Analysis of Shared Memory Multiprocessors*

Bart Sinclair

This work is produced by OpenStax-CNX and licensed under the Creative Commons Attribution License 1.0^{\dagger}

A shared-memory multiprocessor consists of a number of processors accessing one or more shared memory modules. The processors can be physically connected to the memory modules in a variety of ways, but logically every processor is connected to every module.

We can use any of several metrics to evaluate the performance of a multiprocessor system. In order to keep the model and its implementation manageable, we will focus on the degree of parallelism among the memory modules; that is, we are interested in determining the average number of memory modules that are being accessed simultaneously. The model is as follows:

- 1. The system has p processors and m memory modules. Each processor can send requests for access to any memory module.
- 2. Memory module accesses are synchronized; two modules servicing access requests at the same time start and complete their accesses together. Memory accesses always take one time unit (one memory cycle).
- 3. Processors are infinitely fast. When a processor's access request has been serviced by a memory module, the processor immediately generates a new request.
- 4. Processors send requests with equal probability to each module. The module chosen for a new request is independent of the module chosen for any other request.
- 5. A processor may have only one outstanding request at a time.
- 6. A memory module may service only one request at a time. If more than one request is queued at a memory module at the beginning of a memory cycle, the module selects a request to service at random. All requests not serviced during the cycle remain queued at the module at the beginning of the next cycle.

Let B be the average number of memory modules busy servicing an access request during a memory cycle. Regardless of the number of processors and memory modules, we can implement the model as a discrete-time Markov chain in which the information contained in a state includes the number of memory modules that have requests waiting at the beginning of a memory cycle. Hence, a state also describes the number of modules that will be busy during the next memory cycle. We can find \overline{B} by solving for the steady state solution of the Markov chain and computing

$$\bar{B} = \sum_{\text{all states i}} Pr [\text{state i}] \text{ number of busy modules in state i}$$

^{*}Version 2.3: Jun 9, 2005 10:10 am +0000

[†]http://creativecommons.org/licenses/by/1.0

We know that the Markov chain has a steady state solution (is ergodic) since the chain will be finite and at least one state will have a self-loop. The chain is finite because the number of ways that processors' access requests can be distributed among the modules is finite. Any state that represents at least one request at each memory module will have a self-loop, since any processor that has its request satisfied during a cycle beginning in that state has a non-zero probability of sending its next access request to the same module.

As usual, we proceed in three steps. We first determine, for a given p and m, the set of all states in the Markov chain. Then, we compute the probabilities for all allowed single step state transitions. Finally, we solve the Markov chain for the steady state probabilities.

It is not possible in general to implement the model using states that only tells us how many modules are going to be busy during the next cycle. Consider, for example, a model for a system with 4 processors and 2 memory modules. A state that only tells us that two modules will be busy during the next cycle does not include enough information to tell us the probabilities of being in the various states at the beginning of the next cycle. This is because two modules will be busy during a cycle that starts with requests distributed among the modules in either of two ways:

- 1. three requests at one module, one request at the other module
- 2. two requests at one module, two requests at the other module

If we start a cycle with the four requests distributed as in (1), at the end of the cycle we have two requests left at one module, no requests at the other module, and two processors ready to make new requests. The next cycle may start with the four requests distributed as in (1) or (2), or with four requests at a single module. If we start a cycle as in (2), both modules have a request waiting at the end of the cycle. The next cycle cannot begin with all four requests at a single module.

We must use a definition of state that allows us to determine how the access requests that must wait during a cycle are distributed among the modules at the end of the cycle, in addition to telling us how many modules will be busy during the cycle. (1) and (2) above illustrate exactly how this is done. Each state specifies how the p requests are distributed among the modules, without regard to which module is which. It is unimportant in (1) to know which module has three requests and which has two, since the modules and processors are identical.

