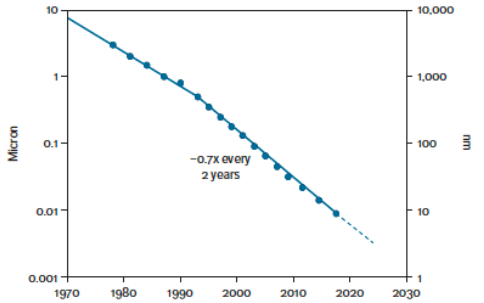


Figure 2. Minimum feature size scaling trend for Intel logic technologies.

The (dynamic) power consumption of a chip can be modelled as:

$$P = QfCV^2$$

where Q # of transistors, f frequency, C capacitance, and V voltage supply

Dennard's scaling law (until early 2000s)

According to Dennard' law, if we scale feature size down by a factor of $\frac{1}{\kappa}$, we can scale up frequency by κ, and scale down the capacitance and voltage by $\frac{1}{\kappa}$, resulting in a (reduced) power consumption of (assuming $Q_\kappa = Q_0$):

$$P_0 = Q_0 f_0 C_0 V_0^2 \rightarrow P_\kappa = Q_\kappa f_\kappa C_\kappa V_\kappa^2 = Q_0 (\kappa f_0) \left(\frac{1}{\kappa} C_0\right) \left(\frac{1}{\kappa} V_0\right)^2 = \left(\frac{1}{\kappa^2}\right) P_0$$

What about if we allow ourselves to keep P_κ constant? How many transistors Q_κ can we fit on the same chip? The answer is:

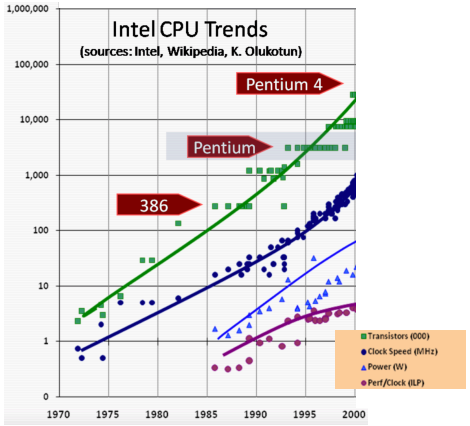
$$Q_\kappa = \kappa^2 Q_0$$

As long as we keep scaling feature size down by $\frac{1}{\kappa}$, we can fit κ^2 more transistors on the same chip, increase their frequency by κ, and use the same power as before!

Until early 2000s – The uniprocessor era

Performance of single instruction stream microprocessors increased exponentially (at a pace of $\approx$ 40% every 24 months)

- Exponential increase of clock rate
- Increase in Instruction Level Parallelism (ILP)
  - Pipelining: from 5 up to 31 (Prescott) stages
  - Superscalarity: from <1 instruction/clock to 4+ instructions/clock
- Many transistors for additional optimizations
  - Increase in cache sizes
  - Prefetching
  - Sophisticated branch prediction logic

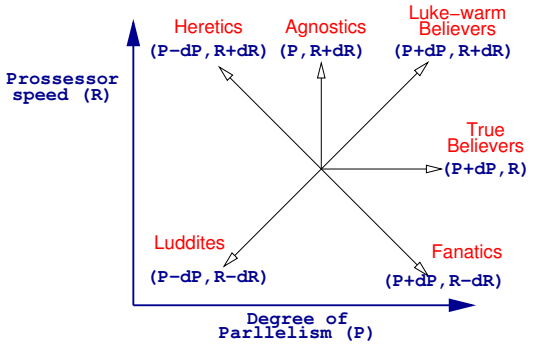


Moore's Law and Dennard Scaling Undermine Parallel Computing

- parallel computing looked promising in the 90's, but many companies failed due to the 'free lunch' from the combination of Moore's Law and Dennard Scaling
  - Why parallelize my codes? Just wait 2 years, and processors will be 1.4x faster!

