Compiler

Information for Optimization

Agenda

- 32 vs. 64-bit and memory management
- Compiler performance optimization
  - Profile and large memory page
  - Optimization level 0-5
  - -qhot
  - Target machine specification
- Many others

Address Mode: -q{32,64}

- Available application modes:
  - -q32 (Default)
  - -q64
  - Also: environment variable OBJECT_MODE
    - export OBJECT_MODE={32,64}
  - Cannot mix -q32 objects with -q64 objects
- AIX kernel modes:
  - 32-bit
  - 64-bit
- Applications address mode is independent of AIX kernel mode

One more thing about 64-bit…

- If you use –q64:
  - Your job can use lots of memory than –q32
  - INTEGER*8 or long long operations are faster
- If you use –q32:
  - You may run a few (~10%) percent faster
    - Fewer bytes are used storing and moving pointers
  - You will have to learn AIX link options –bmaxdata
    - -bmaxdata:0x10000000 = 256 Mbyte = default
    - -bmaxdata:0x80000000 = 2 Gbyte
    - -bmaxdata:0xC0000000 = not widely publicized trick to use more than 2 Gbyte with –q32
      - "C" is the maximum
  - -q64
    - –bmaxdata:0 = default = unlimited
    - Other –bmaxdata values will be enforced if set

Even more on 64-bit...
(because it is so often confused)

- 64-bit floating point representation is higher precision
  - Fortran: REAL*8, DOUBLE PRECISION
  - C/C++: double
  - You can use 64-bit floating point with –q32 or –q64

- 64-bit addressing is totally different. It refers to how many bits are used to store memory addresses and ultimately how much memory one can access.
  - Compile and link with –q64
  - Use `file a.out myobj.o` to query addressing mode

- The AIX kernel can be either a build that uses 32-bit addressing for kernel operations or uses 64-bit addressing, but that does not affect an application's addressibilty.
  - ls –l /unix to find out which kernel is used
  - Certain system limits depend on kernel chosen

Suggested Fortran Compiler Usage

xlf90_r –q64

- Fortran 90 is the most portable standard
  - Consistent storage
    - Dynamic
- Reentrant code (..._r).
  - Required for:
    - phreads
    - Many other programming utilities
- 64-bit addressing:
  - Memory management

Suggested C Compiler Usage

xlc_r –q64

- Reentrant code (..._r).
  - Required for:
    - phreads
    - Many other programming utilities
- 64-bit addressing:
  - Better memory management

Address Modes

- ILP32
  - Integers, Long integers and Pointers are 32 bits
- LP64
  - Long integers and Pointers are 64 bits

- Standard C and C++ relationship:
  - sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long)

C and C++ Data Type Sizes

	-q32	-q64
Data Type	ILP32	LP64
Char	8	unchanged
Short	16	unchanged
Int	32	unchanged
Long	32	64
Long long	64	unchanged
Pointer	32	64
Enum	32	unchanged
Float	32	unchanged
Double	64	unchanged
Long double	64	unchanged

Long and Pointer change size with –q{32,64}

Fortran Data Type Sizes

Data Type	ILP32	LP64
integer	32	unchanged
integer(2)	16	unchanged
integer(4)	32	unchanged
integer(8)	64	unchanged
real	32	unchanged
Real(4)	32	unchanged
Real(8)	64	unchanged
Real(16)	128	unchanged
Double	64	unchanged
Logical	32	unchanged
Logical(2)	16	unchanged
Logical(4)	32	unchanged
Logical(8)	64	unchanged
Pointer	32	64

Pointer change size with –q{32,64}

Fortran Data Type Sizes

	-q32	-q64
Default INTEGER	4 bytes	4 bytes
Default REAL	4 bytes	4 bytes
Loc()	4 bytes	8 bytes

Fortran Data Type Sizes

Type	Default	Kind=1	Kind=2	Kind=4	Kind=8	Kind=16	Kind=32
Logical	4	1	2	4	8	4	4
Integer	4	1	2	4	8	4	4
Real	4	4	4	4	8	16	4
complex	4	4	4	4	8	16	4

Memory Management

	Heap	Stack
Fortran	Static Common Allocate	Dynamic Local
C	Static Malloc	Automatic Local
Default (-q32)	256 Mbyte	64 Mbyte
Loader option control (for –q32)	-bmaxdata	-bmaxstack

-bmaxdata: extend addressability to 2 GB in 32-bit mode.
e.g. “-bmaxdata:0x80000000” (0x70000000 for MPI)
-bmaxstack: similar to –bmaxdata but for stack

ALLOCATE and malloc Arrays

- Allocation occurs at statement execution
  - Heap operation
    - Inexpensive
    - Limited size
- Maximum size specification (for 32-bit only):
  - $(LDR) ...-bmaxdata:0x80000000

```
Subroutine sub(n)
Integer, allocatable, Dimension(:) :: A
…
Allocate(A(n))
…
Deallocate(A)
end
```

```
void my_proc()
{ long *A
…
A = (long *) malloc(n*sizeof(long));
…
free(A);
}
```

Dynamic and Automatic Arrays

- Memory allocation occurs at subroutine entry
  - Stack operation
    - Inexpensive
    - Limited size
- Maximum size specification:

