High Performance Programming with IBM XL Compilers and Libraries

SPXXL/ScicomP-17 2011 Summer Workshop
IBM Rational Disclaimer

© Copyright IBM Corporation 2011. All rights reserved. The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in these materials may change at any time at IBM’s sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. IBM, the IBM logo, Rational, the Rational logo, Telelogic, the Telelogic logo, and other IBM products and services are trademarks of the International Business Machines Corporation, in the United States, other countries or both. Other company, product, or service names may be trademarks or service marks of others.
Agenda

▪ Overview of XL Compiler Family
▪ Major Features of XL C/C++ V11.1 and XL Fortran V13.1
▪ Migration from GNU to XL compilers
▪ XML Compiler Transformation Reports
▪ Compiler Optimizations for Performance
  – Profile Directed Feedback Optimization
  – SIMDization and Vectorization
  – Loop Transformations
  – Data Prefetch
  – Data Reorganization
  – Inlining
  – Parallelization
Overview of XL Compiler Family

- Similar compilation technology used to implement C, C++ and Fortran Compilers
- Supports AIX, Linux on Power, zOS (C/C++ only), BlueGene, Cell
- Advanced optimization capabilities
  - Exploitation and tuning for latest hardware implementations
  - Aggressive loop analysis and transformations (unimodular and polyhedral framework)
  - Whole program optimization
  - SIMD code generation and Vectorization exploitation
  - Parallelization (automatic and user-driven through OpenMP)
  - Profile-driven optimization
Major Features of XLC11.1 / XLF13.1

- POWER7 support and exploitation

- Language standard conformance
  - Full Fortran 2003
  - Full OpenMP 3.0
  - Additional C++0X features

- Optimization enhancements
  - Automatic parallelization
  - Inlining
  - Loop analysis and transformations
  - Delinquent load driven optimizations
  - Profile-directed feedback

- Productivity enhancement
  - XML compiler transformation reports

- Other features
  - Fine-grained strict control
  - ProPolice
  - func_trace
  - Other options and directives
HPC Performance Tuning with XL Compilers

Profiling for hot spot detection:
- Compiler instrumentation: –qpdf1=level={1,2} / pdf2
- -pg for gprof/xprofiler; -qlist for tprof
- User-provided profile functions: -qfunctrace

SIMDization:
- Automatic SIMDization: –O3 or above with –qsimd
- User explicit SIMD program: -qaltivec

Loop transformations:
- Loop transformations: –O3 or above

Parallelization:
- User explicit parallelization only: -qsmp=omp
- Auto parallelization: -qsmp (-qsmp=auto)
- Polyhedral framework: -qsmp with -qhot=level=2

Whole program optimizations:
- -O4 or –O5 for inter-procedural optimization: inlining, code partition, data reorganization

XML Transformation Reports
- -qlistfmt=xml=all
Migration from GCC to IBM XL Compilers

- **Compatibility**
  - Source level
    - Supports many gcc/g++ language extensions and annotations
  - Binary level
    - Link objects from gcc/g++ and XL C/C++

- **gxlc and gxlc++ utilities for compile option mapping**
  - Controlled by the gxlc.cfg configuration file for option mappings from GCC to XL C/C++
  - Modify the contents of the gxlc.cfg to meet your specific compilation requirements
gxlc, gxlc++, gxIC

gxlc
gxlc++  -v  -Wx, <XL compiler options>  <gcc, g++ options>  filename
gxIC

Creating customized configuration file
(XLC_USR_CONFIG environment variable to specify the location of your defined configuration file)

gxlc.cfg format:

abcd  "gcc_or_g++_option"  "xlc_or_xlc++_option"

e.g.
nnnn  "-ansi"  "-qlanglvl=extc89 -qnokeyword=inline -qnokeyword=typeof -qnokeyword=asm -qnocpluscmt -D__STRICT_ANSI__"
nnnn  "-B**"  "-B**"
nnnn  "-C"  "-C"
nnnn  "-c"  "-c"
nnnn  "-dM"  "-qshowmacros"
nnnn  "-D**"  "-D**"
*  "E"  "E"
XML Compiler Transformation Reports

