

What is the difference between concurrency and parallelism?

Concurrency – logically operating at once

Parallelism – actually operating at once

How can parallel computing improve power efficiency of a system?

clock frequency goes down but capacitance goes up; however, lowered frequency also lowers voltage ($P = fCV^2$)

Why do we include multiple levels of caches on processors?

Ultimately everything wants to be on-chip to actually work. You don't want to waste time going down to main memory for small problems when you can shallowly fetch from cache. Ultimately it's the speed/size tradeoff.

In hw1 why were profiled times wrong if we did not print out final index after pointer chasing?

If it's "unneeded" then it could perform out of order or not even optimize.

Explain how simultaneous multithreading (hyperthreading) is related to an out-of-order superscalar processor.

Out-of-order superscalar is to instructions as hyperthreading is to threads. They break up problems to fully fill the pipeline by executing what computations they can wherever they can.

Give two examples of micro-architectural advances that can potentially achieve greater than one instruction issued per cycle.

Pipelining and superscalar

More than 1 IPC started: Superscalar and SIMD

More IPCs finished: Out-of-order and pipelining

Reduced clock cycles: Pipelining and speed up transistors

How does modularity help the chief architect of a software development team?

Easier to organize entire project. You can divvy up tasks and narrow down problems more easily, as well as switch out modules in bits and pieces without breaking everything.

How does modularity help the project leader of a software development team?

Easier to explain tasks to various task teams, organize the project, assign tasks. Rather than forcing everyone to learn everything you can assign stuff to various focused teams.

How does modularity help the individual programmer of a software development team?

You don't have to know how everything works to do your piece. Helps with code reuse.

Data decomposition: Splitting up large amounts of data into independent chunks that can be processed in parallel.

Task decomposition: Breaking up large instructions into smaller independent chunks that can be processed in parallel.

A non-pipelined basic processor has a latency of 24 ns for each instruction. If we split the processor into 6 perfectly pipelined stages, what is the new latency per instruction? What was the throughput before and after the pipelining? Make sure to use the correct units. Also please ignore any additional latency induced by the introduction of pipeline stages.

24ns per instruction; throughput will increase from $1/24\text{ns}$ to $1/4\text{ns}$

What is the difference between SIMD and superscalar?

SIMD = single instruction, multiple data, ex) $1+2+3+4$

Superscalar = multiple scalar processing units, ex) 1+2 and 3+4 on two different units

Almost all programs use which of the structural patterns at their highest level?

Pipe-and-filter

Give two concrete reasons why patterns are valuable in software engineering?

1. You don't need to know the details of a program, just its basic structure
2. Turn programs into modules, which are easier to write and debug

Please rank the following in order of decreasing latency, and estimate latency in cycles.

DRAM(200 cycles, 65.1ns) > L3(38 cycles, 13ns) > L2(10 cycles, 3.4ns) > L1(4-8 cycles, 1.3ns)

Please rank the following in order of decreasing bandwidth, and estimate latency in GB/s.

DRAM(10.1 GB/s) > L3(26.2 GB/s) > L2(31.1 GB/s) > L1(45.6 GB/s)

Please rank the following in order of decreasing memory size, and estimate memory size.

DRAM(12 GB) > L3(8 MB) > L2(128 kB) > L1(32 kB)

Define the variables a, b, and c to be arrays of double-precision floating-point numbers.

double a[N x N], b[N], c[N]

where N is a large power of 2. The variable a is interpreted as an nxn matrix in row-major format. B and c are interpreted as vectors of length n. a and b are initialized to arbitrary values while c is initialized to all zeros. Now consider the following naïve matrix vector multiply routine.

```
For (int i=0; i< n; i++) {  
    For (int j=0; j<n; j++) {  
        C[i] = c[i] + a[i*N + j] * b[j];  
    }  
}
```

What is the total number of flops in the naïve matrix multiply routine in terms of N?

$2N^2$ (a[i*N + j] doesn't count because it's integers, not floating-point)

Pipe-and-Filter: Everything really

Compiler, Image Retrieval, image classification, Logic Optimizer

Iterator/BSP: SVM Training

MapReduce: Image processing, convolution, feature extraction, edge detection, fluid dynamics, heat maps, speech recognition, calculating sample distance from frontier in SVM, indexing web documents

Structured grid: Image convolution, modeling heat diffusion, modeling fluid dynamics

Scheduling

Latency

Throughput

Resource Management

Data parallelism: element-wise ops, communication reductions, nested parallelism, arbitrary

Matrix-multiply, SVM image classifier

Task parallelism: all other parallel strategies can be described in terms of this

