

What Amdahl's Law can tell us about multicores and multiprocessing

By Darryl Koivisto

Embedded.com

(11/24/05, 02:00:00 PM EST)

Two “laws” have been much on the mind of developers of embedded systems faced with bringing higher levels of performance to designs with strict limits on power: Moore’s Law and the law of diminishing returns

In the mid-60s, [Gordon Moore](#) made the observation that component-doubling time on integrated circuits was 12 months, which he later revised to a somewhat slower 24 months. Since then, others have tried to more closely fit his prediction to actual results by changing the doubling interval to 18 months.

This extrapolation has taken on the force of law over the years, despite evidence that as we move toward smaller and smaller geometries that the law of diminishing returns is taking its toll: the tendency for a continuing application of effort or skill toward a particular project or goal to decline in effectiveness after a certain level of result has been achieved.

In the face of this, in order to maintain performance improvements in designs that are severely constrained as to power and space, developers have moved away from the well understood and tested single processor model to system, board and circuit designs with two, three or four processors operating in parallel which they hope will allow them to more efficiently balance power and performance.

But as we make this move, it is important to keep yet another “law” – Gene Amdahl’s – in mind, as a guide to where we can go with this approach, where it is most useful and where it is not.

Simply stated, Gene Amdahl’s law of parallel computing states that the speedup of a parallel algorithm is limited by the fraction of the problem that must be performed sequentially; that is, your design is only as strong as its weakest link.

In this context, parallel computing can be defined as two or more processing units applied to solve a single problem. The processing units can be physical or logical processes. A process is an abstraction provided by the operating system to capture the state of an executing program. Thus, a program containing parallelized tasks in a program can be executed by two or more cooperating processes on one or more processors.

Central to the subject of parallel computing are algorithms where as a general rule there are no naturally occurring parallelisms. Parallelism must be created by removing internal dependencies that exist in most algorithms, or entirely new algorithms must be created in some cases.

Important in many parallel processing environments is the tradeoff between the “speedup” of parallel execution times versus sequential execution times, or simply the ratio of parallel performance divided by sequential performance on a single processing unit.

Suppose there is a particular code that one executes on a processing unit that cannot be parallelized. Amdahl’s Law of parallel computing assumes that no matter how many processing units are available, a sequential portion of the code may need to be executed. This code could be the communication time to send messages to parallel execution units, or synchronization time to synchronize the parallel execution units into a single result.

A closer look at Amdahl’s Law

Amdahl’s speedup law, shown below, relates the total parallel execution time ($T_{parallel}$), plus serial execution time (T_{serial}) to the parallel execution on separate processors ($T_{parallel}/N$), plus serial execution time (T_{serial}).

$$\text{Equation 1. } SpeedUp = (T_{serial} + T_{parallel}) / (T_{serial} + (T_{parallel}/N))$$

If N is one, then this is equivalent to the single processor case and the speedup factor is 1.0, or essentially no speedup. For a given Tserial value, there is an upper bound to the theoretical speedup value. For example, if Tserial = 0.01 of Tparallel, or 1% of the parallel processing time, then the upper bound to the speedup becomes:

$$\text{Equation 2. } \text{SpeedUp} = (0.01T_p + T_p) / (0.01T_p + (T_p/N))$$

If N is a large number, then the SpeedUp factor approaches an upper bound of approximately 50.0, as N approaches 99 processors, a practical limit. If N = 10, then the SpeedUp factor is approximately 9.18, or in general, the SpeedUp factor is less than the number of parallel processors utilized. If Tserial time is larger than the 1% of the Tparallel example, then the SpeedUp factor will be reduced proportionately.

Predicting and measuring parallel performance

Parallel performance is more complex than sequential performance for a number of reasons. First, it is harder to predict or measure when independent processors are cooperating on a single algorithm consisting of parallel tasks. Sequential performance has been historically easier to predict, by examining code execution profiles, or timing code embedded in the application itself.

Second, during parallel execution, message passing and synchronization become factors that are not as easily measured, however, can be added with some effort into existing code. More importantly, predicting parallel execution becomes more difficult on parallel processing units without a common time reference, and detailed execution information for each execution unit.

The notion of speedup as defined in Amdahl's Law relates to fixed size tasks, or when the mean to maximum execution time approaches unity, defined earlier as the optimal task balancing case. In many cases one may deal with parallel tasks of differing size, and Amdahl's Law becomes more difficult to apply to these scenarios.

As used here, a task is part of a program that can be thought of independently from the other parts of a program. One can think of a task as a separate procedure, such as a member function in C/C++, or a method in Java. The smallest parallel task of a set of parallel tasks is the smallest task that can be performed independently of any other computation.

Tasks that can be performed independently of any other computation, and also require no communication messages, are defined as simple parallel tasks. The iteration space for a given program, or simulation, is the set of all possible loop indices or parameter settings.

