Architecture of distributed systems
Summary of the course contents

Lukas Prokop

12.03.18

Contents

1 Fundamentals of Distributed Computing

1.1 Terminology

partitioning Separation of tasks in subtasks (in respect of data dependencies)

mapping Definition which processor executes task

scheduling Define chronological order

Degree of parallelism degree of parallelism (DOP) is a metric which indicates how many operations can be or are being simultaneously executed by a computer[1]

Ubiquitous computing A post-desktop-computing model where information processing is done primarily by small, distributed units

pervasive computing Ubiquitous computing focusing the aspect of adapting computation to the human environment (unlike desktop computing does)

wearable computing Ubiquitous computing focusing units which are part of our clothes and daily things (typically RFID technology)

1.2 Data flow (dependency) graph

A *data flow graph* or *data dependency graph* is a graph (V, E) with

- $e \in E$ symbolizing data flow dependencies between tasks
- $v \in V$ symbolizing tasks

By constructing a data flow graphs we can easily identify tasks, we can parallelize (the ones without edges)[2][3][4]. In terms of reading and writing, the *Bernstein conditions* describe constraints:

True dependence If process P writes to a memory cell M, then no process P_2 can read the cell M. $IN_2 \cap OUT_1 = 0$ (Read After Write).

Antidependence If process P reads from a memory cell M, then no process P_2 can write to the cell M. $IN_1 \cap OUT_2 = 0$ (Write After Read).

Output dependence If process P writes to a memory cell M, then no process P_2 can write to the cell M. $OUT_1 \cap OUT_2 = 0$ (Write After Write).

[1] https://en.wikipedia.org/wiki/Degree_of_parallelism

[2] https://en.wikipedia.org/wiki/Vectorization_%28parallel_computing%29#Building_the_dependency_graph

[3] https://en.wikipedia.org/wiki/Dependence_analysis

[4] https://en.wikipedia.org/wiki/Dependency_graph

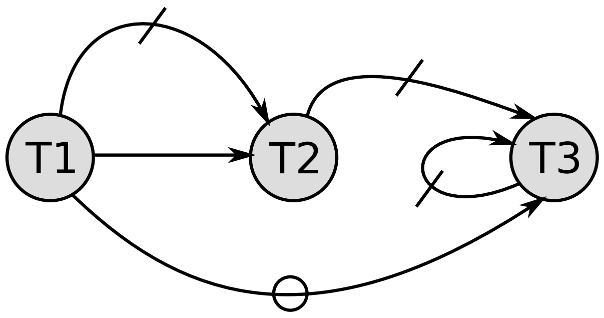


Figure 1: Example dependency graph with Write After Read (between T2 and T3), Write After Write (between T1 and T3) and Read After Write (between T1 and T2).

1.3 Cache coherence

Cache coherence describe strategies in multiprocessor systems assuring that all caches return the same value for a specific memory address. If several processors use the same memory and caching techniques inside the processors are used, synchronization problems might occur. For example processor A updates a value 8 to 5 at memory address 0x10. Now processor B asks for memory address 0x10 and its cache (B_C) will return a deprecated 8 value, because the cache does not have any idea of the update. This is a general problem for multiprocessor systems.

Directory-based (a separate, central list remembers which processors are allowed to write data to the memory) and snooping-based (each cache is in "snooping" mode listening at the communication bus while not writing and listens for value updates) solutions are introduced. Communication protocols are one approach but depend on the strategy the caches are working with. Write policies define *when* data from the cache will be written to the main memory:

Write Through cache A write operation immediately yields a write through the main memory.

Write Back cache A write operation will only affect the cache as long as no replacement is done (and therefore saves memory interactions).

Snooping protocols only work with Bus systems (broadcast operations). Two different cache protocols that use snooping protocols are:

Write invalidate protocol After writing all memory addresses of that block will be flagged as "invalid" and a memory access to one of them will lead to a *cache miss*

Write update protocol The updated memory block will be transfered to all the other caches.

Causes for the need of cache coherence include:

- data sharing
- process migration
- I/O activity

1.4 Transparency of a distributed system

Access transparency Access of local or global objects by the *same* operation

Breakdown transparency User does not recognize partial breakdowns of system

Capacity transparency dynamic capacity distribution

scalability transparency Modification of system size without modification of system structure

1.5 Handshake Protocol

The handshake protocol define a handshake between two communication partners. It states, that the communication opener has to send a signal, that data is available. The partner will receive this data and read it from the communication channel. He sends a signal back symbolizing the success of reading it. The opener reads this signal and therefore stops his own communication.

