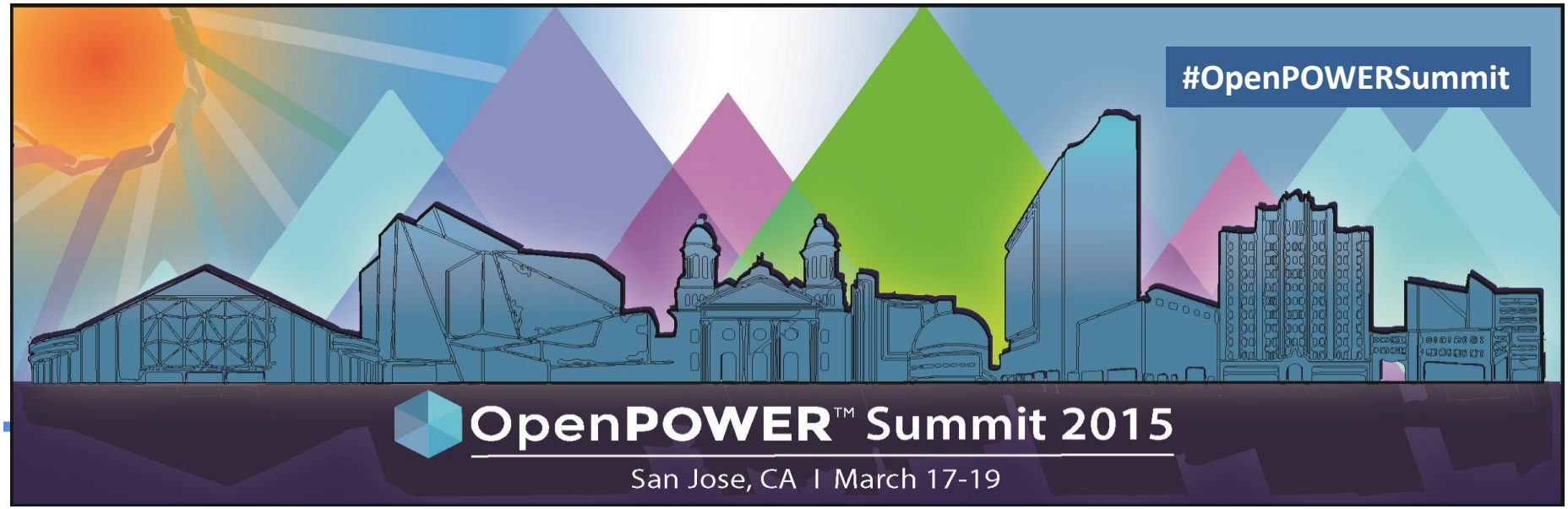
OpenPOWER™

Power and Speed: Maximizing Application Performance on IBM Power Systems with XL C/C++ and Fortran Compiler

Yaoqing Gao, Senior Technical Staff Member
ygao@ca.ibm.com
IBM Canada Lab



Agenda

- **Overview of IBM XL C/C++ and Fortran Compiler**
- **IBM XL Compiler Optimization Capacities**
- **Performance Tuning Tips**

Value of IBM Compilers on Power

- **Maximize return on your investment in Power hardware**
  - Designed to unleash full power of IBM POWER processors
- **Equipped with leading edge optimization technology**
  - Provide dramatic increase to the performance of an application.
  - No expert knowledge required; allows programmers to focus on business logic and produce high performing code
- **Support Application Development, Maintenance and Deployment**
  - Business Applications
  - Compute extensive Analytics and technical computing applications

Why XL Compiler on Power

SPEC benchmark leadership

- Overall **16%** on SPEC2006int over GCC
- Overall **57%** on SPEC2006fp over GCC
- Published leadership 1.7x on SPECint 1.9x on SPECfloat over Ivybridge on P8
- Critical in keeping Power leadership over Intel via proprietary optimizations

Key contributions to Power middleware

- Strong performance gains for major middleware & ISVs, 10-30% (e.g. DB2, SAP, Oracle)
- Mature whole-program and profile-based optimization are used in production by key middleware

Key contributions to Analytics & Technical Computing

- Up to 40% performance gain for ILOG CPLEX, up to 20% gain for SPSS
- Aggressive loop transformations for locality and memory hierarchy optimization
- Optimized OpenMP implementation

Power performance depends on XL Compiler exploitation

- SIMD, TM, and Power features rely on compiler exploitation and optimization
- XL advanced optimization is widely used in production environments by ISVs, Technical Computing and HPC

IBM XL Compiler

- **Target Linux, AIX on Power**
  - Common technology for Blue Gene/Q, and zOS (XL C/C++ only for zOS)
- **Advanced optimization capabilities**
  - Full exploitation of IBM hardware architectures
  - Advanced loop optimization
  - SIMDization and vectorization
  - Whole program optimization (IPA)
  - Profile-directed optimization (PDF)
  - Parallelization
- **Language standard compliance**
  - C99 Standard compliance, selected C11 features
  - C++98 and subsequent TRs, selected C++11 features
  - Fortran 2003 Standard compliance
  - OpenMP 3.1 conformance and selected OpenMP 4.1 features
- **Fully backward compatible with objects compiled with older compilers**
- **GCC affinity**
  - Partial source and full binary compatibility with GCC