An iteration space decomposition consists of all the disjoint subsets of the iteration space whose union is the entire iteration space. Task balancing is defined as iteration space decompositions that are of equal proportion, meaning they can be processed by similar parallel execution units in roughly the same time.

In parallel task execution, task balancing is important in that if the slowest, smallest parallel task has not completed, then the entire parallelized task cannot finish. Statistically speaking, task balancing can be defined as the ratio of the mean to maximum execution time being as close as possible to 1.0.

Applying Amdahl's law to a processor array

Using the [VisualSim modeling software](#), Mirabilis Design has built a Processor Array Model that examines nine processors being processed by a tenth processor that merely distributes tasks to the nine remaining processor elements.

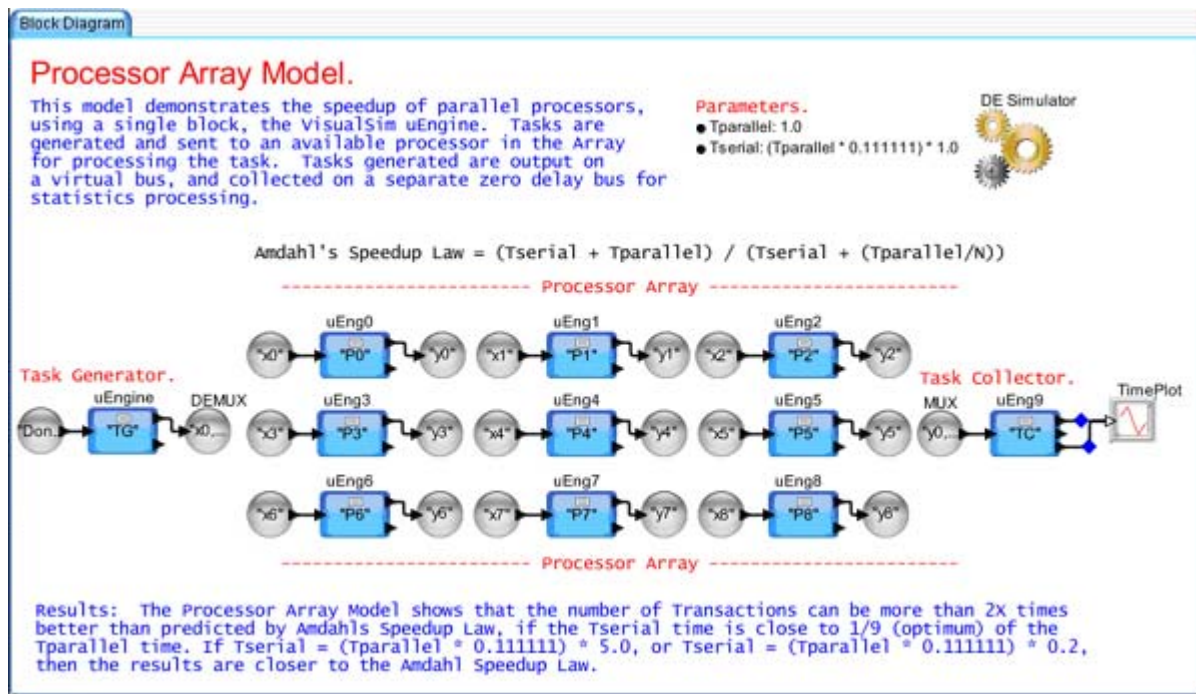


Figure 1

The Processor Array Model, shown in Figure 1 above, illustrates how Amdahl's speedup law can be applied to a model of array processors. The Task Generator on the left-hand side generates sequential tasks destined for each processor in the array in sequential order using a simple SmartMachine script. SmartMachine is a rapid prototyping scripting language that is fully integrated in the VisualSim Architect environment.

```
LBL Virtual_Bus_Thread ;
ACT virtual wait ; /* Wait for Transaction */
ACT DS_Out = Input_Queue ; /* Get from Input Queue */
ACT Tserial wait ;
ACT output send DS_Out ;
GTO Virtual_Bus_Thread ;
```

The processor array elements then process each transaction transmitted by the task generator. Upon completion, each process array element passes the transaction to the Task Collector and also back to the Task Generator to obtain the next transaction. The task Collector collects statistics on the completed transactions and at appropriate times during the simulation, plots values to a Timed Plotter using another simple script.

```
LBL Processor_Thread ;
ACT input wait ; /* Wait for Transaction */
ACT Proc_DS = Input_Queue ; /* Get from Input Queue */
ACT Tparallel wait ;
ACT output send Proc_DS ; /* Collect Stats */
ACT Done send Proc_DS ; /* Signify Done to TG */
GTO Processor_Thread ;
```

The two plots shown below in Figure 2, below represent the Processor Array Model results (red) versus Amdahl's Law (blue). One will notice that the two curves diverge, as the model processes twice as many transactions in the time period. While there are only nine processing elements in the Processor Array Model, the Y Axis, Relative Processing Capacity, indicates a multiplier of 10.0 for nine array processors.