  $(LDR) ...-bmaxstack:256000000

```
Subroutine sub(n)
integer A(n)
....
A(i) = ...
...
return
end
```

```
void  sub(int n)
{
long A[n];
....
A[i] = ...
...
return(0);
}
```

Dynamic and Automatic Arrays

	-q32	-q64	Comment
Default	64 Mbyte	Unlimited	
Maximum	2 Gbyte	Unlimited (ulimit)	-bmaxstack
Salve Default	4 Mbyte	4 Mbyte	
Slave Max.	64 Mbyte	4 Gbyte	XLSMPOPTS \ stack=64000000

Note: Extra concern with pthreads or OpenMP
Default SLAVE stack is only 4 Mbyte.
Use XLSMPOPTS=stack=…

Memory Allocation: Summary

	Heap	Stack
Control	-bmaxdata	-bmaxstack
-q32	2 Gbyte	256 Mbyte
-q64	Unlimited	unlimited

- **Programming advice:**
  - **Fortran ALLOCATE**
  - **C malloc**

Shifting to Next Topic

- Compiler options

Fortran Compiler Options – 10 categories

Categories	Commonly Used Options
Control input to the compiler	-I, -qfixed -qfree
Specify locations of output file	-d –o
Performance optimization	-O0 to –O5, -p, -pg, -qarch, -qtune, -qhot, -qessl, -qipa, -qlargepage, -qsmp=omp, -qstrict, -qthreaded, -qunroll
Error checking and debugging	-C, -g, -qdbg, -qlanglvl,
Control listings and messages	-qlist, -qreport –qversion –S -v
compatibility	-qautodbl, -qbigdata, -qinit, -qrealsize, -qsave, -qxlf77, -qxlf90
Floating-point processing	-qfloat, -qieee, -qstrictieeemod,
Control linking	-c, -L*dir*, -l*key,*
Control other compiler operations	-B, -F*config_file,* -q32, -q64
Obsolete or not recommended	-qcharlen, -qrecur

Summary: Commonly Used Options

- -q32, -q64
- -O0, -O2,-O3,-O4,-O5
- -qmaxmem=-1(allow max mem for comiling)
- -qarch=,-qtune=
- -hot (High order Transformation)
- -g (debugging)
- -p, -pg (profiling)
- -qstrict (no alter the semantics of a program)
- -qstatic
- -qipa (inter procedural analysis)
- -qieee
- -qlist (assembly lang report)
- -qsmp
- -qreport(smp list when –qsmp also used)

Profiling Your Code

1. Compile the code with –p (or –pg)
   compiler will set up the object file for profile (or graph profile)
- Execute the program. A mon.out (or gmon.out) file will be created
- Use prof (or gprof) command to generate a profile
- Or xprofiler a.out gmon.out (if you can open xwindow)

Example

```
➢xlf95 -p needs_tuning.f

➢ a.out
    mon.out created
➢ prof
```

Example

```
➢xlf95 -pg needs_tuning.f

➢ a.out
    gmon.out created
➢ xprofiler a.out gmon.out
```

Large Pages Option
-qlargepage

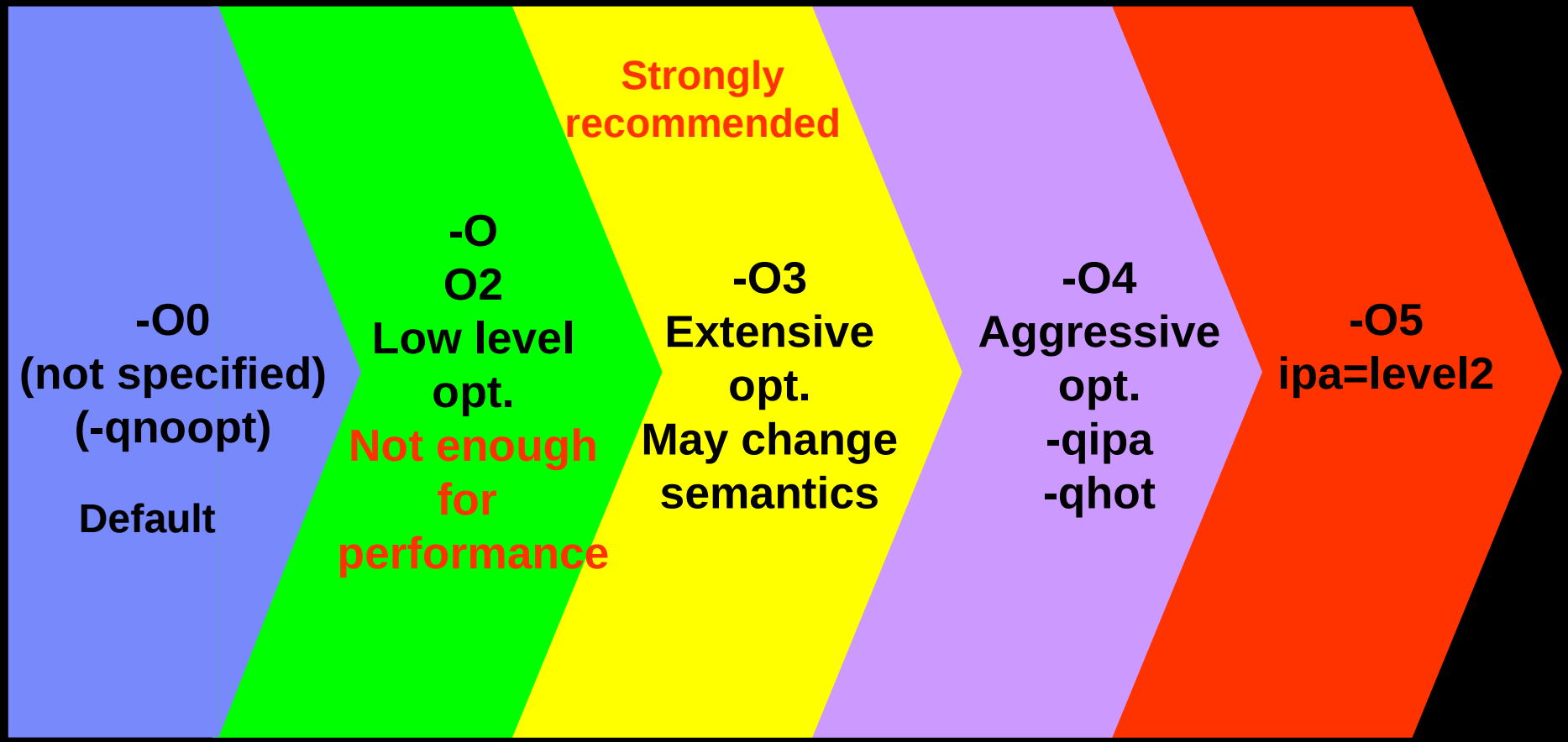
- Instructs the compiler to exploit large page heaps available on POWER4 and POWER5 systems
  - HINT to the compiler:
    - Heap data will be allocated from large page pool
    - Actual control is from loader option –blpdata or LDR_CNTRL=LARGE_PAGE_DATA
  - Compiler will (might) divert large data from the stack to the heap
  - Compiler may bias optimization of heap or static data references

