UNDERSTANDING THE MP EFFECT: MULTIPROCESSING IN PICTURES

Neil J. Gunther

Computer Dynamics Consulting, Mountain View, CA 94040, U.S.A.
codynamo@aol.com

Multiprocessors, whether mainframes or open systems, do not deliver linear-scaling capacity. The real capacity curve is sublinear due to some amount of processing capacity being usurped by the system in order to orchestrate the interaction between physical processors and computational resources. This computational overhead is sometimes referred to as the MP Effect (MPE) and its magnitude is determined by both platform architecture and workload characteristics. Modeling the MPE is important for predicting multiprocessor capacity during procurement and upgrade periods. Detailed models (e.g., a simulation or queueing network model) are difficult and time-consuming to construct and verify. A simpler approach is to use an equational model (e.g, Amdahl's law) and fit the data to this equation. The problem is, there is more than one such equational model from which to choose. Which one is "correct?" We review three capacity models that have been used by various authors at CMG and elsewhere. These models are based on Amdahl, Geometric, and Quadratic scaling, respectively. We present a totally new perspective on this kind of modeling by introducing a pictorial representation of the underlying dynamics expressed by each capacity model. With the dynamics revealed, the capacity planner is in a better position to select the appropriate model by matching it to the platform and workload rather than relying on naive curve fitting. Examples, showing how to apply this new insight, will be presented in the session.

INTRODUCTION†

Computational overhead refers to the fraction of processor cycles usurped by system work in order to execute the user workload. This loss in processing capacity diminishes the potential economies of scale offered by multiprocessor (MP) computers.

Typical sources of MP overhead include:

- O/S code paths (e.g., syscalls in Unix; supervisor calls in MVS)
- Exchanging shared-writeable data between processor caches
- Data exchange between processors and main memory
- Spin-lock synchronization (i.e., serialization) of accesses to shared data
- Waiting for an I/O to complete a DMA (direct memory access)

MP overhead can be regarded as a type of ***interaction*** between the processors as they contend for shared subsystem resources. As more processors are added to the complex, to do more work, the overhead typically increases because the degree of interaction increases. This so-called ***MP Effect*** (MPE) occurs in both proprietary mainframes and open system multiprocessors. Without the MPE, the aggregate processor capacity would scale linearly. The MPE, however, is responsible for the incremental throughput falling below the linear ideal.

MP Interactions

From simple observation we know that MP capacity declines in a ***nonlinear*** fashion such that any attempt to model the MPE requires a nonlinear ***function***; not a single number. By using an appropriate (nonlinear) function, we can formulate simple models which offer significant assistance in setting realistic expectations for MP capacity in the presence of MPE. The practical application of such MP capacity models has been

discussed by various authors at CMG and ICCM. Some notable examples include:

- Geometric scaling in proprietary mainframe MPs [Artis 1991]
- Super-serial scaling for open system MPs [Gunther 1993]
- Amdahl's law and the effective MPL for parallel sysplex [Kraus 1995]
- Regression techniques for MP acquisition [McGalliard 1995]
- Quadratic approximation for open system MPs [Gunther 1995]

The great attraction of such models is that they are much simpler to construct than a more detailed simulation or queueing network model. The detraction is the existence of more than one equational model from which to choose. In view of all these choices, the capacity planner can justifiably ask for the "correct" scaling law to please stand up!

One approach to making the "correct" choice is to see which model best fits the data. Unfortunately, bets-fit is not always the best criterion because it assumes that the data is accurate and that the MP has been tuned for optimal performance at each configuration. To be certain the data are optimal, you'd like to compare the measurements to a model. This is a circular argument.

We address this vexing circularity by revealing the underlying dynamics of three MP capacity models:

- Amdahl scaling capacity denoted by A(p)
- Geometric scaling denoted by G(p)
- Quadratic scaling denoted by Q(p)

where p is the number of physical processors. Once we know the dynamics of processor interaction in each case, we should be in a better position to select the appropriate capacity model based on our knowledge of the workload. As far as this author is aware, making the connection between these simple capacity formulae and their underlying dynamics has never been done before.

ONE PARAMETER MODELS

The three capacity models we discuss are important because they are in common usage by practitioners and they also fall into the simplest class of capacity functions that are characterized by a ***single*** parameter. First, we review the three models: A(p), G(p), and Q(p), in their conventional, equational form.

1. Amdahl Scaling: The Amdahl capacity model is defined as:

$$A(p) = \frac{p}{1 + \sigma (p - 1)} \qquad (1)$$

Where p is the number of processors in the CPU complex. The parameter, $0 < \sigma < 1$, known as the ***seriality*** constant, refers to the serial fraction of the workload that cannot be made to execute in parallel [Amdahl 1967; Hennessy and Patterson 1990]. The asymptotic capacity is $A(\infty) \approx 1/\sigma$. (See Note 1 at the end of this paper)

Example: If the amount of serial work is 3.33% of the workload, then the effective capacity of a 10-way MP is predicted to be:

$$A(p) = \frac{10}{1 + 9 * 0.0333} = 7.69$$

In other words, 2.31 CPU equivalents of capacity is lost to MPE.

2. Geometric Scaling: The Geometric capacity model G(p) can be written as:

$$G(p) = 1 + \Phi + \Phi^2 + \Phi^3 + \ldots + \Phi^{p-1} \qquad (2)$$

where the parameter, $0 < \Phi < 1$, is known as the ***mp factor*** and refers to the fraction of uniprocessor capacity available at each CPU increment. The asymptotic capacity is: $G(\infty) \approx 1/(1-\Phi)$.

