***Chapter 1***

**7 dimensions of ISA:** Class of ISA: Memory addressing; Addressing modes; Types and sizes of operands; Operations; Control flow instr.; Encoding an ISA.

**Amdahl’s Law**: 提升系统部分性能时，整体性能提升取决于这一部分的重要性(设这部分占比, 提升倍，系统提升)

**性能提升的限制**：ILP；Memory；Power。

**SISD**（ILP），**SIMD**（Single Instruction Multiple Data，DLP），**MISD**（useless），**MIMD**（TLP）

**可靠性相关概念**：MTTF平均故障时间，MTTR平均修复时间，FIT故障率=1/MTTF，MTBF平均故障间隔时间=MTTF+MTTR。可用性MTTF/MTBF。对于一个系统Failure Rate=，系统整体MTTF=1/Failure Rate。某时段内故障磁盘数=磁盘数\*时间/MTTF

**MIPS**：每秒百万条指令数。

**处理器性能公式**：CPU time=程序时钟周期数\*时钟周期=指令数\*CPI\*时钟周期时间；IC指令数；CPI每条指令的时钟周期数=CPU时钟周期数/指令数；IPC每时钟周期指令数。

***Chapter 2***

DRAM（用在main memory容量大，便宜），SRAM（用在cache，速度快，贵）

**Cache原理**：时间/空间局部性。

**Cache映射模式**：Direct mapped（address % #blocks in cache）；Fully associative（块可以在cache中任意位置）；Set associative（按组分address % #sets in cache）一组中有n个blocks即为n-way associative。

**Cache Line**：一般有valid位，dirty位，Tag，Data，注意data长度为一个block的长度而不是一个字长。

一个**物理地址**从高到低：Tag/Index（在cache中）/block offset。

**替换策略**：direct-mapped（一个人一个坑直接踢走）；set-associative（随机；LRU踢出最近最少用到的line，一般用近似LRU不熟悉可以写一下；FIFO）

**写策略**：Write-back（仅在成为victim且dirty时写回内存，write hit时只需要写cache更快，同一块多个字被写时memory带宽更小）；Write-through（每次更改同时改Cache和memory，read miss时不需要写memory（即写回），能保证cache和memory中数据的一致性，需要write buffer尽量避免stall）

**Write-miss时**：write-allocate（将要写的块移进cache中再写）；write-around（直接写内存）

**Unified cache**（所有内存请求经过同一个cache）；**Split** I&D cache（区分指令和数据cache，I cache只读）

**CPU Execution Time =** (CPU clock cycles + Memory stall cycles) \* Clock Cycle Time **=** IC \* (ALU\_instr.% \* CPI\_ALU + Mem\_instr.% \* AMAT) \* clock cycle time; **Memory stall cycles** = *IC* \* Mem\_instr.% \* miss rate \* miss penalty; **CPU clock cycles** = *IC* \* CPI

**AMAT**(Average Memory Access Time) = Hit time + (Miss rate \* miss penalty) = (Inst part) + (Data part).

对于Out-of-order处理器，miss penalty需要去除重叠overlapped部分的比例（比如重叠部分30%，则乘0.7）

**提升DRAM性能**：FPM（Fast Page Mode DRAM,同一页的内存访问，只发一次ROW，后多次更改COL）；Synchronous DRAM（重叠数据读出和下一个地址的传送）；DDR（double data rate，正负边沿都进行读写）；RAMBUS（）

**Cache优化方法**

**降低hit time**：small and simple cache；Way-prediction；避免地址转换（TLB，virtual indexed physically tagged cache，此时要求page offset >= block offset + index）;Trace cache(动态指令序列，一个line保存一个最小段的多条指令)

**提高带宽**：pipelined caches；multibanked cache；Nonblocking cache（L2一定要支持，允许在miss处理完之前提供hit的数据，用于OOO execution）

**降低miss penalty**：多级cache（AMAT=L1-hit-time + L1-miss-rate \* (L2-hit-time + L2-miss-rate \* L2-miss-penalty)）；提高read miss优先级（有write buffer，需要注意检查要读的内容是不是还在write buffer中）；Critical word first&early restart（block size较大时有效。分别是：先请求miss的word读到CPU就恢复执行，整块慢慢读到cache中；从头读块，读到需要的字就传给CPU并继续运行）；Merging write buffer；victim caches（给victim添加一个cache，如果很快用到的话直接拿回）

