IBM Compiler Optimization on Bassi

June 14, 2006

Michael Stewart/Richard Gerber

NERSC User Services Group

pmstewart@lbl.gov

510-486-6648

ragerber@lbl.gov

510-486-6820

Introduction

- Why be concerned about choosing the right compiler optimization arguments on Bassi?
- What are the most useful compiler arguments and libraries for code optimization?
- Examples of the effects of various optimization techniques on benchmark codes.

IBM Default: No Optimization!

- When you compile a code on Bassi without any arguments using any IBM compiler: no optimization!
- Can have very bad consequences:

```
do i=1,bignum
   x=x+a(i)
enddo
```

- *bignum* stores of x are done when the code is compiled with no optimization argument.
- When optimized at any level, **store motion** is done: intermediate values of x are kept in registers and the actual store is done only once, outside the loop.

NERSC/IBM Optimization Recommendation

- For all compiles - Fortran, C, C++:
  -O3 -qstrict -qarch=auto -qtune=auto
- Compromise between minimizing compile time and maximizing compiler optimization.
- With these options, optimization only done within a procedure (e.g. subroutine, function).
- Numerical results bitwise identical to those produced by unoptimized compiles.
- Drawback: does not optimize complex or even very simple nested loops well. Add **–qhot** for these routines.
- Generally speaking the best Seaborg optimizations are the best Bassi optimizations.

Numeric Arguments: -O2 and –O3

- **-O0** and **-O1** not currently supported.
- **-O2**: Intermediate level producing numeric results equal to those produced by an unoptimized compile.
- **-O3**:
  - More memory and time intensive optimizations.
  - Can change the semantics of a program to optimize it so numeric results will not always be equal to those produced by an unoptimized compile unless **-qstrict** is specified.
  - No POWER5 processor specific optimizations.
  - Not very good at loop oriented optimizations.
- Most benchmarks achieve 90% or better of their maximum possible performance at the **–O3** level.

Numeric Arguments: -O4

- Equivalent to "**–O3 –qarch=auto –qtune=auto -qcache=auto -qipa -qhot**".
- Inlining, loop oriented optimizations, and additional time and memory intensive optimizations.
- Performs interprocedural optimizations.
- Option should be specified at link time as well as compile time.

Numeric Arguments: -O5

- Equivalent to “**–O4 –qipa=level=2**”.
- Full interprocedural optimization in addition to **–O4** optimizations.
- Option should be specified at link time as well as compile time.

-qstrict: Strict Equality of Numeric Results

- Semantics of a program are not altered regardless of the level of optimization, so numeric results are identical to those produced by unoptimized code.
- Inhibits optimization (in principle) - does not allow changes in the order of evaluation of expressions and prevents other significant types of optimizations.
- In practice, this option rarely makes a difference at the **–O3** level and can even improve performance.

-qarch: Processor Specific Instructions

- **-qarch=auto**: Produces code with machine instructions specific to the POWER5 processor that can improve performance on it.
- Codes compiled with **-qarch=auto** may not run on other types of POWER or POWERPC processors.
- The default at the **–O2** and **–O3** levels is **-qarch=com** which produces code that will run on any POWER or POWERPC processor.
- Default for **–O4** and **–O5** is **–qarch=auto.**
- When porting codes from other IBM systems to Bassi, make sure that the **–qarch** option is either **pwr5** or **auto**.

-qtune: Processor Specific Tuning

- **-qtune=auto**: Produces code tuned for the best possible performance on a POWER5 processor.
- Does instruction selection, scheduling and pipelining to take advantage of the processor architecture and cache sizes.
- Codes compiled with **-qtune=auto** will run on other POWER and POWERPC processors, but their performance might be much worse than it would be without this option specified.
- Default is for no specific processor tuning at the **–O2** and **–O3** levels, and for tuning for the processor on which it is compiled at the **–O4** and **–O5** levels.