- **Generate compilation reports consumable by other tools**
  - Enable better visualization and analysis of compiler information
  - Help users do manual performance tuning
  - Help automatic performance tuning through performance tool integration

- **Unified report from all compiler subcomponents and analysis**
  - Compiler options
  - Pseudo-sources
  - Compiler transformations, including missed opportunities

- **Consistent support among Fortran, C/C++**

- **Controlled under option**
  - `-qlistfmt=xml=inlines` generates inlining information
  - `-qlistfmt=xml=transform` generates loop transformation information
  - `-qlistfmt=xml=data` generates data reorganization information
  - `-qlistfmt=xml=pdf` generates dynamic profiling information
  - `-qlistfmt=xml=all` turns on all optimization content
  - `-qlistfmt=xml=none` turns off all optimization content
Compiler Transformation Report Contents

- **Program characteristics**
  - Compiler version
  - Date of compilation
  - Source file table
  - Function Table
  - Loop table (line number, nest level, iteration count, loop attributes)
  - Transformation table (line number, transformation description)
  - Pseudocode

- **Transformations at both high and low-level optimizations**
  - Intra-procedural transformations
    - Loop transformations
    - Data prefetch
    - Vectorization and SIMDization
    - Parallelization
    - Instruction scheduling
  - Inter-procedural transformations
    - Inlining
    - Data reorganization

- **Profiling information**
  - Basic block counters
  - Call counters
  - Cache miss counters
XL Compiler Assisted Performance Analysis and Tuning

- **Compiler Optimizations for Performance**
  - Compiler static analysis and optimization
  - User guided optimizations through compiler options and directives
  - Automatic compiler optimizations through profile directed feedback

- **XL Compiler and Tooling Integration**
  - Compiler feedback view in PTP
  - Compiler transformation reports and HPCS toolkit help detect bottlenecks and identify solutions
Compiler Feedback View
Multiple-pass Dynamic Profiling Infrastructure

SOURCE CODE

COMPILE AND LINK WITH –qpdf1
Static analysis
Profile based refinement

INSTRUMENTED APPLICATION

SAMPLE INPUTS

PROFILE DATA

COMPILE AND LINK WITH –qpdf2
Profile directed optimizations

OPTIMIZED APPLICATION

Hardware and software constraints
Multiple sample runs for different hardware performance events
Profile based instrumentation refinement
Basic Block and Call Counter Information

Step 1: compile the application with –qpdf1 to generate an instrumented executable
Step 2: run the executable with typical input data set to gather profiling information
Step 3: re-compile the application with –qpdf2 –qlistfmt=xml=all to generate the optimized executable and XML compiler transformation report

Basic Block Counter Information

<table>
<thead>
<tr>
<th>Region #</th>
<th>Region Execution Count</th>
<th>Block Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>81 / 81</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Region Execution Count</th>
<th>Block Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>81 / 81</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block Index</th>
<th>Block Execution Count</th>
<th>Start Line #</th>
<th>End Line #</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>33</td>
<td>34</td>
</tr>
</tbody>
</table>

Call Counter Information

<table>
<thead>
<tr>
<th>Region #</th>
<th>Region Execution Count</th>
<th>Call Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>32 / 49</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Region Execution Count</th>
<th>Call Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32 / 49</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Call Name</th>
<th>Call Execution Count</th>
<th>Line #</th>
</tr>
</thead>
<tbody>
<tr>
<td>smtctl_</td>
<td>0</td>
<td>79</td>
</tr>
<tr>
<td>is_smt_on_</td>
<td>1</td>
<td>55</td>
</tr>
<tr>
<td>jbind_</td>
<td>1</td>
<td>109</td>
</tr>
<tr>
<td>jbind_</td>
<td>0</td>
<td>108</td>
</tr>
<tr>
<td>aff_</td>
<td>1</td>
<td>38</td>
</tr>
</tbody>
</table>
# Cache Miss Information

