



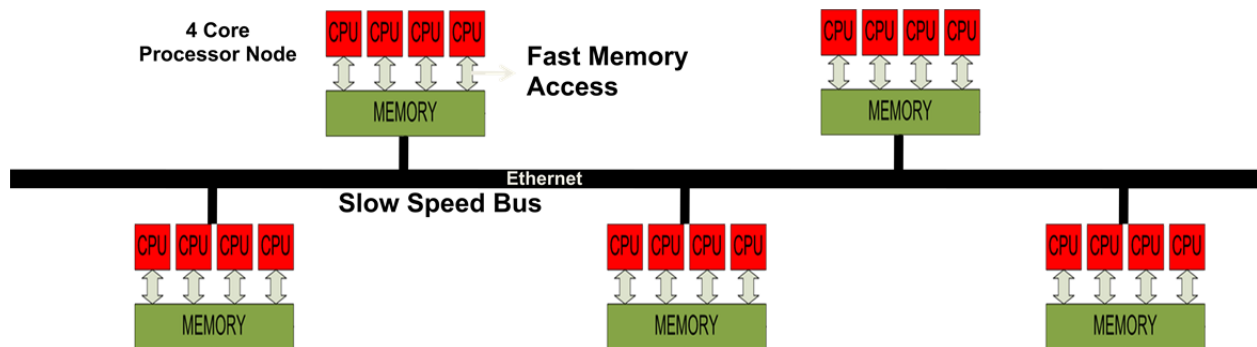
White Paper: Nimbic's Parallelization Methodology

Dipanjan Gope, Vikram Jandhyala and Xiren Wang

Nimbic's parallelization strategy, built upon a two-layer foundation of (a) fine-grained, multithreaded solver architectures for the multicore environment and (b) coarse-grained, distributed architectures for multiple processors connected by high-speed buses is summarized. Nimbic's parallelization methodology is tuned to simultaneously exploit Nimbic's proprietary fast solver technology and to take advantage of emerging multicore and many-core processors.

Computing Trends

Consider the figure below. This hybrid cluster of multicore nodes connected by a relatively slow speed bus, when compared to the fast memory access between cores on the same node, is the norm. With emerging many-core architectures, the number of cores on each node will continue to increase. Efficient use of this architecture requires a combination of *fine-grained* shared-memory parallelism at the individual node level, and coarse-grained distributed-memory parallelism at the cluster level.



Field Solver Parallelism

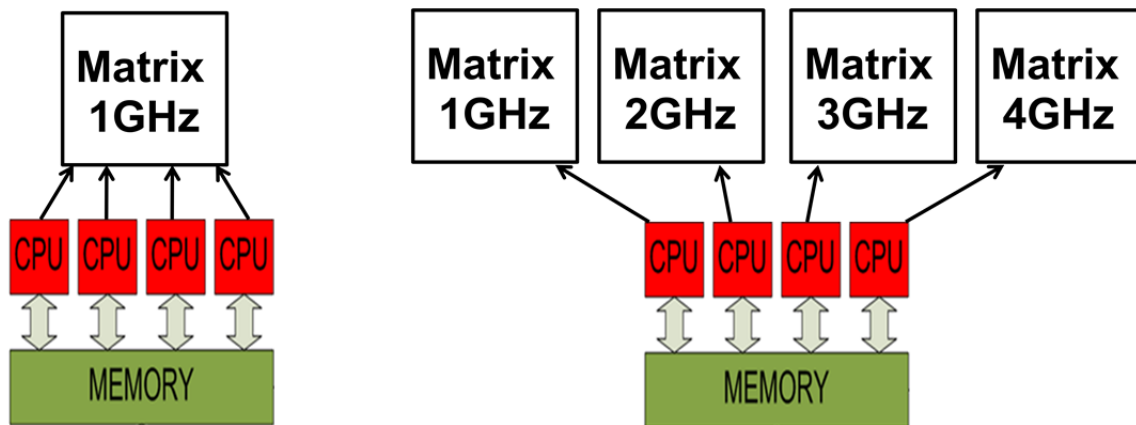
One very common approach to "parallelism" in the industry is coarse-grained distributed-memory parallelism. From the user's perspective, this permits, for example, multiple frequencies to be solved in parallel, and is well-suited for *single-core* machines connected via ethernet connections. Unfortunately this approach, though extremely simple, does not perform very well once multi-core cluster nodes are introduced. The emerging multi-core and many-core shared-memory environment is not suited to this coarse-grained parallelism, as the next example will show.