Example: If the mp-factor = (100 − 3.33)%, then a 10-way will have a predicted capacity of:

$$G(p) = 1 + 0.967 + 0.967^2 + 0.967^3 + \ldots + 0.967^9 = 8.71$$

or 1.29 CPU equivalents is consumed by computational overhead. Applications of this scaling modelcan be found in [Artis 1991] and [McGalliard 1995].

2. Quadratic Scaling: The capacity model is:

$$Q(p) = p - g\, p\, (p - 1) \qquad (3)$$

where the parameter, $0 < g < 1$, refers to the ***overhead factor*** contributed by each CPU in the complex. See [Gunther 1993; 1995] for details.

Example: If the g-factor = 3.33%, then a 10-way has a predicted capacity of:

$$Q(p) = 10 - 0.0333 * 10 * 9 = 7.00$$

or 3 CPUs are consumed by computational overhead. Unlike Amdahl and Geometric scaling, Quadratic scaling does not possess an asymptote. Rather, it predicts a capacity maximum at $p_{max} = (1 + g) / 2g$. See Figure 1.

Figure 1 shows a complete set of capacity projections for up to 20 processors using the same parameter value, 3.33%, for each model. Typically, when fitting these capacity models to real data, the parameter values would be different for each model. For a 10-way the spread in prediction is almost 2 CPUs worth of capacity, while a 20-way has a spread of about 8 CPUs worth of capacity. What is a capacity planner to do?

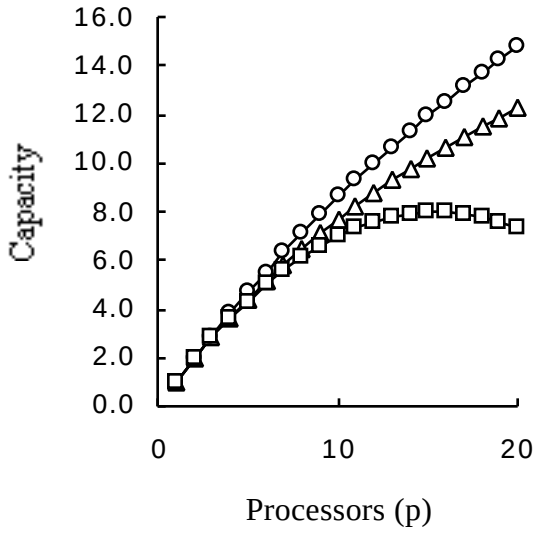

Figure 1. Capacity models: Amdahl (Δ), Geometric (o), and Quadratic (❑), with parameter values equal to 3.33%.

It should be noted that MP scaling models differ from other capacity evaluation methods such as those based on ITR [Fitch 1992] or Relative I/O Content [Major 1986]. There, the relative processor capacity is determined by direct measurement of a specific MP configuration. Scaling models, on the other hand, attempt to predict the relationship between configurations, thus enabling extrapolation (or interpolation) beyond existing capacity measurements. Such capacity projections are useful when assessing benchmark data during a procurement cycle or when upgrading the same MP configuration with faster processors.

Level Crossing

For later comparisons, we set the MP overhead to be equal when the second CPU is added in each of the three capacity models (see Note 2). To ease the arithmetic calculations, we assume that 1/4 of MP capacity is lost when the second CPU is added i.e., A(2) = G(2) = Q(2) = 1.75 CPU equivalents are available capacity. It follows that the values of the respective model parameters must be different to meet this constraint. They are summarized in Table 1.

The corresponding capacity ratios are summarized in Table 2. Note that a g-factor of 1/8 corresponds to a 12.5% overhead; an extremely high value. Hence, the Q(p) capacity has fallen to zero with p = 9 processors in the MP complex. Remember, we are just using these values for the purposes of demonstration.

Table 1. Equal Overhead Constraint

Model	Capacity	Parameter
Amdahl	A(2) = 1.75	σ = 1/7
Geometric	G(2) = 1.75	Φ = 3/4
Quadratic	Q(2) = 1.75	g = 1/8

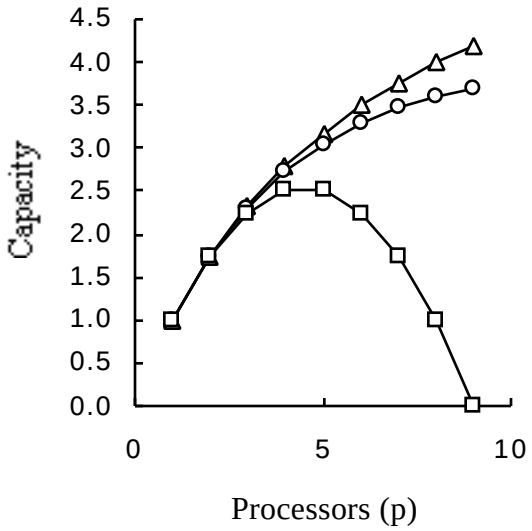

Figure 2. Scaling with Table 1 parameters. Capacity ratios are in Table 2. Legend as in Fig. 1.

The capacity curves corresponding to Table 1 parameter values appear in Figure 2. The ordering of the curves is different with Amdahl scaling being more efficient than Geometric scaling, which is more efficient than Quadratic scaling.

The actual capacity values corresponding to Figure 1 are provided in Table 2. We shall be referring back to these numbers when we make use of the pictorial representation in the next section. It is important to keep in mind that the numbers in Table 2 were calculated using the scaling models as formulated in equations 1 through 3.

Table 2. Capacity Projections

CPUs	A(p)	G(p)	Q(p)
1	1.000	1.000	1.000
2	1.750	1.750	1.750
3	2.333	2.313	2.250
4	2.800	2.734	2.500
5	3.182	3.051	2.500
6	3.500	3.288	2.250
7	3.769	3.466	1.750
8	4.000	3.600	1.000
9	4.200	3.700	0.000

Our objective is to understand why each of these models gives such different capacity projections. Having established a common basis by selecting modeling parameters that result in equal 2-way overhead, we are now in a position to reveal the underlying dynamics of each capacity model.