Large page (16 MB) can improve performance significantly
(2) if the code uses GB+ memory
(3) If the code has TLB misses

Compiler Optimizations for Performance Optimization

- 5 levels of optimization
  - No specification: same as –qnoopt, -O0
  - -O0: no optimization, same as –qnoopt. Eq. to default
  - -O2, -O3, -O4, -O5
- -qhot: (High Order Transformation)
- -qipa: (Inter Procedural Analysis)
- Other
  - INLINE
  - UNROLL

Optimization Levels



Optimization Level 0, 2

-O0 (not specified)	-O2 = -O
Fast compilation	Comprehensive low-level opt.
Full support debugging	Global assign. of user variables
No optimization at all	Elim. redundant or unused code
DEFAULT	Scheduling instr. for target machine

Optimization Level 2 - 5

Strongly recommended for performance

-O2	-O3 *	-O4*	-O5*
Comprehensive low-level opt.	Source manipulation	High Order Transformations (HOT)	More aggressive IPA, inlining
Global assign. Of user variables	Inner loop unrolling	InterProcedural Analysis (IPA), inlining	
Elim. Of redundant or unused code	Software pipelining	Automatic architecture, tuning, cache detection	
Scheduling instr. for target machine	Whole procedure scope		

*** Use –qstrict to ensure semantics is unchanged**

Optimization Level Hierarchy

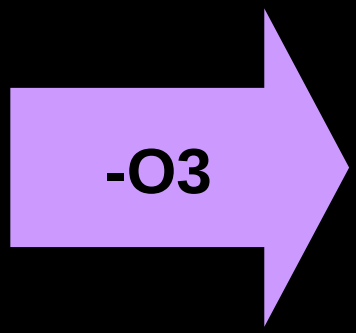
Base Optimization Level	Additional Options Implied by Base Optimization Level	Additional Recommended Options	Additional Options to Try with Base Optimization Level
-O0	None	-qarch -qtune	-g
-O2	-qmaxmem=2048*	-qarch -qtune	-qhot -g
-O3	-qnostrict -qmaxmem=-1*	-qarch -qtune	-g –qhot=vector -qstrict
-O4	All of –O3 plus: -qhot -qipa -qarch=auto -qtune=auto -qcache=auto		
-O5	-qipa=level=2		

*** Limit memory to be used by compiler for optimization**

Effect of –O2 vs. –O3

- Wider optimization scope
- Replaces divide with reciprocal
- Unrolls inner loops
- Precision tradeoffs
  - Not strictly IEEE floating point rules