On several recent occasions, I have been asked whether parallel computing will soon be relegated to the trash heap . . . Ken Kennedy, CRPC Director, 1994

- demography of Parallel Computing (mid 90's, origin unknown)



The end of Dennard scaling and uniprocessor era (2002-2004)

With feature size below $\approx$ 100nm (nowadays around $\approx$ 10nm), we have that:

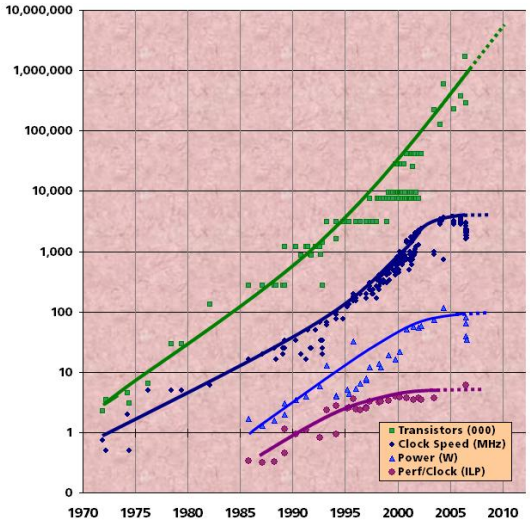
$$P = QfCV^2 + VI_{\text{leakage}}$$

(Note: for "large enough" feature size, the term VI_{leakage} is negligible)

Unfortunately, I_{leakage} grows exponentially with downscaled V as we decrease feature size by $\frac{1}{\kappa}$. Thus, the term VI_{leakage} blows up and dominates power consumption!

To keep power under control, large number of transistors are switched off (dark silicon effect), operated at lower frequencies (dim silicon effect) or organized in different ways

The end of Dennard scaling and the uniprocessor era

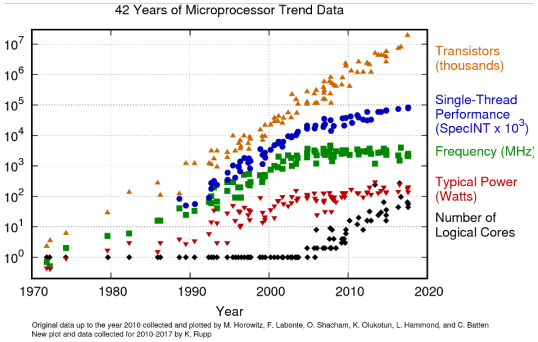


The free lunch is over!

- Clock rate capped ($<$ 4 GHz) by power constraints
- ILP (superscalarity and pipelining) reaches its limits
- Power usage saturated (becomes a major concern)

The multicore era

The only way to sustain exponential growth in computational performance is by efficiently exploiting parallel computers



- This is very much different from the golden years of ILP where hardware architects did all the work for us
- Programmers are now forced to bear the burden of finding and exploiting parallelism
- This is also an exciting era of opportunities for computational scientists: new algorithms and efficient implementations make a difference on what is achievable in computing

Where are we now?

We find ourselves in a chip development era characterized by:

- Transistors galore (this cannot be sustained forever, though, due to physical limits)
- Severe power limitations (maximize transistors utility no longer possible)
- Customization versus generalization

We are now seeing:

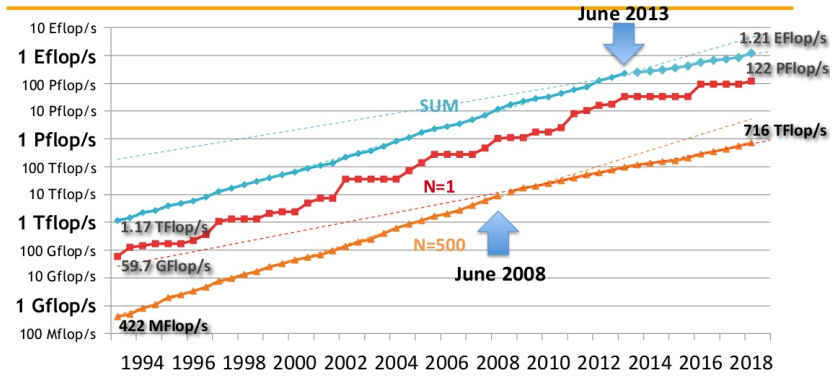
- (customized) accelerators, generally manycore with low clock frequency
  - e.g. Graphics Processing Units (GPUs), customized for fast numerical calculations
- 'dark silicon': need to turn off parts of chip to reduce power
- hardware-software codesign: speed via specialization

The Top 500 Most Powerful Computers: November 2022

The Top 500 provides an interesting view of these trends (click here for full list)

