The shared memory model as closed queueing network

Erik Leonards

March 14, 2023

1 The processor

Consider a processor with n cores that are executing tasks. An active core sometimes want to access the memory. The core does this by sending memory requests to the memory. It might be the case that the return value of the request is only needed in the far future, which means that the core can continue execution of the task while the memory request is being sent and the core doesn't need to wait. It could also be the case that the memory request is necessary in the nearby future, which would imply that the core can only do a little bit of execution of the task before it has to wait for the memory request to return. The measure for waiting that is normally used for processors is utilisation.

1.1 The utilisation

The utilisation for a certain time interval $[t_1, t_2]$ is defined as

$$\text{utilisation} = \frac{\text{active clock cycles}}{\text{total clock cycles}},$$

where the active clock cycles indicate the cycles where the core is actively doing computations. This means that the total clock cycles are made up of the waiting clock cycles and the active clock cycles.

The utilisation changes during the execution of a task, but for now we assume that the utilisation is uniform during the entire execution of the task.

1.2 The bandwidth

It is important to know that the utilisation changes when more cores are active. This is because the memory only has a certain capacity (called bandwidth and denoted with B) with which it can execute the memory requests. The bandwidth is then equal to the number of memory requests it can simultaneously execute. When the rate at which the memory requests arrive is larger then the bandwidth, then a queue in the memory is formed.

Example 1.

- *Let $B = 100$ and suppose there are 2 cores sending memory requests with respectively a rate of 60 and 30. Since the combined rate of the memory requests is $90 < 100 = B$ we have that there is no queue forming in the memory. This means that the isolated execution of the two tasks is equal to the execution of the two tasks when run simultaneously.*
- *Let $B = 100$ and suppose there are 3 cores sending memory requests with respectively rates of $70, 40$ and 10. The combined rate is $120 > 100 = B$, so a queue will form in the memory. Core 1 gets for example only requests back with a rate of 60, core 2 with a rate of 32 and core 3 with a rate of 8. This means that return time of a request is delayed. Since the return values are needed for the execution of the task in the future, the core is slowing down in comparison with the isolated execution. This will have the affect that it takes longer to execute a task and the rate at which memory requests are sent decreases. The rate at which the memory requests are sent will decrease until the arrival rate and the departure rate of requests is the same and an equilibrium is reached.*

1.3 The scenario

We will now consider the case of n tasks running simultaneously without knowing whether the bandwidth is reached.

2 The mathematical model

There are three possible ways of modelling the memory. You can either model it as a queue, where B number of memory requests can be handled at the same time, each with a service rate of μ. This will be called the Deterministic Memory Queue (DMQ) model. This model can be slightly modified to obtain the Probabilistic Memory Queue (PMQ) where the queueing policy is a probabilistic one (the next one to be served by the memory is randomly chosen from all requests currently in the memory).

An alternative model is when the memory has B capacity, where each memory requests receives a service rate of

$$\min\left\{\mu, \frac{B}{m}\mu\right\},$$

when there are m memory requests in the system. This case will be called the Bandwidth Sharing (BS) model, because the bandwidth is equally shared between the requests and each request can receive a maximum service rate of μ.

2.1 The bandwidth sharing model

The Bandwidth Sharing (BS) model is illustrated in Figure 1. The leaving and arriving memory requests are modeled as red and green dots moving between the memory and the cores. The total number of these requests is fixed. For n active cores, let $m_1, m_2, \ldots, m_n$ be the number of requests in the system where m_i is the number of requests belonging to core i. Now define

$$m := \sum_{i=1}^{n} m_i = m.$$

In Figure 1, we have that red dots belong to core 1 and green dots belong to core 2. This means that $m_1 = 13$, because 8 requests (dots) are in the memory being processed and 5 requests (dots) have returned from the memory and are currently in the core. Similarly for core 2, we have $m_2 = 7$.

2.1.1 The departure of requests from the core

The requests in the core have just returned and are waiting to be sent back to the memory. If there are x requests in core i, then requests are send with a rate of $\lambda_i(x)$, with $\lambda_i(x)$ increasing. This means that the rate is dependent on the number of requests in the core. This is in line with the theory, because many requests in the core imply that the the core has just received many requests, so it can probably keep executing the task and sending requests to the memory, without having to wait.