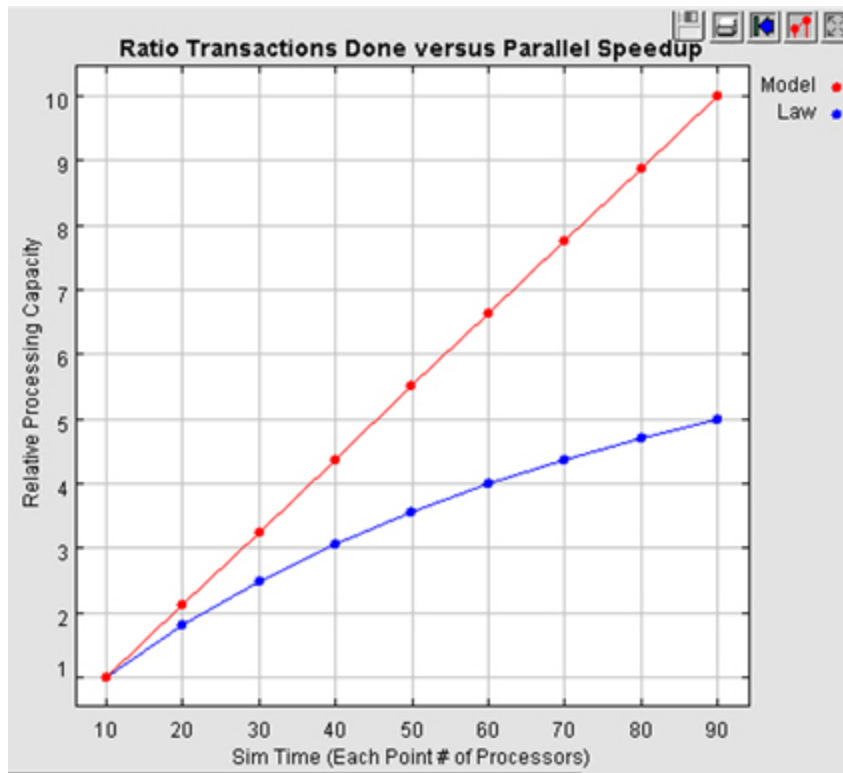


Figure 2

This result is based on the fact that the unity point includes the processing element delay and the distribution delay, set at 1/9 of the processing element delay. This implies that the true multiplier is $10.0 / (1 + 1/9) = 9.0$, or the ideal multiplier for nine array processor elements. The question remains, why is the Processor Array Model more efficient than as predicted by Amdahl's Law of parallel processing? The Processor Array Model is pipelined to send transactions to processor array elements with a separate processor, the tenth processor in the model.

In addition, the tenth processor expends exactly 1/9 the time to send a transaction to each processor array element, thereby being ready to send a new task to the first processor exactly one array processor array transaction delay later. Amdahl's Law assumes each processor performs a serial component as part of internal processing, whereas the Processor Array Model has assigned a separate processor to perform this serial function.

Optimum Parallel Processing

The Processor Array Model indicates that an optimum parallel processing case can be created that improves on Amdahl's Law of parallel computing if the following conditions are met:

- (1) Tasks can be performed independently of any other computation, and require no communication messages, defined as simple parallel tasks.
- (2) Task balancing is optimized if iteration space decompositions are of equal proportion, meaning they can be processed by similar parallel execution units in approximately the same time.
- (3) Serial time of task distribution is equal to the array processor delay divided by the number of processors, and performed on a separate serial task distribution processor.

This is an important finding in that it suggests that parallel processing improvement can be in direct proportion to the number of processors, assuming the above conditions, and little memory interaction. A hierarchical memory structure will have contention and arbitration that will reduce the potential parallel processing improvement.

Conclusions

An optimal parallel processing case can set a theoretical limit equal to the number of added processors, if certain conditions can be met regarding the serial processing of parallelized tasks to processor array elements, the size of the tasks, and the timing of the tasks. Equation 3 below summarizes this intuitive finding for the above conditions:

$$\text{Equation 3} \quad \textit{OptimumSpeedUp} = (T_{\textit{parallel}}) / (T_{\textit{parallel}} / N) = N$$

A real world parallel processing scenario would undoubtedly involve more memory interaction at the cache, SDRAM, or disk level that will reduce the speedup factor from this ideal case and approach the Amdahl Law speedup case. In addition, a real world case would involve more array processor element serial code that would also reduce the potential speedup. Nonetheless, the OptimumSpeedUp equation can serve as a theoretical limit for parallel processing systems.

For more about this subject, go to [More about multicores, multiprocessors, and tools](#)

Dr. Darryl Koivisto is the chief technology officer of [Mirabilis Design Inc.](#)

Please [login or register here](#) to post a comment