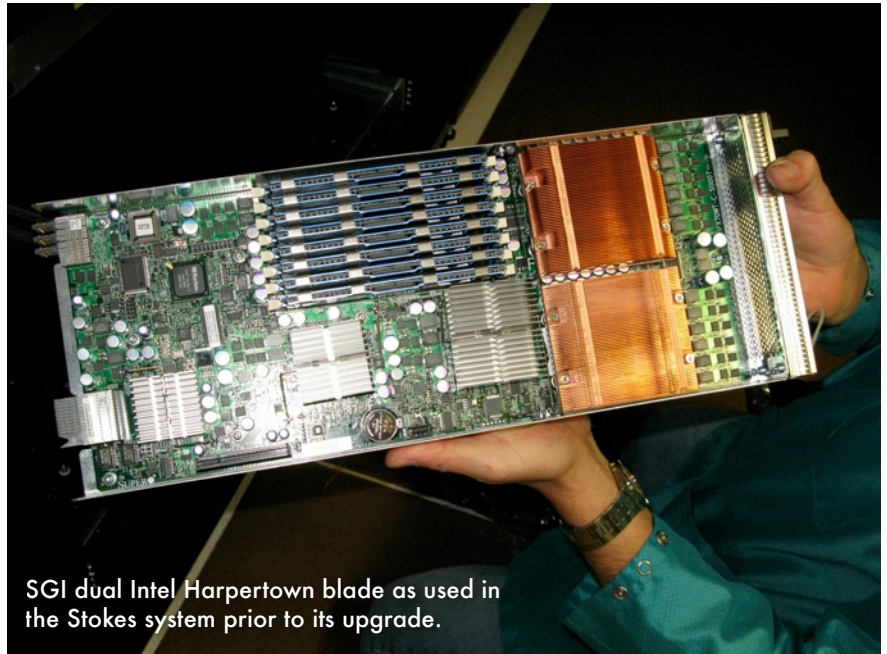


ICHEC

TECHNICAL
REPORT

Dr. Michael Browne
ICHEC Computational Scientist



SGI dual Intel Harpertown blade as used in the Stokes system prior to its upgrade.

Intel Compiler Suite:: Optimisation quick reference guide for C/C++ and Fortran

Introduction

This document is heavily based on Intel's very useful "Quick-Reference Guide to Optimization with Intel® Compilers version 11 For IA-32 processors, Intel® 64 processors and IA-64 processors". The intention is to highlight areas of this document that are most relevant to ICHEC's systems and users.

Intel's guide is itself very well put together and clear. Rather than try to improve on its advice the most useful sections are reproduced and many cases quoted verbatim. Advice specific to HPC and using the compiler suite within the ICHEC environment is included at certain points and features

Getting Started

The Intel compiler suite is installed on the ICHEC Stokes system and is available to all users of the system. Prior to using the Intel C/C++ or Fortran Compiler on Stokes it is necessary to load the appropriate modules. You can load the intel-cc module for working with C/C++ codes or the intel-fc module for Fortran codes. You can also load both

The Intel Compiler

Introduction	1
General optimisation options	2
Parallel performance	3
Processor specific optimisation	4
Floating point arithmetic	4
Fine tuning	6
Profile guided optimisation	7
Interprocedural optimisation	8

modules simultaneously if necessary. As significant updates are released by Intel they are installed on the system. By default the module points to the latest release. You can use an older versions by simply specifying its specific module. Note a report on using modules on ICHEC systems is also available. The intel-cc module is loaded as follows:

```
username@stokes1:~> module load intel-cc
```



ICHEC highly recommends you download the Intel Quick reference guide from:

<http://www.intel.com/cd/software/products/asm-na/eng/222300.htm>

More detailed documentation for the compilers is also available from Intel's website and you should feel free to contact the ICHEC helpdesk if you have specific queries.

General Optimisation Options

Argument	Comment
-O0	<p>No optimisation. Used during the early stages of application development and debugging. Use a higher setting when the application is working correctly.</p> <p>Using this option will result in quicker compile times which can be useful when doing intensive development. If stepping through code in a debugger such as DDT use this option to avoid the “erratic jumping” effect of code reordering.</p>
-O2	<p>Maximise speed. Default setting. Enables many optimisations, including vectorization. Creates faster code than -O1 in most cases.</p>
-O3	<p>Enables -O2 optimisations plus more aggressive loop and memory access optimisations, such as scalar replacement, loop unrolling, code replication to eliminate branches, loop blocking to allow more efficient use of cache and additional data prefetching. The -O3 option is particularly recommended for applications that have loops that do many floating-point calculations or process large data sets. These aggressive optimisations may occasionally slow down other types of applications compared to -O2.</p> <p>Production codes running on ICHEC systems should be compiled at with -O3 unless there is reason not too. However is not guaranteed to result in significant improvements over -O2.</p>
-g	<p>Generates debug information for use with any of the common platform debuggers. This option turns off -O2 and makes -O0 the default unless -O2 (or another O option) is specified.</p>
-debug full	<p>Produces full debugging information including symbol table information needed for full symbolic debugging of unoptimised code and global symbol information needed for linking. It produces the largest size object modules.</p> <p>If this option is used with optimised code, full symbol information will be generated including the local symbol table information, regardless of the optimisation level. This may result in minor performance degradation.</p> <p>If you download development releases of a package its make files will often be configured to include debugging information, which you may not require for optimised production runs.</p>

