Using the Intel Fortran and C/C++ Compilers

R. Bader (LRZ)
H. Bast (Intel)
G. Hager (RRZE)
Intel® Itanium® Architecture: Explicit Parallelism

Original Source Code

Compiler

Itanium Architecture

Compiler Views

Wider Scope

Compile

Parallel Machine Code

Hardware

multiple functional units

More efficient use of execution resources

Increases Parallel Execution

March 2008

©2005-8 LRZ, RRZE, SGI and Intel
Central role of compiler for exploitation of EPIC features

- The **compiler** should play the key role in designing the plan of execution, and the architecture should provide the requisite support for it to do so successfully.
- The architecture should provide features that assist the **compiler** in exploiting ILP.
- The architecture should provide mechanisms to communicate the **compiler’s** plan of execution to the hardware.

---

Intel64 (x86) Architecture: Role of compiler

- Current leading x86 designs (Netburst, Core 2, AMD64) have (in contrast to IA64)
  - Out-of-order cores
  - Hardware prefetching units
  - Double precision floating-point SIMD capabilities (SSE2)

- Consequences
  - Less stringent software pipelining strategies required
  - Compiler does not have to be so smart about prefetching
    - In fact, SW prefetches are not used at all by default by the Intel compilers
  - Compiler must employ “vectorization” strategies to exploit SIMD data parallelism
Generic Features

- Compatibility to standards
  - ANSI C, C99 (partial)
  - Fortran95, Fortran2003 (partial)
- Compatibility to leading open-source tools
  - ICC vs. GCC, IDB vs. GDB
- OpenMP 2.5 support
  - Extension in C/C++: work queuing
- Vectorization support (SSE)
- Automatic parallelization
- Profile-guided optimization
- Multi-file interprocedural optimization
- Support of other Intel tools
  - `-tcheck` for Threading Tools
Generic Features (contd.)

- Support for gprof (\texttt{-p} switch)
- Inlined math intrinsics
- Code Coverage Tool
- Test Prioritization Tool
Using the Intel compilers on LRZ HPC systems

- Use the environment modules fortran and ccomp
  - default version (10.1) is loaded at login
  - switch to a different version – example:
    module unload fortran ccomp
    module load fortran/9.1 ccomp/9.1
  ➢ recommendation: do not mix versions between Fortran and C/C++

- Calling the compiler executables
  - Fortran compiler called via ifort
  - C compiler via icc
  - C++ compiler via icpc (do not use icc for C++)
Using the Intel compilers on RRZE HPC systems

- Use the environment modules \texttt{intel(64)} (C/C++/Fortran) or \texttt{intel(64)-c} and \texttt{intel(64)-f}, respectively
  - No default version loaded at login
  - Use plain \texttt{intel64} module (default version) for a start
  - Specific versions can be used with e.g. \texttt{intel64-c/10.1.012}

- Calling the compiler executables
  - Fortran compiler called via \texttt{ifort}
  - C compiler via \texttt{icc}
  - C++ compiler via \texttt{icpc} (do not use \texttt{icc} for C++)
Compiler documentation

- Extensive documentation for all compilers available from Intel
- `-help` compiler option prints extensive list of options
- Also installed on LRZ web server
  - Start from the URL
    http://www.lrz-muenchen.de/services/compute/linux-cluster/doc/
    and select whatever you’re interested in
  - some parts require Java/Javascript to be activated in the browser
- **Password-protected documents:**
  - obtain user name and password by
    - logging in to any LRZ HPC system you are validated on
    - issuing the command `get_manuals_passwd`
### Some switches valid for all compilers

<table>
<thead>
<tr>
<th>Switch Description</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable optimization (on IA64: may be very slow)</td>
<td>-00</td>
</tr>
<tr>
<td>Optimize for speed (no code size increase), no SWP on IA64: good for “branchy” code, flat profile code</td>
<td>-01</td>
</tr>
<tr>
<td>Optimize for speed (default), includes SWP ( “HLO-light” since 9.0 )</td>
<td>-02</td>
</tr>
<tr>
<td><strong>High-level optimizer</strong>, incl. prefetch (IA64), unroll, flush-to-zero</td>
<td>-03</td>
</tr>
<tr>
<td>Aggressive optimizations (same as <code>-O3 -ipo -static</code> + vectorization on x86)</td>
<td>-fast</td>
</tr>
<tr>
<td><strong>Note:</strong> cannot use for MPT programs on Altix systems</td>
<td></td>
</tr>
<tr>
<td>Create symbols for debugging</td>
<td>-g</td>
</tr>
<tr>
<td>Generate assembly files</td>
<td>-S</td>
</tr>
<tr>
<td>Assume no aliasing</td>
<td>-fno-fnalias -fno-alias</td>
</tr>
<tr>
<td>OpenMP 2.5 support (enables directives and OpenMP RT calls)</td>
<td>-openmp</td>
</tr>
<tr>
<td><strong>Note:</strong> can use <code>-openmp-stubs</code> for dummy stub routines</td>
<td></td>
</tr>
<tr>
<td>Automatic parallelization (shared memory multi-threading)</td>
<td>-parallel</td>
</tr>
</tbody>
</table>
[Proprietary] compiler directives