With this general definition of state, computing the state transition probabilities can be divided into two parts. The first is simple: if $r \leq p$ requests are serviced during a memory cycle, the r processors that generate new requests for the next memory cycle choose any particular set of modules with probability $\left(\frac{1}{m}\right)^r$. They may all choose to send their new requests to the same module, or each may select a different module from the others, or some may choose the same module while others select different modules. The end result will be some distribution of p requests among the m modules at the beginning of the next cycle, with each possible distribution represented by a different state.

The second part of computing state transition probabilities is to count the number of ways in which processors may choose to make new requests that will result in the same distribution of requests at the beginning of the next cycle (the same next state). Consider the following example with 4 processors and 2 memory modules, as above. Suppose the model begins one cycle with two requests at each module. At the end of the cycle, each module has one request remaining. Two processors will make new requests before the start of the next cycle. If they select the same module, the next state has three requests at one module and one request at the other. There are two ways that this can happen, since there are two modules that they can select and each of these modules has one request remaining from the previous cycle.

To put the two parts together, we compute the single step state transition probability for the transition from state A to state B by multiplying the probability of choosing any set of modules following the cycle beginning in state A by the number of ways we can choose a set of modules that takes us to state B. For the example, the probability of making the transition from the state with two requests at each module to the state with three requests at one module and one at the other is $(\frac{1}{2})^2 2 = 1/2$.

Generally, the process of computing state transition probabilities is more difficult that this example would indicate, even though the process just described always works. The difficulty will always come in the second part - counting the number of ways of choosing where new requests go that take the model to the same next state. For this reason, we recommend that you follow the above procedure to compute **all** single step transition probabilities for each state. Because the single step transition probabilities for a given state (including a transition back to the same state) must sum to 1, this will provide a useful check on whether or not you counted all the possibilities.

We illustrate the complete procedure with several examples. Let C(n,r) be the number of combinations of *n* objects taken *r* at a time, and P(n,r) be the number of permutations of *n* objects taken *r* at a time. In the first two examples, we find an expression for the average memory module concurrency as a function of the number of memory modules for a fixed number (2 or 3) of processors.

Example 1: 2 processors, $m \ge 2$ modules

| S | tate | |
|---|------|----------------------------------|
| 1 | | 2 requests at the same module |
| 2 | | 1 request at each of two modules |

Table 1

First, find the single step transition probabilities. At the risk of overkill for this particularly simple case, we explicitly write each probability as the product of the probability that the new requests go to a particular set of modules ($\frac{1}{m}$ if the cycle started in state 1 and $\frac{1}{m^2}$ if the cycle started in state 2) and the number of such sets that result in the specified next state.

$$p_{1,1} = \frac{1}{m} = \frac{1}{m}$$

$$p_{1,2} = \frac{1}{m} (m-1) = \frac{m-1}{m}$$

$$p_{2,1} = \frac{1}{m^2} C(m,1) = \frac{1}{m}$$

$$p_{2,2} = \frac{1}{m^2} C(m,2) = \frac{m-1}{m}$$

This may actually seem to be double overkill. Since we know that the P matrix is singular, we only need one of the Chapman-Kolmogorov equations, and hence only two of the single step transition probabilities - either $p_{1,1}$ and $p_{2,1}$ or $p_{1,2}$ and $p_{2,2}$. The other independent equation is the normalization equation. However, as mentioned above, if you compute all of the single-step state transition probabilities, you can add the probabilities for all transitions from each state as a check on having gotten them correct.

Continuing with the example, choose the first Chapman-Kolmogorov equation. Then

$$\pi_1 = \pi_1 p_{1,1} + \pi_2 p_{2,1} = \pi_1 \frac{1}{m} + \pi_2 \frac{1}{m}$$

$$\pi_2 = (m-1)\,\pi$$

$$(\pi_1 + \pi_2 = 1 = m\pi_1) \Rightarrow \left(\pi_1 = \frac{1}{m}\right) \land \left(\pi_2 = \frac{m-1}{m}\right)$$