PROCESSING IN PICTURES

We are going to introduce a pictorial representation of multiprocessor overhead that will be unfamiliar to everyone, since it has not been done before. (see Note 3) The simplest model to understand using these pictures is the Quadratic capacity model.

Quadratic Pictures

First, consider a dual processor (p = 2) shown in Figure 3 where processors are shaded circles and a message exchange is an arrow.

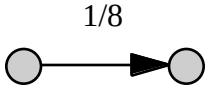

Figure 3. One contribution to MPE in a dual processor system.

The cost of sending one message in the quadratic model is 1/8th of a CPU's capacity. This follows immediately from the fact that the g-factor has the value 1/8 in Table 1. By definition, the g-factor is a direct measure of the overhead per CPU.

At some point, the other CPU may send a reply message to the first processor so that the computation can be completed. The response message is depicted in Figure 4. Since the system is symmetric (the 2 CPUs are indistinguishable), the cost of sending the reply in the other direction is also 1/8th of a CPU.

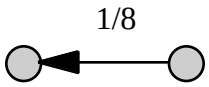

Figure 4. Another contribution to the overhead in a dual processor system. The cost of replying is the same as the initial request since the entire system is symmetric.

The total cost (in CPU cycles) for a request-reply pair is therefore, 2/8 = 1/4 of a CPU per bi-directional message (Figure 5).

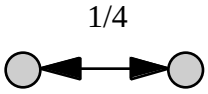

Figure 5. The total contribution to the overhead in a dual processor system is the simply the sum of the request and reply messages.

Let's check this conclusion. From equation (3) we know that the second term refers to the amount of capacity lost to overhead. To determine the remaining computational capacity, we simply subtract off the overhead from the ideal capacity (the first term). In other words,

$$Q(2) = 2 - 2 * \frac{1}{8} = 1\frac{3}{4} \qquad (4)$$

which is in agreement with Q(2) in Table 1.

That was easy. Maybe we just got lucky? Let's check the next level; a 3-way multiprocessor. The corresponding diagrams are shown in Figure 6. Each message still costs 1/8th of a CPU to send

and there are 6 possibilities: 3 requests and 3 replies.

There are 6 messages (arrows), each costing 1/8. The overhead is therefore 6/8. If we now subtract that loss from the physical capacity (3 CPUs), the result should be the available capacity predicted by Q(3).

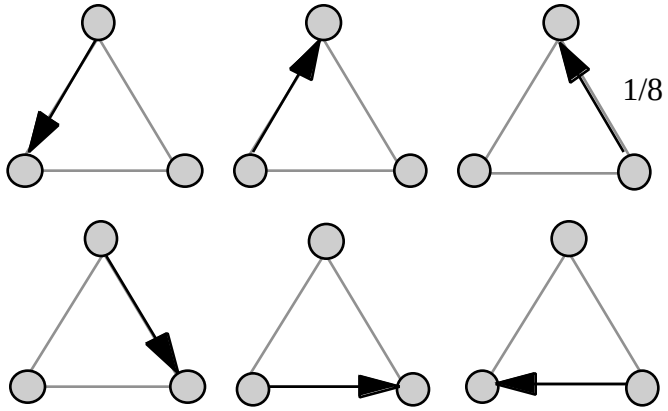

Figure 6. Overhead contributions in a 3-way MP.

Let's do the arithmetic.

$$Q(3) = 3 - 6 * \frac{1}{8} = 2\frac{1}{4} \qquad (5)$$

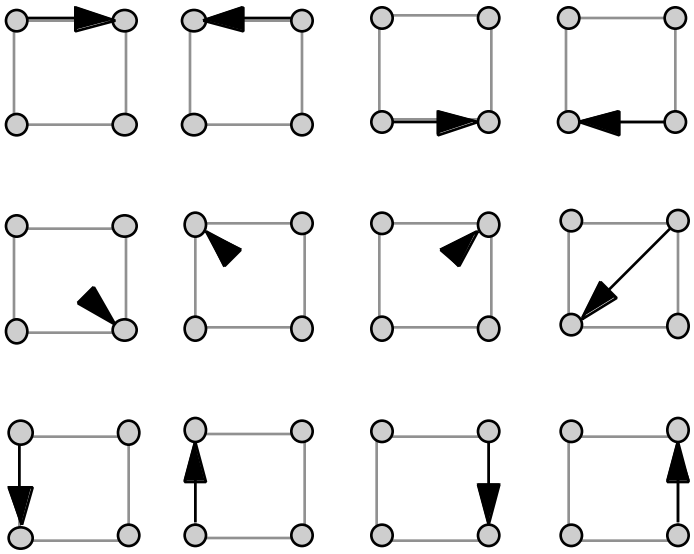

Figure 8. Overhead in a 4-way multiprocessor.

Alternatively, we could calculate the capacity by noting that there are 3 bi-directional messages, each costing 1/4 of a CPU, as shown in Figure 7. But 3/4 is the same as 6/8 which must produce the same arithmetic result as equation (5).

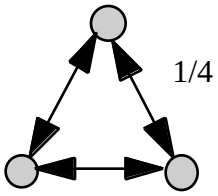

Figure 7. Total overhead in a 3-way MP.

Once again, we have agreement with the Q(3) value in Table 1. This is getting infectious! Let's try a 4-way multiprocessor. The 12 messaging possibilities are shown in Figure 8.

All these cases can be summarized more efficiently using bi-directional arrows as shown in Figure 9.

