



XL Compiler Libraries Tools

Dr. Achim Bömelburg

Unix Server Technical Support



Agenda

- Fortran: XLF & C - C++: XLC

- IBM performance libraries
 - ESSL, MASS, WSMP, MIO

- Monitoring tools



Compiler

Latest versions

- ❑ **VisualAge C++ for AIX, Version 6.0**
- ❑ **C for AIX, version 6.0**
- ❑ **XL Fortran for AIX, Version 8.1**

Older, supported versions

- ❑ **C for AIX, Version 5.0.2**
- ❑ **VisualAge C++ for AIX, Version 5.0.2**
- ❑ **VisualAge C++ Professional for AIX, Version 4.0 (until 12/02)**
- ❑ **XL Fortran for AIX, Version 7.1.1 (update for Power4)**
- ❑ **XL Fortran for AIX, Version 7.1.0.2**

Versions for Linux on pSeries

- ❑ **C, C++ and Fortran for Linux pSeries**

Fortran: XLF Basics

- Compilation

- `xlf <options> program.f`
- predefined options in `/etc/xlf.cfg`
- examples:

`xlf program.f` => generates `a.out` executable

`xlf -c program.f` => generates `program.o` object file for further processing
(archiving, linkage editing)

`xlf program.o subroutine.f -o program` => feeds `program.o` into linkage-editor, `subroutine.f` into compiler and produces executable program

Fortran: XLF Basics

- **Different invocations:**
 - **xlf** => original Fortran compiler
 - **f77** => alias for xlf
 - **fort77** => original Fortran compiler, used for XPG4 compliance
 - **xlf90** => Fortran 90 compiler
 - **f90** => alias for xlf90
 - **xlf95** => Fortran 95 compiler
 - **f95** => alias for xlf95

 - **More fine-control through `-qlanglvl` flag**

C/C++: XLC Basics

- **Different invocations:**
 - **cc** => traditional (K&R) C-compiler
 - **xlc** => ANSI 89 C
 - **c89** => alias for xlc
 - **x1C** => ANSI 98 C++
 - **c99** => ANSI 99 C

Fortran: XLF Basics

- **SMP programs / threadsafe programming:**
- **xlf_r** => reentrant original Fortran compiler (for smp [threadsafe] programs)
- **xlf_r7** => same as above, but compliant to POSIX Draft 7 threads
 - Analogously for the other invocations
xlf90_r **xlf90_r7** **xlf95_r** **xlf95_r7**
- **MPI programs / distributed memory programming:**
- **mpxlf** => original Fortran compiler for parallel environment
 - Analogously for the other invocations:
mpxlf90 **mpxlf95**
- **Mixed programs (MPI and SMP)**
- **mpxlf_r** => threadsafe original Fortran compiler for parallel environment
 - Analogously for the other invocations:
mpxlf90_r **mpxlf90_r7** **mpxlf95_r** **mpxlf95_r7**

XL Fortran: Debug and Check -1

- **-qsource** produces source section of the listing
- **-qlist** produces object section of the listing
- **-S** produces .s assembler files
- **-qattr=full** produces attribute component of the listing
- **-qxref=full** produces cross-reference of the listing

- **-g** generate symbolic debug information
- **-qtbtable=full** full debugging traceback information in object files
- **-qfullpath** record the absolute pathname of source and include files in object files compiled with **-g**
- **-qlanglvl=77std (e.g.)** identifies non-conforming code

XL Fortran: .lst for -qsource

```
>>>>> SOURCE SECTION <<<<<
      1 |      subroutine a(b,c,d,n)
      2 |c
      3 |      real b(n),c(n)
      4 |      real*8 d(n)
      5 |c
      6 |      do i=1,n
      7 |      b(i) = c(i) + d(i)
      8 |      enddo
      9 |c
     10 |      return
     11 |      end
** a      === End of Compilation 1 ===
```

XL Fortran: .lst for -qlist

>>>> OBJECT SECTION <<<<<

GPR's set/used: -s-s ss-- ---- ---- ---- ---- ----s
FPR's set/used: -ss- ---- ---- ---- ---- ---- ----
CCR's set/used: s--- ----

	000000			PDEF	a	
0				PROC	gr3-gr6=.b, .c, .d, .n	
0	000000	stw	93E1FFFC	1	ST4A	#stack(gr1,-4)=gr31
0	000004	stwu	9421FFA0	0	ST4U	gr1,#stack(gr1,-96)=gr1
0	000008	lwz	83E20004	1	L4A	gr31=.&a\$(gr2,0)
0	00000C	stw	90610078	0	ST4A	.b(gr1,120)=gr3
0	000010	stw	9081007C	1	ST4A	.c(gr1,124)=gr4
0	000014	stw	90A10080	0	ST4A	.d(gr1,128)=gr5
0	000018	stw	90C10084	1	ST4A	.n(gr1,132)=gr6
0	00001C	lwz	80610084	0	L4A	gr3=.n(gr1,132)

...

XL Fortran: .lst for `-qattr=full`

```
>>>> ATTRIBUTE AND CROSS REFERENCE SECTION <<<<<
```

```
IDENTIFIER NAME ATTRIBUTES
```

```
a          Subroutine
b          Reference argument, Real(4) (1:?), Offset: 0, Alignment: full word
c          Reference argument, Real(4) (1:?), Offset: 0, Alignment: full word
d          Reference argument, Real(8) (1:?), Offset: 0, Alignment: double word
i          Static, Integer(4), Offset: 0, Alignment: quadruple word
n          Reference argument, Integer(4), Offset: 0, Alignment: full word
** a      === End of Compilation 1 ===
```

XL Fortran: .lst for `-qxref=full`

>>>> ATTRIBUTE AND CROSS REFERENCE SECTION <<<<<

IDENTIFIER NAME	CROSS REFERENCE
a	0-1.18\$
b	0-3.12\$ 0-1.20 0-7.7@
c	0-3.17\$ 0-1.22 0-7.14
d	0-4.14\$ 0-1.24 0-7.21
i	0-6.10@ 0-7.9 0-7.16 0-7.23
n	0-1.26 0-3.14 0-3.19 0-4.16 0-6.14

** a === End of Compilation 1 ===

XL Fortran: .lst for -qreport

>>>> LOOP TRANSFORMATION SECTION <<<<<

```
SUBROUTINE a (b, c, d)
7|   b(1) = real((real(c(1)) + d(1)))
   b(2) = real((real(c(2)) + d(2)))
   b(3) = real((real(c(3)) + d(3)))
   b(4) = real((real(c(4)) + d(4)))
   b(5) = real((real(c(5)) + d(5)))
   b(6) = real((real(c(6)) + d(6)))
   b(7) = real((real(c(7)) + d(7)))
   b(8) = real((real(c(8)) + d(8)))
   b(9) = real((real(c(9)) + d(9)))
   b(10) = real((real(c(10)) + d(10)))
11|  RETURN
END SUBROUTINE a
```

Source File	Source Line	Loop Id	Action / Information
0	6		Loop has been completely unrolled because its iteration count is less than 32.