In the figure below, two scenarios are depicted. On the right, is the well-established, relatively simple to implement, coarse-grained distributed-memory parallelism, operating as it normally does, at multiple

White Paper: Nimbic's Parallelization Methodology

frequencies. On 4 cores, as shown, the electromagnetic problem is solved for 4 different frequencies. In matrix methods such as finite element or boundary element, this means that on each core, a matrix is constructed for one frequency, and solved for a given excitation. Notice that the memory available for any particular solution has now reduced by a factor of 4! In other words, the coarse-grained distributed-memory parallelism when applied to a multicore chip provides speedup as expected, but only a fraction of the memory is available for any given solution! This significantly reduces the scale of the problem such a technique can solve.

Conversely, a fine-grained shared-memory implementation of a matrix solution is shown on the left. In this case, even for a single frequency, the matrix is solved in parallel on 4 cores. There is a proportional speedup associated with using the 4 cores, and all the memory is available for a single solve.



Solver Level Parallel

Distributed Frequency Parallel

Time: Every Frequency is solved ~4times Faster
Memory: Same as sequential code

Time: 4 Frequencies are solved at the time of 1
Memory: 4 times More Memory Required

Therefore, fine-grained parallelism is particularly necessitated for multicore environments, so that the entire shared memory is available for solution, and the scale of the problem that can be solved is not reduced. However, the challenge is that it is easy to take coarse-grained parallelism as shown on the right above and adapt it to multicore scenarios, with the massive memory and scale of problem penalty. Fine-grained parallelism for multithreaded shared-memory multicore implementations is far more difficult, and requires novel parallel algorithms and matrix implementations. The table on the next page shows some of the features associated with the two very different approaches, both of which are labeled generally as "parallel" implementations.

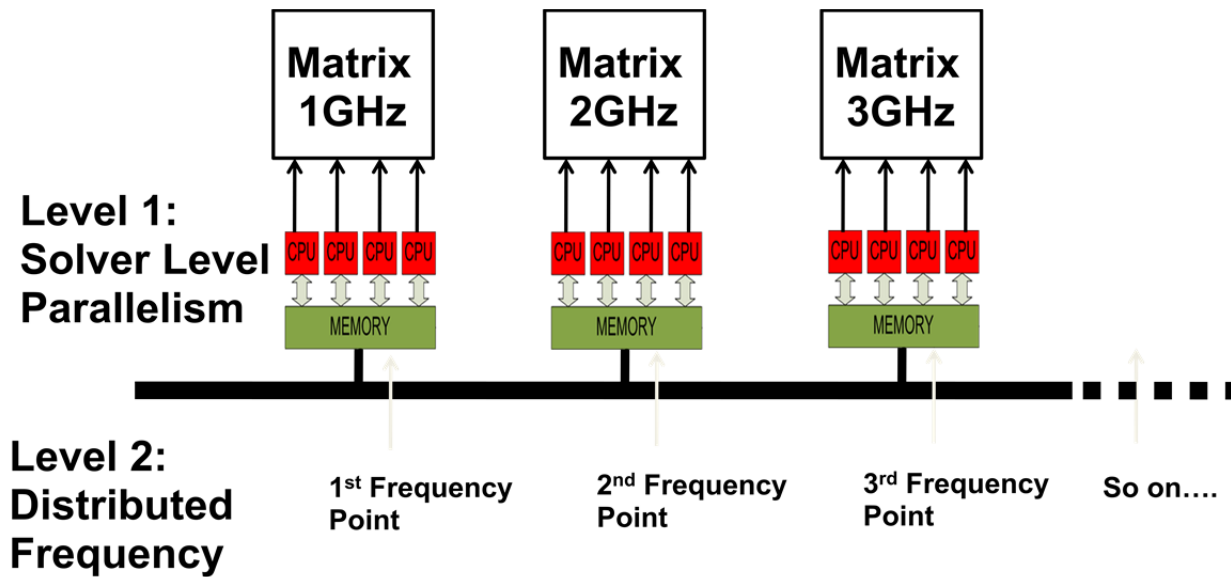
White Paper: Nimbic’s Parallelization Methodology

