Unit 1 – Introduction to Parallel Processing

Q1. Parallelism in uniprocessor vs parallel computer structures; control-flow vs data-flow computers. (≈300 words)

Traditional uniprocessor system me ek hi CPU sequential instructions execute karta hai, lekin phir bhi kaafi level ka parallelism possible hai. Instruction-level parallelism (ILP) me independent instructions overlap hote hain; pipelining, multiple functional units, out-of-order execution isko exploit karte hain. Micro-architectural techniques jaise caches, prefetching, branch prediction bhi effective parallel work increase karte hain. Parallel computer structures explicitly multiple processing elements use karte hain. SISD traditional single instruction, single data model hai. SIMD (Single Instruction, Multiple Data) me ek control unit same instruction ko ek saath multiple data elements par execute karwata hai (vector/array processors). MISD rare theoretical model hai. MIMD (Multiple Instruction, Multiple Data) me multiple autonomous processors different programs/different data par kaam karte hain; multiprocessors, multicomputers is category me aate hain.

Control-flow computers normal von Neumann style systems hain jahan execution flow program counter (PC) aur control instructions (branches, jumps) se decide hota hai. Program memory me stored instructions sequentially fetch–decode–execute pipeline follow karte hain; data compute ke liye PC-driven control stream essential hota hai. Data-flow computers me execution data availability par based hoti hai, PC par nahi. Program ko ek directed graph me represent kiya jata hai jahan nodes operations, edges data dependencies hote hain. Jab kisi operation ke required inputs available ho jate hain, wo asynchronously fire ho sakta hai. Is model me implicit parallelism naturally emerge hota hai kyunki independent nodes simultaneously execute kar sakte hain. Practically, modern superscalar, out-of-order processors internal data-flow style scheduling use karte hain, lekin overall architecture abhi bhi control-flow programming model expose karta hai. Parallel processing ka main goal hardware + model dono level par concurrency ko identify, schedule aur efficiently execute karna hai.

Unit 2 – Memory & I/O Subsystems

Q2. Memory hierarchy, cache characteristics & organization. (≈300 words)

CPU speed aur main memory speed me gap ko bridge karne ke liye memory hierarchy design ki jati hai. Top par sabse fast, small, costly memories hote hain; neeche jaise-jaise size badhta hai, speed kam aur cost per bit kam hoti hai. Typical hierarchy: registers → L1/L2/L3 cache → main memory (DRAM) → secondary storage (SSD/HDD). Registers CPU ke andar sabse fast hote hain; caches recently used data/instructions store karke average memory access time drastically reduce karte hain.

Cache memory small, high-speed SRAM based storage hoti hai jo locality of reference (temporal + spatial) ka fayda uthati hai. Temporal locality ka matlab recently accessed data ko phir jaldi access hone ka chance high hai; spatial locality ka matlab nearby addresses access hone ke chances zyada hain. Cache hit hone par data quickly mil jata hai; miss hone par data lower level memory (main memory) se fetch karke cache me load kiya jata hai. Cache organization mapping policy par depend karta hai: direct-mapped cache me har memory block ek fixed cache line me map hota hai; fully associative cache me block kisi bhi line me aa sakta hai; set-associative cache in dono ka compromise hai jahan cache sets me divide hota hai, har block ek set ke andar kisi bhi line me aa sakta hai.

Replacement policy (LRU, FIFO, Random) decide karta hai ki miss par kaun-sa block evict hoga. Write policy (write-through vs write-back) decide karti hai ki updates memory tak kaise propagate honge. Addressing schemes me physical, virtual ya mixed addressing use ho sakta hai, jahan TLB virtual memory ke sath cache ko coordinate karta hai. Achhi cache design CPU utilization badhati hai, bus traffic optimize karti hai aur effective bandwidth increase karti hai, jo parallel aur high-performance architectures me critical factor hai.

Unit 3 – Pipelining & Vector Processing

Q3. Linear pipeline principles, hazards, aur vector processing ka overview. (≈300 words)
Pipelining assembly line jaisa concept hai jahan instruction execution ko multiple stages (fetch, decode, execute, memory, write-back) me divide karke overlap kiya jata hai. Linear pipeline me har stage ek fixed function perform karta hai; har clock cycle me new instruction enter hota hai, aur ideally n-stage pipeline n× speedup de sakti hai. Practical systems me structural, data, aur control hazards is ideal speedup ko limit karte hain. Structural hazard tab aata hai jab instructions same hardware resource ek time par maangte hain. Data hazards tab aate hain jab ek instruction previous instruction ke result par depend ho; RAW (read-after-write) typical case hai. Control hazards branches ke karan pipeline flow change hone se hote hain; solutions me branch prediction, delayed branching, speculative execution include hain.

Reservation table pipeline stage usage ko time vs resource grid me dikhata hai. Is se job sequencing aur collision prevention plan kiya ja sakta hai; forbidden latencies identify karke valid initiation interval choose kiya jata hai. Instruction pipelining ke alawa arithmetic pipelines (jaise floating-point operations) bhi design kiye jate hain jahan multiplier/adders ko stages me tod kar high throughput achieve ki jati hai.

Vector processing specially scientific and engineering workloads ke liye design ki gayi hai jahan same operation large arrays/vectors par apply hota hai. Vector processors ek instruction se whole vector par operation perform kar sakte hain, jisse loop overhead kam hota hai aur pipeline utilization high rehti hai. Vector supercomputers (CRAY family jaisi) powerful vector registers, deep arithmetic pipelines aur high memory bandwidth provide karte hain. Attached Scientific Processors (ASP) host mainframe ke sath co-processor ke roop me connect hote hain aur heavy numerical tasks offload karte hain. Vector + pipeline combination continuous stream of data ko high throughput ke sath process karne ka best example hai.