XL Fortran: Debug and Check -2

- **-qcheck / -C**
 - check each reference to an array element for correctness (for testing and debugging purposes, should be removed for production code)**
 - if compiler detects already during compilation time that a reference goes out of bounds it produces a severe error**
 - at run time an out-of-bounds reference produces a SIGTRAP signal**

- **-qextchk**
 - sets up type checking information for common blocks, procedure definitions, procedure references and modules**

 - during compile time: checks for consistency of procedure definitions and references**

 - during link time verifies that actual data agree in type, shape, passing mode and class plus declarations of common blocks**

XL Fortran: Debug and Check -3

- **-qinitauto[=hexvalue]** initialize automatic variables to a specific value
(helps to find variables that are referenced before being defined, e.g. set hexvalue to a NaN value and combine with flttrap)
[see /usr/include/fp.h for values](#)
- **-qflttrap[=suboptions]** determines which types of floating point exceptions are to be trapped
- **-qsigtrap** installs trap handler
- **-u / -qundef** equivalent to implicit none

Fortran: Debug and Check

-4

-
- **Subroutine in case of abnormal termination**
 - **-qsigtrap=xl_trcedump**
 - use dbx (or favorite debugger) to investigate core file

 - **Trap floating exceptions: -qflttrap=**
 - **overflow** - Detect and trap on floating-point overflow.
 - **underflow** - Detect and trap on floating-point underflow.
 - **zerodivide** - Detect and trap on floating-point division by zero.
 - **invalid** - Detect and trap on floating-point invalid operations.
 - **inexact** - Detect and trap on floating-point inexact.
 - **enable** - Turn on checking for the specified exceptions in the main program.
 - **imprecise** - Only check for the specified exceptions on subprogram entry and exit.

 - **To activate :**
 - qflttrap=en:ov:und:inv:zero:imp -qsigtrap**

XL Fortran: 32 bit and 64 bit -1

- **-q32** enables 32 bit addressing, i.e. 2 GB of address space
default integer size: 4 bytes
default real size: 4 bytes

to get the 2 GB you need to specify during the link step
-bmaxdata:0x8000000
e.g. **xlf prog.o -o a.out -bmaxdata:0x80000000**

- **-q64** enables 64 bit addressing
still:
default integer size: 4 bytes
default real size: 4 bytes

suboption (for AIX 4.3 compatibility):
-q64=nolargetype
this creates the old 64 bit format
(only for FORTRAN, no equivalent xlc option)

XL Fortran: 32 bit and 64 bit -2

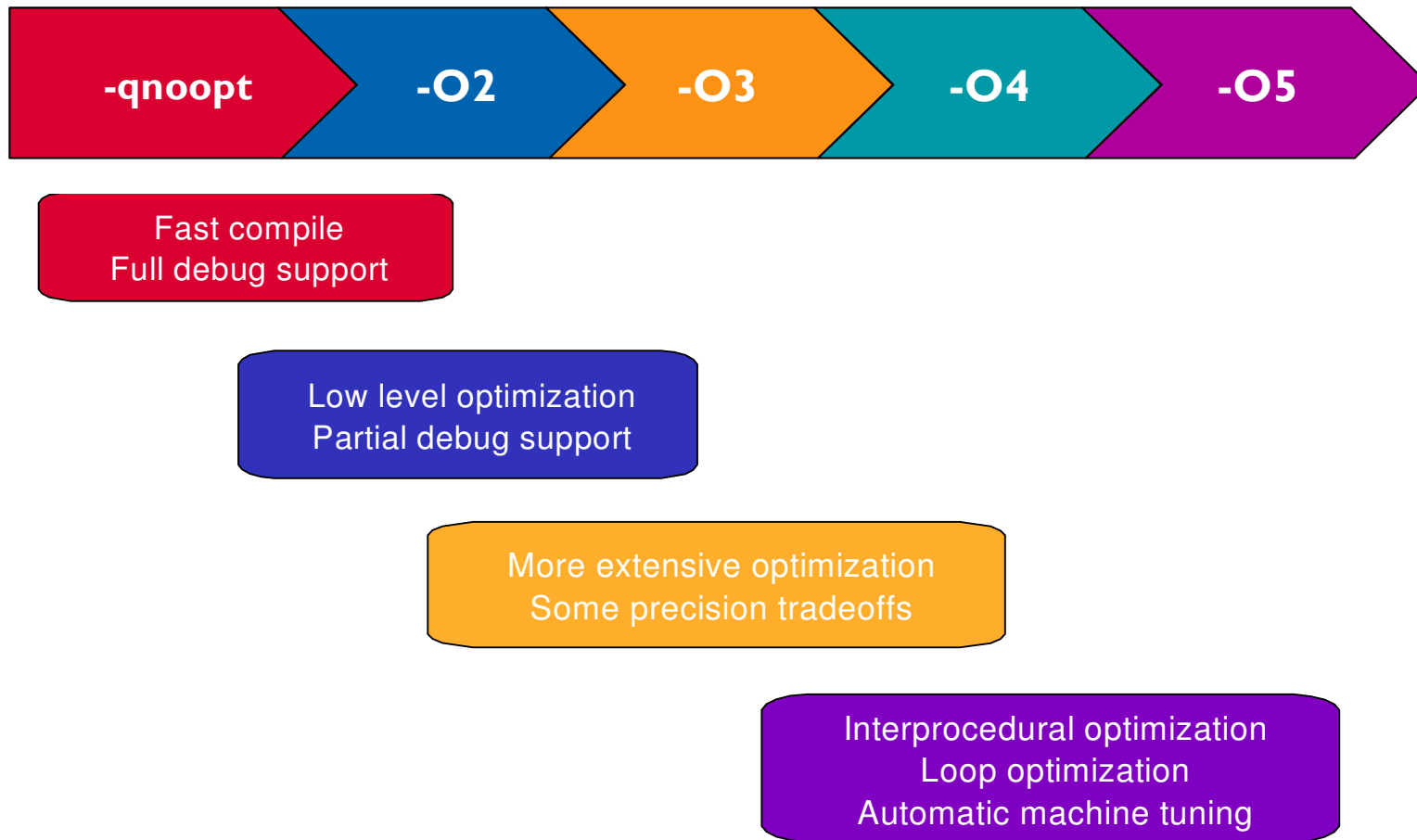
- **-qintsize=4/8** defines default integer size
- **-qrealsize=4/8** defines default real size (but only the **default** real size)
realsize=8 => double precision is real*16!
- **-qautodbl=dbl4** real(4) -- real(8)
complex(4) – complex(8)
- **-qautodbl=dbl8** real(8) – real(16)
complex(8) – complex(16)
- **-qautodbl=dbl** combines dbl4 and dbl8
- **-qautodbl=dblpad4** same as dbl4 and pads objects of other types
if they could share storage with promoted objects
- **-qautodbl=dblpad8** same as dbl8 and pads ...
- **-qautodbl=dblpad** same as dbl and pads ...