- **Fortran compiler**
  - `!DIR$ directive[(...) clause]`
  - can replace DIR by DEC

- **C/C++ compiler**
  - `#pragma directive[(...) clause]`
  - includes non-proprietary pragmas

- **One example of importance in this context**
  - `!DIR$ OPTIMIZE[ :n]`
  - can appear at top of (contained) procedure or module
  - specifies optimization level (n=0,1,2,3)
  - overrides compiler switch if the latter specifies higher optimization level

- **For C/C++, the corresponding pragma is**
  - `#pragma optimization_level n`
  - applies to a single function
High-level Optimizer Overview

- Data dependence analysis
  - Array subscript analysis
- Eliminating memory operations
  - Scalar replacement, unroll-and-jam, register blocking
- Cache optimizations
  - Loop interchange, fusion, distribution, blocking, data layout
- Overlapping memory latency
  - Data prefetch (off by default on x86)
- Instruction-level parallelism
  - Unrolling, interchange, distribution
- Bandwidth optimization
  - Load-pair, array contraction
  - SSE on x86: SIMD arithmetic, SIMD LD/ST, non-temporal stores
Phase ordering inside HLO

This ordering, valid for 9.x, may have changed in more recent compiler releases
Compiler switches and functionality discussed in other talks

- Optimization reporting switches
  → optimization talks

- Compiler directives
  → optimization talks for serial stuff
  → OpenMP talk

- KMP_* environment variables
  → OpenMP talk

- Profile guided optimization
  → optimization talk

- Data format conversion (big/little endian)
  → I/O talk
IPO: Interprocedural Optimization
Extending optimizations across file boundaries

**Without IPO**

- Compile & Optimize → file1.c
- Compile & Optimize → file2.c
- Compile & Optimize → file3.c
- Compile & Optimize → file4.c

**With IPO**

- Compile & Optimize

  - ip
  - ipo

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-ip</td>
<td>Only between modules of one source file</td>
</tr>
<tr>
<td>-ipo</td>
<td>Modules of multiple files/whole application</td>
</tr>
</tbody>
</table>

©2005-8 LRZ, RRZE, SGI and Intel
March 2008
Further remarks on IPO

- **Inline function expansion**
  - Calls, jumps, branches and loops

- **Interprocedural constant propagation**
  - Arguments, global variables and return values

- **Monitoring module-level static variables**
  - Further optimizations and loop invariant code

- **Dead code elimination**
  - Code size

- **Propagation of function characteristics**
  - Call deletion and call movement

- **Multi-file optimization**
  - Same aspects as -ip but across multiple files

---

- **-ipo option:**
  - Creates .o file in intermediate representation
  - Actual optimization done at linkage time
  - Compiler splits objects if complete program gets too large
  - Can suppress this by specifying `-ipo[n]` or `-ipo-separate`

- **Inlining**
  - Consult compiler documentation on how to exert more control over the inlining process
Memory Reference Disambiguation
options/directives related to aliasing

- `[no-]alias-args` (C/C++) | `-assume dummy-aliases` (Fortran)
  - Assume (dummy) arguments may be aliased
  - By default this is not assumed

- `[no-]ansi-alias` | default is `-ansi-alias`
  - Assume program adheres to C/Fortran 95 aliasing rules
  - More aggressive optimization possible
  - Incorrect code generated if assumption violated

- `-fno-alias`: No aliasing in whole program
- `-fno-fnalias`: No aliasing within single units
- `-restrict` (C99): enable restrict attribute
  - Selective pointer disambiguation

- `-safe_cray_ptr` (Fortran): No aliasing introduced by Cray pointers

Related: Switch `-ipo` and directive `IVDEP`

default is to assume aliasing
C++ Compiler for Linux

- Switch-compatible to GNU GCC for all basic options
- Object file interoperability ICC and GCC
  - "C and C" binary mix
    - No restrictions for whatever combination
  - "C++ and C" binary mix
    - No restrictions for whatever combination
  - "C++ and C++" binary mix
    - No restrictions since 8.0
    - Support for GCC STL (libstdC++ library) since 8.0
- Support of most GCC C/C++ language extensions
- Uses GCC run time libraries by default
C++ Compiler for Linux
Some newer features