1.6 Speedup

There are 4 main possibilities to increase processing speed:

- Pipelining
- parallelism
- hyperthreading technology
- new architectures like multiple cores in one processor

When using parallelism always the question of advantage arises, because in general it requires more engineering and programming work. We introduce metrics to measure speedups:

$$S = \frac{T(1)}{T(n)}$$

sequential time over parallel time

This speedup $S(n)$ is smaller or equal to n as the number of parallel computing units ($S(n) \leq n$). If $S(n) > n$ we are talking about a *superlinear speedup*, which is questionable in most cases, but is not impossible. *Efficieny* is defined as speedup per processor (as percentage):

$$E = \frac{S(n)}{n} \cdot 100$$

So speedups are not always linear, because of two reasons:

- Communication of units is a time overhead
- Some parts of the application are not parallelizable

1.7 Amdahl's law

The achievable speedup is limited by the portion of non-parallelizable operations (operations with DOP = 1).

$$S(n) = \frac{n}{1 + (n-1) \cdot f} \leq \frac{1}{f}$$

n	...	number of tasks
f	...	portion of non-parallelizable tasks

1.8 Architectures

RALU Register with an ALU ("arithmetic logic unit")

processor A RALU combined with a control unit

computer architecture A processor with a memory

multi RALU One control unit connected with several RALUs connected with a communication system and a memory

multi processor Multiple processors connected with a communication system to one or more memories

multi computer Several computers interconnected by a communication channel

array machine One control unit controls multiple processing units and they communicate on a separate system

vector processor A processor with a special vectorization arithmetic pipeline which allows to perform parallel operations on vectors (array-like data structures)

2 Classifications

2.1 Classification by Flynn

SISD Single Instruction, Single Data (eg. Von Neumann architecture)

SIMD Single Instruction, Multiple Data (eg. vector processor)

MISD Multiple Instruction, Single Data (eg. pipeline of instruction cycle layer)

MIMD Multiple Instruction, Multiple Data (all architectures supporting real parallelism)

2.2 Extended classification by Tanenbaum

Tanenbaum extended the list of MIMDs:

multi processors UMA, COMA, NUMA

multi computers MPP, COW

2.3 Synchronous vs asynchronous communication

synchronous Synchronous communication means that a processor blocks while waiting for the reponse and therefore cannot proceed

asynchronous In asynchronous communicaiton the processor makes a difference between reading and writing on a communication channel. Techniques like callbacks are introduced.

2.4 Layers of parallelism

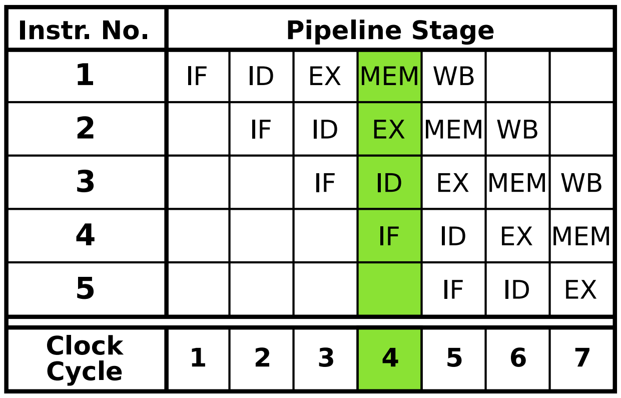
parallelism at machine level machine instructions are executed in parallel (without recognition of programmer)

parallelism at block layer sequences of instructions (in units of "blocks" as defined by programmer) are executed in parallel (explicit interaction of programmer)

parallelism at procedure layer Calling procedures without blocking (ie. without waiting for the result)

parallelism at threading layer Separate program counter, register and stack, but same memory is used to execute lightweight processes ("threads") in parallel

parallelism at process layer Independent processes solve a task in parallel



Instr. No.	**Pipeline Stage**						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	**1**	**2**	**3**	**4**	**5**	**6**	**7**

Figure 2: 5 stages pipeline (Instruction Fetch, Instruction Decode, EXecute, MEMory access, register Write Back) (Image by Wikipedia)

3 Pipelining

- instruction pipelining[5]
- arithmetic pipeline (eg. vectorization)

Once the pipeline is fully pipelined, the pipeline provides one result per cycle. Problems arise in the following situations:

- Conditional jumps changes control flow
- data dependencies must be recognized by the pipeline (eg. Read After Write)

Solutions:

Zero Cycle Branches Instruction after branch gets executed immediately after branch

Branch Delay Slots Pull later instructions forward to get pipeline in correct order