The average memory module concurrency or parallelism is

$$\bar{B} = 1 \times \frac{1}{m} + 2\frac{m-1}{m} = \frac{2m-1}{m}$$

Example 2: 3 processors, m≥3 modules

| State | |
|-------|---|
| 1 | 3 requests at one module |
| 2 | 2 requests at one module, 1 request at another module |
| 3 | 1 request at each of three modules |

Table 2

$$p_{1,1} = \frac{1}{m} 1 = \frac{1}{m}$$
$$p_{1,2} = \frac{1}{m} C(m-1,1) = \frac{1}{m} (m-1)$$

 $p_{1,3} = 0$

$$p_{2,1} = \left(\frac{1}{m}\right)^2 1 = \left(\frac{1}{m}\right)^2$$

$$p_{2,2} = \left(\frac{1}{m}\right)^2 (C(m-1,1) + C(m-1,1)C(2,1)) = \left(\frac{1}{m}\right)^2 3(m-1)$$

$$p_{2,3} = \left(\frac{1}{m}\right)^2 C(m-1,2)P(2,2) = \left(\frac{1}{m}\right)^2 (m-1)(m-2)$$

$$p_{3,1} = \left(\frac{1}{m}\right)^3 C(m,1) = \left(\frac{1}{m}\right)^3 m$$

$$p_{3,2} = \left(\frac{1}{m}\right)^3 C(m,2)C(3,2)P(2,2) = \left(\frac{1}{m}\right)^3 3m(m-1)$$

$$p_{3,3} = \left(\frac{1}{m}\right)^3 C(m,3)P(3,3) = \left(\frac{1}{m}\right)^3 m(m-1)(m-2)$$

 $p_{2,2}$ merits an explanation. Two memory modules are busy during a cycle that starts in state 2. At the end of the cycle, the two processors that had their memory access requests serviced will make new requests. The first term inside the square brackets corresponds to the two new requests going to the same memory module, one of the m-1 modules without a request at the end of the cycle. There are C(m-1,1) ways to select this module. The second term corresponds to one of the new requests going to the module that still has a request pending, and the other new request going to one of the other m-1 modules. There are C(m-1,1) ways to choose the module without a pending request, and there are two ways to order the two new requests so that one goes to the module that already has a request and the other goes to the module that doesn't.

We will make use of the Chapman-Kolmogorov equations for π_1 and π_3 , plus the normalization equation.

$$\pi_1 = \pi_1 \frac{1}{m} + \pi_2 \frac{1}{m^2} + \pi_3 \frac{1}{m^2}$$

OpenStax-CNX module: m10846

$$\pi_3 = \pi_2 \frac{(m-1)(m-2)}{m^2} + \pi_3 \frac{(m-1)(m-2)}{m^2}$$
$$1 = \pi_1 + \pi_2 + \pi_3$$

Solving these three equations gives

$$\pi_1 = \frac{1}{m^2 - m + 1}$$

$$\pi_2 = \frac{(3m - 2)(m - 1)}{m(m^2 - m + 1)}$$

$$\pi_3 = \frac{(m - 1)^2(m - 2)}{m(m^2 - m + 1)}$$

$$\bar{B} = 1 \times \frac{1}{m^2 - m + 1} + 2\frac{(3m - 2)(m - 1)}{m(m^2 - m + 1)} + 3\frac{(m - 1)^2(m - 2)}{m(m^2 - m + 1)} = \frac{3m^3 - 6m^2 + 6m - 2}{m(m^2 - m + 1)}$$

Example 3: 4 processors, 4 modules

| State | |
|---------------|--|
| 1 | all four requests are at one module |
| 2a | three requests are at one module and one request is at another module |
| $2\mathrm{b}$ | two requests are at one module and two requests are at another module |
| 3 | two requests are at one module and one request is at each of two other modules |
| 4 | one request is at each of four modules |