<table>
<thead>
<tr>
<th>Memory Reference</th>
<th>Region #</th>
<th>Line #</th>
<th>Cache Level</th>
<th>Miss Count</th>
<th>Miss Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>((double *)((char *)d-zfaci5%addr + max((long long) klon-&gt;klon.rns0.,0ll) * -8ll - 8ll))- &gt;zfaci5[].rns50.[(long long) .ktdia-&gt;ktdia.rns6. + @CIV11][(long long) .kidia-&gt;kidia.rns4. + @CIV7]</td>
<td>3</td>
<td>408</td>
<td>2</td>
<td>17446</td>
<td>24</td>
</tr>
<tr>
<td>((double *)((char *)d-ztp25%addr + max((long long) klon-&gt;klon.rns0.,0ll) * -8ll - 8ll))- &gt;ztp25[].rns26.[(long long) .ktdia-&gt;ktdia.rns6. + @CIV11][(long long) .kidia-&gt;kidia.rns4. + @CIV7]</td>
<td>3</td>
<td>412</td>
<td>2</td>
<td>9999</td>
<td>14</td>
</tr>
<tr>
<td>((double *)((char *)d-zdqsdtemp5%addr + max((long long) klon-&gt;klon.rns0.,0ll) * -8ll - 8ll))- &gt;zdqsdtemp5[].rns53.[(long long) .ktdia-&gt;ktdia.rns6. + @CIV11][(long long) .kidia-&gt;kidia.rns4. + @CIV7]</td>
<td>3</td>
<td>413</td>
<td>2</td>
<td>9974</td>
<td>14</td>
</tr>
<tr>
<td>((double *)((char *)d-zcld5%addr + max((long long) klon-&gt;klon.rns0.,0ll) * -8ll - 8ll))- &gt;zcld5[].rns72.[(long long) .ktdia-&gt;ktdia.rns6. + @CIV11][(long long) .kidia-&gt;kidia.rns4. + @CIV7]</td>
<td>3</td>
<td>52</td>
<td>2</td>
<td>10336</td>
<td>6</td>
</tr>
<tr>
<td>((double *)((char *)d-zdr15%addr + max((long long) klon-&gt;klon.rns0.,0ll) * -8ll - 8ll))- &gt;zdr15[].rns41.[(long long) .ktdia-&gt;ktdia.rns6. + @CIV11][(long long) .kidia-&gt;kidia.rns4. + @CIV7]</td>
<td>3</td>
<td>557</td>
<td>2</td>
<td>11553</td>
<td>16</td>
</tr>
</tbody>
</table>

- **Delinquent load**: Source code location
- **Cache miss**: Source code location
## Loop information

### Loop Table

<table>
<thead>
<tr>
<th>Loop Index</th>
<th>Start Line #</th>
<th>End Line #</th>
<th>Parent Loop Index</th>
<th>Nest Level</th>
<th>Minimum Cost</th>
<th>Maximum Cost</th>
<th>Iteration Count</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>203</td>
<td></td>
<td></td>
<td>1</td>
<td>19630</td>
<td>19630</td>
<td>75 (array)</td>
<td>well behaved, bump normalized, lower bound normalized</td>
</tr>
<tr>
<td>2</td>
<td>188</td>
<td></td>
<td></td>
<td>1</td>
<td>20413</td>
<td>20413</td>
<td>149 (array)</td>
<td>perfect nest, well behaved, bump normalized, guarded, lower bound normalized</td>
</tr>
<tr>
<td>3</td>
<td>141</td>
<td></td>
<td></td>
<td>1</td>
<td>13300</td>
<td>13300</td>
<td>100 (default)</td>
<td>perfect nest, well behaved, bump normalized, guarded, lower bound normalized</td>
</tr>
<tr>
<td>4</td>
<td>203</td>
<td></td>
<td></td>
<td>1</td>
<td>19630</td>
<td>19630</td>
<td>5 (PDF)</td>
<td>residual, well behaved, bump normalized, guarded, lower bound normalized</td>
</tr>
</tbody>
</table>