Bubbles are empty sequences in the instruction pipeline due to those problems

3.1 Speedup

If the pipeline has k segments and executes n tasks:

$$\left.\begin{array}{l} T(1) = n \cdot k \cdot \tau \\ T(n) = k \cdot \tau + (n-1) \cdot \tau \end{array}\right\} S(n) = \frac{T(1)}{T(n)} = \frac{n \cdot k \cdot \tau}{k \cdot \tau + (n-1) \cdot \tau} = \frac{n \cdot k}{k + (n-1)}$$

[5] `https://en.wikipedia.org/wiki/Instruction_pipeline`

Topology	Hardware costs	Transmission capacity
Bus	small	small
Switch	middle	middle
cross bar	high	high

3.2 Floating-point addition

Floating point addition at pipelines have 4 stages:

- exponent comparison
- mantissa alignment
- mantissa addition
- normalization

4 Connector (communication) systems

4.1 Terminology

Communication latency Total time for transmission

network latency Transmission latency due to infrastructure and software processing

per-port bandwidth Amount of data to be transmitted per second between two ports

aggregate bandwidth Amount of data per second from one half to the other half of the network

4.2 Static and dynamic connector systems

static connector systems connections are fixed. point-to-point connection between processors. eg. Star, Ring, Torus[6], complete graph

dynamic connector systems connections can be swapped during runtime. eg. Busses, crossbars, switch networks

4.3 Bus-oriented communication protocols

4.3.1 Daisy chaining

Setup: Data, Grant, Request and Busy lines[7]

[6] https://en.wikipedia.org/wiki/Torus_interconnect

[7] https://en.wikipedia.org/wiki/Network_topology#Daisy_chain

- Unit write request to line
- Arbiter writes to Grant whether or not the Data line is free to use
- Busy line shows assignment

+ simple units and arbitrary number of them

- static priorities

- error-prone

- slow

4.3.2 Polling

Setup: Request, Busy and Polling Count lines

- On request, Arbiter asks units
- Strategies like Round Robin

+ priorities can be changed

+ more reliable than daisy chaining

- high line costs

4.3.3 Individual Request

Setup: Request and Grant lines for each node

- For requesting, the node asks the arbiter on its individual line

+ fastest strategy

+ flexible for dynamic priorities

+ high reliability

- high effort

4.4 Butterfly network

4.5 Omega network

$$S(i) = 2 \cdot i + \left\lfloor \frac{2 \cdot i}{N} \right\rfloor \bmod N$$

5 Multi computer systems

5.1 MPP ("Supercomputer")

5.2 Network of Workstations ("Cluster")

5.2.1 Models

Workstation-Server Workstations individually per user with enough computational power

Processor-Pool Users communicate with central pool by terminals. Pool processors execute user applications

Hybrid model Pool overtake computational work dynamically if needed

5.3 Distributed computing ("Cloud")

6 Pervasive computing

6.1 Terminology

Smart Device Unit of ubiquitous computing. Device for different purposes with interface to humans (user interface), digital world (network interfaces) and real world (sensors).

Radio Frequency Identification Technology heavily used in logistics and trade. Allows contactless communication by radio technology. Active super supply by battery or passive by external electrical field.

7 Fault-tolerant systems

7.1 Terminology

n-versions programming n different versions of the program are created and distributed among different processing units. They all produce the *same* result and voter accepts this result (high development requirements!)

7.2 Redundancies

- time redundancy
- code redundancy
- hardware redundancy
- software redundancy

8 Projects

8.1 TOP500

TOP500 publishes a list of all top-500 supercomputers[8]. Evaluation by benchmark testing using the LINPACK benchmark (linear equations)[9]

9 Processors

9.1 Difference of CISC and RISC architectures

- CISC (”Complex Instruction Set Computing”)
  - Complex instruction set is implemented
  - slow execution of instructions (in terms of cycles)
  - requires large processor bed
  - few registers
  - little cache
- RISC (”Reduced Instruction Set Computing”)
  - Reduced instruction set is implemented
  - fast execution of instructions (in terms of cycles)
  - large processor bed for storage unit
  - many registers
  - big, separate cache

9.2 Memory Interleaving

Memory Interleaving is used by vector processors and addresses the problem of speed differences between processors and the main memory. The memory is divided into a set of banks. In a block-oriented scheme, address i in a memory of n banks, will be put in bank i/n. Therefore consecutive memory values will be put side by side. In an interleaved memory scheme, address i will be put in bank $i \mod n$ and therefore consecutive values are put in different banks[10].

[8] `http://www.top500.org/`
[9] `http://www.netlib.org/linpack/`
[10] `http://www.phy.ornl.gov/csep/ca/node19.html`