An Effective Distributed Architecture for OPC & RET Applications

Robert Lugg, Mathias Boman, Jim Burdorf, Michael Rieger

Synopsys Inc., 2025 NW Cornelius Pass Road, Hillsboro, OR 97124 (503) 547-6000
Mask Synthesis

ABSTRACT

The computational power needed to generate mask layouts for OPC and resolution enhancement techniques increases exponentially with process node. Rapidly growing design complexity is compounded with the more aggressive methods now required for smaller feature sizes. Layers once considered non-critical now routinely receive correction. While some improvement in code efficiency can be expected, algorithms are maturing to the point where improvements will likely not keep pace with the computational need. To maintain required processing cycle times massively parallel processing methods must be employed.

In this paper we discuss loosely-coupled distributed computing architectures applied to OPC/RET layout synthesis. The degree to which an application is scalable depends on how well the problem can be divided into independent sets of data. Furthermore, data must also be partitioned into reasonably sized blocks so that memory requirements per processor can be bounded. Communication overhead, I/O overhead and serial processeses all degrade scalability, and may increase overall storage requirements. In this paper we analyze behavior of distributed processing architectures with large numbers of processors, and we present performance data on an existing massively parallel system.

Keywords: distributed processing, distributed computing, optical proximity correction, OPC, mask data prep., resolution enhancement technique, RET.

1. INTRODUCTION

1.1 Distributed Processing Model

Effectively distributing a process across processing nodes requires a strategy for partitioning the tasks. In general, the optimum data partitions for each job are dependent on an analysis of all the data in each job. A distributed system will have a component devoted to defining partitions and a component in which parallel processing is highly leveraged. The ability to parallel process is often described using Amdahl's law[1, 2]. It divides a job into two categories: 1) a serial portion that cannot be distributed, 2) a parallel portion that can be divided among many processors. Amdahl's law for execution time is:

$$RunTime = s \cdot T + \frac{(1-s) \cdot T}{N} \quad (1)$$

where:

- T = The amount of (wall clock) time the sequential version of the program takes to execute
- N = The number of processors to be dedicated to the parallel version of the program during its execution
- s = Serial overhead. The fraction of the execution time T that must be spent in sequential code

For a normalized amount of work (T), runtime can be plotted for various amounts of overhead.

Figure 1a shows the improvement in runtime for various levels of serial overhead as processors are added. Amdahl's Law produces a sharp runtime curve; therefore special care must be taken to realize performance gains beyond more than a few processors. The real impact of distributed processing is the fact that it makes impractical calculations practical[3]. Figure 1b shows the runtime for a 100-day (2400 hour) job. The runtime using one processor would be impractical for most situations. With 100 processors, the runtime is reduced to less than 30 hours making the runtime much more reasonable. A well-distributed application naturally enables more computations to be reasonably performed.

22nd Annual BACUS Symposium on Photomask Technology, Brian J. Grenon, Kurt R. Kimmel, Editors, Proceedings of SPIE Vol. 4889 (2002) © 2002 SPIE · 0277-786X/02/$15.00

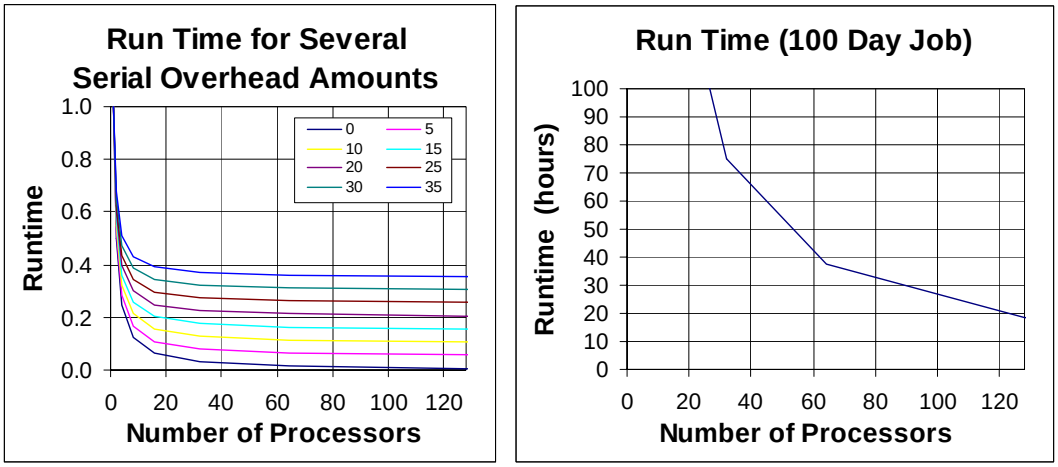

Figure 1:a) Amdahl's law for various serial overhead, b) Example of adding processors to make a long job practical.