XL Compiler Release for Power8

- XL C/C++ V13.1 and XL Fortran V15.1 for AIX and Linux
  - Helped Power S824 (POWER8, 3.5GHz, 24 core) achieve industry leadership performance results for SPECint_rate2006 and SPEC fp_rate2006 *(www.spec.org)*
- XL C/C++ V13.1.1 and XL Fortran V15.1.1 for OpenPOWER
  - **Support Power8 Ubuntu 14.04 & 14.10 and SLES12**
  - **Incorporate Clang technology**
    - Improved C/C++ language conformance
    - Improved GCC compatibility
    - Expressive diagnostics
  - **Continue to use IBM proprietary backend and optimization technology to deliver industry leading performance**
  - **integrated as the Power8 host compiler with CUDA 5.5 on Ubuntu 14.10 (**https://developer.nvidia.com/cuda-downloads-power8)

Optimization Capabilities

- **Platform exploitation**
  - –qarch: ISA exploitation
  - –qtune: skew performance tuning for specific processor, including tune=balanced
  - –Large library of compiler builtins and performance annotations

- **Mature compiler optimization technology**
  - –Five distinct optimization packages
  - –Debug support and assembly listings available at all optimization levels
  - –Whole program optimization
  - –Profile-directed optimization

Summary of Optimization Levels

- **Noopt,-O0**
  - Quick local optimizations
  - Keep the semantics of a program (-qstrict)
- **-O2**
  - Optimizations for the best combination of compile speed and runtime performance
  - Keep the semantics of a program (-qstrict)
- **-O3**
  - Equivalent to –O3 –qhot=level=0 –qnostrict
  - Focus on runtime performance at the expense of compilation time: loop transformations, dataflow analysis
  - May alter the semantics of a program (-qnostrict)
- **-O3 –qhot**
  - Equivalent to –O3 –qhot=level=1 –qnostrict
  - Perform aggressive loop transformations and dataflow analysis at the expense of compilation time
- **-O4**
  - Equivalent to –O3 –qhot=level=1 –qipa=level=1 -qnostrict
  - Aggressive optimization: whole program optimization; aggressive dataflow analysis and loop transformations
- **-O5**
  - Equivalent to –O3 –qhot=level=1 –qipa=level=2 -qnostrict
  - More aggressive optimization: more aggressive whole program optimization, more precise dataflow analysis and loop transformations

Basic Optimization Techniques

- **Inlining**
  - Replaces a call to a procedure by a copy of the procedure itself. It is done to eliminate the overhead of calling the function, and also to allow specialization of the function for the specific call point
- **Redundancy detection**
  - Identify computations that are redundant or partially redundant with values previously computed, so their value can be reused rather than recomputed
- **Platform exploitation**
  - Use a model of the target processor to determine the best mix of instructions to use to implement a certain program sequence
- **Flow restructuring**
  - Reorganize the code to increase the density of the hot code or to make it less frequent for conditional branches to be taken

Loop Optimization

- **Analyze and transform loops to improve runtime performance**
  - Analyze memory access patterns to improve cache utilization
  - Tailor instruction schedule for specific loop and target processor
  - Interleave execution of multiple loop iterations

- **Most effective on numerical applications, e.g. analytics, technical computing**
  - Depends on loops with regular behavior that can be analyzed and restructured by the optimizer

- **Enabled at O3 and above. Aggressive loop optimization with –O3 -qhot**

SIMDization/Vectorization

- **POWER8 SIMD Hardware Improvements**
  - Major improvements of misaligned vector load/store
  - Fully symmetric VMX units and SP enhancements of P7+ doubles throughput
  - New 2-way 64b integer operations
  - Direct move facility for VSR/GPR transfers

- **Explicit SIMD programming with –qaltivec (=BE|LE)**

- **Compiler Auto-SIMDization**
  - New SIMD infrastructure for simplicity and performance, enabled by default at –O3, -qhot
  - More aggressive SIMDization for both loops and basic blocks for POWER8

MASS and MASSV Libraries

- **Libraries of mathematical routines tuned for optimal performance on various POWER architectures**
  - General implementation tuned for POWER
  - Specific implementations tuned for specific POWER processors (pwr5, pwr6, pwr7)

- **Compiler will automatically insert calls to MASS/MASSV routines at higher optimization levels**
  - Users can add explicit calls to the library

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

Transformation report

Loop vectorization was performed.

```
__vsqrt_P8(b,a,n);
```

Parallelization

- **User-driven parallelism**
  - All optimization levels interoperate with POSIX Threads implementation
  - Full OpenMP 3.1 implementation provides simple mechanism to write parallel applications
    - Based on pragmas/annotations on top of sequential code
    - Industry specification, developed by OpenMP consortium (www.openmp.org)
- **Compiler-driven parallelism**
  - Mechanism for the compiler to automatically identify and exploit data parallelism
  - Identify parallelizable loops, performing independent operations on arrays or vectors
    - Best results on loop-intensive, compute-intensive workloads
    - Aided by program annotations, fully interoperable with OpenMP