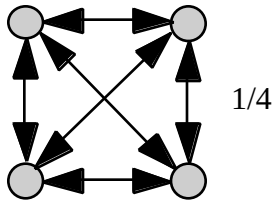

Figure 9. Total overhead in a 4-way MP.

Repeating the arithmetic step for the 4-way MP we have 6 bi-directional messages, each costing 1/4. Subtracting off this overhead produces:

$$Q(4) = 4 - 6 * \frac{1}{4} = \frac{8}{2} - \frac{3}{2} = 2\frac{1}{2} \qquad (6)$$

Once again, we have agreement with Q(4) in Table 1. Going any further, we become overwhelmed by an explosion of diagrams. For example, a 10-way MP has nearly 2 million pictures!

But we are not proposing these pictures as a calculational method. Instead, we are going

to use them to interpret the dynamics of multiprocessor overhead in each of the scaling models defined by equations (1)-(3). Can we say anything, so far, about the dynamics of Quadratic scaling? Yes we can.

Quadratic scaling is a model in which each processor communicates with the other processors, one at a time. That's why there are 12 message arrows in Figure 8. More formally, this is known as a ***pair-wise exchange*** or ***point-to-point*** protocol. Each time a processor sends a message it cost 1/8 of a CPU's capacity because the message has to be set up individually every time.

Moreover, as more processors are added to the system, the cost of communication increases dramatically. Any processor has to reach (p-1) other processors. Hence, the overhead grows quadratically; as reflected in the second term of equation (3). Eventually, the total cost becomes such a burden, the capacity actually becomes reduced (Figures 1 and 2) as the processor interaction soaks up more and more CPU capacity.

As you might guess, this is not the most efficient communication protocol but it does occur in real shared-memory multiprocessors. One CPU will have data needed by another CPU. The first puts a data request on the bus and the other CPU responds with data from its cache. This happens over and over again, between different CPU pairs, during the course of executing work -- particularly in the case of commercial workloads [Gunther 1993].

Amdahl Pictures

We could design a more efficient messaging protocol in the following way. Instead of setting up and sending a message to other processors, one at a time, suppose a processor sends its message to every other processor, ***simultaneously***.

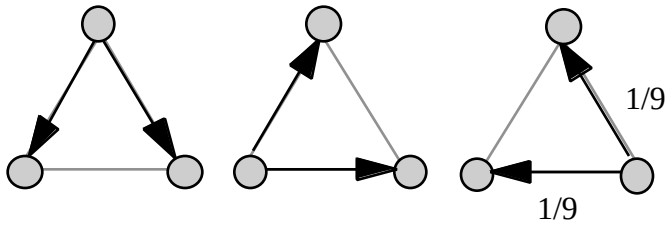

Figure 10. Broadcast overhead in a 3-way MP.

That way the cost of setting up the message is incurred just once. How does this modified protocol look in terms of our processing pictures?

The 2-way diagrams are identical to those in Figures 3 and 4. This has to be true since we demanded the overhead be the same for all three models at p = 2. The interesting difference occurs in the 3-way case.

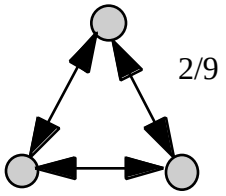

Figure 11. Aggregate broadcast in a 3-way MP.

In the pair-wise protocol (Figure 6) we had 6 diagrams (3 requests and 3 replies). With the modified protocol, we only have 3 diagrams (Figure 10) because one processor sends two messages simultaneously. In addition, the cost of sending the message is less since two messages are sent for the price of one. This is reflected in the 3-way message cost being reduced from 1/8 to 1/9.

By the way, there is not enough space here to go into an explanation of where the cost number 1/9 comes from in this modified protocol. I have to ask you to take it on faith. You will see, however, that things do work out correctly. If you are still overcome by an irrepressible urge to understand the numerical details, Refer to the handout material for a deeper explanation. The aggregate cost is shown Figure 11. Each bi-directional arrow costs 2/9ths of a CPU.

Now, we check the arithmetic using the same procedure we used for the Quadratic model in equation (5). Subtracting off the overhead, we find:

$$3 - 3 * \frac{2}{9} = \frac{9}{3} - \frac{2}{3} = \frac{7}{3} = 2\frac{1}{3} \qquad (7)$$

which is exactly the result we obtained for Amdahl scaling, A(3) = 2.333, in Table 1. Note, we did not use the usual form of Amdahl's law, equation (1), at all. This is quite remarkable!

Next, we check the 4-way overhead. There are only 4 diagrams (Figure 12) and the cost is further reduced from 1/9th to 1/10th of a CPU.

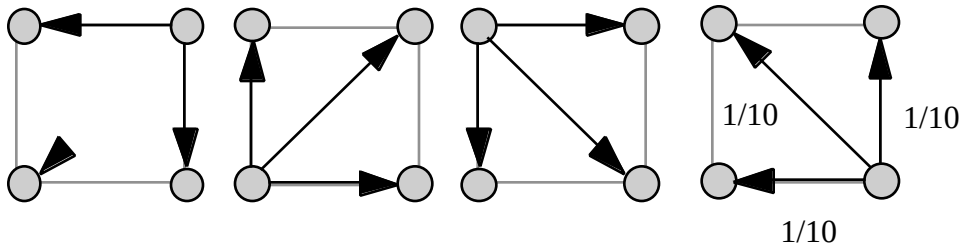

Figure 12. Broadcast overhead in a 4-way MP.