XL Fortran: 32 bit and 64 bit -3

- You can't mix 32bit and 64bit objects in the same executable
- The other tools needed during programming (ar, nm, ...) need **-X** flag, e.g. **ar -X32 cv myarchive ...** or **ar -X64 cv myarchive ...**

Fortran: XLF Optimization



Fortran: XLF Optimization

- **-O0 or -qnoopt:** Default no optimization
- **-O1** reserved for future use
-O1 does not do any optimization and is ignored!
- **-O or -O2:**
 - Same level of optimization.
- **-O3:**
 - Lengthen compilation time and needs more memory and disk resources
 - Reordering of floating point exceptions
 - Rearrange floating expressions: $A*(B*C)$ to $(A*B)*C$
 - Suppress constant expressions from DO loops
 - Replace division by inverse multiplication (rsqrt option from -qfloat)
 - Speedup conversion of floats to integers (fltint option from -qfloat)
 - Set -qmaxmem to -1



Fortran: XLF Optimization

- **-O4:**
 - Group of optimization oriented options:
 - *-O3 -qhot -qipa -qarch=auto -qtune=auto -qcache=auto*

- **-O5:**
 - Highest native optimization level
 - *-O4 -qipa=level=2*

- As a start :
 - *-g -O3 -qarch=pwr4 -qtune=pwr4 {-qstrict}*are usually sufficient to get ideas



Fortran: options for performance: -qarch and -qtune

- -qarch: controls which instructions to generate
- **-qarch=auto** => automatically detect the machine architecture of the compiling machine
- **-qarch=com** => executable can run on any POWER or PowerPC machine
- **-qarch=pwr4** => binaries will not necessarily run on every previous machine

- -qtune: tunes instruction selection and scheduling
- **-qtune=auto** => automatically detect the machine architecture of the compiling machine
- **-qtune=pwr4** => tune for POWER4

- **-qcache** => behaviour defined by -qtune settings



Fortran: options for performance:

-qfloat

- **-qfloat**: selects different strategies for floating point execution
- **-qfloat=fltint** => round double precision values only when they are stored to memory, used for real to integer conversions
- **-qfloat=fold** => evaluate constant floating point expressions at compile time
- **-qfloat=hsflt** => prevent rounding for single precision and replace floating point division by multiplication with the inverse and use fltint technique
- **-qfloat=hssngl** => single precision expressions are rounded only when the results are stored
- **-qfloat=maf** => allow multiply add instructions

Fortran: options for performance:

-qhot

- **-qhot:** higher order transformations on loops and pad array dimensions and data objects (may help if your first array dimension is a power of 2)
loop-interchange – unrolling – loop-fusion
to enhance memory locality, HW prefetch, loop / computation balance (ld/st vs. float)

- **-qhot=arraypad** => pad array by whatever the compiler chooses
- **-qhot=arraypad=n** => pad each array by n elements
- **-qhot=vector** => convert certain operations in a loop on successive elements of an array into a call to routines in libxlopt.a

Fortran: options for performance:

-Q and -qipa

-1

- **-Q**: specifies that inlining can / should be done
only affects procedures if both caller and callee are in the same
(source) file
if more is needed specify -qipa

- **-Q** => try to inline all appropriate procedures
- **-Q+names** => try a little harder to inline on names
- **-Q-names** => try a little harder to inline on names

- **-qipa** => enhances optimization through analysis across procedures
(interprocedural analysis)

Fortran: options for performance:

-Q and -qipa

-2

-qipa : ipa uses a 2 phase mechanism (compile-time phase and link-time).

During compile time ipa information is stored in the object-files,
during link-time ipa causes a complete reoptimization of the program

Specify during compile **and** link step

Suboptions:

-qipa=inline=auto

=> automatically inline procedures

-qipa=inline=limit=number

=> changes the size limits used by -Q
number is the size in bytes that will be
generated by the code

-qipa=inline=threshold=number

=> upper limit for procedures to be inlined

-qipa=inline=<proc-names>

=> try to inline these procedures



Fortran: options for performance: profile directed feedback -qpdf -1

Aim: optimize code near conditional branches and in frequently executed code sections

HowTo:

1: Compile all or some of your sourcecode with `-qpdf1` (and `O2` or higher)

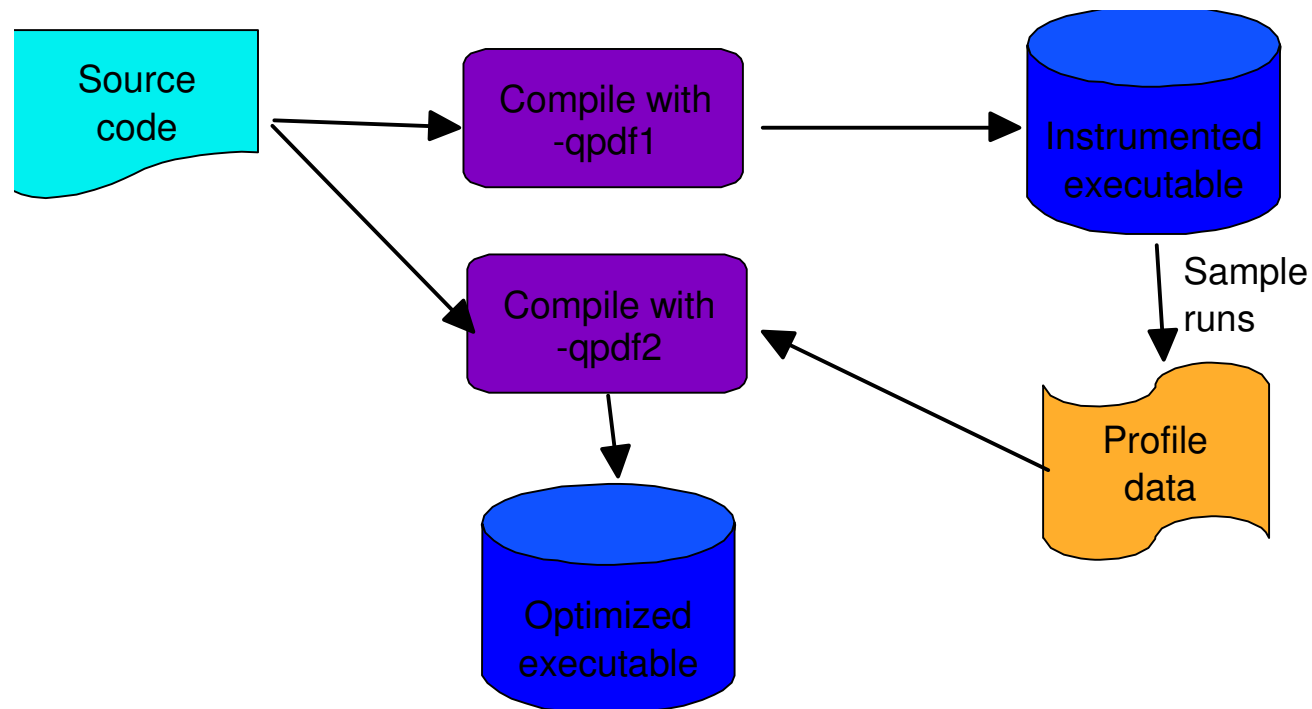
2: Run your executable with a typical / representative data set (once or more times, then the profiling info is accumulated)

3: Relink your program with the same settings as before, but replace `-qpdf1` with `-qpdf2`. In this step the information gained in step 2 is used to finetune your application. (No more profiling overhead!)

PS: Profile data is located in your working directory or in `$PDFDIR`. If the 3rd step can't locate your profile data we wasted time ...