Parallel Performance

Argument	Comment
-openmp	<p>Instructs the parallelizer to generate multi-threaded code when OpenMP directives are present.</p> <p>Note a carefully written OpenMP code can support being compiled without enabling OpenMP and should produce a working serial code.</p>
-openmp-report {0 1 2}	<p>Controls the OpenMP parallelizer's diagnostic levels. The default level 1 reports loops, regions and sections successfully parallelized.</p>
-parallel	<p>Detects simply structured loops capable of being executed safely in parallel and automatically generates multi-threaded code for these loops.</p> <p>This mechanism is an alternative to traditional OpenMP coding and while it may suffice for simple situations it is unlikely to achieve the performance of hand written OpenMP code.</p>
-par-report {0 1 2}	<p>Controls the auto-parallelizer's diagnostic levels as follows:</p> <ul style="list-style-type: none"> 0 Displays no diagnostic information. 1 Indicates loops successfully parallelized (default). 2 Adds information on loops that were not parallelized. 3 Adds information about any proven or assumed dependencies inhibiting auto-parallelization (reasons for not parallelizing).
-par-threshold[n]	<p>Sets a threshold for the auto-parallelization of loops based on the probability of profitable execution of the loop in parallel, n=0 to 100. Default: n=100.</p> <ul style="list-style-type: none"> 0 Parallelize loops regardless of computation work volume. 100 Parallelize loops only if profitable parallel execution is almost certain. <p>This argument must be used in conjunction with -parallel.</p>
-par-schedule-keyword[=n]	<p>Specifies scheduling algorithm for parallel DO loops. n is the chunk size (in contiguous loop iterations). Possible keywords:</p> <ul style="list-style-type: none"> static allocates predetermined chunks to each thread in turn. dynamic allocates fixed chunks to threads dynamically at runtime guided divides up loops dynamically into variable chunks of decreasing size, with a minimum chunk size of n runtime scheduling algorithm and chunk size may be specified by the environment variable OMP_SCHEDULE at runtime.

Processor Specific Optimisation

Argument	Comment
<code>-x{sse4.2 sse4.1 ssse3 sse3 sse2 host}</code>	<p>Processor-specific targeting. Generates specialised code for the indicated Intel processor. The executable should only be run on the targeted or later Intel processors.</p> <p>host May optimise and generate any instructions that are supported by the compilation host.</p> <p>On the Stokes system the processors on the frontend nodes are older, slower and have fewer cores than those on the backend Westmere based nodes. This means that when working with the compiler on the frontend one needs to take care to exploit the more advanced instruction set of the compute nodes. Using the <code>-axsse4.2 -msse3</code> flags will create a binary that supports the SSE 4.2 instructions on the backend but with fallback instructions for the frontend nodes should you need to run it the resulting binary there.</p>

Floating Point Arithmetic Optimisation

Argument	Comment
<code>-[no-]fast-transcendentals</code>	<p>Enables the use of faster, slightly less accurate versions of math functions such as sin or exp. The default is -fast-transcendentals unless -fp-model precise or -fp-model strict is specified in which case the default becomes -no-fast-transcendentals.</p>
<code>-[no-]prec-div</code>	<p>Improves precision of floating point divides with a slight impact on speed.</p>
<code>-[no-]prec-sqrt</code>	<p>Improves precision of square root computations with a slight impact on speed.</p>



The options in this section allow you to exercise some control over the trade offs made by the compiler between speed and accuracy of floating point calculations.

Recommended Reading:

What Every Computer Scientist Should Know About Floating-Point Arithmetic
<http://dlc.sun.com/pdf/800-7895/800-7895.pdf>