2.1.2 The departure of requests from the core

The requests in the memory have been send by the core and the memory must fulfill the requests. In the bandwidth sharing model the bandwidth of the memory is equally divided between all requests in the memory. This

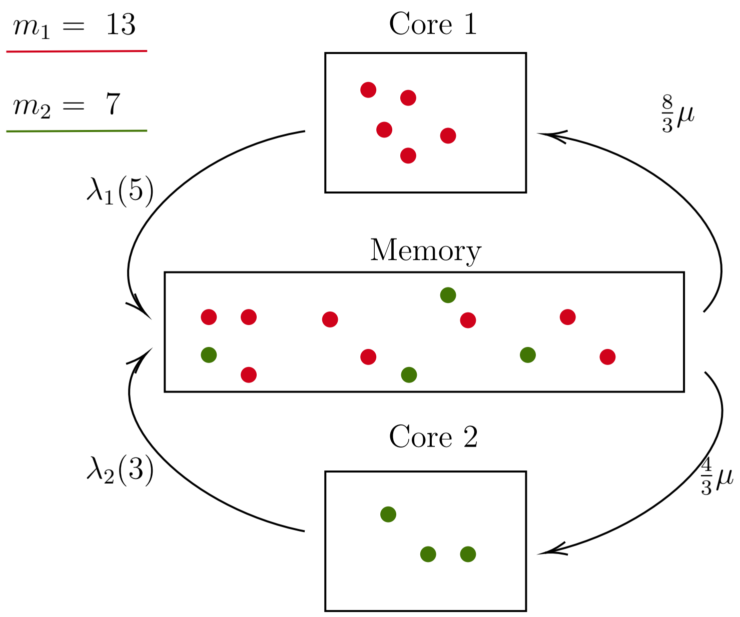

Figure 1: The overview of the bandwidth sharing model where the red and green dots are the requests. We have $B = 4$.

means that a memory running at full bandwidth is serving each of the m requests with a rate of $\frac{B}{m}\mu$, where μ is the rate at which isolated requests are served. In Figure 1 each red dot in the memory is served with a rate of $\frac{B}{12}\mu = \frac{4}{12}\mu = \frac{1}{3}\mu$, because there are 12 total requests in the memory. Since there are currently 8 core1-requests in the memory, the rate at which core1-requests are returning to core 1 is equal to $\frac{8}{3}\mu$. Similarly, the core2-requests are returning to core 2 with a rate equal to $\frac{4}{3}\mu$.

2.1.3 The equilibrium

Suppose a system has n active cores and let C_i and M_i be the number of core-i-requests in respectively core i and the memory. Now define

$$C := \sum_{i=1}^{n} C_i,$$
$$M := \sum_{i=1}^{n} M_i.$$

If the core-i-memory rate $\lambda_i(C_i)$ is greater than the memory-core-i rate, then C_i will decrease. This means that $\lambda_i(C_i)$ will decrease. Furthermore, we have that M_i will increase and the memory-core-i rate will increase. This will happen until the memory-core-i rate is equal to the core-i-memory rate and an equilibrium is reached.

To determine the equilibrium we need to specify the function $\lambda_i(x)$. We must have that $\lambda_i(x)$ is increasing, because more requests in core i means that the core has more information and can faster execute the task and send requests to the memory. A possible approach would be to let $\lambda_i(x)$ be linear, which means that $\lambda_i(x) = \lambda x$ for a certain $\lambda > 0$. Notice how we assume that this value of λ is equal for all $i \leq n$.

To determine the equilibrium we need to distinguish between two cases ($M \leq B$ and $M > B$). If $M \leq B$, then the bandwidth of the memory is enough to serve all requests at the maximum service rate of μ, while this is not the case if $M > B$.

2.1.4 No memory interference

No memory interference between requests in the memory occurs if $M \leq B$, then each request has a service rate of μ in the memory. In the equilibrium it must be the case for all $i \leq n$ that the arrival and departure rate at the core are equal:

$$\lambda_i(C_i) = \lambda C_i = M_i \mu.$$

which means that

$$\lambda(m_i - M_i) = M_i \mu \qquad \Longrightarrow \qquad M_i = \frac{\lambda m_i}{\mu + \lambda}.$$

Moreover, we obtain

$$C_i = \frac{\mu m_i}{\mu + \lambda},$$

because $C_i + M_i = m_i$. Since $M < B$ we must have

$$M = \frac{\lambda}{\mu + \lambda} \sum_{i=1}^{n} m_i = \frac{\lambda m}{\mu + \lambda} < B.$$

2.1.5 Memory interference

Memory interference between requests in the memory occurs if $M > B$. We must have an equal arrival and departure rate of requests in the core and memory:

$$\lambda C_i = \frac{M_i B \mu}{M}.$$

Furthermore, we must have $M_i + C_i = m_i$. This set of equations can be solved to obtain the solution of

$$\begin{aligned} C_i &= \frac{m_i B\mu}{\lambda m}, \\ M_i &= m_i - \frac{\mu B m_i}{\lambda m}. \end{aligned}$$

Since $\sum_{i=1}^{n} M_i > B$, we must have

$$\begin{aligned} &\sum_{i=1}^{n} m_i \left(1 - \frac{\mu B}{\lambda m}\right) > B \\ \implies &\sum_{i=1}^{n} m_i > B + \frac{\mu B}{\lambda m} \sum_{i=1^n} m_i \\ \implies &m > B + \frac{\mu B}{\lambda} \\ \implies &m > B \frac{\mu + \lambda}{\lambda} \\ \implies &\frac{\lambda m}{\mu + \lambda} > B. \end{aligned}$$