*Loop iteration count based on static analysis or dynamic profiling*
# Loop Transformation Reports

<table>
<thead>
<tr>
<th>Seq #</th>
<th>Type</th>
<th>Phase</th>
<th>Region #</th>
<th>Line #</th>
<th>Loop Index</th>
<th>Description</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>LoopVector (success)</td>
<td>High Level Optimizer</td>
<td>4</td>
<td>217</td>
<td>1</td>
<td>Loop vectorization was performed.</td>
<td>not available</td>
</tr>
<tr>
<td>3</td>
<td>LoopFusion (success)</td>
<td>High Level Optimizer</td>
<td>4</td>
<td>108</td>
<td>3</td>
<td>Loops were fused.</td>
<td>Loop Line Number: 108 • Loop Line Number: 206</td>
</tr>
<tr>
<td>4</td>
<td>LoopVector Version (success)</td>
<td>High Level Optimizer</td>
<td>4</td>
<td>108</td>
<td>3</td>
<td>Vector versioning was performed.</td>
<td>not available</td>
</tr>
<tr>
<td>20</td>
<td>ModuloSchedule (success)</td>
<td>Low Level Optimizer</td>
<td>12</td>
<td>3499</td>
<td>26</td>
<td>Loop was modulo scheduled.</td>
<td>Initiation Interval: 12</td>
</tr>
</tbody>
</table>
Performance Tuning with Compiler Transformation Reports

```
file.c

foo (float *p, 
     float *q, 
     float *r, 
     int n) {
    for (int i=0; i< n; i++) {
        p[i] = p[i] + q[i]*r[i];
    }
}
```

```
file.xml
Loop cannot be automatically parallelized. 
A dependency is carried by variable aliasing
```

```
file.xml
Loop was automatically parallelized
Loop was modulo scheduled
```

```
file.c

foo (float * restrict p, 
     float * restrict q, 
     float * restrict r, 
     int n) {
    for (int i=0; i< n; i++) {
        p[i] = p[i] + q[i]*r[i];
    }
}
```

Original source file

Tuning

modified source file
Explicit SIMD programming for POWER7
Enabled under -qaltivec

- Successor to altivec programming extensions on POWER6/PPC970
  - Altivec data types
    - vector char
    - vector short
    - vector pixel
    - vector int
    - vector float
  - VSX Altivec extensions
    - vector double
    - vector long long

- Altivec built-in functions extended to new data types
  - vec_add(vector double, vector double),
  - vec_sub(vector long long, vector long long),

- New vector operations: vec_mul, vec_div, ...

- Unaligned load and store operations
  - Altivec truncating loads/stores still available: vec_ld, vec_st
  - New non-truncating loads/stores: vec_xld2, vec_xstd2
Automatic SIMDization

- **Automatic SIMDization for VMX and VSX**
  - Supports data types of INTEGER, UNSIGNED, REAL and COMPLEX

- **Features:**
  - Basic block level SIMDization
  - Loop level aggregation
  - Data conversion, reduction
  - Loop with limited control flow
  - Automatic SIMDization with -qstrict (VSX) and -qnostrict
  - Support of unaligned vector memory accesses (VSX)
  - Automatic SIMDization enabled at -O3 -qsimd
SIMDization Tuning

**Transformation report**

**Loop** was SIMD vectorized

**It is not profitable to vectorize**

- Use __attribute__((aligned(n))) to set data alignment
- Use __alignx(16, a) to indicate the data alignment to the compiler
- Use -qassert=refalign if all references are naturally aligned
- Use array references instead of pointers where possible