Features	Solver Level Parallelism: Fine-Grained Shared-Memory Multithreading	Distributed Frequency Parallelism: Coarse Grained Distributed-Memory Multithreading
Method	Matrix solution is parallelized at a single frequency	Matrix solution is itself serial, multiple matrix solutions at different frequencies are solved in parallel
Speed-up for single frequency	Up to nearly N times on N core	No Speedup
Memory Requirement	Same as sequential code with small overhead	N times that of sequential code
Advantage on multicore machines	High	Limited—Massive scale reduction on largest problem that can fit in memory
Availability in the industry	Cutting-edge; Requires bottom-up multithreaded fine-grained parallel software architecture and algorithm development	Commodity; Reasonably simple to implement

Summary: The word “parallel” is used extensively in the industry but can mean very different things, from relatively straightforward coarse-grained frequency-level distributed-memory parallelization to significantly more challenging fine-grained solver-level shared-memory parallelization. The second approach is necessary for using multicore and many-core architectures (also including emerging general-purpose GPUs) to their full potential in a scalable manner.

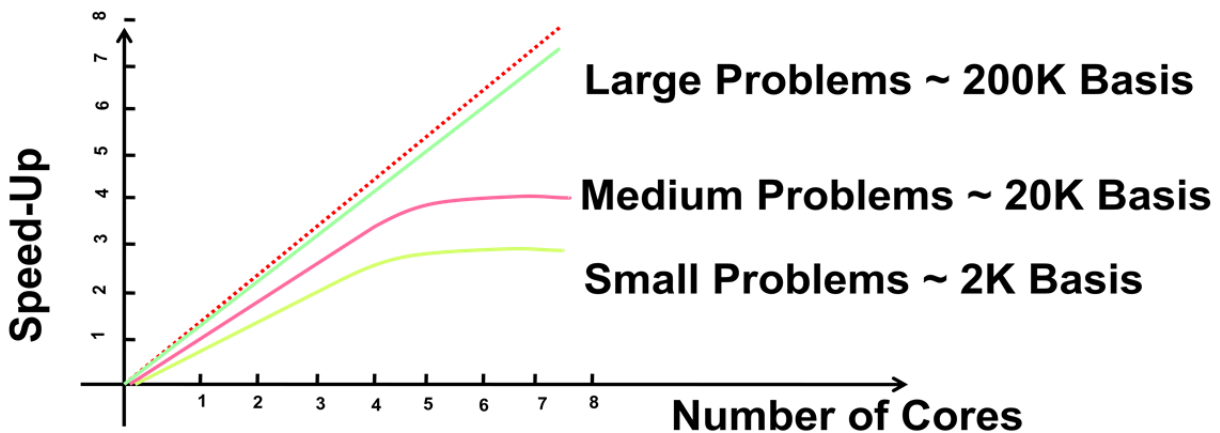
White Paper: Nimbic's Parallelization Methodology

Nimbic's technology, which is used in the full-wave product nWave, is built from the ground-up to use fine-grained, shared-memory parallelism (Level 1 in the figure below) at the solver level. In addition, for distributed-memory systems, Nimbic's parallelization methodology is to offer coarse-grained, distributed-memory parallelism (Level 2 in the figure below). The methodology is therefore a hybrid of fine-graining and coarse-graining, utilizing the best of shared-memory multicore nodes and distributed-memory clusters.



