Presentation 6

Queuing Theory: Memory Buffer Limits on Superscalar Processing

תורת התורים עוסקת בהתנהגות מערכות שבהן חוצץ (buffer) יוצר תור של בקשות לטיפול. מערכות כאלו נפוצות מאוד במחשבים, מהתור לגישות לדיסק הקשיח ועד למערכות שרת/לקוח מבוזרות. ניתן להתייחס למפענח (decoder) במעבד כלקוח ששולח בקשות לשירות ליחידות הביצוע (EUs) שפועות כשרת, באמצעות ה- ROB שמתפקד כחוצץ. על פי המודל, תורת התורים נותנת חסמים על היעילות של שיטת superscalar.

Slide 2 – 5 Client/Server Model of I/O

Queuing theory assumes that a fast CPU acts as a client for the services of a slower device over an I/O channel (input/output). The buffer stores client requests and forwards them to the server at the slower server response rate. Some basic definitions are:

Latency	The time between the client request and the server response.
Throughput	The actual number of services provided per unit time by the server.
Bandwidth	The maximum data transfer rate on the I/O channel (including the buffer).
Capacity	The maximum throughput of the server receiving requests through buffer (depends on the bandwidth and the service rate).
Utilization	The request rate as a proportion of capacity.

The client makes requests to the server intermittently. Most of the time the client makes no requests, but while requests are made the volume of requests per second can reach the peak capacity of the client. The client's peak request rate is much higher than its average request rate over a long time. This is called **bursty** behavior (the client emits requests in short, high speed bursts). Generally, the peak client request rate is also higher than the server's maximum service rate.

The FIFO buffer stores client requests in order to forward them to the server at the maximum rate that the server can accept them. The buffered requests form a **queue**.

The **minimum latency** is determined by the server's maximum service rate. Since the service rate is expressed as services per second, the latency is the inverse, expressed as seconds per service. The minimum of seconds per service is the time required to service one request.

If the client continues to fill the buffer with requests at a rate faster than the service rate of the server, the buffer will eventually overflow. The **net buffer fill rate** = fill rate – empty rate. For example, a CPU can send a hard disk 10^9 read requests per second. The disk can only respond to 10^7 read requests per second (it takes 10^{-7} seconds to service one request). If the buffer can hold 1000 read requests, and the CPU makes requests continuously, the buffer will overflow in about $1000 / (10^9 – 10^7)$ seconds.

As utilization grows, the client makes more requests per second. But the server cannot work faster, so more requests are held in the buffer (the queue length grows). The total latency for any given request is the time required to process that request plus the time required to process the requests in the queue when the request is made. As the queue grows longer, so does the latency for any given request. A basic result of queuing theory is that **latency and utilization grow together**, so that low latency requires low utilization.

Slide 6 Queuing Theory

Queuing theory makes the following **assumptions**:

1. **Client Requests** arrive independently and obey Poisson statistics. Each request has a random length (bytes to transfer) and requires a random service time.
2. The **buffer** stores requests and forwards them in order of arrival (FIFO) at the maximum available service rate. Over a long time, the buffer reaches an average "**steady state**" level of stored requests.
3. The server provides services to each request independently with a **service time** obeying Poisson statistics.

The **principal results** are:

1. Average latency = 1 / (average service rate – average arrival rate).
2. Average buffer level = average arrival rate × average buffer latency (Little's Law).

Slide 7 – 16 Queuing Theory Details

Slides 7 to 16 show the details of the **derivation** of the basic queuing theory results.

Slide 7 reviews the Poisson statistics for randomly distributed independent events. The average arrive rate is given by α (the Greek letter alpha).

Slide 8 describes the client requests as Poisson events with an average request rate given by λ (the Greek letter lambda) and describes the servicing of a request as Poisson events with an average response rate of μC (the Greek letter mu times C).

Slide 9 shows the possible transitions for a buffer whose level (number of buffered requests) is given by n at time $t + \Delta t$:

1. The level at t was n – 1 and 1 request arrived during the time interval Δt
2. The level at t was n + 1 and 1 request was serviced during the time interval Δt
3. The level at t was n, 0 requests arrived and 0 requests were serviced during the time interval Δt
4. The level at t was n, 1 requests arrived and 1 requests were serviced during the time interval Δt

Using the rules for independent events, the total probability of buffer level n at time $t + \Delta t$ is given on slide 10 as the sum of these four possibilities. Each possibility is the probability of the previous buffer level at time t and the probabilities of new requests and requests serviced.