Amdahl's Law is also useful for determining job speedup as additional processors are added. Figure 2 plots speedup versus various serial overhead amounts.

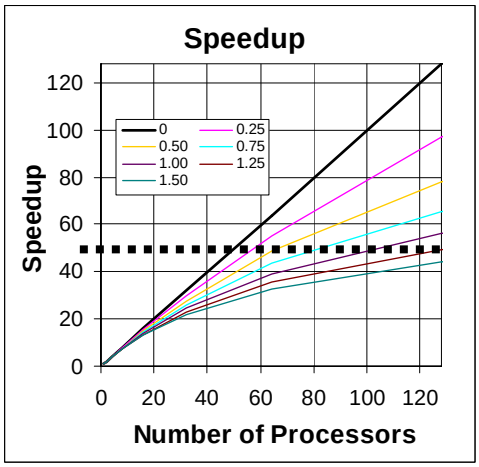

Figure 2: Speedup as processors are added for various serial overhead values.

Serial overhead values have been reduced from those in Figure 1. It is clear that even one percent serial time will seriously affect performance on distribution to many processors. This curve is useful in examining real systems. If test results follow the same trends as the theoretical functions, the serial overhead 's' can be calculated. The theoretical curves can then be used to predict system performance for any number of processors. It has been realized that the use of speedup curves is useful when studying real systems as long as those systems follow Amdahl's law. This is because it is easier to discern small differences in serial overhead using speedup curves. The maximum serial overhead allowed depends on the application. Processor and software costs must be balanced with runtime gains. For OPC/RET applications, a 50% efficiency level was selected as a minimum threshold. At this level, a system's serial overhead must be less than 1% to efficiently distribute to 100 processors. This may be counter-intuitive if one considers runtime on a single processor.

The cause of serial overhead is also significant. If the serial time remains constant and the parallel processing work increases, the serial time as a percentage (‘s’) will decrease. Such is the case observed with OPC/RET applications where the required work is growing at a rate faster that the size of the input data.

1.2 Communications Overhead

Modeling a distributed system as serial and parallel components is over simplistic. As a practical matter, some level of communication among the parallel nodes is required, which adds a third component to the cycle time equation. As the number of processors increase, the work spent communicating (control and data transfer) will increase. When performing model-based operations[4], for example, polygons in proximity to the feature being evaluated must be considered. Consider a simple case where a flat design is divided spatially in order to distribute it.

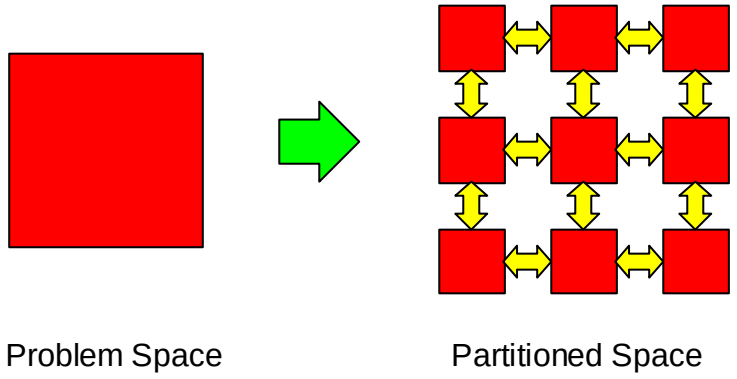

Figure 3:Problem is divided into 9 partitions for distributed processing.
Communication is required between partitions.

In practical situations, partitions need to communicate as illustrated in Figure 3 above. As additional partitions are added, runtime may increase. When interactions are confined to a known, fixed range, the communication involves data in bands at the boundaries of each partition. In this case, data communication can be modeled by duplicating data at the partition boundaries as shown in Figure 4.