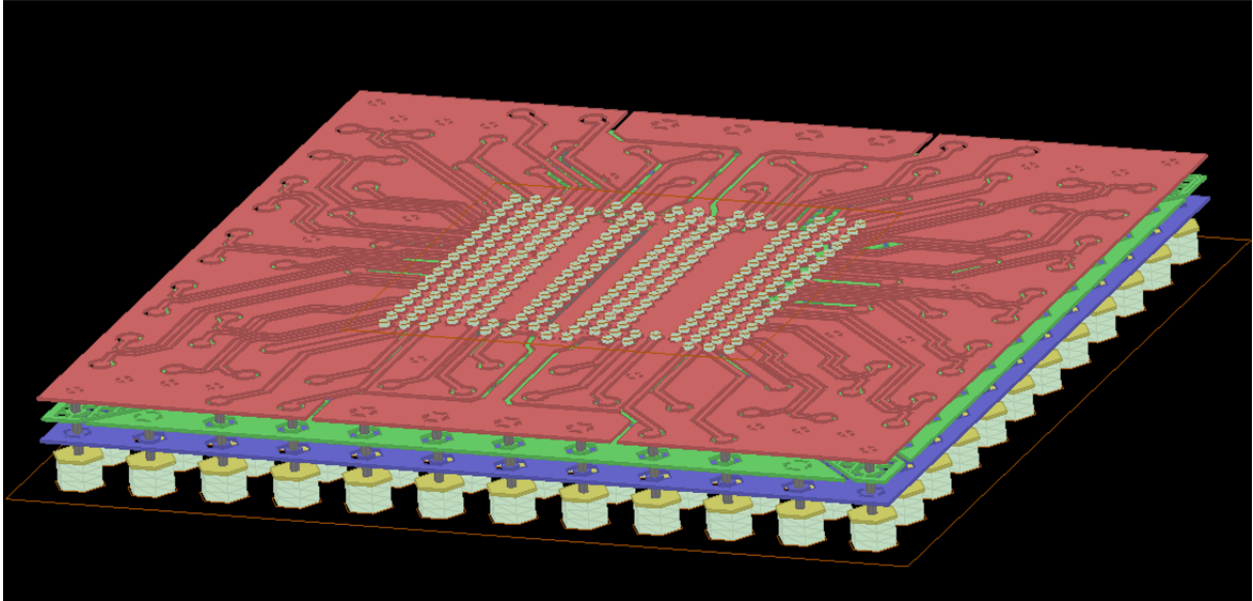
White Paper: Nimbic's Parallelization Methodology

While distributed-memory frequency-level parallelization is relatively easy to accomplish, and essentially involves spawning multiple solver threads and collating results, as mentioned shared-memory fine-grained parallelism is more difficult. In particular, Amdahl's law states that even a small amount of serial content in an algorithm can effect the speedup, so getting ideal speedup is extremely difficult. In addition, granularity of a problem is also important. In other words, even an excellent parallel code may have some small serial sections that are relatively more important when the size of the problem being solved is small. In other words, for very small problems, it is more difficult to achieve ideal speedup with too many cores! Is this an issue? No! Small problems, represented by a few thousand basis functions are solved in few seconds to a minute depending on the number of cores. For larger problems, where the speed and scale of multiple cores is critical, the speedup is significantly closer to ideal speedups as seen in the curves below (the red line shows ideal speedup).



Finally, we show an example of how nWave performs on a typical problem of full-wave extraction of a real structure, on a multicore platform. The structure below has 16 ports and is modeled with 120000 mesh elements.

