Chapter 1 - Motivation and History

Data parallelism: same operation to different elements of the data set (e.g. vector addtion)
Function parallelism: different operations to different data elements

1. 5. A task can be divided into m subtasks.
How much time is needed for a m-stage pipeline to process n tasks?
-> n + m - 1

1.6. Printing n pages takes 5+10+3+n s. What is the minimum capacity of the feeder tray so that asympotic throughput reaches 40 pages/min?

$40 \frac{\text{pages}}{\text{min}} \rightarrow 1.5 \frac{s}{\text{page}}$ $\quad 1.5 = \frac{18+n}{n} \rightarrow 1.5n = 18+n \rightarrow n = 2 \cdot 18 = \underline{\underline{36}}$

1.8 Performance increases x10 every 5 years, how long does it take to double?

t: time in years $\quad 10^{\frac{t}{5}} = 2 \quad \frac{t}{5} = \log_{10} 2 \quad t = 5 \log_{10} 2 = \frac{5}{\log_2 10} \approx 1.505$

1.10: Calculate new problem size that can be solved in the same time when the computer is 100x faster. (Old problem size: 100 000.)

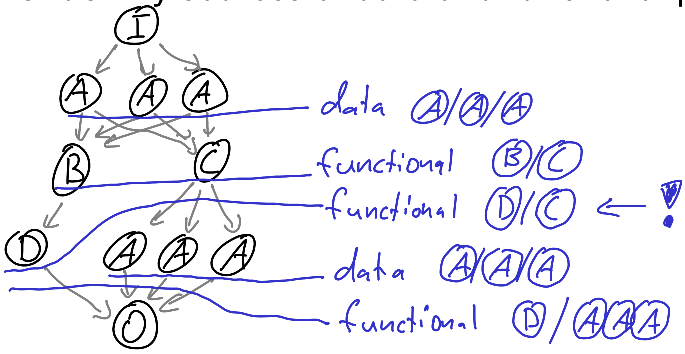
a) $\Theta(n) \quad \frac{n}{10^5} = 100 \quad n = 100 \cdot 10^5 = 10^7$

b) $\Theta(n \log_2 n) \quad \frac{n \log_2 n}{10^5 \log_2 10^5} = 100$ -> solve for n

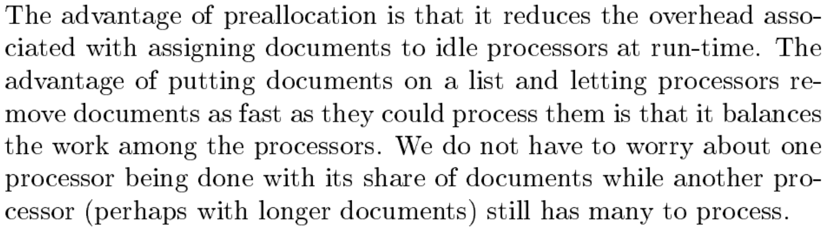
c) $\Theta(n^2) \quad \frac{n^2}{(10^5)^2} = 100 \quad n = \sqrt{10^{12}} = 10^6$

d) $\Theta(n^3) \quad \frac{n^3}{(10^5)^3} = 100 \quad n = \sqrt[3]{10^{17}}$

1.13 Identify sources of data and functional parallelism



1.14 preallocate N/p documents vs. put documents in a list and let processors remove documents as fast as they can process them



The advantage of preallocation is that it reduces the overhead associated with assigning documents to idle processors at run-time. The advantage of putting documents on a list and letting processors remove documents as fast as they could process them is that it balances the work among the processors. We do not have to worry about one processor being done with its share of documents while another processor (perhaps with longer documents) still has many to process.