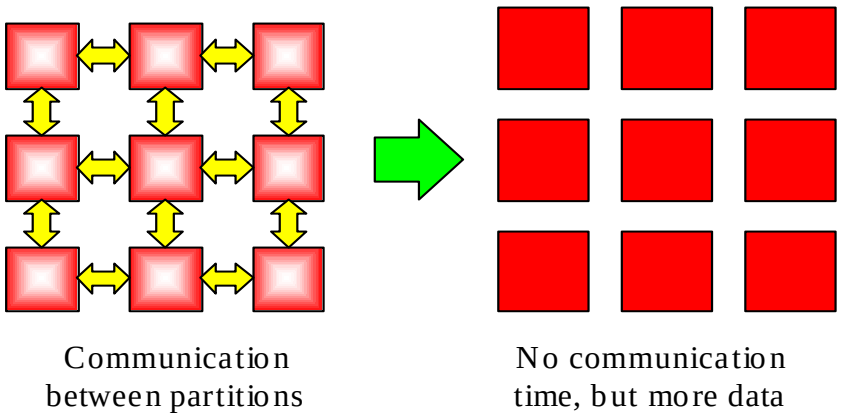

Figure 4: Modeling communication overhead with data redundancy.

This increase in work will be a function of the range of influence. Assuming that work is proportional to added area, Figure 5 plots the work increase as a function of range of influence (given as percent of partition size) and number of partitions.

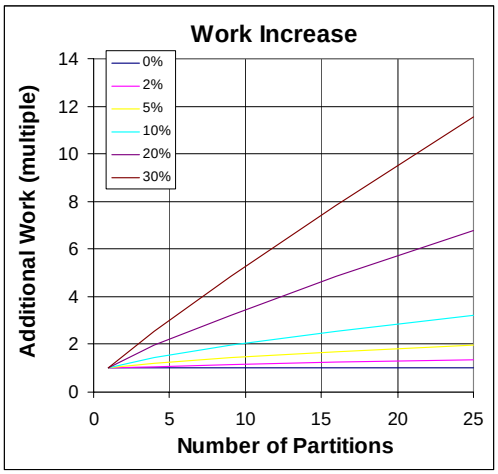

Figure 5: Additional work as a result of data redundancy.

From this model it is clear that as communication needs increase, the acceleration provided by distributed processing decreases. On a real system the communication overhead work can be inferred by comparing observed scalability in the distributed components to ideal, 1/N, scalability.

2. METHODOLOGY/DATA

2.1 Determination of communication overhead

Communication overhead is difficult to directly predict. It is a lumped factor that can include external network factors such as speed, bottlenecks, and other running processes not associated with correction (operating system operations, for example). Because of this, and empirical approach is taken. Improvement results are therefore conservative. The total run time on 12 to 97 processors was recorded. One-processor tests were not performed due to the long run times. The results were then compared to the theoretical curves of Figure 1. Any deviation from the curves would indicate possible communications overhead in the system. Figure 6 below shows early performance on an SMP machine.

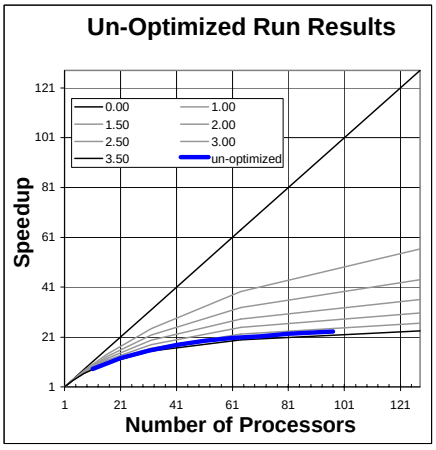

Figure 6: Initial results showing sub-optimal performance, but negligible communications overhead.

The curve indicates that test data follows the theoretical curves. Overhead seems low (roughly 3.4%), but this percentage is far too high to distribute much beyond 20 processors. Because test results have consistently followed theoretical functions, a single run with a given number of processors may be used to study behavior. Serial overhead can then be calculated and used to predict performance.

2.2 Improvements to achieve 100 process scalability.
Results from above indicate that overhead in this system is dominated by serial processing. Several parameters were studied to understand their effect on overall performance. Parameter testing was performed on an 8 CPU SMP machine due to availability. Results of the first three tests are summarized in Table 1.

Design size was first examined. A hierarchical design with multiple replications was used for testing. Eight and sixteen replications yielded similar test results. This can be explained by the fact that larger designs increase both the serial and parallel operations of a correction. Thus the percentage serial overhead remains the same.

Task complexity was also examined. As the math involved in performing correction increases, efficiency also increases. This is because the serial portion as a percentage of overall correction time decreases. 4, 8, and 12 functions were tested. Overhead decreased with increasing numbers of functions. Eight functions are in a typical recipe. But, as work increase, so too can the number of processors used.

Designs are divided into chunks to be distributed. Changing the chunk size demonstrates an interesting difference between "fast" and "scalable". It was found that reducing chunk size reduced serial time and increased parallel processing time. These both reduce the percentage of serial overhead. However, the one-processor correction runtime was higher than for larger chunk sizes. The best chunk size is therefore a function of the number of processors. Relative chunk sizes of .5X, 1X, 2X, and 4X were tested. The overhead is reduced at the cost of runtime on a small number of processors.