-qhot: Loop Specific Optimizations

- Works with C/C++ as well as Fortran.
- Loop specific optimizations: padding to minimize cache misses, "vectorizing" functions like sqrt, loop unrolling, etc.
- Works best on loop dominated routines, if the compiler has some information about loop bounds and array dimensions.
- Operates by transforming source code: **-qreport=hotlist** produces a (somewhat cryptic) listing file of the loop transformations done when **-qhot** is used.
- Can double or triple compile time and may even slow code down at run time, but improves with each compiler release.
- Included by default with **–O4** or **–O5** compiles.

-qipa: Interprocedural Analysis

- Examines opportunities for optimization across procedural boundaries even if the procedures are in different source files.
- **Inlining** - Replaces a procedure call with the procedure itself to eliminate call overhead.
- **Aliasing** - Identifying different variables that refer to the same memory location to eliminate redundant loads and stores when a program's context changes.
- Can significantly increase compile time.
- Many suboptions (see man page).
- 3 ipa numeric levels: **-qipa=level=n**.

-qipa=level Optimizations

- Determines the amount of interprocedural analysis done.
- The higher the number the more analysis and optimization done.
- **-qipa=level=0**: Minimal interprocedural analysis and optimization.
- **-qipa=level=1** or **-qipa**: Inlining and limited alias analysis. **(-O4**)
- **-qipa=level=2**: Full interprocedural data flow and alias analysis. **(-O5)**

ESSL Library

- Single most effective optimization: replace source code with calls to the highly optimized Engineering and Scientific Subroutine Library (ESSL) .
- The ESSL library is specifically tuned for the POWER5 architecture and has many more optimizations than those that can be obtained with **–qarch=auto** and **–qtune=auto**.
- Contains a wide variety of linear algebra, Fourier, and other numeric routines.
- Supports both 32 and 64 bit executables.
- Not loaded by default, must specify the **–lessl** loader flag to use.

-lesslsmp: Multithreaded ESSL Library

- When specified at link time ensures that the multi-threaded versions of the essl library routines will be used.
- Can give significant speedups if not all processors of a node are busy.
- **Important**: Default for a program linked with **–lesslsmp** is to use 8 threads when run on Bassi. Change the number of threads by setting the **OMP_NUM_THREADS** environment variable to the desired number of threads.

Fortran Intrinsics

- Fortran intrinsics like matmul and random_number are multi-threaded by default at run time when a “thread safe” compiler (_r suffix) is used to compile the code.
- 8 threads are used by default on Bassi at run time for each task regardless of the number of MPI tasks running on the node – can lead to 128 threads running on a node.
- Can control the number of threads used at run time by setting the environment variable XLFRTEOPTS=intrinthds=n where n is the number of threads desired.
- The non-thread safe compilers (no _r subscript) produce code that is single threaded at run time.
- The performance of both the single and multi-threaded versions of the intrinsics are worse than their ESSL equivalents.

-qessl: Optimize Fortran Intrinsics

- **-qessl**: replace Fortran intrinsics with the equivalent routine from the ESSL library.
- Must link with **–lessl** (single threaded) or **–lesslsmp** (multi-threaded).
- For the multi-threaded version it uses the same number of threads as any ESSL or OpenMP routine in the code: 8 by default on Bassi or the value of the environment variable **OMP_NUM_THREADS**.

Other Useful Compiler Options

- **-qsmp=auto** – Automatic parallellization. The compiler attempts to parallelize the source code (runs with 8 threads by default at run time or the number of threads specified by the environment variable **OMP_NUM_THREADS**).
- **-Q+proc** – Inline specific procedure proc.
- **-qmaxmem=n** – Limits the amount of memory used by the compiler to n kilobytes. Default n=2048. n=-1 memory is unlimited.
- **-C** or **–qcheck** – Check array bounds.
- **-g** – Generate symbolic information for debuggers.
- **-v** or **–V** – Verbosely trace the progress of compilations.