Unit 4 – SIMD Array Processors & Parallel Algorithms

Q4. SIMD array processor structure, interconnection networks, aur matrix multiplication ka parallel idea. (≈300 words)
SIMD (Single Instruction, Multiple Data) array processors me ek central control unit hota hai jo

same instruction broadcast karta hai, jabki multiple processing elements (PEs) alag-alag data elements par parallel operate karte hain. Har PE generally local registers/memory ke sath simple ALU hota hai. Is tarah ke architectures image processing, matrix operations, signal processing jaise data-parallel tasks ke liye ideal hote hain.

PEs ko connect karne ke liye different interconnection networks use hote hain. Linear array aur 2D mesh sabse simple topologies hain jahan har PE apne neighbors se directly connect hota hai. Advanced multistage networks me log N stages hoti hain: shuffle-exchange, butterfly, omega network, barrel shifter, cube, Illiac patterns, etc. In networks ka design permutation, routing cost, diameter (max hops) aur hardware complexity ka trade-off manage karta hai. Crossbar full connectivity deta hai par cost prohibitive hoti hai; multistage networks thode indirect paths ke price par hardware bacha lete hain.

SIMD matrix multiplication me common approach hota hai har PE ko result matrix C ke ek element ya block assign karna. Suppose N×N matrix multiplication hai; PE(i,j) ko C[i][j] compute karna hai. Control unit loop ke through row A[i][k] aur column B[k][j] ke elements broadcast/route karta hai, PE partial products multiply karke local accumulator me add karta hai. Network ka role yahan data ko efficiently shift/broadcast karna hai—jaise, systolic arrays me data rhythmic pattern se pipeline hota hai. Parallel sorting ke liye bitonic sort ya odd-even transposition algorithms array processors par implement kiye jate hain, jahan compare-exchange operations neighbors ya network-connected PEs ke beech parallel perform hote hain. Key idea hai deterministic communication pattern + uniform operations, jisse SIMD model efficiently exploit ho sakta hai.

Unit 5 – Multiprocessors & Scheduling

Q5. Loosely vs tightly coupled multiprocessors, UMA/NUMA/COMA memory, aur scheduling basics. (≈300 words)

Multiprocessor systems multiple CPUs ke sath single system image provide karte hain. Tightly coupled multiprocessors ek shared main memory aur usually common OS instance ka use karte hain; processors high-speed interconnection (bus, crossbar, multiport memory) se linked hote hain. Ye fine-grain sharing ke liye suitable hain, jahan threads same address space share karte hain. Loosely coupled multiprocessors (multicomputers, clusters) me har node apna local memory aur OS chala sakta hai; nodes messages ya network (LAN) ke through communicate karte hain, jisse coarse-grain parallelism support hota hai.

Memory organizations: UMA (Uniform Memory Access) me har processor ke liye main memory ka average access time same hota hai—simple programming model, but scalability limited. NUMA (Non-Uniform) me memory physically modules me distributed hoti hai; processor apne local memory ko fast access karta hai, remote module ko slower; hardware cache-coherent NUMA (CC-NUMA) consistency maintain karta hai. COMA (Cache-Only Memory Architecture) me main memory bhi distributed caches ke roop me treat hoti hai, jahan data dynamically migrate/replicate hota hai.

Multiprocessor OS ko tasks scheduling, process synchronization, shared data protection, load balancing aur fault tolerance handle karna hota hai. Scheduling strategies multiprocessor ke dimensions par based hoti hain: centralized vs distributed scheduler, static vs dynamic load balancing, space-sharing vs time-sharing, gang scheduling for communicating threads, etc. Language features (threads, parallel loops, message passing, synchronization primitives) concurrency expose karte hain. Classic example matrix multiplication on concurrent processors: each processor ko matrix ke rows/blocks assign karna, shared or distributed memory me data layout plan karna, aur minimal communication ke sath result combine karna. Achhi scheduling CPU utilization high rakhti, waiting time kam karti aur scalability improve karti hai.

Unit 6 – RISC & Superscalar

Q6. RISC architecture ke features, SPARC windowed registers, aur superscalar processors ka overview. (≈300 words)

RISC (Reduced Instruction Set Computer) architecture ka focus simple, fixed-length instructions, load/store model, large register set aur hardwired control par hota hai. Key features: limited addressing modes, few instruction formats, single-cycle execution for most instructions, aggressive pipelining, compiler-friendly design. Is simplicity se high clock rate, efficient pipeline aur easier hardware design possible hota hai.

SPARC typical RISC scalar processor ka example hai. Iski sabse unique feature register window mechanism hai. Instead of small fixed register set, SPARC large overlapping register windows use karta hai; har procedure call per ek naya window allocate hota hai jisme parameters, local variables, return values ke liye registers milte hain. Windows overlap se caller–callee ke beech arguments pass karna efficient ho jata hai, memory stack access kam ho jata hai. Procedure nesting me windows circular buffer jaisa behave karte hain; deep recursion par window overflow/underflow trap ke through memory spill/fill ki jati hai.

Superscalar processors ek clock cycle me multiple instructions issue aur execute karne ke liye design hote hain. Hardware multiple parallel pipelines aur functional units provide karta hai. Front-end instruction fetch/decode kar ke dependencies analyze karta hai; independent instructions ko same cycle me alag units par schedule kiya jata hai. To achieve this, techniques like register renaming, dynamic scheduling, out-of-order execution, speculative execution, branch prediction use hote hain. Objective hai instruction-level parallelism ko maximum exploit karna bina programmer ko explicit parallel code likhwaye. RISC philosophy + superscalar hardware combination modern high-performance CPUs ka base hai, jahan simple instructions pipeline/superscalar execution ke through very high throughput achieve karte hain.