Table 3

We've chosen the state names to make explicit the number of memory modules busy in each state. The state transition probabilities are

$$p_{1,1} = \frac{1}{4}$$

$$p_{1,2a} = \frac{1}{4}C(3,1) = \frac{3}{4}$$

$$p_{1,2b} = p_{1,3} = p_{1,4} = 0$$

$$p_{2a,1} = \left(\frac{1}{4}\right)^2$$

$$p_{2a,2a} = \left(\frac{1}{4}\right)^2 C(3,1) P(2,2) = \frac{6}{16}$$

$$p_{2a,2b} = \left(\frac{1}{4}\right)^2 C(3,1) = \frac{3}{16}$$

$$p_{2a,3} = \left(\frac{1}{4}\right)^2 C(3,2) P(2,2) = \frac{6}{16}$$

$$p_{2a,4} = 0$$

$$p_{2b,1} = 0$$

$$p_{2b,2a} = \left(\frac{1}{4}\right)^2 C(2,1) = \frac{2}{16}$$

$$p_{2b,2b} = \left(\frac{1}{4}\right)^2 P(2,2) = \frac{2}{16}$$

$$p_{2b,3} = \left(\frac{1}{4}\right)^2 C(2,1) + \left(\frac{1}{4}\right)^2 C(2,1) C(2,1) P(2,2) = \frac{10}{16}$$

$$p_{2b,4} = \left(\frac{1}{4}\right)^2 P(2,2) = \frac{2}{16}$$

$$p_{3,1} = \left(\frac{1}{4}\right)^3 C(3,1) + \left(\frac{1}{4}\right)^3 C(3,1) C(3,2) = \frac{12}{64}$$

$$p_{3,2b} = \left(\frac{1}{4}\right)^3 C(3,1) C(3,2) = \frac{9}{64}$$

$$p_{3,3} = \left(\frac{1}{4}\right)^3 C(3,2) P(3,3) + \left(\frac{1}{4}\right)^3 C(3,2) C(3,2) P(2,2) = \frac{36}{64}$$

$$p_{4,1} = \left(\frac{1}{4}\right)^4 C(4,2) C(4,3) P(2,2) = \frac{48}{256}$$

$$p_{4,2b} = \left(\frac{1}{4}\right)^4 C(4,2) C(4,2) P(2,2) = \frac{124}{256}$$

$$p_{4,3} = \left(\frac{1}{4}\right)^4 C(4,3) C(3,1) C(4,2) P(2,2) = \frac{144}{256}$$

$$p_{4,4} = \left(\frac{1}{4}\right)^4 P(4,4) = \frac{24}{256}$$

Combining all of these probabilities into the single-step transition probability matrix:

$$P = \left(\begin{array}{ccccccc} 1/4 & 3/4 & 0 & 0 & 0 \\ 1/16 & 6/16 & 3/16 & 6/16 & 0 \\ 0 & 2/16 & 2/16 & 10/16 & 2/16 \\ 1/64 & 12/64 & 9/64 & 36/64 & 6/64 \\ 4/256 & 48/256 & 36/256 & 144/256 & 24/256 \end{array}\right)$$

Solve $\pi P = \pi$ plus the normalization equation.

The solution is

$$\begin{pmatrix} \pi_1 \\ \pi_{2a} \\ \pi_{2b} \\ \pi_3 \\ \pi_4 \end{pmatrix} = \begin{pmatrix} 0.0323 \\ 0.2419 \\ 0.1452 \\ 0.5081 \\ 0.0726 \end{pmatrix}$$

.

The average memory module concurrency is

$$B = \pi_1 1 + (\pi_{2a} + \pi_{2b}) 2 + \pi_3 3 + \pi_4 4 = 2.2610$$

The three examples allow us to compare the average memory module concurrency with four modules for two, three, and four processors:

| р | \bar{B} |
|---|-----------|
| 2 | 1.750 |
| 3 | 2.269 |
| 4 | 2.261 |
| | |

Table 4

In going from 2 to 3 processors, we see about a 30% increase in memory module concurrency. When we go from 3 processors to 4, the performance improves only about 15.5%.