Optimization Example: Dense Matrix Multiply

- Multiply two 1000 by 1000 real*8 dense matrices.
- Directly: **-O3 –qarch=auto –qtune=auto –qstrict**
- Fortran: c(i,j)=c(i,j)+a(i,k)*b(k,j)
- C: c[i][j]=a[i][k]*b[k][j]+c[i][j]

Performance depends on the order of the index variables.

	ijk	ikj	jik	jki	kij	kji	
Fortran	139	54	140	1446	54	1359	MFlops
C	159	1465	158	58	1337	59	MFlops

- Add **–qhot** to compile and performance differences disappear among the different index variable orders.

NPB2.3-serial Class B Benchmarks

- Serial versions of the moderate sized Class B NAS Parallel Benchmarks.
- 8 benchmark problems representing important classes of aeroscience applications written in C and Fortran 77 with Fortran 90 extensions.
- Designed to represent “real world codes” and not kernels.
- Revision 2.3 from 8/97.
- Information at http://www.nas.nasa.gov/NAS/NPB/.
- Has internal timings: time in seconds and Mop/s (million operations per second).
- Designed to run with little or no tuning.
- Timings are the best attained from multiple runs.

BT Simulated CFD benchmark

- Solves block-tridiagonal systems of 5x5 blocks.
- Solves 3 sets of uncoupled equations, first in the x, then in the y, and then in the z direction.
- A complete application benchmark, not just a kernel.
- Time and memory intensive (>1GB).
- 3700 source lines of Fortran.

BT Timings

Optimization	Mop/s	Compile (secs)	Seaborg Mop/s
None	264	1.10	48
-O2	1005	3.75	142
-O3 –qarch=auto –qtune=auto	1057	8.47	139
-O3 –qarch=auto –qtune=auto -qstrict	1049	8.35	140
-O3 –qarch=auto –qtune=auto -qstrict -qhot	**1094**	**29.69**	137
-O4	970	37.59	129
-O5	1050	63.92	143

CG Kernel

- Estimates the largest eigenvalue of a symmetric positive definite sparse matrix by the inverse power method.
- Core of CG is a solution of a sparse system of linear equations by iterations of the conjugate gradient method.
- 1100 lines of Fortran 77.

CG Timings

Optimization	Mop/s	Compile (secs)	Seaborg Mop/s
None	62	.32	18
-O2	166	.84	52
-O3 –qarch=auto –qtune=auto	**231**	**1.10**	53
-O3 –qarch=auto –qtune=auto -qstrict	213	1.19	52
-O3 –qarch=auto –qtune=auto -qstrict -qhot	229	2.46	53
-O4	224	3.64	54
-O5	210	4.74	54

EP Kernel

- 2 dimensional statistics are accumulated from a large number of Gaussian pseudo-random numbers.
- 250 lines of Fortran 77.

EP Timings

Optimization	Mop/s	Compile (secs)	Seaborg Mop/s
None	7.85	.44	2.72
-O2	10.13	.68	3.19
-O3 –qarch=auto –qtune=auto	13.36	.78	3.10
-O3 –qarch=auto –qtune=auto -qstrict	13.00	.79	3.12
-O3 –qarch=auto –qtune=auto -qstrict -qhot	13.38	1.07	3.70
-O4	**13.44**	**1.42**	3.69
-O5	**13.44**	**1.70**	3.70

FT Kernel

- Contains the computational kernel of a 3 dimensional FFT-based spectral method.
- Uses almost 2 GB of memory.
- 1100 lines of Fortran 77.

FT Timings

Optimization	Mop/s	Compile (secs)	Seaborg Mop/s
None	129	.51	30
-O2	776	1.08	138
-O3 –qarch=auto –qtune=auto	1035	1.18	139
-O3 –qarch=auto –qtune=auto -qstrict	**1048**	**1.18**	136
-O3 –qarch=auto –qtune=auto -qstrict -qhot	874	3.32	145
-O4	932	5.25	133
-O5	933	7.77	147

IS Kernel

- Integer sort kernel.
- 750 lines of C.