The aggregate diagram is the same as the one in Figure 9 but with each bi-directional arrow costing 3/10ths of a CPU instead of 1/8th. Doing the arithmetic, we find:

$$4 - 4 * \frac{3}{10} = \frac{20}{5} - \frac{6}{5} = \frac{14}{5} = 2.80 \qquad (8)$$

which is precisely the 4-way result A(4) for Amdahl scaling in Table 1.

So, now we know the hidden dynamics in Amdahl's law: it models a ***broadcast*** protocol. It is more efficient than Quadratic scaling because the cost of setup is a one-shot per message cycle. On the other hand, as the system is scaled up, there are more CPUs to broadcast to, hence the overhead grows more slowly than for the pair-wise protocol that underlies Quadratic scaling. Hence, the capacity growth under a broadcast protocol reaches an asymptote just like Amdahl's law (See the remarks following equation 1).

Although this broadcast protocol is clearly more efficient than pairwise exchange, there is a problem in applying it. Most commercial workloads do not induce this kind of processor interaction very often. Although CPU broadcasts do occur, they occur relatively infrequently in the execution of commercial workloads. In this sense, modeling MP capacity with a purely broadcast protocol is likely to be unrealistic in most cases.

Some exceptions might include scientific or financial workloads. For certain numerical workloads, a high degree of parallelism is possible e.g., calculating a matrix inverse where each matrix element or sub-block is assigned to a processor. After each block is calculated, the other processors need to get each other's results before the next phase of computation can take place. Broadcasting the numerical updates is clearly the most efficient performance solution.

Geometric Pictures

Finally, we apply our processing pictures to reveal the dynamics of Geometric scaling as defined by equation (2). This is the least intuitive protocol to understand because it involves some complicated mathematics. Once again, I'm just going to state the results for you. Further justification can be found in the handouts provided during this session and in [Gunther 1997]. As usual, let's start with the 2-way protocol (Figure 13).

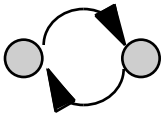

Figure 13. Nearest-neighbor overhead in a 2-way multiprocessor.

Geometric scaling corresponds to a ***nearest-neighbor*** protocol. For the 2-way case, as you you might have already anticipated, the cost per message is the same as for the other models but I draw the diagram in Figure 13 a little differently. The 3-way case is shown in Figure 14.

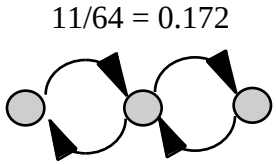

Figure 14. Nearest-neighbor overhead in a 3-way multiprocessor.

The tricky part is determining the cost per message (arrow). It turns out to be 11/64ths of a CPU. This is not at all obvious (see [Gunther 1997] for a complete discussion). Similarly, the cost a bi-directional message is 2 * 11/64 = 11/32nds of a CPU. Doing the arithmetic (there are 2 bi-directional arrows) we have:

$$3 - 2 * \frac{11}{32} = \frac{48}{16} - \frac{11}{16} = \frac{37}{16} = 2.31 \quad (9)$$

which is exactly the 3-way result for G(3) in Table 1. The cost per message in the 4-way system (Figure 15) is 81/384 (don't ask) and the bi-directional cost is shown in Figure 16.

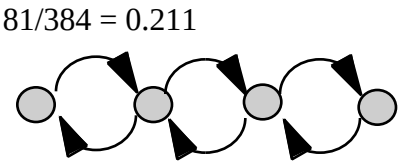

Figure 15. Nearest-neighbor overhead in a 4-way multiprocessor.

You see now why Figure 13 is drawn differently. The overhead in an N-way multiprocessor is represented by a linear chain of (N-1) bi-directional arrows. Performing the, by now, familiar arithmetic (there are 3 bi-directional arrows) gives:

$$4 - 3 * \frac{81}{192} = 2.73 \quad (10)$$

in agreement with G(4) in Table 1. Note also that we did not sum a series as would be required by the definition of Geometric scaling in equation (2).

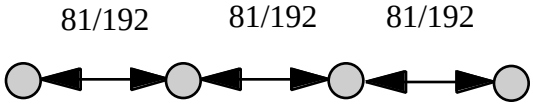

Figure 16. Bi-directional overhead in a 4-way Geometric scaling multiprocessor.

I call this the ***bucket brigade*** protocol. If a CPU wants to send a message to its immediate neighbor, it is the same cost as the pairwise protocol. But if that CPU needs to send a message to another CPU further down the chain, all the intermediate CPUs incur some cost for passing the original message along. Like Amdahl (broadcast) scaling, the capacity reaches an asymptote.

THE BIG PICTURE

In the preceding discussion, we uncovered the hidden dynamics that belong to some common models used to predict MP capacity in the presence of the MP effect. We were motivated to find an alternative to the usual approach, of just fitting the data to a model, because it involves a logical circularity that is often overlooked and can render misleading capacity predictions.

All three capacity models assume a homogeneous workload i.e., every processor does the same kind of work, and they assume there is an inherent cost for processor interaction. That cost is reflected in a single parameter in each capacity equation (Table 1). In summary, we are now able to draw the following conclusions about the applicability of these models:

Quadratic scaling: The MPE arises from a **PAIRWISE** exchange interaction. A CPU can only interact with the other CPUs, one at a time. As the system is scaled up, the cost per message remains fixed because the CPU needs to setup for each message to every other CPU. Compounding this, there are (p-1) paths to all the other CPUs. This growing aggregate cost gives rise to the possibility of retrograde capacity (See Figures 1 and 2).

Amdahl scaling: The MPE arises from a **BROADCAST** interaction. A CPU broadcasts to all other CPUs simultaneously. The suspension of computation during the broadcast cycle causes all CPUs to take longer to complete their work. As the system is scaled up, the cost per message decreases because it is only necessary to setup the message once per broadcast cycle. Contrary to this reduction in scaled message cost, each CPU is required to broadcast to more processors using (p−1) paths. These two opposing effects are responsible for the asymptotic capacity bound $A(\infty)$ in equation (1). Amdahl scaling is most appropriate for modeling workloads containing many similar threads, each of which must occasionally be updated before continuing. Numerical or financial workloads have this property.