Game engines

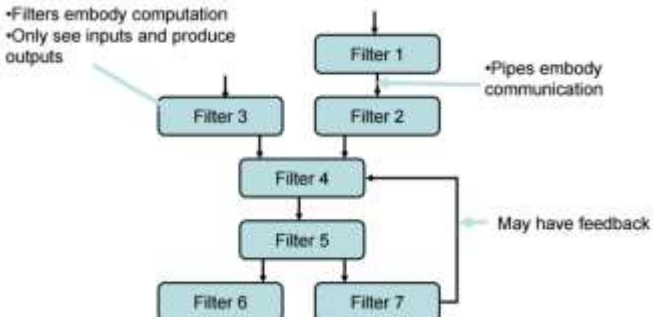
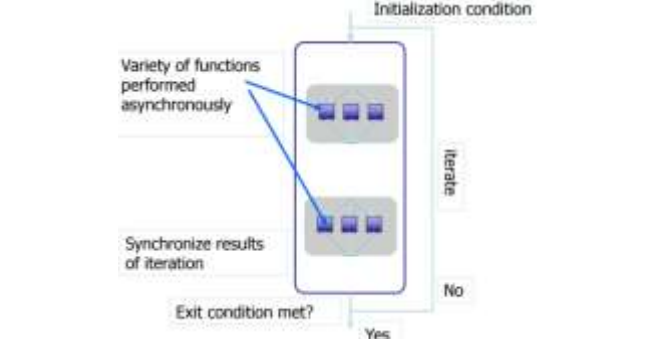
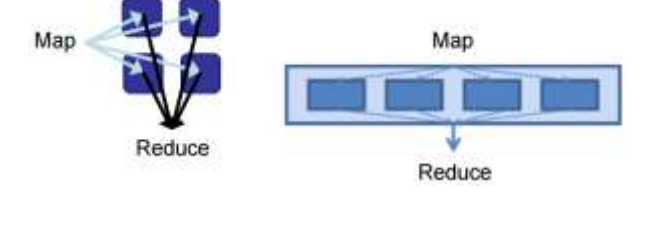
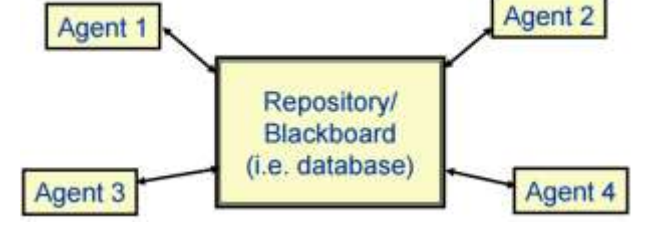
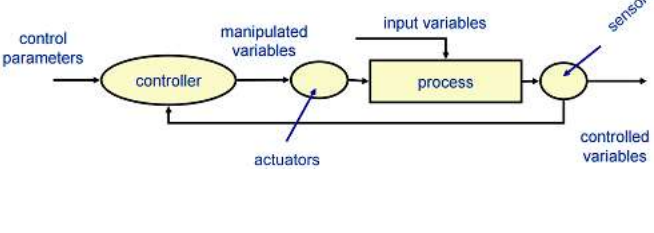
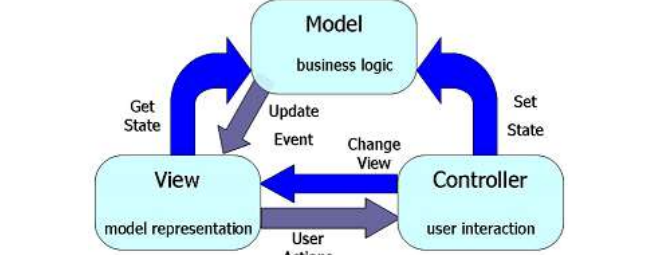
BLAS

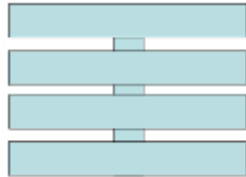

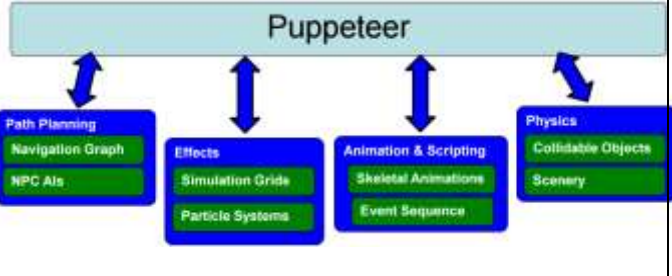
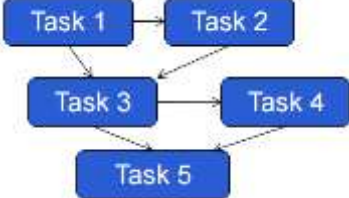
Level Example #mem refs #flops q

1	$y += ax$	$3n$	$2n$	$2/3$
2	$y += Ax$	n^2	$2n^2$	2
3	$C += AB$	$4n^2$	$2n^3$	$n/2$