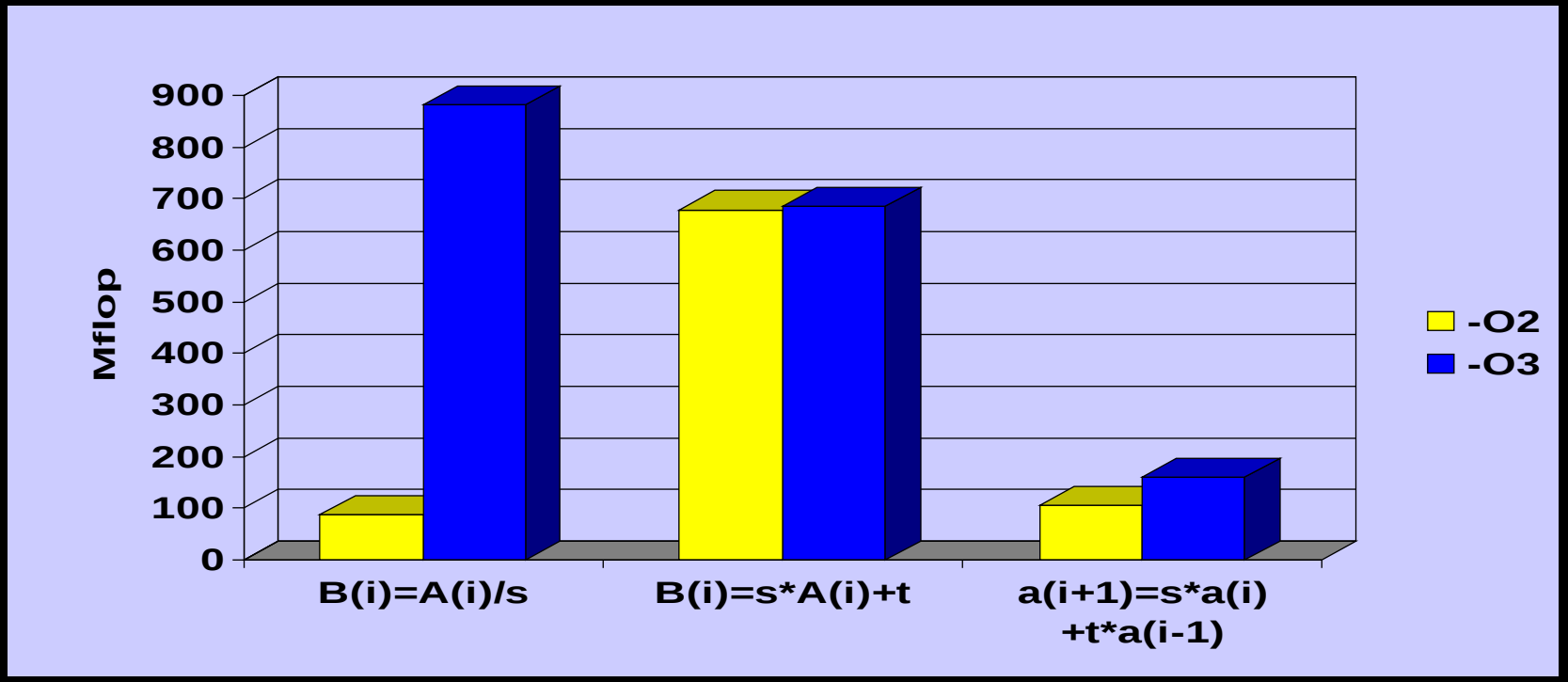
```
for (i=0;i<n;i++)
  b[i] = a[i]/s
```




```
rs=1/s
for (i=0;i<n;i=i+1)
  {b[i] = a[i]*rs
   b[i+1] = a[i+1]*rs}
```

Effect of –O2 vs –O3

- Replaces divide with reciprocal
- Unrolls inner loops
- "Regular" code runs well with -O2



1.3 GHz POWER4

-qessl Option

- -qessl allows the use of the ESSL routines in place of Fortran 90 intrinsic procedures
- Rules
  - Be sure to add –lessl (or –lesslsmp) to link command
  - Use thread save version of compiler xlf_r, xlf90_r or xlf95_r, since libessl.so and libesslsmp.so have a dependency on libxlf90_r.so, or specify –lxlf90_r on link command line
  - Example: c=MATMUL(a,b) may use ESSL routines
  - ESSL libraries are not shipped with the XLF compiler

High Order Transformations (HOT)

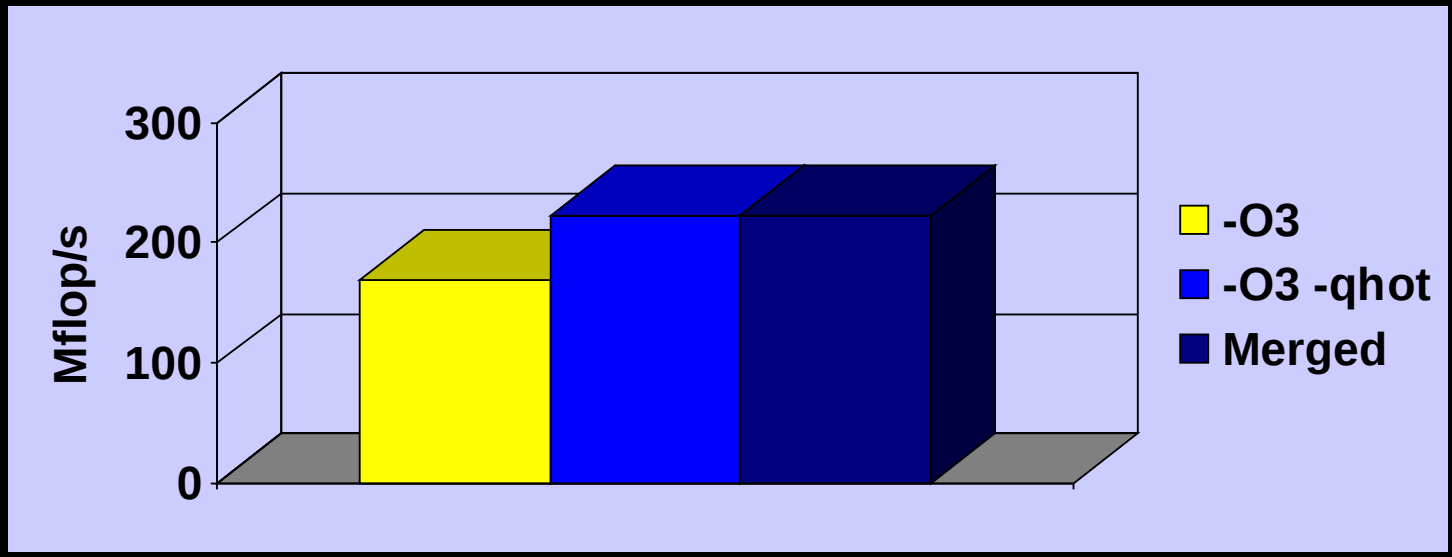
- -qhot [=[no]vector | arraypad[=n]]
  - Transformation of loop nests
  - Hardware prefetch
  - Balance loop computation
  - Vector intrinsic library
- Included at -O4 and higher level optimization