- C99 support
  - Many features but not all since 8.1
- New pragmas
  - e.g. #pragma member alignment
- -Wcheck
  - Diagnostics for problems that could manifest as runtime problems
    - Variable “x” is used before its value is set
    - Conversion from “int” to “short” may lose significant bits
- Variable length arrays with arbitrary types
- Optimized copy constructors
- Version 10.x improvements:
  - mudflap pointer checking
  - static verifier for improved identification of coding errors
  - semantic checking of template definitions
    - g++ performs this only at instantiation
  - rapid type analysis for improved virtual function performance
**Motivation:** Large Project typically contains a common subset of includes

**What:** “Memory Dump” of the compiler after header file processing.

**Goal:** Create the maximum subset of common header files

- **`-pch` option** creates and uses PCH files (extension `.pchi`)
  - can also specify additional options for specific naming and targeting PCH files – see compiler docs for details

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Compile Time Improvement by PCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Povray (-O2)</td>
<td>39%</td>
</tr>
<tr>
<td>Eon (-O2)</td>
<td>29%</td>
</tr>
</tbody>
</table>
Intel Fortran Compiler
Fortran Compiler for Linux [Windows]

- **Full compatibility to ANSI Fortran 77, Fortran90, Fortran95**
  - Switches support language conformance

- **Many extensions like**
  - Options on ATTRIBUTES directives
    - DECORATE, DEFAULT
  - REAL(16), COMPLEX(16)
    - software emulation
    - performance impact
    - no 10 byte floating point kind supported (as in C/C++)

- **Some extensions of Fortran2003 already implemented**

- **Successor product to Compaq Visual Fortran 6.6**
  - Intel Visual Fortran on Windows
  - Windows specific features not discussed here

- **Releases available on LRZ/RRZE HPC systems:**
  - default production: **10.1.13/9.1.45**
  - older versions: **8.1, 9.1, 10.0**
    - not all features described here may be available
### Language Semantics: Controls and checks

<table>
<thead>
<tr>
<th>Control</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turn warnings to errors</td>
<td>-warn errors</td>
</tr>
<tr>
<td>Enforce strict typing (IMPLICIT NONE by default). Can also use –implicitnone or –u</td>
<td>-warn declarations</td>
</tr>
<tr>
<td>issue <strong>warnings</strong> for source syntax that does not conform to the argument standard version (default –stand none)</td>
<td>-stand [f90</td>
</tr>
<tr>
<td>for Fortran 77 legacy code: Generate interfaces (separate files!) and check correct usage</td>
<td>-gen-interfaces -warn interfaces</td>
</tr>
<tr>
<td>Enforce fixed / free format (default by file extension .f / .f90)</td>
<td>-fixed / -free</td>
</tr>
<tr>
<td>For fixed format source: extend max. line length (72)</td>
<td>-extend_source 132</td>
</tr>
<tr>
<td>run preprocessor (default for extension .F90) e.g., for use of <strong>FILE</strong>, <strong>LINE</strong> macros, OpenMP</td>
<td>-fpp</td>
</tr>
<tr>
<td>check program syntax only; do not generate code</td>
<td>-syntax</td>
</tr>
</tbody>
</table>
Run Time settings

<table>
<thead>
<tr>
<th>Setting</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add debug symbols</td>
<td><code>-g</code></td>
</tr>
<tr>
<td>Generate profiling information for use with <code>gprof(1)</code></td>
<td><code>-p</code></td>
</tr>
<tr>
<td>Generate stack traceback upon crash</td>
<td><code>-traceback</code></td>
</tr>
<tr>
<td>Array bounds checking at run time</td>
<td><code>-check bounds</code></td>
</tr>
<tr>
<td>Check match of data vs. format descriptor at run time</td>
<td><code>-check format</code></td>
</tr>
<tr>
<td>Warn against too large data items formatted I/O</td>
<td><code>-check output_conversion</code></td>
</tr>
<tr>
<td>RT Error on read references to uninitialized variables</td>
<td><code>-check uninit</code></td>
</tr>
<tr>
<td><strong>Note:</strong> Module variables not included.</td>
<td></td>
</tr>
<tr>
<td>RT Error on spurious references to uninitialized pointers / unallocated allocatables</td>
<td><code>-check pointer</code></td>
</tr>
<tr>
<td>Inform about copy-in/copy-out temporaries</td>
<td><code>-check arg_temp_created</code></td>
</tr>
</tbody>
</table>
## Module and library handling

| Specifies location for writing `.mod` files (default is working directory). Also added to search path | `-module <path>` |
| Add to search path for `.mod` files | `-I<path>` |
| Link dynamically (default) / statically | `-dynamic / -static` |
| Notes: | |
| `-static` will not work for MPT programs on Altix | |
| use `-Bstatic` for specific libraries (if available) | |
| Linkage option: Add to search path for subsequently specified libraries | `-I<path>` |
| Link Intel-provided libraries dynamically. Default is `-i-dynamic` | `-i-static / -i-dynamic` |