Floating Point Arithmetic Optimisation

Argument	Comment
-fp-model <i>name</i>	<p>This method of controlling the consistency of floating point results by restricting certain optimisations is recommended in preference to the -mp and -mp1 switches which are deprecated. The possible values of name are:</p> <p>fast=[1 2] Allows more aggressive optimisations at a slight cost in accuracy or consistency. (fast=1 is the default).</p> <p>precise Enables only value-safe optimisations on floating point code.</p> <p>double/extended/source Intermediate results are computed in double, extended or source precision. Implies precise unless overridden. The double and extended options are not available for Intel Fortran.</p> <p>except Enforces floating point exception semantics.</p> <p>strict Strictest mode of operation, enables both the precise and except options and disables fma contractions.</p> <p>Recommendation: -fp-model precise -fp-model source is the recommended form for the majority of situations where enhanced floating point consistency and reproducibility are needed.</p>
-fp-speculation <i>mode</i>	<p>Enables floating-point speculations with one of the following modes:</p> <p>fast Speculate floating-point operations (default).</p> <p>off Disables speculation of floating-point operations.</p> <p>safe Do not speculate if this could expose a floating-point exception.</p> <p>strict This is the same as specifying off.</p>
-[no-]ftz	<p>Flushes denormal results to zero. This option flushes denormal results to zero when the application is in the gradual underflow mode. It may improve performance if the denormal values are not critical to your application's behaviour.</p> <p>This option only has an effect when the main program is being compiled. It sets the FTZ/DAZ mode for the process. The initial thread and any threads subsequently created by that process will operate in FTZ/DAZ mode.</p> <p>Every optimisation level except -O0, sets -ftz. If this option produces undesirable results of the numerical behaviour of your program, you can turn the FTZ/DAZ mode off by using -no-ftz in the command line while still benefiting from the -O3 optimisations. Note: -ftz is a performance option. Setting it does not guarantee that all denormals in a program are flushed to zero. Only denormals generated at run time are flushed to zero.</p>

Fine Tuning

Argument	Comment
-unroll[n]	Sets the maximum number of times to unroll loops. -unroll0 disables loop unrolling. The default is -unroll , which uses default heuristics.
-[no-]opt-prefetch	Enables or disables prefetch insertion.
-opt-block-factor=n	Specifies preferred loop blocking factor n , the number of loop iterations in a block, overriding default heuristics. Loop blocking is enabled at -O3 and is designed to increase the reuse of data in cache.
-opt-streaming-stores mode	Specifies whether streaming stores may be generated. Values for mode: always Encourages the compiler to generate streaming stores that bypass cache, assuming application is memory bound with little data reuse never Disables generation of streaming stores auto Default compiler heuristics for streaming store generation.
-[no]restrict	Enables or disables pointer disambiguation with the restrict keyword. Off by default. (C++ only).
-fno-alias	Assumes no aliasing in the program. Off by default.
-fno-fnalias	Assumes no aliasing within functions. Off by default.
-fargument-[no]alias	Implies function arguments may be aliased [are not aliased]. On by default. (C++ only).
-f[no-]exceptions	-f-exceptions , default for C++, enables exception handling table generation -fno-exceptions , default for C or Fortran, may result in smaller code. For C++, it causes exception specifications to be parsed but ignored. Any use of exception handling constructs (such as try blocks and throw statements) will produce an error if any function in the call chain has been compiled with -fno-exceptions .
-opt-report[n]	Generates an optimisation report directed to stderr. n specifies the level of detail, from 0 (no report) to 3 (maximum detail). Default is 2 .
-opt-report-help	Displays all possible values of name for -opt-report-phase above. No compilation is performed.

Fine Tuning

Argument	Comment
-opt-report-routine str	Generates reports only for functions or subroutines whose names contain the string str . By default, reports are generated for all functions and subroutines.
-[no-]opt-prefetch	Enables or disables prefetch insertion.
-vec-report [n]	Controls the vectorizer's diagnostic levels as follows: n = 0 no information n = 1 indicates vectorized loops (default) n = 2 indicates vectorized and non-vectorized loops n = 3 indicates vectorized loops and explains why other loops were not vectorized.

Profile-Guided Optimisation (PGO) Options

Argument	Comment
-prof-gen	Instruments a program for profile generation. Profile guided optimisation allows you to first compile your code with automatic instrumentation, then run this instrumented code which will produce log files. Then recompile this time without the instrumentation however the compiler will use the log files to help to optimise the final version.
-prof-use	Enables the use of profiling information during optimisation.
-prof-dir dir	Specifies a directory for the profiling output files, *.dyn and *.dpi.

Interprocedural Optimisation (IPO)

Argument	Comment
-ip	Single file interprocedural optimizations, including selective inlining, within the current source file.
-ipo[value]	<p>Permits inlining and other interprocedural optimisations among multiple source files. The optional value argument controls the maximum number of link-time compilations (or number of object files) spawned. Default for value is 0 (the compiler chooses).</p> <p>Caution: This option can in some cases significantly increase compile time and code size.</p>
-ipo-jobs[n]	Specifies the number of commands (jobs) to be executed simultaneously during the link phase of IPO. The default is 1 job.
-finline-functions -finline-level=2	<p>This option enables function inlining within the current source file at the compiler's discretion. This option is enabled by default at -O2 and -O3.</p> <p>Caution: For large files, this option may sometimes significantly increase compile time and code size. It can be disabled by -fno-inline-functions.</p>
-finline-factor=n	This option scales the total and maximum sizes of functions that can be inlined. The default value of n is 100 , i.e., 100% or a scale factor of one.



IPO includes function-inlining to reduce function call overhead and expose more optimisation opportunities. IPO can increase code size and compile time.. IPO is best used in conjunction with PGO to guide which functions to inline.