Fortran: XLF Optimization

- **Branch Optimization:**
 - Compile with **-qpdf1** and execute on significant data
 - Compile with **-qpdf2** which will use former measurements



Fortran: XLF Optimization

large pages

-1

`-qlargepage` => prepares the objects for largepage usage
in effect if optimization level is O or higher

Not sufficient!

- A) machine must be configured to have large pages available
`/usr/samples/kernel/vmtune -g 16777216 -L 256`
(this prepares for 256 segments a 16 MB or 4 GB for large pages)
`bosboot -a`
if we want to use large page segments for shared memory:
`/usr/samples/kernel/vmtune -S 1`

Fortran: XLF Optimization

large pages

-2

- B) user must be entitled to use large pages:
chuser capabilities=CAP_BYPASS_RAC_VMM,CAP_PROPAGATE \
<lpuserid>
- C) Executable must be linked with large pages:
-blpdata flag for the link step of xlf or ld:
xlf myprog.o -o a.out -blpdata

or

be link-edited afterwards:
/usr/ccs/bin/ldedit -blpdata a.out

Fortran: XLF Optimization

large pages

-3

or

use an environment variable for control (this takes precedence over the `lpdata` flag in the XCOFF header):

```
export LDR_CNTRL=LARGE_PAGE_DATA=Y
```

(this is equivalent to `lpdata-flag` in the executable)

```
export LDR_CNTRL=LARGE_PAGE_DATA=N
```

(this overrides the `lpdata-flag` in the executable)

```
export LDR_CNTRL=LARGE_PAGE_DATA=M
```

(this requires large pages for data and heap mandatorily)

Fortran: XLF Optimization

large pages

-4

- Do I use large pages?
- `head -10 | svmon -U <userid>`

shows me the large page usage of my application

(this requires root permission ;-)

`vmstat -l`

shows me large page usage for a non-root user



Fortran: XLF Profiling

- Simple profiling support code: **-p**
 - **xlf myprogram.f -p -o myprogram**
 - **myprogram => mon.out Outputfile**
 - **prof mon.out**

- Simple profiling support code: **-pg**
 - **xlf myprogram.f -pg -o myprogram**
 - **myprogram => gmon.out Outputfile**
 - **gprof gmon.out**

Fortran: XLF Profiling

big_normal_mrci

ngranularity: Each sample hit covers 4 bytes. Time: 11592.41 seconds

%	cumulative	self	self	total		
time	seconds	seconds	calls	ms/call	ms/call	name
25.0	2892.64	2892.64				.datb [1] ESSL
8.0	3817.30	924.66				.uncompress_double_ [2]
8.0	4740.25	922.95				.addmx_ [3]
7.8	5639.56	899.31				.dab [4] ESSL
5.1	6232.11	592.55				.mxvbx_ [5]
4.6	6764.37	532.26				.mxmas_ [6]
4.5	7285.80	521.43				.fzero_ [7]
3.2	7662.18	376.38				.dtrsa [8] ESSL
3.2	8032.28	370.10				.mxvb_ [9]
3.1	8391.59	359.31				.mxml5b__ [10]
2.9	8732.25	340.66				.daxpy_ [11] ESSL
2.7	9046.44	314.19				.dmv46 [12] ESSL
2.2	9306.16	259.72				.dtrca [13] ESSL
2.2	9563.28	257.12				.locate_ [14]
1.3	9712.80	149.52				.mxvbs_ [15]
1.3	9860.51	147.71				.copmx_ [16]
1.2	9996.56	136.05				.sp_mlt_ [17]
1.2	10130.39	133.83				.mxmb4__ [18]



Compilers & Optimizations

- Fortran: XLF & C - C++: XLC
- IBM Performance Libraries
 - ESSL, MASS, WSMP, MIO



S&T Libraries :ESSL

-1

- Engineering and Scientific Subroutine Library, V3.3)
- Prog.Nr. 5765-C42
- Support for AIX 5 32bit and 64 bit kernel
- Tuning for POWER4
- Serial and threadsafe versions
- More than 400 highly optimized subroutines



S&T Libraries :ESSL

-2

- **Compatible BLAS1, BLAS2, BLAS3 implementation**

- **Linear Algebra Equations**
 - **Dense, Banded, Sparse**

- **Eigensystem Analysis**
 - **General, Real Symmetric, Complex Hermitean, Generalized Real**

- **Fourier Transforms**

- **Sorting and Searching**

- **Interpolation, Numerical Quadrature**

- **Random Number Generation**

S&T Libraries :ESSL

-3

- **New Dense Linear Algebraic Equations Subroutines (*Lapack*):**
 - General Matrix Inverse
 - Positive Definite Real Symmetric or Complex Hermitian Matrix Factorization
 - Positive Definite Real Symmetric or Complex Hermitian Matrix Multiple Right-Hand Side Solve
 - Positive Definite Real Symmetric Matrix Inverse
 - Triangular Matrix Inverse
- **New Linear Least Squares Subroutines:**
 - Linear Least Squares Solution for a General Matrix
 - Additional functionality has been added to the Packed Dense Linear Algebraic subroutines.



S&T Libraries :PESSEL

-4

- **Parallel version: PESSL Vers. 2.3**
- **Compatible calling sequences with PBLAS and Scalapack**
- **More than 100 highly tuned subroutines**

S&T Libraries: ESSL

-5

USAGE (*Limitation : integer*4*)

- Single link for 32-bit and 64-bit applications
 - `/usr/lib/libessl.a, /usr/lib/libesslsmp, /usr/lib/libpessl.a`

- **Compiler** (add `-q64` for 64-bit applications)
 - **Scalar version**
 - `xlf` or `xlc` `{-q64}` `-o executable.... -lessl`

- **OpenMP version** `{thread safe}`
 - `xlf_r` or `xlc_r` `-qsmp=omp` `{-q64}` `-o executable -lesslsmp`

- **Parallel version (MPI)** `{+OpenMP}`
 - `mpxlf_r` `{-q64 -qsmp=omp}` `-o executable ... -lpessl {-lessl}`



S&T Libraries: ESSL and LAPACK

Note:

- For users of LAPACK: simple integration of highly tuned BLAS subroutines plus some subroutines replacing the LAPACK equivalents is possible through the CCI (Call Conversion Interface) tool provided through the LAPACK site www.lapack.org

S&T Libraries: MASS

-1

- Fast transcendental functions
 - exp, pow, log
 - sin, cos, tan, sinh, cosh, tanh, atan, atan2
 - rsqrt, dnint
- Scalar and vector versions
- Different tradeoff between accuracy and performance than libm or compiler generated code
- Accuracy differences very small and usually tolerable, assume round to nearest mode
- Free for download from:
- <https://techsupport.services.ibm.com/server/mass>

S&T Libraries: MASS

-2

	PWR4 speedup	PWR4 vector speedup
sqrt	1.74	1.00
rsqrt	1.61	2.46
exp	2.12	6.29
log	1.99	9.87
sin	2.25	7.57
cos	2.30	8.14
tan	2.44	5.08
atan	0.98	
sinh	4.16	
cosh	3.91	
tanh	4.30	
dnint	3.51	5.00
x**y	2.21	

