In general, using -O3, IPO, and/or PGO, as well as the optimization reports described in the Fine-Tuning section of this document, can correct aliasing and improve memory utilization provides the best performance for Itanium-based systems.

### Fine-Tuning

Once you have identified performance hot spots, you may need to provide the compiler with more information to further optimize functions. The optimization and vectorization reports may show places where loops could not be optimized fully due to pointer aliasing or memory-access overlaps. For example, the loop

```c
for (int i = 0; i < 10; ++i) {
    for (int j = 0; j < 10; ++j) {
        //...</n```

Targets optimization for the Itanium processor. Detects parallel loops capable of being parallelized. Enables the parallelizer to generate multithreaded code for these loops. The option is turned ON with -ipo[n].

### Parallel Performance

The following options allow the compiler to help you parallelize your application for multi-processor, multi-core or processors with Hyper-Threading Technology:

1. *Add in intraprocedural optimization (IPO) and/or profile-guided optimization (PGO), and measure performance again to determine whether your application benefits from either of them.
2. *Fine-tune performance with processor-specific options to target systems based on Pentium 4 and Itanium processors, including those with Intel EM64T.
3. *Before you begin performance tuning, ensure that your application runs as intended with a base set of options or in debug mode (-Od and -Zi).
4. *Run your applications on multi-core, multi-threaded, or processors with Hyper-Threading Technology - capable systems using the Parallel Performance options.
Automatic Optimization Options

Before you begin performance tuning, ensure that your application runs as intended with a base set of optimization options. Use the following options to improve your code's performance.

**/O1 (No Optimization)**

No optimization. Useful during application development and debugging.

**/O1 (Optimizes for size)**

Optimizes options that tend to increase object size. Creates the smallest optimized code in most cases. On Linux systems with IA-32 processors only, there is no difference between /O1 and /O2.

This option is useful in many large server/database applications where memory paging due to larger code size is an issue.

**/O2 (Maximize speed)**

Default setting. Creates the fastest code in most cases, but may increase code size significantly over /O1. On Linux systems with IA-32 processors, /O1 and /O2 are equivalent.

**/O3 (Maximize optimization)**

n/a Equivalent to /O2 except that /O2 does not do library function packaging or /O2 (string pooling).

**/O3 (High-level optimizations)**

Same as /O2, plus loop transformations and data prefetching for improved memory usage efficiency. For the full benefit of /O3, use the /O3, /O4, /Qvec, and /Qvec options with Pentium III and Pentium 4 processors and subsequent IA-32 processors. This option is useful for a broad range of applications, particularly for the body-kernel-based code common in high-performance computing.

**/O1 -g**

The /O1 option maximizes speed across the entire program. For Intel® Xeon® based systems, use /O2, //O3, and –static.

For IA-32 and systems using Intel® EMMAP, use -O2, -O3, -static, and -static -ME.

**/Z2**

Generates debug information for use with the official debugger.