### Mixed Fortran-C++ applications

| Mixed Fortran/C++ applications require C++ libraries in addition | `-cxxlib` |
| Link Intel libcxa C++ library statically (default) or dynamically | `-static-libcxa` / `-shared-libcxa` |
| Propagate C++ exceptions through intermediary Fortran call | `-fexceptions` |
Fortran 2003 features available (10.1 compiler)

- Allocatable extensions (TR15581)
  - dummy arguments
  - function results
  - type components
  - `move_alloc` intrinsic

- IMPORT
  - access host information from interfaces

- Source form extensions
  - variable name length 63
  - 255 continuation lines
  - character set extension
  - `[ ]` form of array constructor syntax
  - parameter in COMPLEX constant
  - BOZ constants
Fortran 2003 features available (cont’d)

- **object attributes**
  - `protected` attribute
  - `intent` for pointers
  - `volatile` attribute, statement and scoping
  - `value` attribute

- **rename user-defined operators in `use` statement**

- **I/O extensions**
  - `flush` statement
  - asynchronous I/O (effective for unformatted I/O)
  - stream I/O

- **access to operating environment**
  - mostly available
  - intrinsic module however not yet available
Fortran 2003 features available (cont’d)

- C interoperability
  - `bind(C)`
  - nearly complete implementation of iso_c_binding intrinsic module

- The really interesting parts
  - type extension, polymorphism, TBPs, abstract types and interfaces, destructors, procedure pointers
  - derived-type I/O, recursive I/O
  - parameterized types
  - further extensions to dynamic memory management
  - extensions to pointer assignment
  - IEEE exception handling
  - submodules

are still missing
## Fortran Compiler: Intel extension routines/intrinsics

Portability routines e.g., GETENV, GETPID, SIGNAL, DTIME, ETIME, ...

| USE IFPORT |

POSIX Routines like PXFWAIT (wait for child process)

| USE IFPOSIX |

Only Windows compiler: Routines to access Microsoft Windows environment

| USE IFQWIN |
| USE IFLOGM |
| USE IFCOM |
| USE IFAUTO |
| ... |

Misc routines like `FOR_GET_FPE` and `FOR_SET_FPE` to get and set the floating point status registers

| USE IFCORE |
Some special Features
Example of code coverage summary for a project. The workload applied in this test exercised 34 of 143 blocks, representing 5 of 19 functions in 2 of 3 modules. In the file, SAMPLE.C, 4 of 5 functions were exercised.

Clicking on SAMPLE.C produces a listing that highlights the code that was exercised. In this example, the pink-highlighted code was never exercised, the yellow was run but not exercised by any of the tests set up by the developer and the beige was partially covered.
How to generate code coverage output

- A multi-step process:
  1. Compile with the `-prof-genx` option
     - also generates `*.spi` file
  2. Run instrumented application
     - possibly more than once
     - each run generates a `*.dyn` file
  3. Run `profmerge -prof_dir .` to generate a dpi file
     - `pgopti.dpi` by default

Run codecov with appropriate suboptions
- generates HTML file(s) for viewing results
- for example:
  `codecov -prj mystuff -spi pgopti.spi -dpi pgopti.dpi`
“Black Belt” options for Intel compilers

- Intel memo documenting numerous compiler options not listed in official product documentation; useful for
  - Compiler debugging
    - e.g., finding the application module where the compiler optimizer fails
  - Benchmarking/tuning in very special situation
- **Warning: These options are**
  - Not supported
  - Not part of the usual regression testing
  - Are not intended to become “supported” options
  - Can be non-functional in future releases or have a different function in the future
- The memo is no longer Intel-confidential information but should not be made public
  - only selected options presented here
  - some of these have been converted to supported options in the most recent compiler release
Black Belt: Some selected options

- `mP2OPT_hlo_prefetch=TRUE/FALSE`
  - Enable/disable data pre-fetching
- `mP2OPT_hlo_distribution=TRUE/FALSE`
  - Enable/disable loop distribution
- `mP3OPT_ecg_mm_fp_ld_latency=<value>`
  - Default=11; cache miss latency
- `mP3OPT_ecg_non_ieee_div=TRUE/FALSE`
  - Move to a faster, non-IEEE division
  - see also `-no-prec-div` (not available on IA-64)
- `mP2OPT_opt_threshold_enforce=TRUE/FALSE`
  - Consider/ignore internal limits for optimizer; (do not) give up on optimization when certain threshold for resource usage is reached
  - can also use `-override-limits`
Thank you for your attention