

The SunTune Seminar An Overview

Summary – *The SunTune Optimization Seminar shows end-users of SUN systems how to get good performance out of their application. The seminar covers both how to use the SUN development environment in an optimal way, as well as manual optimization techniques to improve performance.*

The general philosophy of the seminar is to build up understanding of key concepts that are relevant to obtain good application performance. Once this is achieved, it is much easier to use the development environment in the best possible way.

The seminar covers serial optimization and shared memory parallelization with the OpenMP programming model. Other than some programming experiences (preferably in Fortran or C), no specific background in application tuning is assumed.

Examples given are in Fortran and/or C, but they are simple enough to be understood by anybody with some programming experience in another programming language.

In addition to the presentations, lab sessions are organized. These are meant to challenge the attendees to apply the theory in practice.

1. Objectives

After this three day seminar, attendees should be much more knowledgeable how to:

- Get good performance out of a Sun Fire system
 - i. Use the Sun ONE Studio Compiler Collection software development environment to optimize an application
 - ii. Manually tune a technical–scientific application
- Use the compilers to automatically parallelize a program
- Use the shared memory OpenMP programming model to explicitly parallelize an application

Attendees should also be able to give practical advise to others what to do and what not to do when it comes to making their program(s) run more efficiently on SUN systems.

2. Target audience

This seminar primarily targets people that are interested to learn more about the technical details how to tune a technical–scientific application.

We do assume you have some practical programming experiences, but do not assume any background in application tuning. All concepts will be built up from the ground up.

The examples and the lab exercises are in Fortran and/or C. Therefore some experience in either one of these languages is of help, but it is not a requirement.

Even though C++ is not covered explicitly, we believe that C++ programmers may benefit from the seminar as well.

The focus is on serial performance and shared memory parallelization using the OpenMP programming model. The Message Passing Interface (MPI) programming model is not covered, other than a very brief overview.

3. Lab Sessions

The seminar is practically oriented. In order to benefit most from the material presented, several lab sessions are included as part of the seminar. Fortran and C lab exercises will be made available. These are very simple to understand, but have been chosen to highlight a specific feature discussed in the seminar. Attendees are encouraged to work on these examples.

4. Dress code

The seminar will be rather intense, so comfortable cloths may be most appropriate. However, we prefer not to impose any preferences regarding this. Please dress in the way you feel most comfortable with.

5. Seminar language

The seminar will be given in English.

6. The Seminar Topics

In this section we list the topics covered in the seminar. Per topic, a short summary description is included.

The Memory Hierarchy

Today's microprocessors make heavy use of caches to bridge the gap between the speed of the processor and main memory. The resulting memory hierarchy can be rather complex. Often, tuning an application is about using the entire memory hierarchy in the best possible way.

In this module we explore the memory hierarchy to a reasonable level of detail. The concepts and characteristics presented here will be needed in the remainder of the seminar.

The Solaris Operating Environment

The Solaris Operating System has many powerful and useful commands that can assist the application developer. A small subset of these commands is presented in this module.

Compiler Options

The compilers included in the Sun ONE Studio Compiler Collection provide a comprehensive set of options. In this extensive module we will focus on those options that are relevant for performance. As will be explained and discussed, several categories of optimization switches are available.

In addition to compiler options, one can also use the highly tuned mathematical libraries that are included in the environment.

Frequently used intrinsic operations have been optimized extensively. In addition to this, vectorized versions of important intrinsic functions are available.

The SUN Performance Library not only provides tuned (and shared memory parallelized) versions of the BLAS1, BLAS2, BLAS3, LAPACK, FFTPACK and VFFTPACK public domain packages, but also contains additional functionality for sparse matrices, convolutions, correlations, etc.

The library can be called both from Fortran and C programs. Fortran 95 applications can take advantage of parameter type checking and use generic calls. C programmers will benefit from the native C interfaces.

We conclude this module with some options that can be helpful when porting and/or debugging an application.

Performance Analysis Tools

The Sun ONE Studio Compiler Collection offers a comprehensive program development suite, including a symbolic debugger and a source code browser.

In this module we focus on one specific component of the environment: the performance analyzer.

The performance analyzer is a powerful and useful tool to analyze the performance of an application. It can be used on existing, non-instrumented, executables and provides important information on the behavior of the application. If one is willing to recompile the application with the `-g` option added, performance and compiler optimization information at the source line level will be given.