**data dependence prevents SIMD vectorization**

- Use fewer pointers when possible
- Use #pragma independent if it has no loop carried dependency
- Use #pragma disjoint (*a, *b) if a and b are disjoint
- Use restrict keyword or compiler option –qrestrict

**memory accesses have non-vectorizable alignment.**

- Use __attribute__((aligned(n))) to set data alignment
- Use __alignx(16, a) to indicate the data alignment to the compiler
- Use -qassert=refalign if all references are naturally aligned
- Use array references instead of pointers where possible

**User actions**

- Use #pragma simd_level(10) to force the compiler to do SIMDization

It is not profitable to vectorize
SIMDization Tuning

Transformation report

- Loop structure prevents SIMD vectorization
- Memory accesses have non-vectorizable strides
- Either operation or data type is not suitable for SIMD vectorization.

User actions

- Convert while-loops into do-loops when possible
- Limited use of control flow in a loop
- Use MIN, MAX instead of if-then-else
- Eliminate function calls in a loop through inlining

- Loop interchange for stride-one accesses, when possible
- Data layout reshape for stride-one accesses
- Higher optimization to propagate compile known stride information
- Stride versioning

- Do statement splitting and loop splitting
MASS enhancements and Auto-vectorization

- MASS enhancements for POWER7
  - POWER7 vector MASS library (libmassvp7.a)
    - Internally exploit VSX instructions
      - SP: average speedup of 1.99 vs Power5 MASSV
      - DP: average speedup of 1.27 vs Power5 MASSV
  - POWER7 SIMD MASS library (libmass_simdp7.a)
    - Tuned math routines operating on vector data types
    - Over 35 frequently used mathematical functions
    - Both simple and double precision
    - To be used in conjunction with explicit SIMD programming

- Auto-vectorization at optimization level –O3 or above
- -qstrict=vectorprecision to maintain precision over all loop iterations

```
for (i=0;i<n;i++) {
    b[i]=sqrt(a[i]);
}
```

Transformation report

Loop vectorization was performed.

```
__vsqrt_P7(b,a,n);
```
Software-controlled data prefetching for POWER7

- Software control over POWER7 prefetch engine, supporting up to 12 data streams

- Fine grained software controlled data prefetch including stream type, stream length, stream stride, prefetch depth at optimization level -O3 –qhot or above
  - More aggressive exploitation under option – qprefetch=aggressive

- Global analysis for coarse grained prefetch engine control at optimization level -O5
Built-in functions for POWER7 data prefetching and cache control

- Transient cache line touch
  ```c
  void __dcbtt(void *address);
  void __dcbtstt (void * address);
  ```

- Partial cache line touch
  ```c
  void __partial_dcbt(void *address);
  ```

- Stride-N stream prefetch
  ```c
  void __protected_stream_stride(offset, stride, stream_ID);
  ```

- Transient stream prefetch:
  ```c
  void __transient_protected_stream_count_depth(unit_count, depth, stream_ID)
  void __transient_unlimited_protected_stream_depth(prefetch_depth, stream_ID)
  ```
Example of POWER7 data prefetching

Store stream prefetch for array a;
transient stream prefetch for array b

```c
for (i=0; i< n; i++) {
    a[i] = b[i] + ...;
}
__protected_store_stream_set(FORWARD, &a, 11);
__protected_stream_count_depth(n*sizeof(double)/128, DEEPER, 11);
__protected_stream_set(FORWARD, &b, 0);
__transient_protected_stream_count_depth(n*sizeof(double)/128, DEEPER, 0);
__eieio();
__protected_stream_go();
```

Stream direction
Stream id
Stream length
Prefetch depth
Start stream prefetch
Loop Optimization

- **Traditional unimodular loop transformations for prefect regular loop nests**
  - Compiler loop transformations including loop fusion, loop distribution, unroll-and-jam, loop tiling, loop rerolling, loop collapsing, loop unrolling
  - Compiler pragmas: `unroll`, `stream_unroll`, `block_loop`, `unrollandfuse`
  - Under the optimization level O3, O3 –qhot or above