IS Timings

Optimization	Mop/s	Compile (secs)	Seaborg Mop/s
None	24	.40	3
-O2	51	.53	9
-O3 –qarch=auto –qtune=auto	47	.66	9
-O3 –qarch=auto –qtune=auto -qstrict	47	.67	9
-O3 –qarch=auto –qtune=auto -qstrict -qhot	59	.91	9
-O4	58	1.52	10
-O5	**63**	**2.22**	9

LU Benchmark

- Lower-Upper symmetric Gauss-Seidel decomposition.
- 3700 lines of Fortran.

LU Timings

Optimization	Mop/s	Compile (secs)	Seaborg Mop/s
None	248	1.29	52
-O2	1150	3.84	183
-O3 –qarch=auto –qtune=auto	1291	6.84	197
-O3 –qarch=auto –qtune=auto -qstrict	1143	7.77	188
-O3 –qarch=auto –qtune=auto -qstrict -qhot	**1581**	**27.71**	208
-O4	1570	35.67	220
-O5	1563	65.97	225

MG Benchmark

- Multi-grid method for 3 dimensional scalar Poisson equation.
- 1400 lines of Fortran.

MG Timings

Optimization	Mop/s	Compile (secs)	Seaborg Mop/s
None	168	.55	39
-O2	1212	1.46	214
-O3 –qarch=auto –qtune=auto	1458	2.66	221
-O3 –qarch=auto –qtune=auto -qstrict	**1461**	**2.61**	221
-O3 –qarch=auto –qtune=auto -qstrict -qhot	1283	5.18	229
-O4	1457	9.68	178
-O5	1432	14.42	177

SP Benchmark

- Multiple independent systems of non-diagonally dominant, scalar pentadiagonal equations are solved.
- Similarly structured to the BT benchmark.
- 3000 lines of Fortran.

SP Timings

Optimization	Mop/s	Compile (secs)	Seaborg Mop/s
None	143	1.12	28
-O2	477	4.03	99
-O3 –qarch=auto –qtune=auto	545	10.79	100
-O3 –qarch=auto –qtune=auto -qstrict	530	10.60	100
-O3 –qarch=auto –qtune=auto -qstrict -qhot	649	27.41	110
-O4	640	37.71	107
-O5	**684**	**58.71**	122

NPB2.3 Parallel Class C FT Benchmark

- Implements the time integration of a three-dimensional partial differential equation using the Fast Fourier Transform.
- MPI Fortran90 code.
- Revision 2.3 from 8/97.
- Same timings as the NAS B serial benchmarks.
- 32 processors on 4 nodes on Bassi.

NAS FT C 32 Processor Timings

Optimization	Mop/s	Compile (secs)	Seaborg Mop/s
None	118	1.43	22
-O2	669	1.80	86
-O3 –qarch=auto –qtune=auto	**816**	**2.17**	84
-O3 –qarch=auto –qtune=auto -qstrict	801	2.25	84
-O3 –qarch=auto –qtune=auto -qstrict -qhot	726	9.39	91
-O4	729	15.21	92
-O5	724	21.04	85

SuperLU 64 Processor Benchmark

- Set of subroutines to solve a sparse linear system A*X=B.
- Based on the ACTS SuperLU library. See http://acts.nersc.gov/superlu/.
- Implemented in ANSI C, using MPI to communicate data
- Double-precision real arithmetic.
- 64 processors on 8 nodes on Bassi.
- Timing is in elapsed wallclock seconds. The lower the better.