This means the two cases can be summarised to obtain

$$\begin{aligned} C_i &= \begin{cases} \frac{\mu m_i}{\mu+\lambda} & \text{if } \frac{\lambda}{\mu+\lambda} \le B \\ \frac{m_i B \mu}{\lambda m} & \text{otherwise} \end{cases}, \\ M_i &= \begin{cases} \frac{\lambda m_i}{\mu+\lambda} & \text{if } \frac{\lambda m}{\mu+\lambda} \le B \\ m_i - \frac{m_i B\mu}{\lambda m} & \text{otherwise} \end{cases}. \end{aligned} \tag{1}$$

2.2 The Probabilistic Memory Queue model

The Probabilistic Memory Queue (PMQ) model is illustrated in Figure 3. This model is very similar to the BS model, but the difference is that at most B number of requests are served with a rate of μ. Each request in the memory is now either in the waiting area, or it is being served with service rate μ. In Figure 2 there are 3 red dots being served by the memory, so the core1-arrival rate is 3μ. Similarly, there is 1 green dot that is being served, so the core2-arrival rate is μ.

2.2.1 The equilibrium

Just like the BS model we are interested in finding the equilibrium of the model when the system has n active cores. Let C_i, W_i and E_i be the number

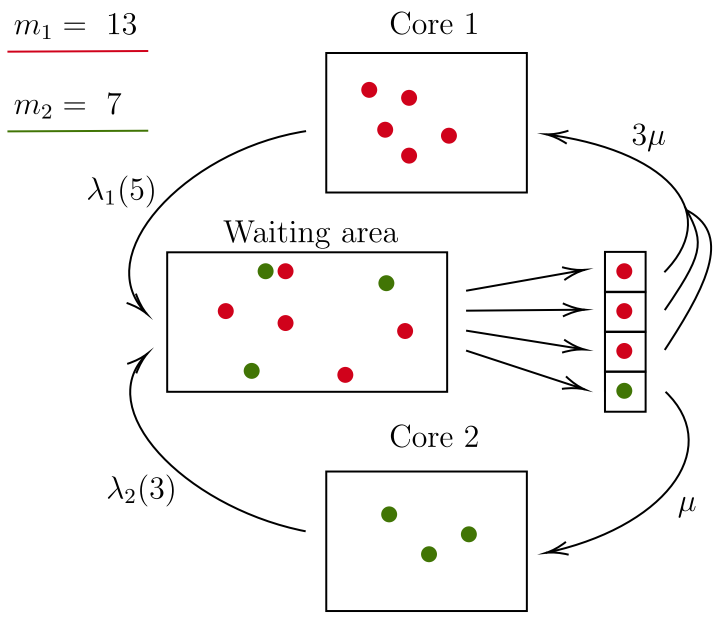

Figure 2: The overview of the Probabilistic Memory Queue (PMQ) model where the red and green dots are respectively the core1- and core2-requests. We have $B = 4$.

of core-i-requests in respectively core i, the waiting area and the execution area. Define

$$C := \sum_{i=1}^{n} C_i,$$

$$W := \sum_{i=1}^{n} W_i,$$

$$E := \sum_{i=1}^{n} E_i.$$

We have $m_i = C_i + W_i + E_i$ and $E \leq B$. We will once again consider the functions $\lambda_i(x) = \lambda x$ for all $i \leq n$. In equilibrium we must have an equal arrival and departure rate in all cores:

$$\lambda C_i = E_i \mu. \tag{2}$$

To calculate the equilibrium we need to distinguish between two cases. The first case is $W = 0$ (no requests in the waiting area) and the second case is $W > 0$. If $W > 0$ we must have $E = B$, because waiting requests imply that all execution slots are filled. If $W = 0$, then $E \leq B$. We will start by calculating the equilibrium in case of $W = 0$, because that is the easier case.

2.2.2 Empty waiting area

If $W = 0$ then $W_i = 0$ for all i, so $m_i = C_i + Q_i$ and the core-i-requests from core-i are directly send to the execution area. We can use this together with Equation 2 to obtain

$$\lambda(m_i - E_i) = E_i\mu \qquad \Longrightarrow \qquad E_i = \frac{\lambda m_i}{\mu + \lambda},$$

and we obtain

$$C_i = m_i - \frac{\lambda m_i}{\mu + \lambda} = \frac{\mu m_i}{\mu + \lambda}.$$

Since $Q \leq B$, we must have

$$\frac{\lambda \sum_{i=1}^{n} m_i}{\mu + \lambda} = \frac{\lambda m}{\mu + \lambda} \leq B.$$