**降低miss rate**：大block size（减少compulsory miss，增加capacity&conflict miss，增加miss penalty）；大cache（减少miss rate，增加hit time，增加功耗）；高关联度（减少conflict miss，增加hit time，增加能耗）；编译优化（合并数组（类似结构体），交换循环行列，合并循环，分块）

**指令预取（不明白的话需要看ppt）**

***Chapter 3***

**Latency**：一个指令产生结果到另一个指令可以使用这个结果的cycle差值。

**Initiation interval**：一个功能单元接收两条指令的cycle差值（完全流水化是1，完全不流水化是latency+1）

RAW（真依赖），WAR（反依赖），WAW（输出依赖）

**动态调度**的核心思想：允许被stall指令后边的指令先执行。按序发射，乱序执行，乱序完成。

**Scoreboard**：有一个集中控制的单元，IF IS（issue，避免WAW和结构冲突：功能模块可用，正在飞的指令中没有相同目的寄存器）RO（Read Operands，避免RAW：两个操作数都可用）EX（直到execution结束，功能单元才可用） WB（检测WAR，必要时stall）

**Scoreboard添加结构**：Instruction status（每一条指令在哪步IS RO EX WB）；Function Unit Status（busy，op，Fi目标寄存器，Fj，Fk源寄存器，Qj，Qk操作数将从哪个功能单元得到结果），Rj，Rk（操作数是否可用）；Register result status（那个功能单元将写这个寄存器）

**Tomasulo**：保存站（解决WAW和WAR，存放待执行的指令，分布在每个功能部件，指令中的源操作数被替换成值或者RS的编号，一条的内容为busy，op，Vj, Vk源操作数的值，Qj，Qk产生源操作数的保存站，A地址值），CDB（Common Data Bus，通过RS通知运算结果，而非从寄存器堆中读），存储器读写指令也有RS。Register Result Status（记录哪一个功能部件将改写这个寄存器）**具体步骤复习PPT**

**Tomasulo的阶段**：Issue（保存站可用，renaming，ROB可用，记录目标寄存器的ROB no.），Execute（两个操作数可用），Write Result（在CDB上广播，标记这条指令对应保存站可用，通知ROB），commit（ROB头产生结果，更新寄存器并移除该ROB项，；如果错误预测，需要回刷跳转指令及之后的ROB项）