S&T Libraries: MASS

-3

- Usage:
 - `cc -o foo foo.c -lmass -lm ...` (for scalar only)
 - `cc -o foo foo.c -lmassv -lm ...` (for vector only)
 - `cc -o foo foo.c -lmassvp3 -lm ...` (for POWER3 vector only)
 - `cc -o foo foo.c -lmass -lmassv -lm` (for scalar and vector)
 - `cc -o foo foo.c -lmass -lmassvp3 -lm` (for POWER3 scalar and vector)
- Vector Usage: Examples (after installation) in `/usr/lpp/mass/libmassv.f`:

e.g.:

```
subroutine vexp(y,x,n)
  real*8 x(*),y(*)
  do 10 j=1,n
    y(j)=exp(x(j))
  10 continue
  return
end
```



S&T Libraries: WSSMP

Software package for solving large sparse system using a direct method

- <http://www.cs.umn.edu/~agupta/wsmp.html>
- serial and parallel package, in a shared&distributed memory
- WSMP is composed of two parts
 - part I replaces the older software called WSSMP
 - part II deals with the solution of general sparse systems
- Numeric schemes based on Multifrontal algorithm
- Currently, message-passing general/unsymmetrical system not supported
- Serial/Multithreaded Linux version G1 2003
- Distributed-Memory Parallel version for Linux Q2 2003



S&T Libraries: MIO

-1

-
- developed by John Bauer at the ACTC (Advanced Computing Technology Center) of the Watson Research Center at IBM
 - direct/non-direct IO
 - asynchronous/synchronous IO
 - user specified buffer(s) for prefetching
 - included in AIX Version 5.1 Bonus Pack

S&T Libraries: MIO

-2

-
- **Implementation of MOI**
 - **#define open(a,b,c) MIO_open(a,b,c,0)**
 - **#define close MIO_close**
 - **#define lseek64 MIO_lseek**
 - **#define read MIO_read**
 - **#define write MIO_write**
 - **#define ftruncate64 MIO_ftruncate**
 - **#define fstat MIO_fstat**
 - **#define fcntl MIO_fcntl**
 - **#define ffinfo MIO_ffinfo**
 - **#define fsync MIO_fsync**

 - **setenv MIO_DEFAULTS " trace/stats/mbytes , pf/async/direct/pref=4/stats/mbytes "**
 - **setenv MIO_FILES " *.DMS* [trace | **
pf/global=1/cache_size=500m/page_size=2m/norelease | trace] "
 - **setenv MIO_DEBUG ALL**



Agenda

- Fortran: XLF & C - C++: XLC

- IBM performance libraries
 - ESSL, MASS, WSMP, MIO

- Monitoring tools



Tools

□ **Monitoring**

- Monitor
- Topas
- nmon

□ **Debugging**

- dbx, pdbx
- Xldb
- Idebug

□ **Profiling**

- gprof
- Xprofiler
- Hpmcount, libhpm

Monitoring: monitor

-1

-
- **monitor** reports:
 - cpu usage
 - load average (from kernel or by using loadavgd program)
 - virtual and real memory usage (both process and file pages)
 - paging information
 - process events
 - Disk I/O / TTY I/O
 - Network activity
 - top cpu users
 - NFS operations
 - more detailed disk I/O screen (with -disk option)
 - more detailed information on multiprocessor SMP machines

 - Invocation: **monitor** [options] :
 - **monitor** [-s sleep] [-top [nproc]] [-u] [-all] [-d] [-n|-p] [-O] [-F|-M] [-o|-i] [-P pidfile]

 - More information: **monitor -help**

Monitoring: monitor

-2

```
AIX monitor v1.12: leka.hut.fi          Wed Jun 15 18:48:04 1994
Sys  5.3% Wait  0.0% User 94.7% Idle  0.0%   Refresh: 10.56 s
0%      25%      50%      75%      100%
```

```
Runnable processes  8.33 load average:  8.85,  9.01,  8.76
```

```
Memory      Real      Virtual    Paging (4kB)   Process events   File/TTY-IO
free        85.6 MB  431.6 MB    1.6 pgfaults   189 pswitch     51 iget
procs       84.7 MB  168.4 MB    0.0 pgin        338 syscall     24 namei
files       21.7 MB           0.0 pgout       32 read         0 dirblk
total      192.0 MB  600.0 MB    0.0 pgsin       23 write        42237 readch
                                0.0 pgsout      0 fork          30173 writch
DiskIO      Total Summary
read         0.0 kByte/s
write        0.0 kByte/s      Client Server NFS/s   39 xmtint    882 ttyoutch
transfers    0.0 tps              0.0   0.9 calls
active       0/11 disks           0.0   0.0 retry
                                0.0   0.0 getattr  Netw read  write
TOPdisk read write      busy   0.0   0.9 lookup  lo0   0.0   0.0 kB/s
hdisk1      0   0 kB/s  0%    0.0   0.0 read   en0   3.8  34.3 kB/s
hdisk2      0   0 kB/s  0%    0.0   0.0 write
hdisk3      0   0 kB/s  0%    0.0   0.0 other
hdisk4      0   0 kB/s  0%
```

```
leka.hut.fi      load averages:  1.05,  1.10,  1.14   Sun May 15 18:41:25 1994
Cpu states:      93.1% user,  5.0% system,  2.0% wait,  0.0% idle
Real memory:     110.9M free  68.4M user  12.6M numperm 192.0M total
Virtual memory:  466.6M free 133.4M used 600.0M total
```

```
  PID USER      PRI NICE   SIZE   RES STAT    TIME   CPU% COMMAND
 53453 mheinila 105  0    482K   616K run     95:26  94.1% prgl
   514 root      127  21    28K    8K run    8157:10  2.0% wait (kproc)
 96310 jmaki     61  0    299K   332K run       0:00   2.0% monitor
16397 amiettin 60  0    258K   88K sleep    0:03   1.0% ybiff
55547 jhi       60  0    152K   52K sleep    0:00   1.0% rlogin
12353 root      60  0    885K  220K sleep   349:49  0.0% glbd
11836 root      60  0    504K  172K sleep   306:59  0.0% llbd
```

Monitoring: Topas

- **Topas** is a « performance monitoring » tool introduced with AIX 4.3.3, and enhanced with AIX 5L.

- **Invocation: topas**

```
Topas Monitor for host: tonga
Thu Aug 8 16:43:09 2002 Interval: 2

Kernel 0.0 |
User 1.0 |
Wait 0.0 |
Idle 99.0 |#####|

EVENTS/QUEUES FILE/TTY
Cswitch 24 Readch 6282
Syscall 115 Writech 78
Reads 32 Rawin 0
Writes 0 Ttyout 63
Forks 0 Igets 0
Execs 0 Namei 3
Runqueue 0.0 Dirblk 0
Waitqueue 2.2

Interf KBPS I-Pack O-Pack KB-In KB-Out
tr0 1.1 5.4 2.9 0.3 0.8
en0 0.7 1.9 2.9 0.6 0.1

PAGING MEMORY
Faults 0 Real,MB 1023
Steals 0 % Comp 58.0
PgspIn 0 % Noncomp 34.0
PgspOut 0 % Client 0.0
PageIn 0
PageOut 0
Sios 0 PAGING SPACE
Size,MB 512
% Used 1.3
% Free 98.6

Disk Busy% KBPS TPS KB-Read KB-Writ
hdisk0 0.0 0.0 0.0 0.0 0.0
hdisk1 0.0 0.0 0.0 0.0 0.0
hdisk3 0.0 0.0 0.0 0.0 0.0
hdisk4 0.0 0.0 0.0 0.0 0.0
hdisk7 0.0 0.0 0.0 0.0 0.0

topas (231482) 1.0% PgSp: 0.5mb root
nmbd (11108) 0.0% PgSp: 0.3mb root
syncd (3916) 0.0% PgSp: 0.3mb root
ksh (47750) 0.0% PgSp: 0.3mb root
dtscreen (242580) 0.0% PgSp: 0.3mb root
dtexec (233906) 0.0% PgSp: 0.4mb root
perl5.005 (10836) 0.0% PgSp: 2.5mb root
nfsd (8586) 0.0% PgSp: 0.1mb root
gil (1290) 0.0% PgSp: 0.0mb root
X (4392) 0.0% PgSp: 9.1mb root

Press "h" for help screen.
Press "q" to quit program.
```