- **Polyhedral framework for any loop nests**
  - Provide abstract representation for aggressive analysis and complicated transformations of arbitrary loop nests and shapes under option control –qhot=level=2 with -qsmp
    - Loop skewing, loop tiling for triangular loop shapes
  - Perform exact dependence testing through unified dependence formulation to enable more aggressive loop transformations in both traditional and polyhedral frameworks under at all hot levels (-O3 –qhot or above)
Polyhedral Loop Transformation Examples

- Dependence analysis
  
  ```
  do i = 1,N
    do j = 1,i
      do k = i+1,N
        c(k) = c(j) + b(j,k)
  ```

  - Enable interchange of j and k loops to improve access locality for b
    - Identifies independence of memory accesses to c

- Loop transformations
  
  - Tiling of triangular matrix multiplication
    ```
    do i = 1,N
      do j = i+1,N
        do k = i+1,N
          c(j,i)+=a(k,i)*b(j,k)
    ```

  - Also allows transformation of imperfect loop nests
    - Intervening code between loops

  - Only available at -qhot=level=2
Polyhedral Loop Transformation Example

Input

Sequence of Imperfect Loop nests
for j ...
  for k ...
    A[j] = B[k]
  for i ...
    C[i][k] = ...
for j ...
  for k ...
    M[j] = N[k]
  for i ...
    X = C[i][k]

Output

Parallelism & Locality
Optimized Loops
for jT ...
  #omp parallel for
  for kT ...
    for j ...
      A[j] = B[k]
      M[j] = N[k]
    for i ...
      C[i][k] = ...
    X = C[i][k]
Data Reorganization

- **Data reorganization transformations to reduce memory latency**
  - Context sensitive and insensitive safety analysis
  - Data affinity analysis and shape analysis
  - Data splitting, data transposing, data interleaving to reshape data layout

- **Enabled at O5**

### Data Reorganization Report

<table>
<thead>
<tr>
<th>Seq #</th>
<th>Type</th>
<th>Phase</th>
<th>Data Name</th>
<th>Category</th>
<th>Region #</th>
<th>Line #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ArraySplitting</td>
<td>High Level Optimizer</td>
<td>iplus</td>
<td></td>
<td></td>
<td>9</td>
<td>An array of a large aggregated data-type was split into multiple arrays of smaller data-types.</td>
</tr>
<tr>
<td>27</td>
<td>ArrayCoalescing</td>
<td>High Level Optimizer</td>
<td>net</td>
<td></td>
<td></td>
<td></td>
<td>Global variables were aggregated.</td>
</tr>
</tbody>
</table>
Array splitting

Array merging

Array transposing

Data locality, cache utilization
User Explicit Parallelization with OpenMP

- Full OpenMP 3.0 implementation on C, C++ and Fortran
  - Full OpenMP task parallelization
  - Privatization of Fortran descriptor-based arrays

- Efficient Threadprivate implementation using OS supported Thread-Local Storage TLS by default
  - Bypass expensive pthread key mechanism
  - pthread-based implementation available under option control -qsmp=nOSTLS for backward compatibility

- Improved interaction between OpenMP and automatic SIMD
Automatic parallelization

- Improvements to automatic parallelization
  - More effective array data flow analysis for array privatization
  - Automatic privatization of descriptor-based Fortran arrays
  - Runtime-dependence testing

- SMP runtime improvements
  - Leverage of TLS on SMP runtime implementation

- Enable automatic parallelization with the compiler option -qsmp
XLSMPOPTS Environment Variable for Runtime Tuning

- **XLSMPOPTS** environment variable allows you to tune runtime behavior of OpenMP and autoparallel programs.