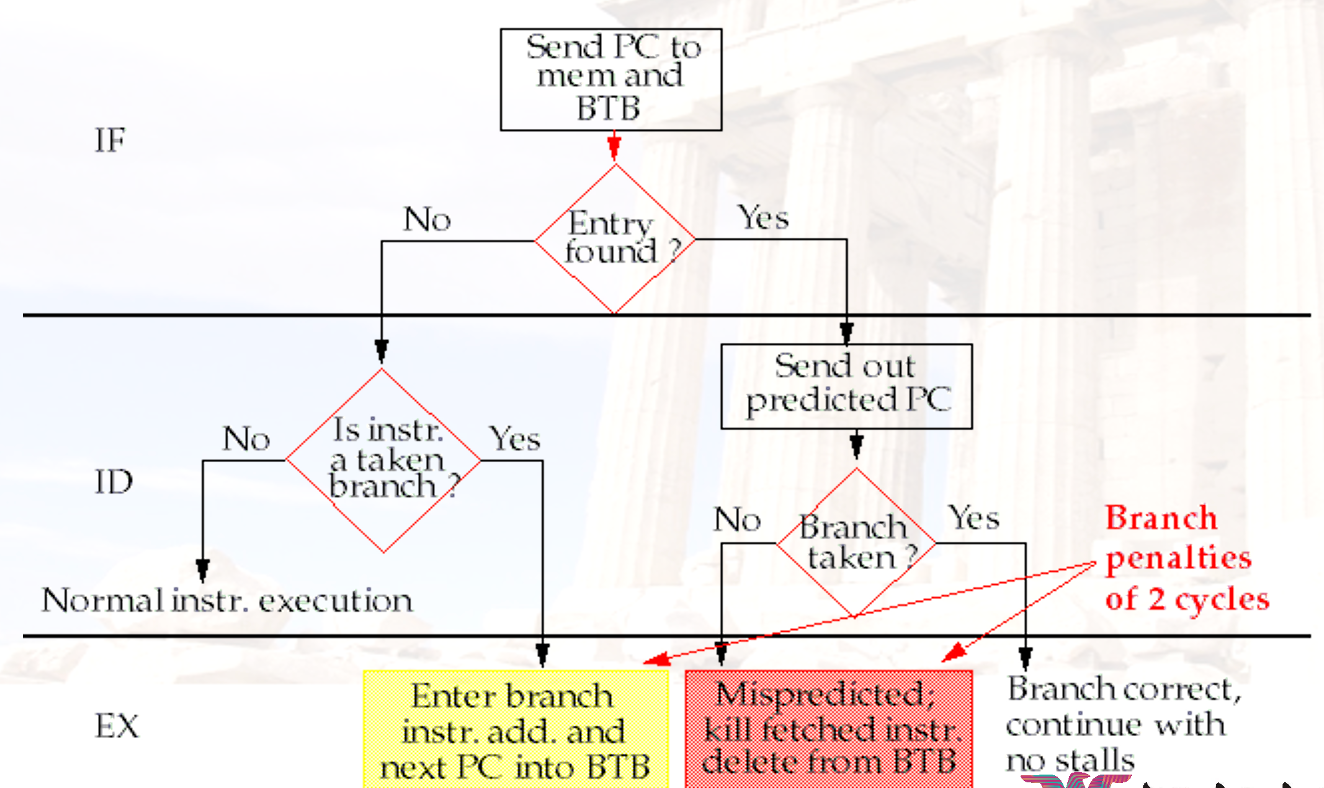
Tomasulo的**好处与缺点**：分布式的冲突检测逻辑，消除了WAR和WAW的停顿；比较复杂，需要高速的CDB，同一周期可以完成的指令数限制为1（可以设置多条CDB），非精确中断