Slide 11 makes a linear approximation of the probabilities for very small interval Δt.

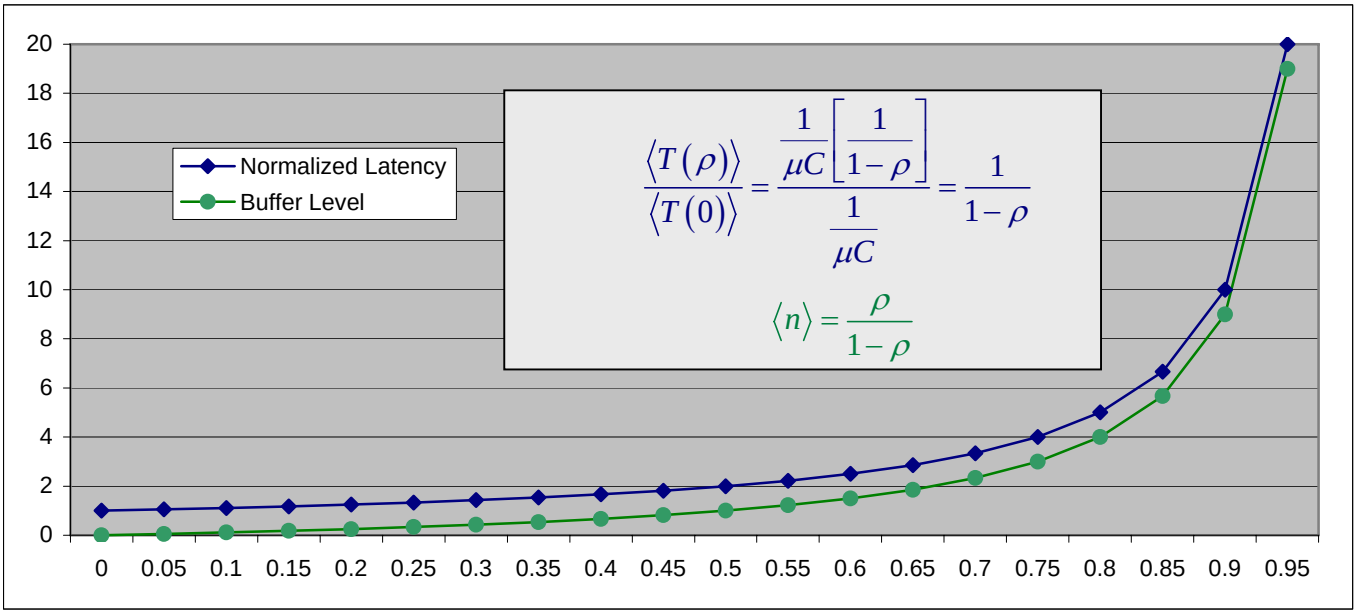
Slide 12 uses the steady state assumption to show that the probability of a 0 to 1 transition must exactly balance the probability of a 1 to 0 transition. Using the approximate expressions for the probabilities, this leads to $P_1 = \rho P_0$, where ρ is the system utilization.

Slide 13 uses the steady state assumption for the general case, following a few algebraic manipulations to set the time-derivative of the probability to 0 (steady state ⇒ no change). This leads to a recursion relation for P_{n+1} in terms of P_n and P_{n-1}.

Slide 14 inserts the relationship between P_1 and P_0 to show that $P_n = \rho^n P_0$ for all n. Since the sum of all probabilities must sum to 1. The probability of buffer level n is found to be $\rho^n(1-\rho)$.

On slide 15, the average buffer level is shown to demonstrate the results claimed on slide 6.

Slide 16 shows the relative values of buffer level and latency as a function of utilization.



Slide 17 Memory Queuing

Slide 17 shows three **examples of a buffer queue** within a single computer system.

The memory ordering buffer (**MOB**) in the x86 microarchitecture buffers data writes to L1 data cache, so that writing can be interrupted to perform higher priority reads.