Monitoring: nmon

-2

```
nmon v6q [H for help]  Hostname=boemelb  Refresh=2.0secs  17:04.45
RS/6000 Details
Hardware-Type(NIM)=CHRP=Common H/W Reference Platform Bus-Type=PCI
Logical partition=No
CPU Architecture=PowerPC Implementation=630, 64 bit with 4 CPUs (4 CPUs activated)
CPU Level 1 Cache is Combined Instruction=32768 bytes & Data=65536 bytes
      Level 2 Cache size=4194304
AIX 5.1.0.37                Kernel=Multi-Processor 64 bit
uname=boemelb hostname=boemelb
```

Monitoring: nmon

-3

nmon v6q [H for help] Hostname=boemelb Refresh=2.0secs 17:04.45

Adapter I/O	read	write	xfers	Disks	Adapter	Type
scsi0 10-60	0.0	0.0 KB/s	0.0	2	N/A	
ssar	0.0	654.9 KB/s	5.1	1	N/A	
ssar	0.0	639.4 KB/s	5.0	1	N/A	
ssar	0.0	655.4 KB/s	5.1	1	N/A	
ssar	0.0	655.4 KB/s	5.1	1	N/A	
ssar	0.0	639.4 KB/s	5.1	1	N/A	
ssar	0.0	639.4 KB/s	5.0	1	N/A	
ssar	0.0	639.4 KB/s	5.0	1	N/A	
ssar	0.0	623.4 KB/s	4.9	1	N/A	
ssar	0.0	623.4 KB/s	4.9	1	N/A	
ssar	0.0	623.4 KB/s	4.9	1	N/A	
ssar	0.0	639.4 KB/s	5.0	1	N/A	
ssar	0.0	639.4 KB/s	5.0	1	N/A	
ssar	0.0	639.4 KB/s	5.1	1	N/A	
ssar	0.0	639.4 KB/s	5.0	1	N/A	
ssar	0.0	639.4 KB/s	5.1	1	N/A	
ssar	0.0	639.4 KB/s	5.1	1	N/A	
TOTALS	0.0	10230.9 KB/s	80.4	TOTAL=	10230.9	



Debugging: dbx and pdbx

- **dbx:**

- symbolic debugger for Fortran, C, C++ and Pascal
 - command line mode

- examine objects and corefiles,
 - control program-execution
 - put breakpoints into your program,
 - execute the program line by line,
 - debug using symbolic variables, set their values,...

- **pdbx:**

- parallel version of dbx

- Incorporating the necessary parallel functionalities

- Online Documentation: **man dbx, man pdbx**



Debugging: Xldb

- **Graphical debugger** for Fortran, C et C++

- **Functionality**
 - source code window (active window is active procedure)
 - Change the values of variables and registers
 - Set breakpoints
 - Step by step execution...

xldb [Option...] Program [ProgramArgument...]

- Online Documentation: **xldb -help**

Debugging: Xldb

The screenshot shows the Xldb debugger interface with the following components:

- Locals for `_main()` in `polystrm.f`**:
 - `i: +0`
 - `j: +0`
 - `m: +0`
 - `phosphate: []`
 - `compound: []`
 - `benzine: []`
- Callers**:
 - `_main()`
 - `_start()`
- Subprograms**:
 - `_main()`
 - `print() [polystrm.f]`
- Files**:
 - `polystrm.f`
- Command**:
 - Continu
 - Next
 - Step
 - Return
 - Signal
 - Edit
 - Restart
 - Exit
 - Options
- Code Window (./polystrm.f)**:

```
⇒ do i=2,21
  read(7,fmt="(20a1)") (phosphate(i,j),j=2,21)
end do

where (phosphate.ne.' ') ! i.e., if there's compound in the
● compound=1
  elsewhere
  compound=0
end where

do m=1,20
● call print
```
- Breakpoints**:
 - `main [line 4 in polystrm.f]`
 - `main [line 17 in polystrm.f]` (highlighted in cyan)
 - `main [line 46 in polystrm.f]`
 - `main [line 47 in polystrm.f]`

Debugging: idebug

- **Graphical debugger**
 - All standard debugging functionalities
 - client/server application : local or via TCP/IP

- Local machine (p690 par ex): start “debug engine”
 - **irmtdbgc -qhost=<nom_machine_remote ou adr_IP>**

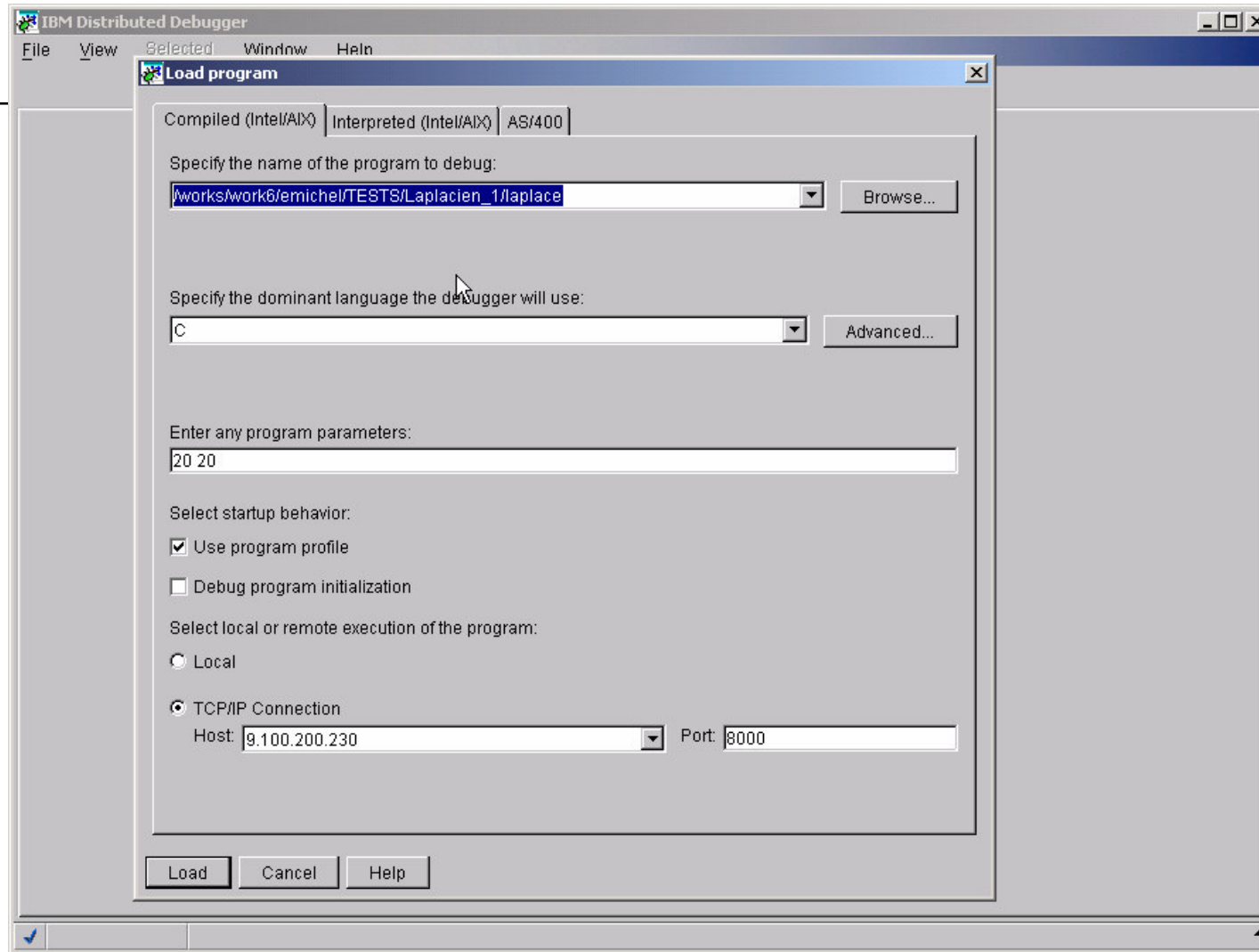
- Remote machine (PC Windows par ex): start client :
 - **Start -> Run -> idebug**