-qhot Transformation: Merge

```
do i=1,n
      A(i) = A(i) + B(i)*s
end do
do i=1,n
      C(i) = C(i) + A(i)*s
end do
```

-qhot

```
do i=1,n
      A(i) = A(i) + B(i)*s
      C(i) = C(i) + A(i)*s
end do
```



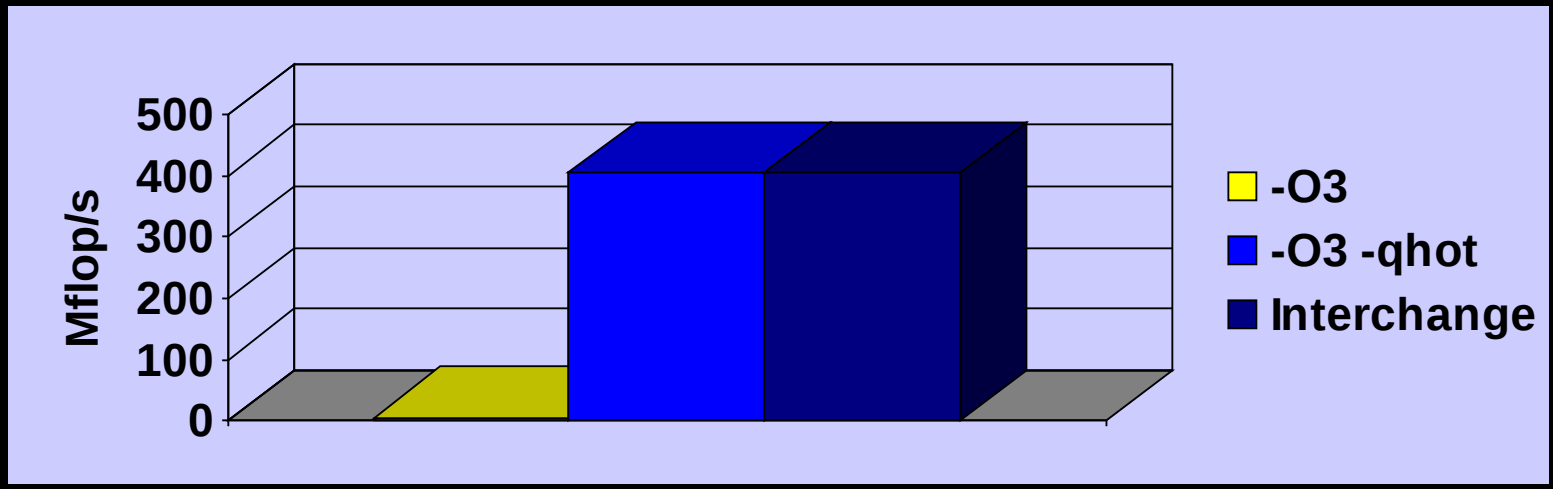
1.3 GHz POWER4

-qhot Transformation: Loop Interchange

```
do i=1,m
    do j=1,n
        sum = sum + X(i)*A(i,j)
    end do
end do
```

-qhot

```
do j=1,n
    do i=1,m
        sum = sum + X(i)*A(i,j)
    end do
end do
```



1.3 GHz POWER4

-qhot Transformation: Vectorization

- Extract intrinsic function, compute in batches
  - Lower latency
  - Better register utilization
  - Pipelined

Vectorization Example

```
SUBROUTINE VD(A,B,C,N)
REAL*8 A(N),B(N),C(N)
DO I = 1, N
  A(I) = C(I) / SQRT(B(I))
END DO
END
```