Extensive post-processing features on the statistics gathered allow the user to obtain insight into the performance characteristics of the application.

Several metrics (CPU time, wall clock time, system time, etc.) can be obtained. On UltraSPARC-III based systems, the performance analyzer also gives easy access to the on-chip hardware event counters.

As of Solaris 8, the `cpustrack` command is available. This command provides an easy to use interface into the hardware event counters available on the UltraSPARC-II and UltraSPARC-III processors. We will explain how to use this command and show examples as well.

Architecture Overview

This relatively short module covers the UltraSPARC-II and UltraSPARC-III microprocessors and the server architectures.

Only those characteristics relevant when tuning an application will be covered. This includes details on the memory hierarchy and a description of the most important instructions.

Serial Optimization Techniques

One may be tempted to think that serial optimization is not worth considering, because that is an area well under control. Even though a lot of progress has been made, there is often room for significant improvement in many applications.

Serial performance is important for several reasons:

- Existing resources are used more optimally, typically without additional cost
- The performance increase can be noticeable
- If parallelized, a program tuned for sequential performance will usually also give better performance using many processors

A variety of techniques to optimize specific programming constructs have been developed over the years. Nowadays, the most important ones have been integrated into compilers and are part of standard optimizations. However, even though compilers get more sophisticated over time, they sometimes lack information to make the optimal decision.

Therefore we will cover all these key optimization techniques such that users can decide for themselves what should be done to assure good performance, with or without the help of the compiler.

Modulo Scheduling

Modulo Scheduling is used by the compiler as a technique to exploit the superscalar nature of the microprocessor and to hide instruction latencies. This module covers the concepts of modulo scheduling and the diagnostic messages given by the compiler. Several examples how to use this information in order to tune an application are presented.

Case Studies Serial Optimization

By now, one should have a good understanding of what the compilers can do, how to use them in the best way and what possible source code optimization techniques could be applied to enhance performance even further.

This section concludes the serial performance tuning part. We will present some case studies through which we hope to show the tuning methodology. The optimizations implemented go beyond what can be expected from a compiler.

We will demonstrate that, with a modest effort, rather dramatic performance improvements can be realized. Several examples are based on real-life applications.

Introduction Into Parallelization

The basics of parallel processing are introduced and explained. This module serves as a basis for the remainder of the seminar. Several important concepts (speed-up, Amdahl's law, load balancing, cache coherency, parallel architectures, programming models, etc.) are introduced and discussed.

Introduction MPI on SUN

This is a short overview talk, included for completeness only. After a brief introduction into MPI, we will cover the main features of the MPI implementation on SUN., as included in the HPC Clustertools product.

Data Dependency Analysis

In order to safely parallelize a specific program construct, one has to know whether a data dependency exists or not.

With automatic parallelization, the compiler has to answer the question whether the candidate loop (nest) includes a data dependency.

If the user explicitly parallelizes a program fragment, he or she has to be able to determine whether there is a data dependency or not.

In this module we will study source code fragments that may, or may not have, a data dependency.

Automatic Shared Memory Parallelization

The SUN compilers support automatic parallelization. We will present and discuss the compiler options for this, and show several examples.

The Shared Memory Toolbox

This short section is meant as an introduction into the lengthy OpenMP module. Several important concepts needed when explicitly parallelizing an application will be introduced and discussed.

Explicit Shared Memory Parallelization with OpenMP

It might happen that the compiler lacks information to resolve a data dependency, or that the user would like to implement a rather high level of parallelization. In such a situation, one can pass on additional information to the compiler through so-called directives (pragmas in C).

Nowadays, the de-facto OpenMP standard provides a compact, but powerful, environment to implement such explicit parallelization.

We will cover the OpenMP model in quite some detail, both for Fortran and C. Examples to illustrate the most important concepts will be presented and discussed.

Tuning OpenMP Applications

Similar to serial optimization, parallel performance becomes relevant once the program is ready from a functional point of view and produces correct answers.

Therefore, after having used OpenMP to parallelize the application, it makes sense to look into performance.

We will discuss general inhibitors to scalable performance and also take a somewhat closer look at the OpenMP multi-tasking library as implemented on top of the Solaris Operating Environment.

OpenMP Case Studies

We will use examples to show how to efficiently parallelize a program using the OpenMP model¹.

In particular we will focus on how to apply OpenMP constructs to implement a higher level of parallelism.

¹We will not discuss how to design and write parallel algorithms. This is beyond the scope of the seminar.