Geometric scaling: The MPE arises from a **BUCKET BRIGADE** interaction. Each CPU can only interact with its nearest (inline) neighbor. As the system is scaled up, the overhead per message increases due to the amount of work done by each CPU in passing messages along to remote processors. Counterbalancing this increasing cost per message are the vastly fewer message paths available as CPUs are added. Looking at Figure 15, for example, you can see there are only 2 arrows per CPU. Like Amdahl scaling, these two competing effects are responsible for the

asymptotic capacity bound G(∞) n equation (2). Geometric scaling is probably most suited to architectures like binary hypercube MPPs. There, all communication occurs via hops between neighbors (vertices) in the respective cube.

Multiprocessing -- The Movie

Figure 17 shows a typical shared-memory MP architecture. It comprises a set of processors (aka CPUs or engines) each with their own private memory, also called a cache memory (CM) or High Speed Buffer (HSB), and all connected to each other and main store by a common bus. This bus also connects the I/O subsystem to the processors and main memory. This architecture is common to most mainframes and open systems multiprocessors. Certain variations, such as the use of multiple memory buses, are based on this theme [Gunther 1997].

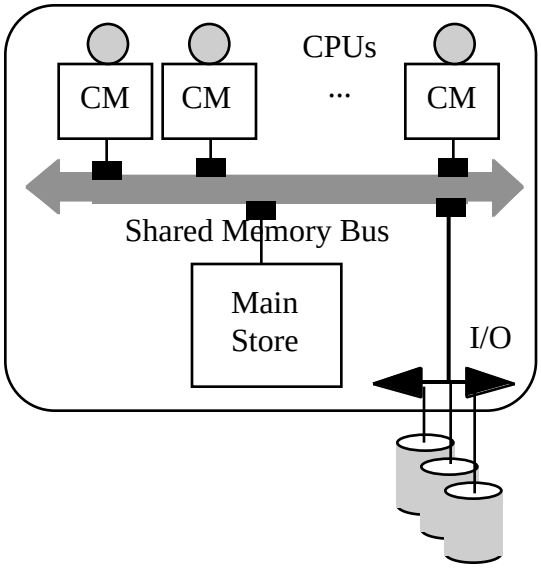

Figure 17. A typical multiprocessor architecture.

To better understand our pictorial analysis, we consider a much simplified schematic representation in Figure 18. In particular, we drop the I/O subsystem by assuming that all workloads are CPU-intensive and we reduce the bus to a simple line. The simplified MP schematic can be used to enumerate some of the most common processor interactions that occur in a real MP.

There are two basic operations that can be performed on an MP: read data or write data. The details of how these basic operations are implemented varies somewhat across vendor platforms. In this paper we are only interested in the gross features of MP interactions and they can be summarized as follows. First, consider a read operation.

A processor (CPU 1 in Fig. 18) needs new data, so it puts a read-request on the bus (usually costing 1 bus cycle). Since every device listens to all operations that go across the bus, the read acts like a ***broadcast*** request to all processors and main memory. Figure 18 depicts the case where main memory has the most recent copy of the requested data, so it responds by addressing the data to CPU 1, and placing it on the bus for delivery.

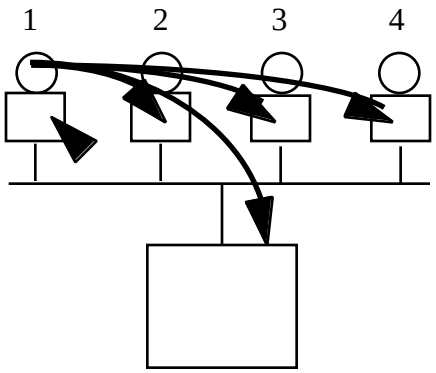

Figure 18. Main memory responds to the read-data request broadcast by processor 1.

An alternative scenario is shown in Figure 19. In this case CPU 3 had the most recent copy of the data (because it just updated that datum in its cache) and it responds instead of main memory (a pairwise exchange).

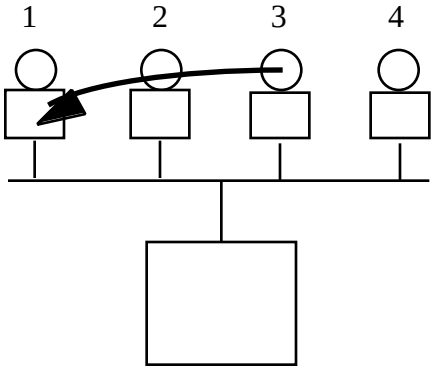

Figure 19. CPU 3 responding to CPU 1.

The other basic operation is a write to main memory. This operation is depicted in Figure 20. At some point, the processor caches need to be informed that their data is now stale with respect to main memory. This is accomplished with an invalidate operation being put on the bus. The invalidate may be broadcast simultaneously with the write (called write-through) or it may occur later (called write-back). This subtlety is a fine point for our discussion and we can ignore it here.

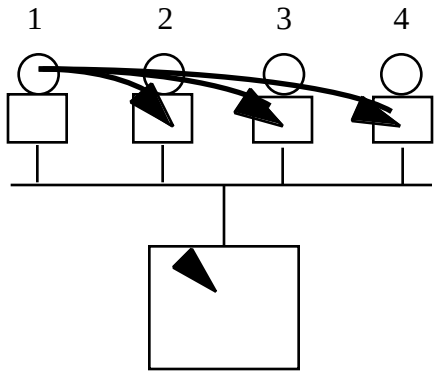

Figure 20. Write with cache invalidate broadcast.