Inter-Procedural Analysis (IPA)

- **Optimize the whole program at module scope**
  - Intercept the linker and re-optimize the program at module scope
- **Three levels of aggressiveness (-qipa=level=0/1/2)**
  - Balance between aggressive optimization and longer optimization time
- **Enables additional program optimization**
  - Cross-file inlining (including cross-language)
  - Global code placement based on call affinity
  - Global data reorganization
- **Reduction in TOC pressure, through data coalescing**
  - TOC: global directory used to access global variables in a module
    - Limited to 8K entries in 64-bit mode
  - TOC overflow triggers alternate linker scheme at performance cost

Profile-Directed Optimization (PDF)

- **Collect program statistics on training run to use on subsequent optimization phase**
  - Minimal impact on execution time of instrumented program (10% - 50%)
  - Static program information: Call frequencies, basic block execution counts
  - Value profiling: collect histogram of values for expressions of interest
  - Hardware counter information (optional)
- **Supports multiple training runs and parallel instances of the program**
  - Profiling information from multiple training runs aggregated into single file
  - Locking used to avoid clobbering of the profiling data on file
- **Integrated with IPA process (implies ipa=level=0)**
  - PDF synchronization point at beginning of link-time optimization phase
  - No need to recompile source files for PDF2, only relink with qpdf2 option
- **Tolerates program changes between instrumentation/optimization**
  - Compiler skips profile-based optimization for any modified functions
  - Shows an estimate of the relevance of the profiling data

Debugging Optimized Code

- **Debug levels**
  - Tradeoff between compiler optimization and debug transparency
  - Compiler optimizations hide program state from the debugger
    - Users have to choose between full debug at no-opt, or marginal debug at full opt

- **Compiler to provide control over tradeoffs between optimization and debug**
  - Debug levels: -g0 to –g9
    - -g1 minimal debug to maintain full performance
    - -g2 the default to provide maximal performance with unreliable debug
    - -g8 preserves most performance and allows examination of program state through debugger
    - -g9 provides full debug capability, at runtime performance cost
  - Expect better runtime performance from -g9 -O2 than -g –O0

Performance Tuning Tips

- **Compiler must be pessimistic when determining potential side effects**
  - Procedure calls may access or modify any visible variables
  - Accesses through pointers may modify any visible variables
- **Pessimistic side effect analysis prevents compiler optimizations**
  - Must re-compute expressions with operands which may have been modified
  - Must compute values that otherwise might be unneeded
- **Help the compiler identify side effects will improve application performance**
  - Use suitable optimization levels, -O2 minimum, better O3
  - Include appropriate header files for any system routines in use
  - Use local variables to maintain values of global variables across function calls or pointer dereferences
  - Avoid using global variables when local variables are suitable
  - Avoid reusing local variables for unrelated purposes
  - Follow ANSI C/C++ language pointer aliasing rules
    - An object of a certain data type can only be accessed through a pointer of the same (or compatible) data type

Performance Tuning Tips

- Obey all language aliasing rules (avoid –qalias=noansi in C/C++)
- Avoid unnecessary use of globals and pointers; use restrict keyword (XLC supports multiple level and scope restricted pointer) or compiler directives/pragmas to help the compiler do dependence and alias analysis
- Use “const” for globals, parameters and functions whenever possible
- Group frequently used functions into the same file (compilation unit) to expose compiler optimization opportunity (e.g., intra compilation unit inlining, instruction cache utilization)
- Limit exception handling
- Excessive hand-optimization such as unrolling and inlining may impede the compiler
- Keep array index expressions as simple as possible for easy dependency analysis
- Consider using the highly tuned MASS and ESSL libraries

Performance Tuning Tips

- **POWER8 exploitation**
  - POWER8 specific ISA exploitation under –qarch=pwr8
  - Scheduling and instruction selection under –qtune=pwr8:SMTn (n=1, 2, 4, 8)
- **Automatic SIMDization at O3 –qhot**
  - Limited use of control flow
  - Limited use of pointers. Use independent_loop directive to tell the compiler a loop has no loop carried dependency; use either restrict keyword or disjoint pragma to tell the compiler the references do not share the same physical storage whenever possible
  - Limited use of stride accesses. Expose stride-one accesses whenever possible
- **Data prefetch**
  - Automatic data prefetch at O3 –qhot or above.
  - Problem-state control of DSCR (data stream control register) to control data prefetch

Performance Tuning Tips

- **"-O2 –qipa" or "-O3 –qipa" for commercial workloads and "-O3 –qhot" for technical computing workloads**
- **Make use of visibility attribute**
  - Load time improvement
  - Better code with PLT overhead reduction
  - Code size reduction
  - Symbol collision avoidance
- **Inline tuning**
  - Call overhead reduction
  - Load-hit-store avoidance
- **Whole program optimization by IPA**
  - Across-file inlining
  - Code partitioning
  - Data reorganization
  - TOC pressure reduction