- **Some suboptions of interest:**
  - `spins` and `yields` to define the behavior of idle threads
  - Thread binding using `startproc` and `stride` suboptions
  - New `bind` suboption on AIX, `bind=SDL=<start resource>,<number of resources>,<stride>;bindlist=SDL=i0,i1,…,ix`
  - `schedule` to define the runtime scheduling algorithm used for parallel loops (static, dynamic, guided)
  - Note that the default schedule has changed from `runtime` to `auto` in V11/V13
Inlining

- **Single control knob to enable inlining**
  - Simplifies inlining control for programmer
  - `-qinline=level=X`    `X=0..10` (default 5)
  - Consistent across all languages and optimization levels
  - Previous mechanisms still available

- **User inline control with** `-qinline{+|-}<function_name>`

- **Automatic inlining before loop optimization**
  - Previously only available at `-O5`, or user inlining on C++
  - Available at all levels of `-qhot` (default at `-O3` and up)
  - Enables early inlining of Fortran module procedures
Control over optimizations that may affect program results
-qstrict suboptions

- **Aggressive optimization may affect the results of the program**
  - Precision of floating-point computation
  - Handling of special cases of IEEE FP standard (INF, NAN, etc)
  - Use of alternate math libraries

- **-qstrict guarantees identical result to noopt, at the expense of optimization**
  - Suboptions allow fine-grain control over this guarantee
  - Examples:
    - `qstrict=precision` Strict FP precision
    - `qstrict=exceptions` Strict FP exceptions
    - `qstrict=ieeefp` Strict IEEE FP implementation
    - `qstrict=nans` Strict general and computation of NANs
    - `qstrict=order` Do not modify evaluation order
    - `qstrict=vectorprecision` Maintain precision over all loop iterations

- **Can be combined: -qstrict=precision:nonans**
The IBM Rational C/C++ Café on IBM developerWorks

ibm.com/rational/cafe/community/ccpp

Overview

Compilers are the bridge between your applications and the hardware architectures on which you run your business. They are integral to application efficiency, programmer productivity and code portability.

IBM C/C++ compilers are designed to unleash the full power of IBM processors, including those shipped in industry-leading IBM Power, IBM System z, IBM Blue Gene and Cell Broadband Engine™ server systems. IBM’s C/C++ compiler products are developed specifically to deliver unprecedented performance and reliability on these systems.

We devised this online café community to provide in-depth information to help you take full advantage of IBM C/C++ compiler products. We provide blogs and on-line forums to facilitate conversations in the spirit of fostering community and collaboration. We hope that you will also share your experiences with us, both for the benefit of other community members and to help us deliver features that better suit your needs.

Tags: c, c++, cafe, rational

Blog
Wiki
Bookmarks
Feeds
Fortran Cafe on IBM developerWorks

Overview

Fortran is the language of choice for scientists and engineers. In addition to its common use as a powerful language for scientific and medical research, it lends itself well to industrial applications in fields such as weather forecasting, aeronautics, aerospace, and the military. Fortran's rich set of mathematical computation features, array manipulation, and optimizable constructs help you quickly create complex yet efficient modelling algorithms. You can use IBM's modern standards-compliant XL Fortran compiler to unlock the full potential of IBM Power and Blue Gene systems running AIX and Linux.

Tags:
Feature Request

- Request for a feature to be supported by our compilers

- **C/C++ feature request page:**

- **Fortran feature request page:**

- Or send e-mail to xl_feature@ca.ibm.com
Documentation

- An information center containing the documentation for the XL Fortran V13.1 and XL C/C++ V11.1 versions of the AIX compilers is available at:
  http://publib.boulder.ibm.com/infocenter/comphelp/v111v131/index.jsp
  –Now downloadable as fully searchable package

- Whitepaper “Code optimization with the IBM XL Compilers”

- Whitepaper “Overview of the IBM XL C/C++ and XL Fortran Compiler Family” available at:

- Please send any comments or suggestions on this information center or about the existing C, C++ or Fortran documentation shipped with the products to compinfo@ca.ibm.com.