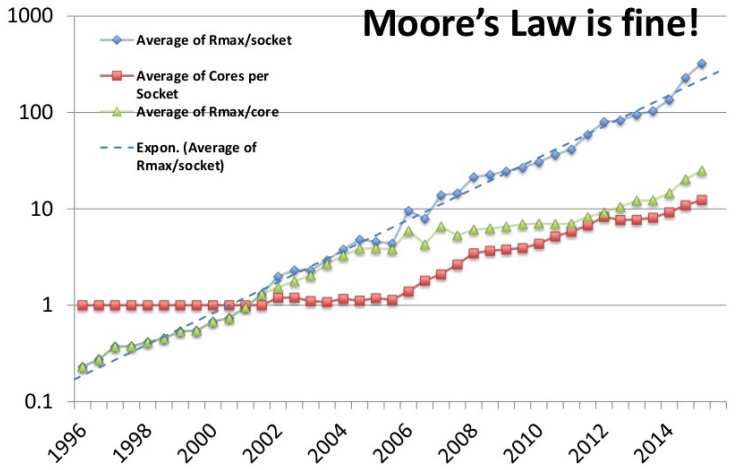
Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,730,112	1,102.00	1,685.65	21,100
2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,220,288	309.10	428.70	6,016
4	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, Atos EuroHPC/CINECA Italy	1,463,616	174.70	255.75	5,610
5	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148.60	200.79	10,096

Top500: Performance Trends



(http://www.top500.org/resources/presentations/ (51st TOP500))

The Top500: Multicore Emergence



(http://www.top500.org/blog/slides-highlights-of-the-45th-top500-list/)

Exascale and Beyond: Challenges and Opportunities

Level	Characteristic	Challenges/Opportunities
among nodes	sheer #nodes, although different balance among #nodes and performance/node in top-ranked supercomputers (e.g., Frontier has "only" 9.4K nodes but Fugaku has 159K nodes)	• programming languages/environment • fault tolerance
within a node	High heterogeneity (e.g., Frontier, LUMI and many more use CPUs and GPUs)	• what to use when • co-location of data with the unit processing it
within a chip	energy minimization (e.g. processor cores have already dynamic frequency and voltage scaling)	• minimize data size and movement (e.g., use just enough precision) • specialized cores

Why Parallel Programming is Hard

- writing (correct and efficient) parallel programs is hard!
  - hard to expose enough parallelism; hard to debug!
- getting (close to ideal) speedup is hard! Overheads include:
  - idling (e.g., caused by load unbalance, synchronization, serial sections, etc.)
  - redundant/extra operations when splitting a computation into tasks
  - communication time among processes
- Amdahl's Law: Let f the fraction of a computation that cannot be split into parallel tasks. Then, max speedup achievable for arbitrary large p (processors) is $\frac{1}{f}$!!!
  - Let t_s and t_p denote serial and parallel exec times
  - $t_p = f t_s + \frac{(1-f)t_s}{p}$
  - $S_p = \frac{t_s}{t_p} = \frac{p}{pf+(1-f)}$ (Speedup)
  - $\lim_{p\to\infty} S_p = \frac{1}{f}$
  - e.g., if $f = 0.05$, then $\frac{1}{f} = 20$
- counterargument (Gustafson's Law): $1 - f$ is not fixed, but increases with the data/problem size N

Recommended further reading

The following two articles provide a crystal clear view of past, present, and future:

- Computing Performance: Game Over or Next Level? Computer, 44(1), 2011. Publisher: IEEE.
- Time Moore: Exploiting Moore's Law From The Perspective of Time. IEEE Solid-State Circuits Magazine, 11 (1), 2019. Publisher: IEEE.

Only for intrepid readers:

- Computing Performance: Game Over or Next Level?, Full report from US National Research Council, National Academies Press, 2011. Bonus: reprints for Moore's and Dennard's Laws seminal papers.

Additional references related to this lecture

- R. Dennard et al., "Design of Ion- Implanted MOSFETs with Very Small Physical Dimensions". IEEE J. Solid State Circuits, vol. 9, no. 5, 1974, pp. 256–268.
- V. Agarwal, M.S. Hrishikesh, S.W. Keckler, and D.A. Burger, "Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures". In Proceedings of the 27th Annual International Symposium on Computer Architecture, 2000, pp. 248-259.
- M. T. Bohr and I. A. Young, "CMOS Scaling Trends and Beyond", in IEEE Micro, vol. 37, no. 6, pp. 20-29, November/December 2017, doi: 10.1109/MM.2017.4241347.
- M. B. Taylor, "A Landscape of the New Dark Silicon Design Regime", in IEEE Micro, vol. 33, no. 5, pp. 8-19, Sept.-Oct. 2013, doi: 10.1109/MM.2013.90.
- The Free Lunch Is Over. A Fundamental Turn Toward Concurrency in Software. Herb Sutter. `http://www.gotw.ca/publications/concurrency-ddj.htm`