2.2.3 Non-empty waiting area

If $W > 0$, then $E = B$. We can obtain that the arrival and departure rate of core-i-requests must be equal in core i and in the waiting area:

$$\begin{aligned} \lambda C_i &= E_i \mu && \text{(equal departure and arrival rate in core } i\text{)} \\ \lambda C_i &= \frac{W_i B \mu}{W} && \text{(equal arrival and departure rate in waiting area).} \end{aligned}$$

Furthermore, we must have $C_i + W_i + E_i = m_i$. This set of equations can be solved to obtain the solution

$$\begin{aligned} C_i &= \frac{m_i B \mu}{\lambda m} \\ W_i &= m_i - \frac{m_i(\mu + \lambda)B}{\lambda m} \\ E_i &= \frac{m_i B}{m}. \end{aligned}$$

Since $W > 0$, we must have

$$1 - \frac{(\mu + \lambda)B}{\lambda m} > 0 \qquad \Longrightarrow \qquad \frac{\lambda m}{\mu + \lambda} > B.$$

This means the two cases can be summarised to obtain

$$\begin{aligned} C_i &= \begin{cases} \frac{\mu m_i}{\mu+\lambda} & \text{if } \frac{\lambda m}{\mu+\lambda} \leq B \\ \frac{m_i B\mu}{\lambda m} & \text{otherwise} \end{cases} \\ W_i &= \begin{cases} 0 & \text{if } \frac{\lambda m}{\mu+\lambda} \leq B \\ m_i - \frac{m_i(\mu+\lambda)B}{\lambda m} & \text{otherwise} \end{cases} \\ E_i &= \begin{cases} \frac{\lambda m_i}{\mu+\lambda} & \text{if } \frac{\lambda m}{\mu+\lambda} \leq B \\ \frac{m_i B}{m} & \text{otherwise} \end{cases}. \end{aligned} \tag{3}$$

2.2.4 Similarities between BS and PMQ model

Let $(C_i^{(BS)}, M_i^{(BS)})$ and $(C_i^{(PMQ)}, W_i^{(PMQ)}, E_i^{(PMQ)})$ be the equilibrium of core-i-requests in respectively the BS and PMQ model. These values can be found in Equations 1 and 3. For both the $\frac{\lambda m}{\mu+\lambda} \leq B$ and $\frac{\lambda m}{\mu+\lambda} > B$ case we have $C_i^{(BS)} = C_i^{(PMQ)}$ and $M_i^{(BS)} = E_i^{(PMQ)} + W_i^{(PMQ)}$. Since $E_i^{(PMQ)} + W_i^{(PMQ)}$ is the total amount of core-i-requests in the memory, it seems that the PMQ model is an extension of the BS model.

2.3 The Deterministic Memory Queue model

The Deterministic Memory Queue (DMQ) model is illustrated in Figure 3. This model is very similar to the PMQ model, but the difference is that there is a memory queue instead of a waiting area. The policy for the queue will be First In First Out (FIFO). In Figure 3 there are 3 red dots being served by the memory, so the core1-arrival rate is 3μ. Similarly, there is 1 green dot that is being served, so the core2-arrival rate is μ.

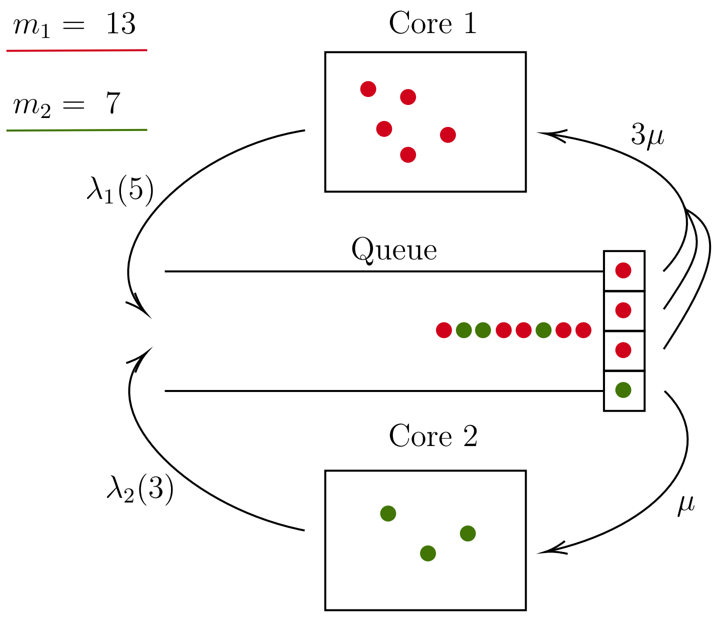

Figure 3: The overview of the Deterministic Memory Queue (DMQ) model where the red and green dots are respectively the core1- and core2-requests. We have $B = 4$.