<http://www-106.ibm.com/developerworks/library/debug/>

<http://scv.bu.edu/SCV/IBMSP/TOOLS.html>

<http://www.cineca.it/manuali/sp3/idebug/>

Debugging: idebug



Debugging: idebug

The screenshot displays the IBM Distributed Debugger interface. The main window shows the source code of a C program named 'laplace.c'. The code is as follows:

```
97     vecteur B;  
98     matrice A;  
99     /* lecture des arguments */  
100 →  if (argc!=3) {  
101         printf("Syntaxe : %s Nx Ny\n",argv[0]); exit(1);  
102     }  
103     nx=atoi(argv[1]);  
104     ny=atoi(argv[2]);  
105     printf("\t\t Resolution d'un laplacien 2D avec %d *  
106     /* creation des tableaux */  
107     X=CreeVecteur(nx,ny);  
108     B=CreeVecteur(nx,ny);  
109     A=CreeMatrice(nx,ny);  
110  
111     /* assemblage, calcul du second membre et CI */  
112     Assemblage(A,nx,ny);  
113     SecondMembre(B,nx,ny);  
114     Climite(A,B,nx,ny);  
115 ●  GaussSeidel(X,A,B,nx,ny);  
116     WriteSol(X,nx,ny);  
117     /* fin */  
118     FreeVecteur(X,nx,ny);  
119     FreeVecteur(B,nx,ny);  
120     FreeMatrice(A,nx,ny);  
121     return (0);  
122 }  
123  
124
```

The debugger interface includes a menu bar (File, View, Selected, Debug, Source, Modules, Storage, Window, Help), a toolbar with various icons, and a status bar at the bottom that reads "Debugger ready...".

On the left side, there is a tree view showing the project structure:

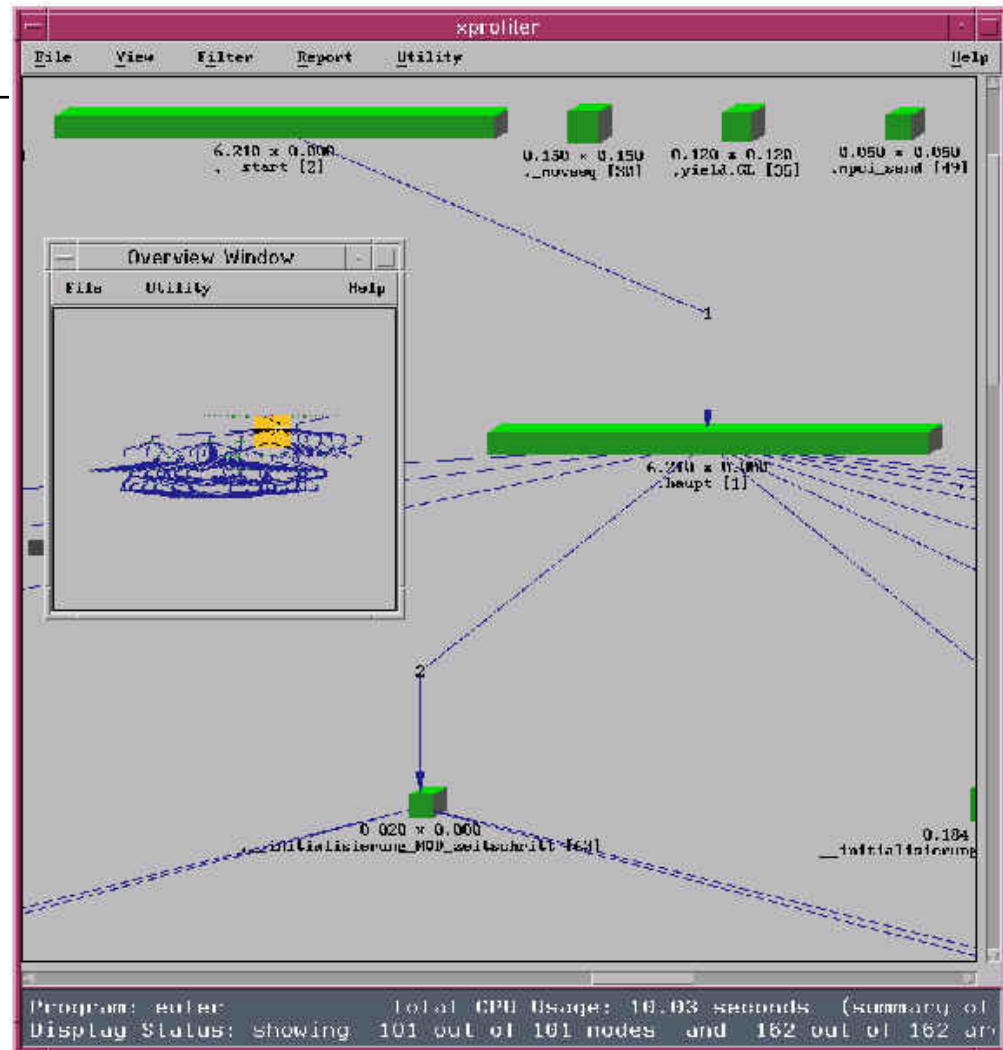
- laplace
 - laplace.c
 - Affiche
 - Assemblage
 - Climite
 - main
 - SecondMembre
 - WriteSol
- util.c
 - CreeMatrice
 - CreeVecteur
 - FreeMatrice
 - FreeVecteur
 - GaussSeidel
 - Norme