Parameter Changed	Serial Overhead (%)
Baseline*	2.88
16x design size	3.00
8 function	2.12
12 function	1.15
0.5X chunk size	1.27
2x chunk size	2.99
4x chunk size	3.44

* Baseline is an 8X design size, 4 function, 1X chunk size

Table 1: Serial Overhead after adjusting various parameters (Independently)

Varying the design size, recipe complexity and chunk size improves overhead, but not to the level needed to distribute to 100 processors. In practice, design size and function counts are fixed will not be modified to improve scalability. Chunk size had the side effect of increasing one-processor correction time, but may still be desirable. The one remaining parameter of interest is code efficiency. For a one-processor program, functions that use the most absolute time should be optimized. This is not true for the 100-processor case. For the distributed case, a detailed analysis of the serial portions of the code was conducted. Optimizations were implemented which resulted in a theoretical one-processor speedup of 2%. When using 100 processors, the same optimizations resulted in a predicted improvement of more than 200% (2x).

To confirm predicted improvements, tests were conducted on the three hardware platforms. Sun and HP systems were examined because they dominate the EDA market. The Sun testing was on a 900MHz SunFire 6800 SMP system. For the HP testing, a 64cpu and 32cpu Superdome were used. The HP systems were configured to act as 6 systems, each with 16cpus each. Both platforms had approximately 32GB of RAM. A cluster of Linux machines was also used. The cluster consisted of Ethernet connected, Pentium 4, 1.7GHz machines with 256MB of RAM each.

Results of the three tests are graphed in Figure 7. The same baseline conditions as above (Figure 6) were used. The serial overhead reduction resulted in a more scaleable system. Serial overhead for the HP and Sun systems was less than 0.5%. Serial overhead for the Linux cluster was less than 1.0%.

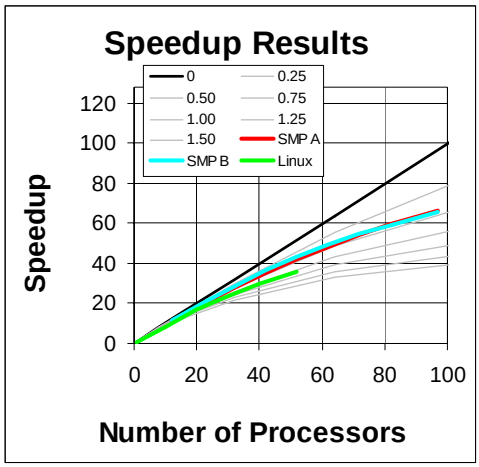

Figure 7:Results on Sun, HP, and Linux

All testing data agreed with Amdahl's, therefore, it was confirmed that there was negligible communications overhead during the testing. This was true for both the SMP and clustered systems. Serial overhead was different between SMP and clustered machines. Several factors could have contributed to this difference. First, the processor speed (in MHz) was roughly double in the Linux case. Second, disk access (performed primarily during serial portion of run) may be optimized on SMP systems as they are designed for enterprise applications.

3. CONCLUSION

It was confirmed that OPC software scalability follows Amdahl's law with negligible communications overhead. Cycle time with N parallel processes is always less than 1/N. Various run parameters were tested to determine their effect on scalability. Coding improvements in the serial portion of code were found to dominate the other parameters. Tests on three platforms (HP, Sun, and Linux) were used to confirm predicted results. The Proteus corrector was shown to scale to 97 processors while maintaining at least a 50% efficiency level.

ACKNOWLEDGEMENTS

The authors would like Bill Kielhorn for valuable discussions, Michael Ehrig from Hewlett-Packard Co. and Morgan Herrington from Sun Microsystems.

REFERENCES

1. G. Amdahl, *Validity of the single-processor approach to achieving large-scale computational capabilities*, in AFIPS Conference Proceedings, **Volume 30**, page 483, AFIPS Press, 1967.
2. Anita Osterhaug, *Guide to Parallel Programming on Sequent Computer Systems*, Sequent Computer Systems, Inc., Beaverton, Oregon, 1986
3. J. Gustafson, "Reevaluating Amdahl's Law," *Communications of the ACM*, **Volume 31**, May 1988.
4. Stirniman, John P., Rieger, Michael L., "Fast proximity correction with zone sampling", Proc. SPIE **Volume 2197**, p. 294-301, Optical/Laser Microlithography VII, May 1994.