Vectorization Example Pseudocode

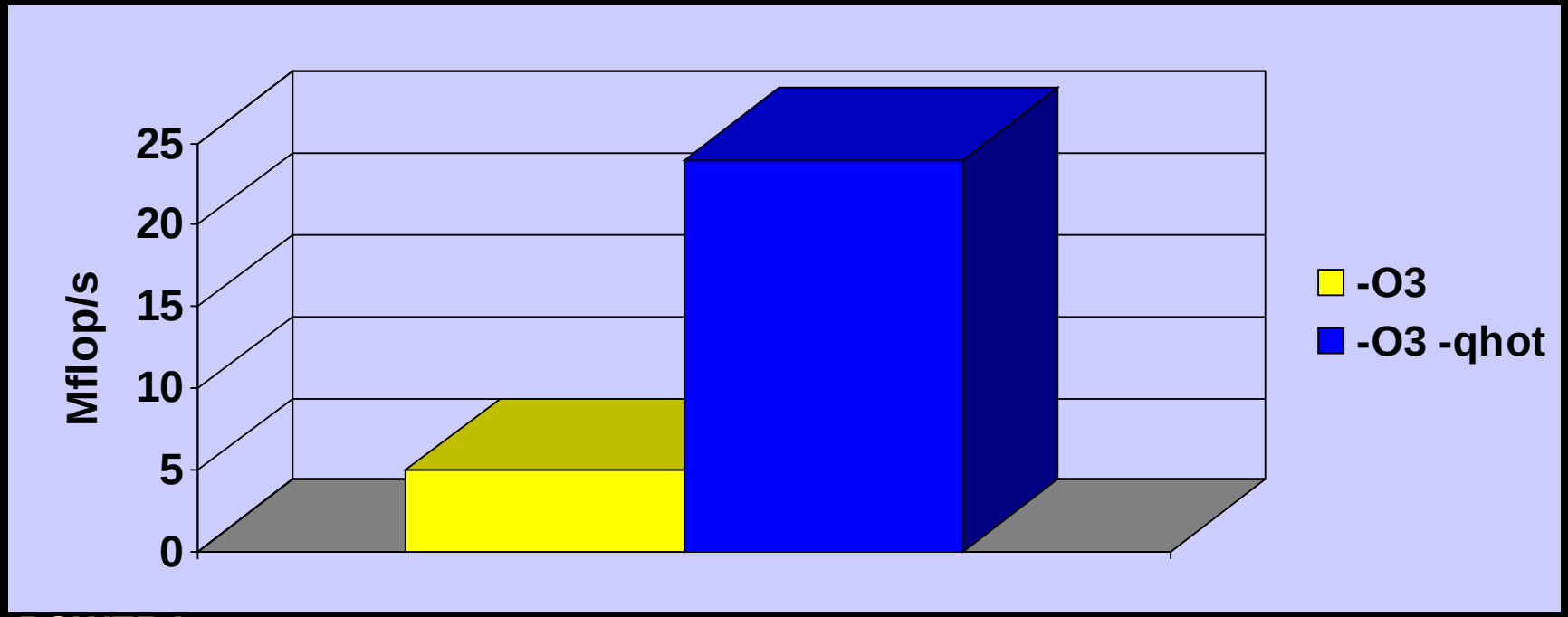
```
 SUBROUTINE vd (a, b, c, n)
3|    IF (n > 0) THEN
4|       CALL __vrsqrt_630(a,c,&n)
3|       @CIV0 = 0
         DO @CIV0 = @CIV0, n-1
            ! DIR_INDEPENDENT loopId = 0
4|        a(@CIV0 + 1) = b(@CIV0 + 1) * a(@CIV0 + 1)
5|       ENDDO
     ENDIF
6|   RETURN
     END SUBROUTINE vd


...
 Vectorization applied to statement.
```

-qhot Transformation: Vectorization

```
do i=1,n
      A(i) = exp(B(i))
 end do
```