**Correlating predictor**：(m, n)预测器利用最近m个分支行为在个n位分支预测器中选择，比如共64项的(2, 2)缓冲区中，分支的低4位和全局2位组成6位索引。

**Tournament predictor**：用地址索引局部预测器和选择器，用分支历史索引全局预测器。

**gshare**：全局分支历史与分支地址异或做索引。

**Branch Target Buffer**：使用PC索引预测的PC。



**Integrated Instruction Fetch Units**：branch prediction; instruction prefetch; instruction memory access and buffering.

**Return address predictors**：用栈存储return address

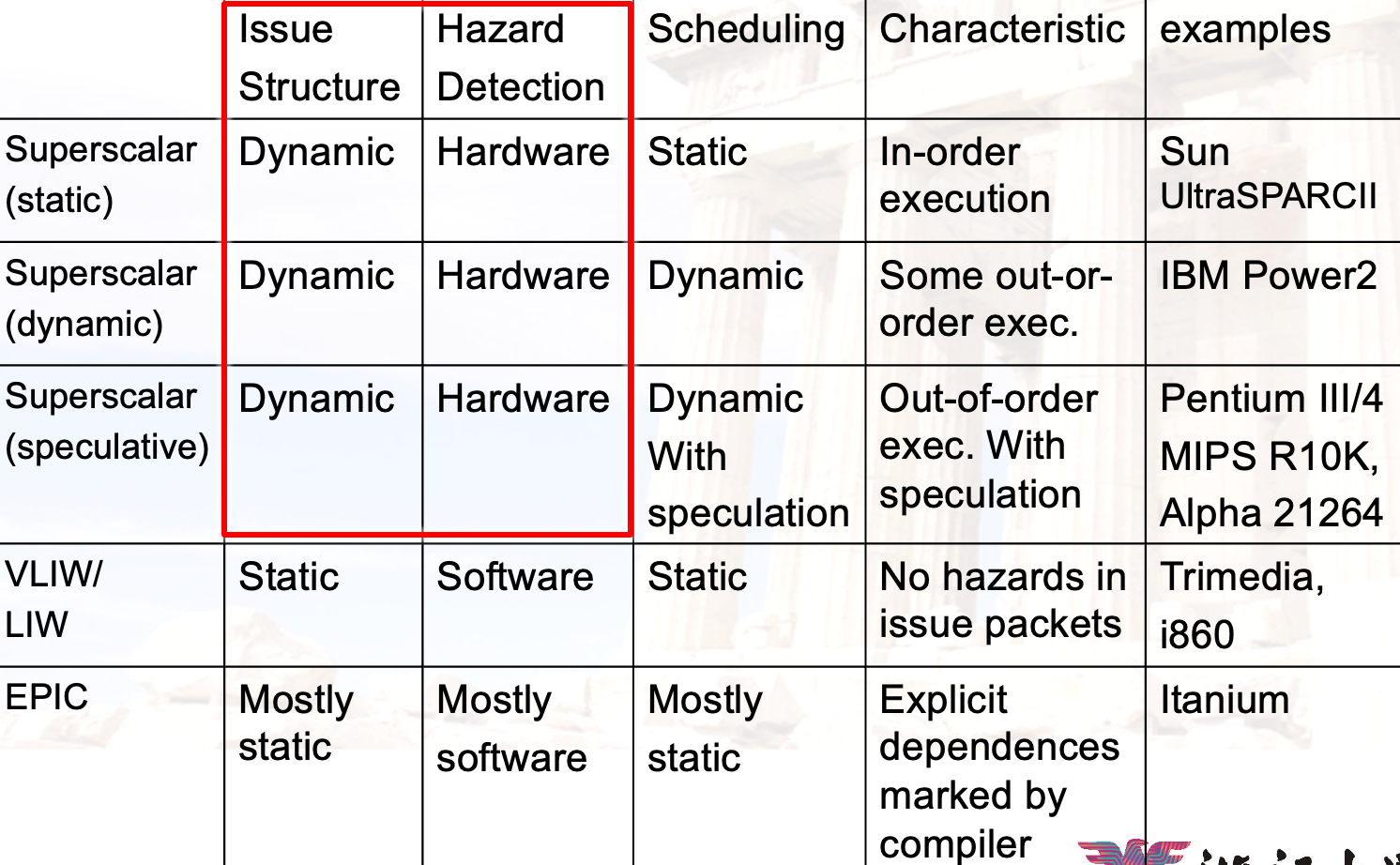
**Speculation**：乱序执行，按序提交。在Tomasulo的基础上添加ReOrderBuffer，取消store buffer，通过ROB换名，RS仅在issue和exe之间保存op和操作数，ROB负责exe结束到commit。

**ROB line**：指令类型，目标，value，ready，exception vector。

**Commit**：非跳转（更新寄存器或mem，移除ROB项）；错误跳转（刷新ROB，从正确PC重新开始）

**避免内存RAW**：buffer还未写入的store指令的地址与值，每一次load指令（还有store没有算出来地址：stall）（有store地址与load相同，取用记录下来的值，如果值还没有，记录这条store对应ROB项并等待）

**记分牌停顿**于issue（结构竞争和WAW）WB（WAR）；**tomasulo停顿**于issue（RS结构竞争）；**带speculation停顿**于issue（RS和ROB结构竞争）。



**Static superscalar的问题**：对指令序列要求高，换名逻辑复杂，需要多条总线或旁路。

SMT（同时多线程）

***Chapter 4***

**向量处理器**：一个向量指令含有多个任务；每个任务之间是独立的，不依赖之前的结果；内存访问有固定的形式；减少了分支跳转。有memory-memory（难以重叠指令：需要检查内存地址的依赖，有更高的启动延迟，需要较高的内存带宽）和vector register两类。