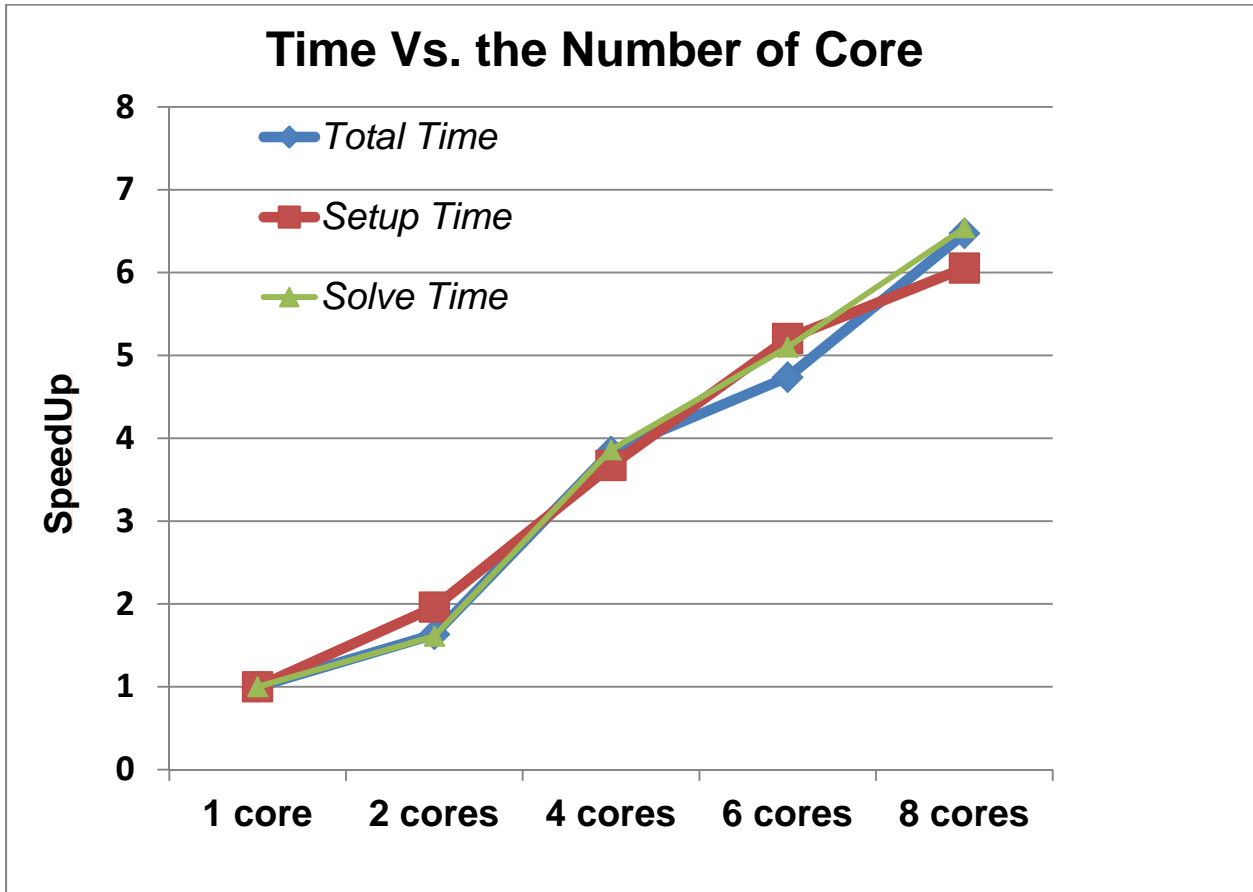
White Paper: Nimbic's Parallelization Methodology



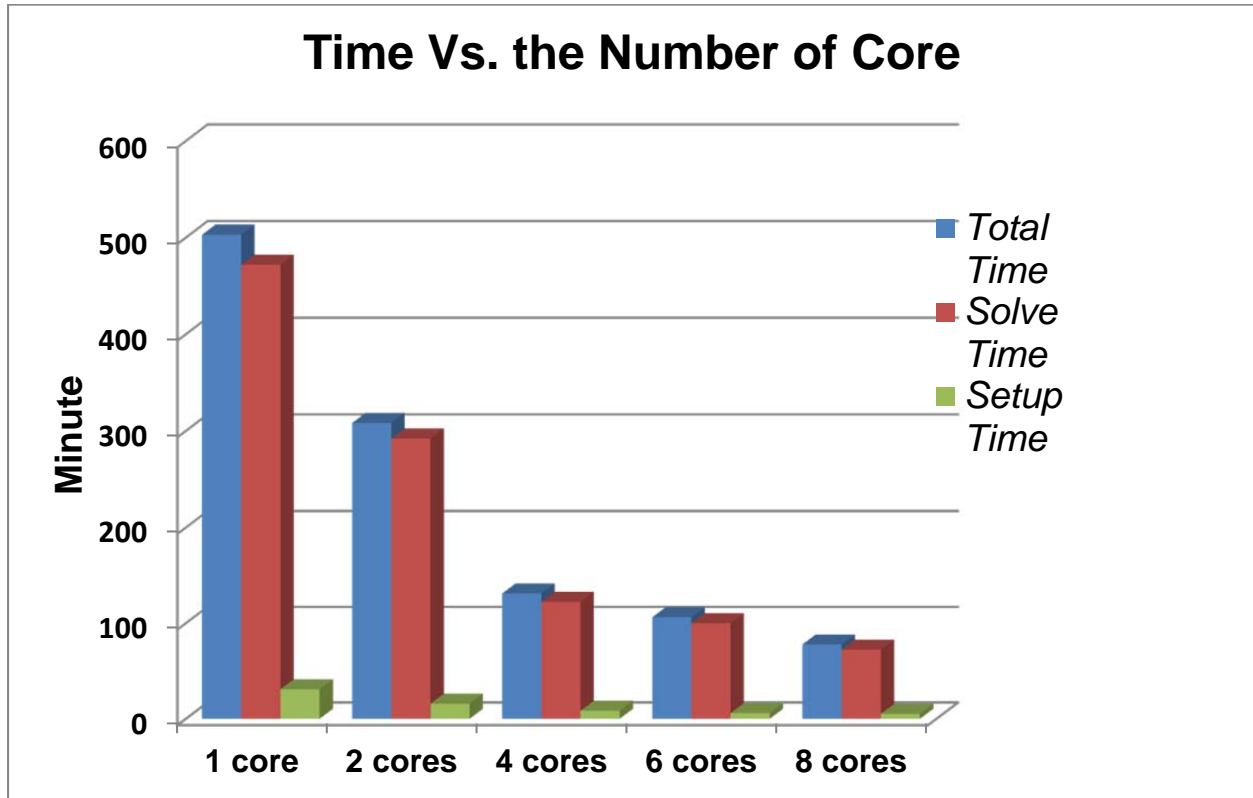
The table below shows the setup, solve, and total times on different number of cores.