Another gross interaction arises when new data is needed by the processor because that data is not resident in its cache (Figure 21). In that event, some resident cache blocks need to be replaced with blocks from main memory that contain the required data. A replacement request is made and main memory responds directly.

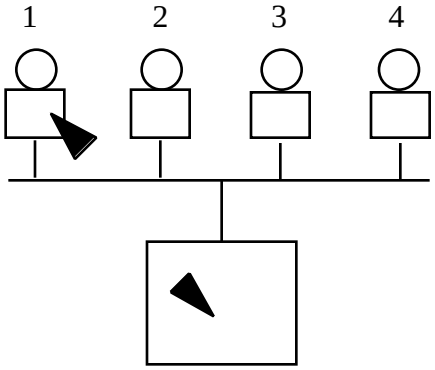

Figure 21. Cache line replacement in CPU 1.

Clearly, the frequency of cache block replacement will depend on (i) the locality of the data in each cache, and (ii) the size of each cache. How well the data fits in the cache is a function of the way the application is written and compiled (optimized) i.e., software can play a big role in the MPE.

The coding of software locks can also be significant in determining the magnitude of the MPE. Figure 19, for example, might also depict a lock being transferred from CPU 3 to CPU 1, but in the next few cycles CPU 3 might want to test if the lock is free yet, and that causes it to be transferred back to its cache. This "ping-ponging" of locks can cause serious MPE, and the more processors, the worse it can become.

Two Thumbs Up?

Now let's compare the "movie" with our pictures. We knew at the outset that the three equational models were simple-minded; that's what makes them easy to use. What has not been understood before, is the nature of the naiveté in each model. Our parallel pictures reveal it for the first time.

Relative to what goes on in real MPs, we can state the following differences:

- Bus-reads and bus-writes typically have very different cycle times (e.g., a factor of 2) in a real MP. In the capacity models, reads and writes are weighted equally.
- Main memory and bus access times are usually different in real MPs. In the capacity models, they are implicitly lumped into the cost of arrows.
- Some real MP interactions, such as cache-line replacement, are unaccounted for in the models.
- The dispatcher (MVS) or scheduler (Unix) will contribute to MPE depending on the setting of various O/S tunables such as, affinity and priority controls.

So the MPE is determined by both the workload instruction-stream and the details of the platform implementation e.g,. cache size and O/S scheduling. None of the three capacity models, however, can distinguish the hardware and software contributions to MPE. We'll return to this point in the next section.

In light of these discrepancies, it is astounding that such simple-minded equational models can be useful at all! The full explanation for this would take us too far afield, suffice to say, it hinges off relative time-scales (see Chapter 0 of [Gunther 1997]). The overall conclusions are:

- None of the actual MP interactions reflect the **nearest-neighbor** exchanges that underlie the Geometric model. In general, this model appears unrealistic.

- There are **broadcast** interactions in real MPs. These typically take about 1 bus cycle and perhaps a fraction of a cycle to set a 'dirty' flag on the datum in the cache. The Amdahl model should be valid when broadcasting (or serialization) dominates the workload e.g., long-running numerical (financial) calculations.
- **Pairwise** interactions occur with the highest frequency but typically between processors and main memory. The Quadratic model views all interactions as occurring between CPUs only. This apparent oversimplification may be a blessing rather than a cardinal sin.

From the dynamics revealed by our processing pictures, we see the Amdahl scaling model corresponds to the most efficient but least frequent interaction. Geometric scaling is almost as efficient as Amdahl scaling but appears quite unrealistic for MP architectures. Quadratic scaling is the least efficient but possibly the most realistic interaction model, since CPU-to-CPU and CPU-to-Memory interactions are a common occurrence in real MPs.

Finally, we indicate how these new insights can be applied to a realistic example of capacity planning.

ILLUSTRATIVE CAPACITY CALCULATION

We follow the CICS transaction example discussed in [Kraus 1995]. Six independent regions can be dispatched under MVS. Hence, there is potential to use 6 engines (one per region) but MPE will be a limiting factor.

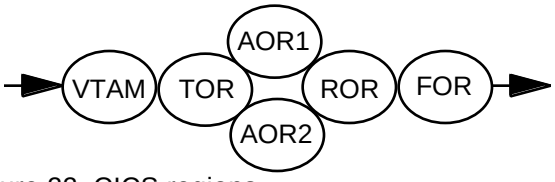

Figure 22. CICS regions.

How can we use our new insights about MPE to determine the optimal number of engines? First, we need to determine the respective model parameters. On the basis of the foregoing discussion, we ignore Geometric scaling as unphysical. That leaves two parameters: the seriality constant σ in the Amdahl model, and the g-factor in the Quadratic model. These parameter values can be determined from the CPU service-demands shown in Table 3 [Kraus 1995].

Table 3. CICS service times

Region	CPU-msecs
VTAM	1
TOR	5
AOR1	38
AOR2	40
ROR	6
FOR	10

The minimum time for a CICS transaction to be processed is Rmin = (1 * 1) + (1 * 5) + (2 * 18) + (1 * 6) + (1 * 10) = 61 msecs. The underlined term arises from the fact that the CICS transaction is processed by AOR1 or AOR2, but not both. The time for the AOR step is therefore the average: (40 + 38)/2 = 39 but each region can be running concurrently on 2 separate CPUs. Hence, the time is 18 msecs for each and the number of CPUS is 2. The other regions run on a single CPU.