-qhot

```
CALL __vexp(a ,b,n)
```



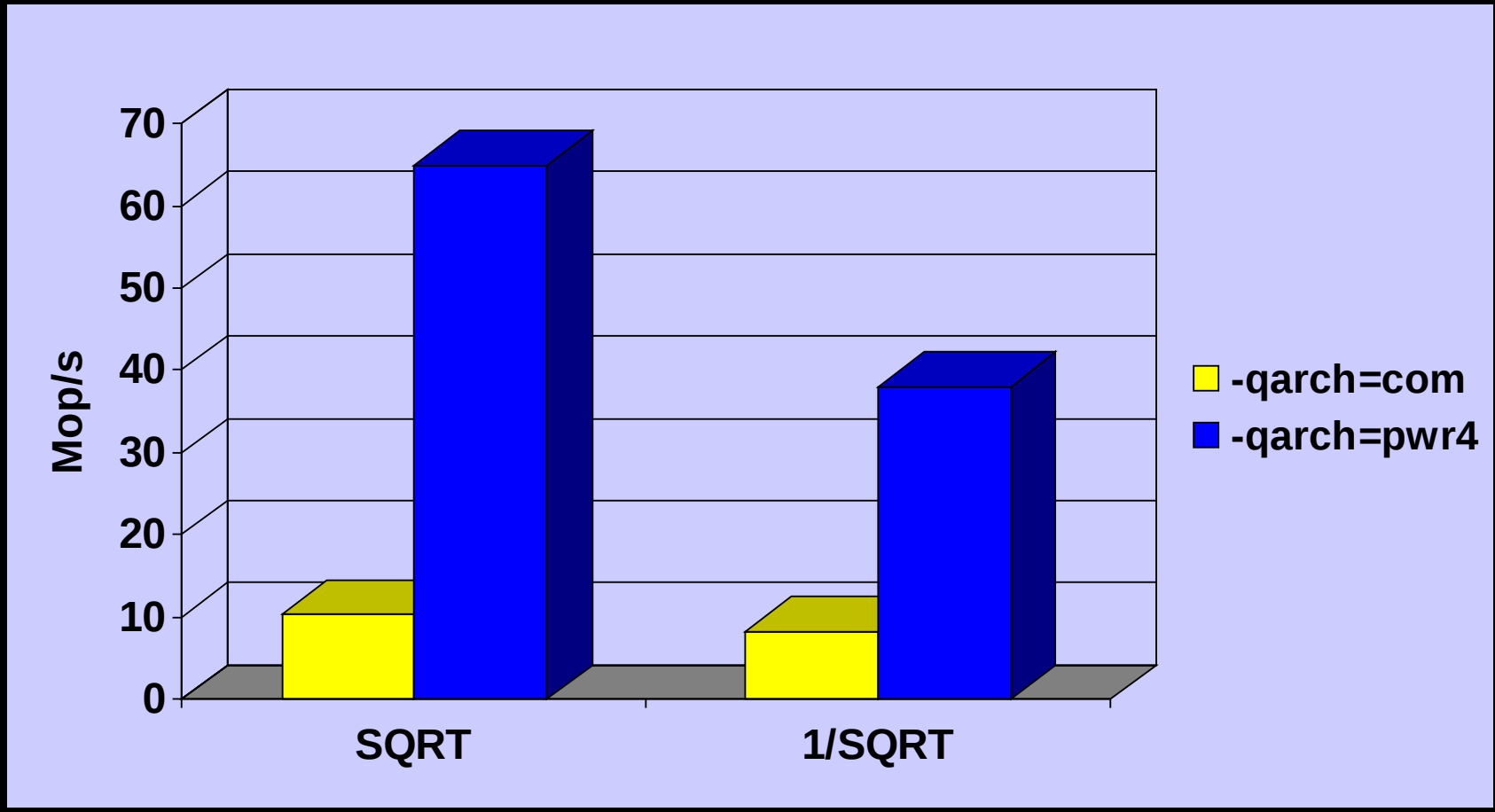
1.3 GHz POWER4

Target Machine Specification
-qarch

- **-qarch=[com,auto,ppc,pwr3,pwr4,pwr5,...]**
- **Generate a subset of the Power instruction set**

Arch	Example
pwr4	POWER4 (recommend for iHPC)
Com (default)	Code can run on any 64-bit ppc hardware platform
ppc	Code can run on any 32-bit ppc hardware platform
Auto	Same architecture for compilation and execution
pwr3, pwr4, pwr5, pwr5x	POWER3 POWER4 POWER5 POWER5+

Effect of -qarch



1.3 GHz POWER4

Target Machine Specification -qtune=processor

- Tunes instruction, scheduling, etc, for a given h/w
- Does not imply anything about the ability to run correctly on a given machine
- Affects performance but not instructions
- Independent of –qarch option
  - If 90% iHPC users are using PWR5, but 10% users are using PWR4, you can do –qarch=pwr4 –qtune=pwr5

Tune	hardware	Example for iHPC
pwr4	POWER4	Compile and run on either neumann, or merlin
Auto	Same arch for compile and execution	Compile and run on neumann, or Compile and run on merlin
pwr3, pwr4, pwr5,	POWER3 POWER4 POWER5	

Auxiliary Slides

Precision

- INTSIZE:
  - Define the default size of INTEGER variables
    - -qintsize=1|2|4|8
    - When using -q64, try -qintsize=8 for improved performance
- REALSIZE:
  - Define the default size of REAL variables
  - Specified as -qrealtsize=4|8
    - Examples: -qrealsize-=8
    - Does not promote explicit "REAL*4"
    - Does not reduce precision of REAL*16

-qrealsize=8

- **Default is 4 bytes**
- **-qrealsize=8 only promotes "real ", "default"**

Declaration	Default	-qrealsize=8
Undeclared	4	8
REAL	4	8
REAL*4	4	4
REAL*8	8	8
DOUBLE PRECISION	8	8
REAL*16	16	16

Long Double

- Special case
- Use xlc128 ... -ldbl128 ...
- (Else, cannot print long double, but can manipulate.)

Diagnostic Options: Listings

- -qlist
  - Object listing
  - Hex and pseudo-assembly code
  - Tag tables and text constants
- -qreport=
  - hotlist
  - smplist
    - Pseudo-Fortran annotations describing what transformations were performed
    - Information about dependences and other inhibitors to optimization

Assembly Language Report

- xlf ... -qlist file.f
- -> file.lst
- GPR: General Purpose Register
- FPR: Floating Point Register
- CCR: Communication and Control Register
- Look for excessive register use (overflows)

```
>>>>> OBJECT SECTION <<<<<

GPR's set/used:   ssus ssss ssss s---  ---- -sss ssss ssss
FPR's set/used:   ssss ssss ssss ssss  ssss ssss ssss ssss
CCR's set/used:   ss-- ----
```

Assembly Language: SPILL

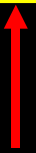
- xlf ... -qlist file.f
- -> file.lst
- Look for register spills

```
>>>>> OBJECT SECTION <<<<<
.....
0| 000168 fmadd    FCB62BFA   0     FMA      fp5=fp5,fp22,fp15,fc
0| 00016C stfd     D8810070   0     STFL     #SPILL5(gr1,112)=fp4
0| 000170 fmul     FDE605B2   1     MFL      fp15=fp6,fp22,fcr
0| 000174 stfd     D8410078   0     STFL     #SPILL6(gr1,120)=fp2
0| 000178 stfd     D9010068   0     STFL     #SPILL4(gr1,104)=fp8
```