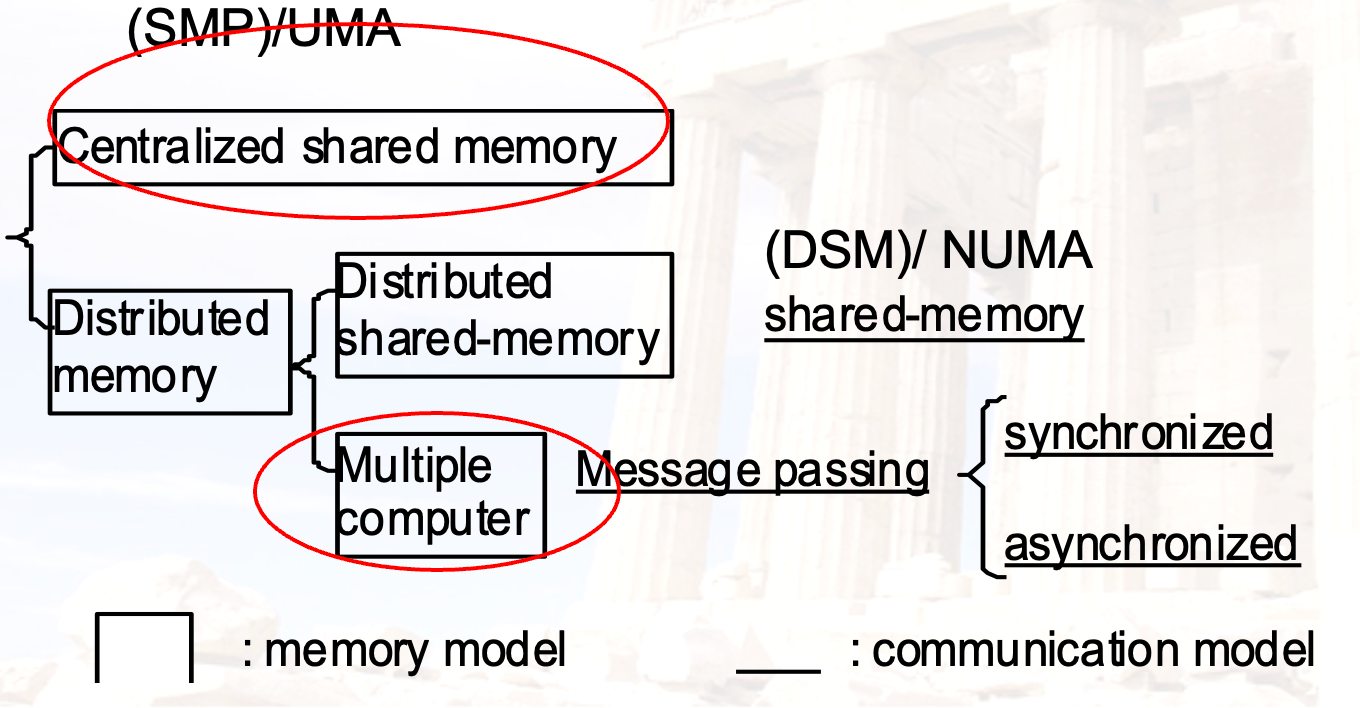
**Chaining**：允许下一条向量指令在上一条开始出结果时就启动。要求两个指令是有RAW依赖的。

当序列长度大于MVL（Maximum Vector Length）时需要建立循环处理。

向量运算不能只看峰值速率，还要考虑启动时间。

All GPU loads are gather instructions and all GPU stores are scatter instructions

***Chapter 5***



DSM（Distributed shared memory）

UMA（Uniform Memory Access）：不同处理器接触内存时间相同，数据存放位置不重要，整体系统延迟大，适用于小型多处理器；NUMA：接触自己的内存延迟低，数据存放位置重要，冲突更少，适用于大型。

**编程模型**：multiprogramming（无交流），Shared address space（通过memory共享），message passing（通过消息传递），data parallel（在不同数据上同时处理，并全局实时交换信息）

**交流模式**：shared address space（load/store/atomic swap，每个处理器都能识别机器中任何一个物理地址，每个进程都能识别和其他进程共享的内存）；message passing（send/receive/library calls）