Chapter 2 - Parallel Architectures

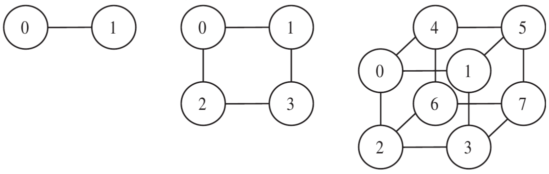
Shuffle-exchange: - shuffle link to LeftCycle(i)
- exchange link to xor(i, 1)

Vector computers:
Pipelined vector processor: streams data through pipelined arithmetic units
Processor array: many identical, synchronized arithmetic processing units

Multiprocessors: multiple-CPU computer with shared memory
centralized (bus, same memory for all CPUs) or distributed (interconnection network)

Multicomputer: distributed memory multiple CPU computer, message passing
asymmetrical: one front-end computer and many back-end computers
symmetrical: all nodes are equal

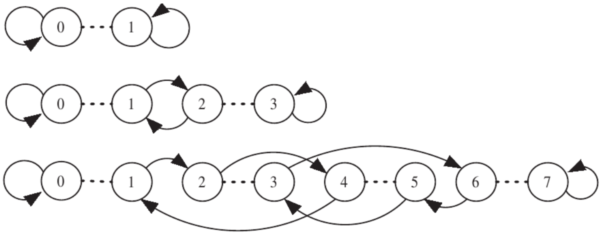
2.1. Draw hypercube with 2, 4, 8 nodes



- edge between nodes if exactly one bit is different
- network with n nodes is subgraph of one with 2n nodes

dim: n edges: $2^{n-1} \cdot n$ nodes 2^n
$1, 2, 3$ $1, 4, 12$ $2, 4, 8$

2.8. Shuffle-exchange network with 2, 4, 8 nodes



dashed: exchanged links
solid: shuffle links

network with n nodes is not a subgraph of network with 2n nodes

2.13. Why are processor arrays well suited for executing data-parallel programs?
-> every processing element performs the same operation, but with different data

2.14. Processor array with 8 processing elements, each 10M integer ops/s. Determine performance for adding two vectors of size 1 to 50.
(According to the solution, the answer should be integer ops/s, not vector ops/s!)

$$\text{performance} = \frac{\text{number of ops}}{\text{time}} = \frac{n \text{ ops}}{\frac{\lceil n/8 \rceil}{10 \text{ million}} \text{ s}} = \frac{10n}{\lceil n/8 \rceil} \text{ million ops/s}$$

2.15. Processor array, case statement with k cases.
a) Efficiency if each case contains same number of elements
-> The tricky thing is that the processor array always executes the same instructions on all elements and that means that for a case statement, it will be really inefficient.

$$\text{efficiency} = \frac{\text{sequential execution time}}{\text{processors used} \times \text{parallel execution time}} = \frac{n}{n \times k} = \frac{1}{k}$$

b) Efficiency if case i has I_i instructions and probability P_i

sequential time: $n \times$ average time $= n \sum_{i=1}^{n} P_i I_i$

$$\text{efficiency} = \frac{n \sum_{i=1}^{n} P_i I_i}{n \times \sum_{i=1}^{n} I_i} = \frac{\sum_{i=1}^{n} P_i I_i}{\sum_{i=1}^{n} I_i}$$

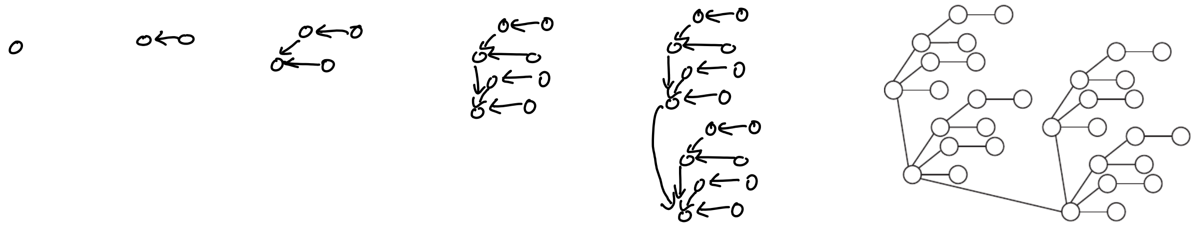
2.16 Why are large data and instruction caches desirable in multiprocessors?
Large caches reduce the load on the memory bus, enabling the system to utilize more processors eficiently.

2.17 Why is the number of processors in a centralized multiprocessor limited to a few dozen?
-> All processors communicate over the bus, which is the bottleneck.