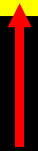
Assembly Language: Timing

- xlf ... -qlist file.f
- -> file.lst
- Look for bottlenecks

```
>>>>> OBJECT SECTION <<<<<
.....
310| 000A18 stfd    D83700A8  0    STFL    c(gr23,168)=fp1
302| 000A1C fdiv    FDB19824  14   RCPFL   fp13=fp17,fp19,fcr
```



Source Line



Delay

Bitfield and char sign

	Default	Option
bitfield	-qbitfields=unsigned	-qbitfields=signed
char	-qchars=unsigned	-qchars=signed

- Default is unsigned
- Compiler options:
  - -q{bitfields,chars}={un}signed

C++ Comments in C

- C++ comment sentinel: //
  - xlc –qcpluscmt …
    - Allow use of “//” comment in C

Problem: .f90 file extenstions

- Not consistent with Fortran90 standard definitions
- Valid file extensions (see "make -p")
  - .o .c .f .y .l .s .sh .h .C .a
- Xlf compiler option:
  - -qsuffix=<option>=<suffix>
    - f=<suffix>
    - o=<suffix>
    - s=<suffix>
    - cpp=<suffix>
  - Xlf –qsuffix=f=f90:cpp=F90
    - Fortran compiler invoke for .f90 and .F90 files; cpp for .F90
  - Need explicit makefile rules for new <suffixes>
  - XL Fortran 10.1 allows .f90 extension by default

Solution: .f90 Makefile rule

- Makefile:
- FFLAGS = ...
- .SUFFIXES: .f .f90 .my_ext

- .f.o:
- $(FC) $(FFLAGS) -c $<
- .f90.o:
- $(FC) $(FFLAGS) -qsuffix=f=f90 -c $<
- .my_ext.o:
- $(FC) $(FFLAGS) -qsuffix=f=my_ext -c $<

Using .F files

- Non-standard
  - Not recommended
- Compiler option:
  - -W<x>,<option1>[,<option2>...]
- Example:
  - xlf .... -WF,-DMYSWITCH
- "-d" option saves source file with "F" prefix
  - myfile.F --> Fmyfile.f

Calling C procedures by value

- Default Fortran convention is “call by reference”
  - Pass ADDRESS, not value
- %val(...) interface
  - ...
  - ptr = malloc(%val(nword*8))
  - ...
- This NOT done automatically by linker
  - Some compiling systems perform this task.

Call C Procedures by Value

Alternate interface:

```
interface
  function malloc(%val(n))
  integer n
  end function malloc
end interface
```

Allows calling "malloc(n)" instead of "malloc(%val(n))"

Xlf Fortran Symbolic Names

- Procedure names are lower case
  - C procedures names with upper case letters need to be changed
  - Some compilers use all capitals for all symbols
    - Unfortunately, there is NO STANDARD
  - Example:
    - Fortran call:
    - CALL MY_C_PROC(A,N)
    - C procedure:
      - void my_c_proc(double A, int N)
  - If something isn't working examine .o files with /usr/bin/nm
    - /usr/bin/nm my_fortran_proc.o | grep –i my_c_proc
    - /usr/bin/nm my_c_proc.o | gep –i my_c_proc

Xlf Fortran Names

- Use linker to change called name:
- xlf ... -brename:.old_name,.new_name
  - Note "." prefix in name
- Example
  - Fortran call:
    - CALL my_c_proc(A,N)
  - C procedure:
    - void MY_C_proc(double A, int N)
  - xlf ... -brename:.MY_C_proc,.my_c_proc

Fortran Namelist

- Default style Is "new" (Fortran 95)
- Example, new style:
  - &today I=1234 ! this is s comment./
  - x(1)=1234, x(3:4)=2*0.0
  - P="!isn't this great", Z=(123,0)
- Example, old style:
  - $today I=1234 ! this is s comment.
  - x(1)=1234, x(3:4)=2*0.0
  - P="!isn't this great", Z=(123,0) $END
- Conversion:
  - export XLFRTEOPTS=namelist=[new|old]
    - (for "XLF Run Time Environment Options").

Traceback

- Compile with "-g"
- Produce debug information
- Low overhead
- Run program
- Produce core file
- dbx {a.out}
- Analyze core file
- where
- Print out a stack trace

```
$ dbx a.out
 ...
 ...
(dbx) where
renorm(...), line 4 in
   "renorm.f"
calc1(...), line 14 in
   "calc1.f"
shallow(), line 76 in
   "shalow.f"
(dbx) quit
```

Parallel Makefile

- AIX makefile is not parallel
- Use gmake
  - /usr/bin/gmake
- gmake -j {# processes}
- gmake -j 20
  - ~5x -10x faster
  - Need "valid" makefiles
  - Explicit dependencies

Floating Point: -float

- [no]maf:
- Enable generation of multiple-add type instructions
- [no]hsflt:
- Replacement of division by multiplication by a reciprocal
- Other various fast floating point optimizations
- [no]rsqrt:
- .../SQRT(s) -> ...*rsqrt(s)
- Allow computation of a divide by square root to be replaced by a multiply of the reciprocal square root

Floating Point: -qfloat

- [no]fold:
- enable compile time evaluation of floating point calculations
- [no]rrm: specifies that rounding mode may not be round-to-nearest
- default is norm