The sustained throughput (maximum long term write speed) to a **SATA 3.0 hard drive** is 6 Gbps, which is slightly less than 600 Mbytes per second. The sustained long term write speed of the PCI Express 4.0 I/O bus is slightly less than 16 Gbytes per second. The typical write buffer on a SATA hard drive is 64 MB, so that a typical PC I/O system can overflow the disk buffer in 4.16 ms.

External hard drives, especially with a USB interface, must be safely removed before disconnection. Safe removal guarantees that the most recent writes to open files are written to the external device from the USB buffer, and the files are marked closed before disconnection. Unsafe removal can leave unwritten data in the buffer, leading to data loss.

Slide 18 Client/Server Model of CPU

A superscalar CPU can be modeled as a **client/server** system. The fetch and decode stage is modeled as a client that sends λ requests (instructions) per clock cycle to a server (EUs) through a buffer (the ROB). The **buffer level** is the number of instructions in the ROB waiting to be executed. The **instruction window** is the total number of instructions in some stage of

execution in all the EUs in any clock cycle. The **maximum service (execution) rate** μC in the EUs is one instruction window per clock cycle.

The goal of the superscalar CPU is to maximize the CPU utilization $\rho = \lambda / \mu C$ for a fixed maximum execution rate μC without overflowing the ROB. This means making the average decoder rate λ as close as possible to the execution rate μC (but always smaller).

Slide 19 – 20 General Superscalar Model

A **general superscalar** is made up of several execution units (EUs). Each EU may be pipelined with a number of stages. Slide 19 shows an example of a CPU with 2 ALUs (each ALU has 1 stage), 1 load/store EU (2 stages), and 1 FPU (3 stages).

In this example, there can be $2 \times 1 + 1 \times 2 + 1 \times 3 = 7$ instructions in some stage of some EU at the same time. For this CPU, the **instruction window** is 7.

The general case is shown on slide 20, where u_i is the number of EUs for instruction type i, and s_i is the number of pipeline stages in an EU for type i. The instruction window is given by

$$IC_{EU} = \sum_i u_i \times s_i = \text{total pipeline stages in CPU}$$

Slide 21 – 23 Queuing Theory and Decoder Rate

The service rate μC for the EUs is the total number of **instructions per cycle** (IPC) it can execute. For instructions of type i, with each EU operating in parallel,

$$IPC_i = u_i \times IPC_i^{scalar} = u_i / CPI_i^{scalar} = u_i / (1 + CPI_i^{stall})$$

where CPI_i^{scalar} is the number of cycles per instruction on a scalar (single pipeline) processor.

Summing over instruction types, the service rate for the EUs is

$$\mu C = IPC = \sum_i IPC_i = \sum_i \frac{u_i}{1 + CPI_i^{stall}}$$

The queuing theory result is that the **average ROB level** (number of instructions in the ROB waiting to execute) is

$$\langle n \rangle = \frac{\lambda}{\mu C - \lambda}$$

where $\mu C = IPC$. To keep the EUs busy, we need to have the ROB level approximately equal to the instruction window IC_{EU}. Setting

$$IC_{EU} = \frac{\lambda}{IPC - \lambda} \qquad \rightarrow \qquad \lambda = \frac{IC_{EU} \times IPC}{1 + IC_{EU}} = \frac{s \times u \times IPC}{1 + s \times u}$$

Neglecting CPI^{stall}, IPC = u, and so the ideal decoder rate is approximately

$$\lambda^{ideal} = \frac{s \times u^2}{1 + s \times u}$$

Slide 22 shows that in a CPU with 2 integer ALUs (1 stage), 2 FPUs (3 stages), 2 load/store units (2 stages), the instruction window is 12 instructions and the decoder should provide and average of 5.54 decoded instructions per clock cycle (the approximate number in an x86 CPU).

Slide 23 considers the **scaling behavior** of this model. Scaling up from 6 to 15 EUs with and average of 8 superpipelined stages, the decoder would need to provide around 15 instructions per clock cycle, and maintain an instruction window of 120 independent instructions.

Of 120 instructions, around 25 to 30 instructions would be branches, leading to a large probability of mispredictions, and therefore a decoder bottleneck. For this reason, larger superscalar CPUs are not practical.