*everything is task parallelism on some level

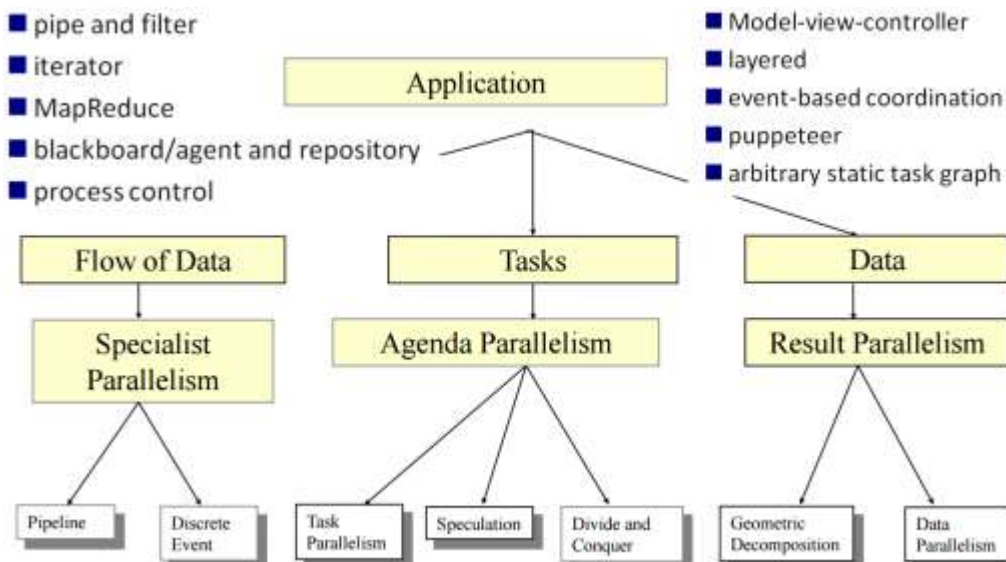
*potential for more parallelism within individual agents, tasks, filters, layers, puppets (but not puppeteer)

<p>Pipe-and-filter most things pipe/filter@highest level</p> <p>Strategy: Pipeline</p> <p>Amt. Parallel: as much as there are filters</p> <p>Examples: Compiler, Image Retrieval, Logic Optimizer</p> <p>[L]</p>	
<p>Iterator [D]</p> <p>Strategy: Data, Task, Pipeline...depends on what's inside</p> <p>Amt. Parallel: All parallelism inside blocks within iterator loop</p> <p>Examples: Support Vector Machine Training</p>	
<p>Map-Reduce [H]</p> <ul style="list-style-type: none"> computations mapped onto independent data results then summarized (reduce) <p>Strategy: Data</p> <p>Amt. Parallel: A LOT; potential as much as # datasets mapped onto</p> <p>Examples: SVM, Speech Rec, Img proc, convolution, feature extraction, edge detection, fluid dynamics, heat maps, index web docs</p>	
<p>Agent/Repository [M]</p> <ul style="list-style-type: none"> Blackboard structural pattern Blackboard=repo shared by all agents (circuit database) Agents=act on blackboard (optimization) Manager=control blackboard access/collect results (scheduler) <p>Strategy: Task; Data if repo can be partitioned into datasets</p> <p>Amt. Parallel: As much as #agents; data in repo; repo is bottleneck</p> <p>Examples: Compiler Optimization</p>	
<p>Process Control [L]</p> <ul style="list-style-type: none"> Processor=underlying phenomena to be controlled/computed Actuator=tasks affecting process Sensor=tasks analyzing state of process Controller=tasks controlling which actuators should be affected <p>Strategy: Task, Discrete Event</p> <p>Amt. Parallel: For each significant task</p> <p>Examples: Thermostat</p>	
<p>Model-View-Controller [M]</p> <ul style="list-style-type: none"> Model=data/intelligence (business logic) of system Controller=capture input, translate to action on model View=renders current state of model for user <p>Strategy: Task</p> <p>Amt. Parallel: One task for each of model, view, controller</p> <p>Examples: Popup charts thing</p>	

Layered Systems [L] <ul style="list-style-type: none"> Individual layers big; interface between layers narrow Nonadjacent layers can't directly communicate Strategy: Task Amt. Parallel: One task for each layer Examples: OSI Network Protocol	
Event-Based Systems [L] <ul style="list-style-type: none"> Agents interact via events/signals in medium Event manager manages events Interaction among agents dynamic; no fixed connection Strategy: Task Amt. Parallel: 1 task for each agent + event manager Examples: Internet	
Puppeteer [H] <ul style="list-style-type: none"> Guide interaction between simulation code No central repo; data transfer between simulators Strategy: Task Amt. Parallel: 1 for each puppet + puppeteer Examples: Blood flow through vessels computations	
Static Task Graph [L] <ul style="list-style-type: none"> Tasks receive inputs; produce outputs All data sharing through explicit messaging Task config statically defined; cannot be changed at runtime Strategy: Task Amt. Parallel: 1 for each task Examples: One game architecture	

Layers of Parallelism

Top level: iterator little parallelism
 Mid level: pipe-and-filter some parallelism
 Low level: mapreduce lots of parallelism
 *Agent/repo, static-task, event-based



Logic Optimization:	Pipe-and-filter
Representing 1 circuit:	Agent-and-repository
Optimizing Circuit Area:	Process Control
Remap Circuit Gates:	Dynamic Programming
Static Time Analysis:	Graph Algorithm/Dynamic Programming