Number of cores	Setup Time (seconds)	Solve Time (seconds)	Total Time (seconds)
1 Core	30.8546	471.817	503.015
2 Core	15.7394	292.051	308.189
4 Core	8.4072	122.256	131.064
6 Core	5.9291	99.8589	106.19
8 Core	5.096	72.2022	77.72

The graph below shows the speedup (i.e. ratio of single core time to multicore time) for the total, setup, and solve times, showing a speedup of 6-6.5 on 8 cores.



Finally, the figure on the next page shows the time versus number of cores. The objective of significant speedups through fine-grained parallelization on multicore shared-memory CPUs is achieved. This nWave project is available upon request for users to benchmark on their own systems.



Summary: As future microprocessors from many of the world's leading chip manufacturers are going to have even more cores, it is important for a user of EDA software to understand the impact that upgrading hardware will have: are the EDA software tools being used scalable with number of cores, and in the right manner without losing scale and simply distributing the task repeatedly and creating a huge memory overhead? This document aims at discussing these questions and to confirm that Nimbic's tools scale excellently with emerging multicore and many-core chip paradigms.

White Paper: Nimbic's Parallelization Methodology

For Further reading

1. M.D. Hill and M. R. Marty, "Amdahl's Law in the Multicore Era," *IEEE Computer Magazine*, July 2008.
2. X. Wang and V. Jandhyala, "Enhanced MPI-OpenMP parallel electromagnetic simulations based on low-rank compressions," invited for special session on tools and techniques for EMC based on parallel processing, *Proceedings of the IEEE International Symposium on Electromagnetic Compatibility*, pp. 1-5, Detroit, August 2008.
3. X. Wang and V. Jandhyala, "Parallel algorithms for fast integral equation based solvers," invited for special session on parallel algorithms and techniques for signal and power integrity, *Proceedings of the IEEE-EPEP Topical Meeting on Electrical Performance of Electronic Packaging*, pp. 249-252, Atlanta, October 2007.
4. V. Jandhyala and X. Wang, "Accelerated integral equation solution methods on clusters and multicore processors," invited for special session on algorithms and techniques for parallel processing for EMI/EMC, *Proceedings of the IEEE International Symposium on Electromagnetic Compatibility*, pp. 1-5, Hawaii, July 2007.
5. C. Yang, S. Chakraborty, D. Gope, and V. Jandhyala, "3D accelerated electromagnetic integral equation solvers on parallel processors for microelectronic simulation," invited for special session, *Proceedings of the IEEE International Symposium on Electromagnetic Compatibility*, pp. 539-543, Portland, OR, August 2006.
6. C. Yang, S. Chakraborty, D. Gope, and V. Jandhyala, "A parallel low-rank multilevel matrix compression algorithm for parasitic extraction of electrically large structures," *Proceedings of the 43rd ACM/IEEE Design Automation Conference*, pp. 1053-1056, San Francisco, CA, July 2006.
7. V. Jandhyala, C. Yang, S. Chakraborty, I. Chowdhury, J. Pingenot, and D. Williams, "A framework and simulator for parallel fast integral equation simulation of microelectronic structures," invited, IBM special session on large-scale package simulation, *Proceedings of the IEEE 15th Topical Meeting on Electrical Performance of Electronic Packaging*, pp. 287-290, Scottsdale, AZ, October 2006.