SuprerLU 64 processor Timings

Optimization	Seconds	Compile (secs)	Seaborg Seconds
None	197.29	3.80	1132.82
-O2	112.09	10.66	672.05
-O3 –qarch=auto –qtune=auto	105.34	14.02	665.15
-O3 –qarch=auto –qtune=auto -qstrict	105.77	13.95	648.69
-O3 –qarch=auto –qtune=auto -qstrict -qhot	105.98	38.28	634.16
-O4	106.17	39.21	619.41
-O5	**104.46**	**38.66**	608.35

CAM 32 Processor Benchmark

- Benchmark based on version 2.0.2 of the CAM Global Climate Model T42 Benchmark.
- Implemented in Fortran90 using OpenMP and MPI to communicate data.
- OpenMP is used to communicate among the 8 processors on a node and MPI to communicate among the 4 nodes.
- 32 total tasks: 4 nodes x 8 processors.
- Timing is in elapsed wallclock seconds. The lower the better.
- Version built with –O5 crashed on Bassi.

CAM 32 processor Timings

Optimization	Seconds	Compile (secs)	Seaborg Seconds
None	23.800	.18	76.533
-O2	23.807	134.08	76.468
-O3 –qarch=auto –qtune=auto	22.699	124.31	75.670
-O3 –qarch=auto –qtune=auto -qstrict	23.777	134.89	75.517
-O3 –qarch=auto –qtune=auto -qstrict -qhot	20.720	278.26	75.725
-O4	**20.237**	**626.27**	71.099
-O5	----	----	70.444

CAM Serial Benchmark

- 1 Processor version of CAM.
- Separately compiled version without MPI or OpenMP.
- Timing is in elapsed wallclock seconds. The lower the better.
- -O5 version crashes on Bassi.

CAM SerialTimings

Optimization	Seconds	Compile (secs)	Seaborg Seconds
None	353.037	.16	1272.419
-O2	354.675	66.42	1270.664
-O3 –qarch=auto –qtune=auto	329.578	85.94	1239.414
-O3 –qarch=auto –qtune=auto -qstrict	348.499	85.46	1235.977
-O3 –qarch=auto –qtune=auto -qstrict -qhot	297.163	180.89	1238.201
-O4	**288.341**	**479.27**	1095.652
-O5	-----	----	1094.667

Chombo 32 Processor Benchmark

- Solution of Laplace's equation in 3D on a multi-level block structured Adaptive Mesh Heirarchy.
- Written in C++ and Fortran using MPI to communication among the processes.
- Run on 32 processors on 4 nodes.
- Timing is in elapsed wallclock seconds. The lower the better.

Chombo 32 processor Timings

Optimization	Seconds	Compile (secs)	Seaborg Seconds
None	1868.23	27.84	6530.13
-O2	174.16	261.58	638.02
-O3 –qarch=auto –qtune=auto	168.07	498.98	605.56
-O3 –qarch=auto –qtune=auto -qstrict	165.30	497.53	601.67
-O3 –qarch=auto –qtune=auto -qstrict -qhot	162.27	484.29	580.46
-O4	**155.58**	**1520.93**	----
-O5	160.32	1145.21	----

Chombo Serial Timings

Optimization	Seconds	Compile (secs)	Seaborg Seconds
None	1268.19	27.84	4752.33
-O2	163.26	261.58	715.24
-O3 –qarch=auto –qtune=auto	150.18	498.98	678.05
-O3 –qarch=auto –qtune=auto -qstrict	150.20	497.53	689.15
-O3 –qarch=auto –qtune=auto -qstrict -qhot	157.46	484.29	654.38
-O4	**137.63**	**1520.93**	----
-O5	140.66	1145.21	----

Conclusions

- There is no one set of optimization arguments that is best for all program, but there should always be some level of optimization specified, even if only at **–O2** level.
- The NERSC/IBM recommended levels of optimization: **-O3 –qarch=auto –qtune=auto –qstrict** work well for most routines, but one should experiment with **–qhot** for numerically intensive and loop dominated routines.
- Use ESSL whenever possible.

Finis

End of this presentation.