Chapter 3 - Parallel Algorithm Design

Task/Channel Model: Tasks (Program, local memory, IO ports) communicate with channels
Fosters Design Methodology: Partitioning, Communication, Agglomeration, Mapping

3.3. Draw binomial tree with 16 and 32 nodes
-> always duplicate smaller version and connect with one line



-> subgraph of hypercube, used for reduction etc

3.8. Prove that n-element reduction has time complexity Ω(n)
TODO look at homework and solution (different)

3.9a Efficient parallel algorithm implementing broadcast
-> Use binomial tree, i.e. do inverse of reduction, ceil(log p) communication steps

Chapter 11 - Matrix Multiplication

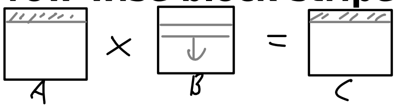
recursive, block oriented algorithm (not parallel)

This is used as the core for the parallel algorithms to do the actual multiplication of parts.

$$A \times B = C \qquad A = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} \quad B = \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix} \qquad C = \begin{pmatrix} A_{00}B_{00} + A_{01}B_{10} & A_{00}B_{01} + A_{01}B_{11} \\ A_{10}B_{00} + A_{11}B_{10} & A_{10}B_{11} + A_{11}B_{11} \end{pmatrix}$$

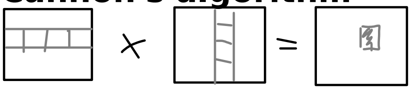
recursively divide matrix into smaller submatrices until they fit in the cache

row-wise block-striped matrix decomposition



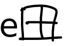
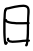
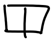
row-shaped blocks of B are passed in a circular way between processes,
every process has row-shaped part of A and calculates row-shaped part of C

Cannon's algorithm



Every process calculates block of C
Blocks of A and B are passed around

11.2: (row-wise) why not replicate B across all processes?
B would have to fit in primary memory of each processor, limits max. problem size

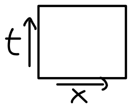
11.3: a) why is Cannon better for the block oriented algorithm?
-> submatrices are more or less square
b) How to modify block oriented algorithm for block striped algorithm?
-> don't divide like  , but  or  depending on largest dimension

Chapter 12 - Solving Linear Systems

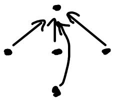
-> stupid

Chapter 13 - Finite Difference Methods

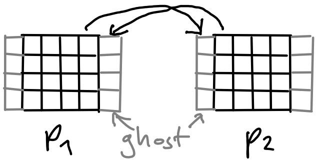
discretisize PDE -> matrix, e.g.:



calculation step e.g.:



parallelize: neighboring elements are needed for calculation step
-> use ghost points that store redundant copies of data



Chapter 14 - Sorting

Quicksort

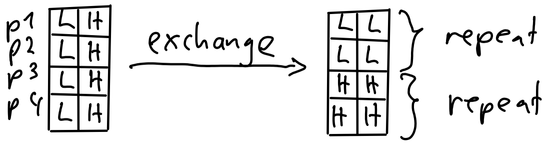
- choose pivot value
- partition the list in low and high sublists
- recurse on low and high sublist
- return [low, pivot, high]

Parallel sorting

- input is evenly distributed along processes
- output is distributed, but not necessarily evenly

First algorithm

- one process globally broadcasts pivot, each process partitions and the lists are exchanged



- afterwards, each process does sequential quicksort of its list

problem: list sizes are not balanced, since pivot is usually not the median

Hyperquicksort

- first, each proces sorts its data using quicksort
- one process chooses median as pivot, all processes split and exchange lists
- afterwards, the two sorted sublists are merged
- communication pattern follows hypercube structure

Parallel Sorting by Regular Sampling (PSRS)

-> more balanced, avoid repeated communication of keys, does not require p=2,4,8,...

- first: quicksort
- each process samples at $0, \frac{n}{p^2}, 2\frac{n}{p^2}, \dots, (p-1)\frac{n}{p^2}$
- one process gathers and sorts samples, selects p pivots
- each process uses pivots to split the lists and sends sublists to corresponding process
- each process merges the p sublists

14.1: quicksort is not stable

14.5: Is hyperquicksort or PSRS less disruped if list is already sorted?
PSRS performs best, since the pivots are globally selected and almost no data needs to be transferred