**Performance Matters**

Today’s complex computer architectures and their deep memory hierarchies are a poor match for most applications. Due to the wide memory gap, processors often devote more than half of their time waiting for data to arrive. This problem is intensified with the introduction of multicore processors due to decreased cache area per thread and more concurrent threads contending for bandwidth.

Until now performance analysis tools have forced developers to wade through a mass of data before having any idea where the performance problems are located. Even then, the process seems to require more magic than engineering skills to identify the specific nature of the problems. And worst of all, developers may spend a huge amount of time trying to identify and fix problems without first being able to prioritize and attack the most high-value fixes.

ThreadSpotter automatically analyzes the application as it’s running, lists the performance problems in order of importance, suggests fixes and gives the developer valuable insights and statistics needed to quickly assess and fix the problems.

ThreadSpotter makes performance experts more productive, and helps less experienced developers become more educated in which techniques work well with the underlying hardware.

**Optimization Workflow**

ThreadSpotter works in conjunction with most compiled languages and most parallelization paradigms. First, it helps the developer optimize the code when it’s still sequential. Then, after the code is parallelized, it helps optimize the thread interactions. Finally, it helps the developer find the optimal thread placement and assists in removing all the cache pollution effects. No other tool can cover all those bases in such a straight-forward way.

ThreadSpotter far surpasses simply collecting raw performance data – instead, it identifies, classifies and instructs the developer in ways to remove specific issues.

ThreadSpotter offers a solid, detailed understanding of performance problems to allow a quick resolution on the system under examination. It also models other execution environments and provides performance optimization guidance for other systems.

ThreadSpotter models thread communication and interaction effects, giving advice on how to resolve the resulting performance issues.
How are your threads performing?
Threads interact directly through communication of data values and synchronization. Unwanted interaction between threads may be introduced through false sharing. Indirect interactions are induced through the sharing of some resources, such as bumping each other’s data from a cache or monopolizing the memory bus. This is a major cause of application degradation, especially on a multicore architecture.

Can the problem be fixed? The easiest way to find out is to ask ThreadSpotter for analysis and advice.

How is your memory system performing?
Effective use of the memory system is critical for efficient thread execution on multicores. The caches in the memory system are only effective if the “right” data happens to be in the “right” cache at the “right” time, otherwise performance degradation and memory interface bottlenecks can make applications run significantly slower.

Two enemies in the quest for optimal performance are wasted cache space and lack of data reuse, also known as locality. Are the memory accesses causing memory bottlenecks in your application? Multicore architectures force you to answer the question: Can these problems be solved? Fortunately, with ThreadSpotter, the answer is a definitive yes.

How does it work?
ThreadSpotter will efficiently monitor the execution of unmodified application binaries and capture sparse memory fingerprints representing the essence of the application’s locality properties. Contrary to most optimization tools, no restart or recompilation of the application is necessary.

The memory fingerprint carries all relevant information for ThreadSpotter to be able to extrapolate the right metrics to any cache. It can also accurately predict these metrics for cache constellations that are different from the architecture where the fingerprint was acquired.

Fortunately, the user does not need to understand the underlying memory system detail. ThreadSpotter leverages the information to direct the user in modifying code for effective use of the memory system and increased application performance.

ThreadSpotter then correlates all problems to their respective source code instructions and data structures. ThreadSpotter captures and displays call stack information leading up to the location of the problem and utilizes this information in the analysis and presentation of relevant advice and specific metrics for each unique call graph leading into a particular instruction. In other words, it understands the hardware intricacies and their relation to performance so you can concentrate on your application.