Amdahl: The seriality constant is computed from the **average** of the CPU service demands (Davg) as follows:

$$Davg = Rmin/6 = 10.17 \text{ msecs}$$
$$\sigma = 1/Davg = 0.0984$$

The asymptotic capacity is $A(\infty) \approx 1/\sigma = 6.00$ CPU equivalents if there were an infinite number of physcal CPUs available.

Quadratic: What about the g-factor? Table 1 tells us how to compute that quantity. If $\sigma = 1/Davg$, then $g = 1/(1 + Davg) = 1/11.17 = 0.0895$. The maximum capacity will occur at $p_{max} = 1.0895/0.17890 = 6$ physical CPUs and produce an effective capacity of $Q_{max}(6) = 6 * (1 - 0.0895 * 5) = 3.32$ CPU equivalents.

So, the bounds on capacity at 6 physical CPUs is 4.02 if we use Amdahl scaling and 3.32 if we use the Quadratic model. In other words, the MPE represents a loss of 1.98 CPU units according to the (optimistic) Amdahl model, and 2.68 CPU units according to the (pessimistic) Quadratic model. Now you, as the capacity planner, get to choose which model is more accurate. But you should no longer be making that judgement based simply on curve-fitting. You only need to ask yourself, Does CICS processing induce more ***broadcasting*** (or serialization), or more ***pairwise*** interactions? I leave that for you to picture.

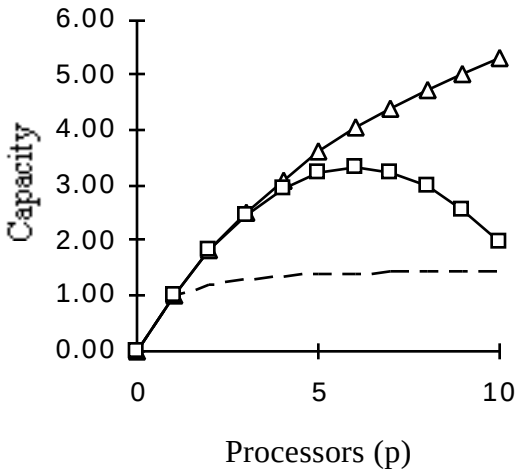

Figure 23. Capacity projections for the CICS workload. The curves are: Amdahl (Δ), and Quadratic (❑), hardware (balanced) bounds on capacity, and the dotted line is the software lower bound on capacity due to workload imbalance.

Finally, it should be noted that the capacity models we've been discussing give the ***harware*** (upper) bound on capacity because they implicitly assume that the workload is well balanced. If it isn't, as in this CICS example, the software imbalance may dominate the available capacity. The ***software*** lower bound for the CICS example is shown as the dashed line in Figure 23. Depending on the actual degree of MPE, the realized capacity will fall somewhere within these two bounds.

ACKNOWLEDGMENTS

Thanks go to Mark Friedman, Irwin Kraus, Jim McGalliard, Rich Olcott, Lois Robinson, and the CMG reviewers for comments that improved the clarity of this paper.

NOTES

1. Asymptotic capacity occurs in the (unphysical) limit of an infinite number of processors. We can track progress toward the asymptote using equation (1) e.g., A(1000) = 29.38, and A(10,000) = 29.91. Clearly, A is approaching the value $1/\sigma = 30$, denoted $A(\infty)$ in the text.
2. We need a common basis for comparing the ***divergence*** in the capacity predicted by each model. If we actually ***measured*** σ, Φ, and g, we'd expect them to be different from each other. We also expect least divergence (due to overhead) at 2 CPUs. A simple expedient is to assume that each model predicts ***identical*** capacity at p = 2, then we can simply calculate the values shown in Table 1 with little error.
3. What's really new about these diagrams is the weight assigned to each arc. That requires intimate knowledge of each capacity model expressed as a ***discrete series*** [Gunther 1997]. As far as this author aware, this is completely novel. The Amdahl and Quadratic diagrams are particular planar, fully-connected graphs called ***complete*** graphs [Chartrand 1977] p.30. The Geometric graphs are simply one-dimensional Markov chains.

REFERENCES

Amdahl, G. 1967. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. ***AFIPS Conf. Proc.*** **30**: 483-485.

Artis, H. P. 1991. Quantifying Multiprocessor Overheads, ***Proceedings CMG'91 Conference*** pp. 363-365.

Chartrand, G. 1977. ***Introductory Graph Theory,*** Dover.

Fitch, J. L. 1992. LSPR Processor Benchmarks: IBM's Approach to Large Processor Capacity Evaluation. Document GG66-0232. IBM Washington Systems Center. IBM Corporation.

Gunther, N. J. 1993. A Simple Capacity Model for Massively Parallel Transaction Systems. ***Proceedings CMG '93 Conf***, Dec. 5-10, San Diego, Cal. pp. 1035-1044.

Gunther, N. J. 1995. Assessing OLTP Scalability for Massively Parallel Processors. ***Capacity Management Review***. **23**(11): 1-22.

Gunther, N. J. 1997. ***The Practical Performance Analyst***. McGraw-Hill. In press.

Hennessy, J. L., and Patterson, D. A. 1990. ***Computer Architecture: A Quantitative Approach.*** San Mateo: Morgan Kaufmann.

Kraus, I. F. 1995. The Role of Symmetry in the Parallel Sysplex. ***Proceedings CMG '95 Conf***, Dec. 3-8, Nashville, Ten. pp. 106-117.

McGalliard, J. W. 1995. Case Study of Table-Top Sizing with Workload-Specific Estimates of the Multiprocessor Effect. ***Proceedings CMG '95 Conf***, Dec. 3-8, Nashville, Ten. pp. 208-217.

Major, J. B. 1986. A Methodology of Processor Capacity Comparison. ***Proceedings CMG '86 Conference*** pp. 241-24