Below the tree view, there are tabs for "Monitors", "Locals", "Storage", and "Registers". The "Storage" tab is active, showing a table of memory addresses and their contents:

Flat	00	01	02	03	HEX and Character
20000100	40	28	23	29	@(#)
20000104	36	31	09	31	61□1
20000108	2E	31	34	20	.14
2000010C	20	73	72	63	src
20000110	2F	62	6F	73	/bos

Outils Profiling: Xprofiler

- ❑ Bundled with IBM Parallel Environment
- ❑ Compile with
 -g -pg -qfullpath
- ❑ Shows dynamic call tree
- ❑ Height of box is time spent in the subroutine
- ❑ Width of box is time spent in subtree below
- ❑ Links show # Calls
- ❑ Small Overview Window for navigating around in the tree
- ❑ Don't forget:
 - `xrdb -load /usr/lib/X11/app-defaults:Xprofiler`





Profiling: hpmcount

- Utility for performance measurement of applications
- Starts a program and at the end of the execution, provides a summary output with:
 - wall clock time
 - resource usage statistics
 - hardware performance counters information
 - derived hardware metrics

Outils Profiling: hpmcount

□ *poe* hpmcount [-o <file name>] [-n] [-g <group>]
 <program>

OR

□ hpmcount [-h] [-l] [-c]

where:

- <program> program to be executed
- -h displays this help message
- -o <file name> output file name
- -n no hpmcount output in stdout when "-o" flag is used
- -g <group> PM_API group number
- -l list groups
- -c list counters and events

□ Set group of hardware counters

HPM_EVENT_SET=[1-60]

Hpmcount output

Resource Usage Statistics

```
Total amount of time in user mode      : 193.350000 seconds
Total amount of time in system mode    : 2.490000 seconds
Maximum resident set size              : 56760 Kbytes
Average shared memory use in text segment : 147559860 Kbytes*sec
Average unshared memory use in data segment : 940654384 Kbytes*sec
Number of page faults without I/O activity : 13935
Number of page faults with I/O activity  : 0
Number of times process was swapped out  : 0
Number of times file system performed INPUT : 0
Number of times file system performed OUTPUT : 0
Number of IPC messages sent             : 0
Number of IPC messages received          : 0
Number of signals delivered              : 0
Number of voluntary context switches     : 1885
Number of involuntary context switches    : 21536

PM_CYC (Cycles)                        : 72249390399
PM_INST_CMPL (Instructions completed)   : 97146499224
PM_TLB_MISS (TLB misses)                : 32100883
PM_ST_CMPL (Stores completed)           : 9353911991
PM_LD_CMPL (Loads completed)            : 20041093389
PM_FPU0_CMPL (FPU 0 instructions)       : 4524355614
PM_FPU1_CMPL (FPU 1 instructions)       : 1458257288
PM_EXEC_FMA (FMAs executed)             : 1639958421

Average number of loads per TLB miss    : 624.316
Load and store operations                : 29395.005 M
Instructions per load/store              : 3.305
MIPS                                      : 480.564
Instructions per cycle                    : 1.345
HW Float point instructions per Cycle   : 0.083
Floating point instructions + FMAs      : 7622.571 M
Float point instructions + FMA rate     : 37.707 Mflip/s
FMA percentage                           : 43.029 %
Computation intensity                    : 0.259

PM_CYC (Cycles)                        : 324686119
PM_INST_CMPL (Instructions completed)   : 886011552
PM_LD_DISP (Loads dispatched)          : 282527993
PM_LD_MISS_L1 (Load misses in L1)      : 4598688
PM_ST_DISP (Stores dispatched)         : 7596408
PM_ST_MISS (Store misses in L1)        : 237357
PM_LSU_IDLE (Load store unit idle)     : 73663035
PM_TLB_MISS (TLB misses)                : 300078

Average number of loads per TLB miss    : 941.515
Total loads and stores                   : 290.124 M
Instructions per load/store              : 3.054
Average number of loads per load miss   : 61.437
Average number of stores per store miss : 32.004
Average number of load/stores per D1 miss : 59.992
L1 cache hit rate                        : 98.333 %
% Cycles LSU is idle                     : 22.687 %
Cycles per instruction                   : 0.366
Instructions per cycle                    : 2.729
```



Outils Profiling: libhpm_r

- Instrumentation library developed at ACTC for performance measurement of Fortran, C, and C++ applications

- For each instrumented point in a program, libhpm provides:
 - total count
 - total duration (wall clock time)
 - hardware performance counters information
 - hardware derived metrics

- Provides resource usage statistics for the total execution of the instrumented program



Outils Profiling: libhpm_r

- Supports:
 - OpenMP and threaded programs
 - Multiple instrumentation points
 - Nested instrumentation
 - provides exclusive duration for the outer points
 - Multiple calls to an instrumented point.
 - average and standard deviation is provided when an instrumented point is activated multiple times.



Outils Profiling: libhpm_r

- Link with **-lhpm_r -lpmapi -lm**
- Visualization tool: **hpmviz**

- Hardware Events Selection
 - Set of hardware events to be used can be selected via the environment variable `HPM_EVENT_SET`
- Environment variables:
 - `HPM_OUTPUT_NAME`, `HPM...`



Outils Profiling: libhpm_r

□ Code instrumentation C

declaration:

```
#include "libhpm.h"
```

use:

```
hpmInit( taskID, "my program" );
```

```
hpmStart( 1, "outer call" );
```

```
do_work();
```

```
hpmStart( 2, "computing meaning of life" );
```

```
do_more_work();
```

```
hpmStop( 2 );
```

```
hpmStop( 1 );
```

```
hpmTerminate( taskID );
```

Outils Profiling: libhpm_r

□ Code instrumentation Fortran

declaration:

```
#include "f_hpm.h"
```

use:

```
call f_hpminit( taskID, "my program" )
```

```
call f_hpmstart( 1, "Do Loop" )
```

```
do ...
```

```
  call do_work()
```

```
  call f_hpmstart( 5, "computing meaning of life" );
```

```
  call do_more_work();
```

```
  call f_hpmstop( 5 );
```

```
end do
```

```
call f_hpmstop( 1 )
```

```
call f_hpmterminate( taskID )
```

Outils Profiling: libhpm_r

□ Code instrumentation Multi Threads

```
!$OMP PARALLEL
!$OMP&PRIVATE (instID)
    instID = 30+omp_get_thread_num()
    call f_hpmtstart( instID, "computing meaning of life" )
!$OMP DO
    do ...
        do_work()
    end do
!$OMP END DO
    call f_hpmtstop( instID )
!$OMP END PARALLEL
```



Backup



New Options and Directives

- **-qxlines** - debug lines in fixed source form
- **-qsmp=noopt**
 - performs minimal optimization in parallelization to facilitate debugging capabilities
- allow **-qhot** on compilation unit
- snapshot directive
 - make variables visible in debugger inside parallel environment
- collapse and subscriptorder directives
 - array layout specifications for performance



Other XLF Options and Directives

- ❑ **-qlargepage** - executed in large memory page environment
- ❑ **-qsmallstack** - minimize stack usage
- ❑ **-qunwind** - behaviour of volatile registers during procedure calls
- ❑ **-qsclk=centimicro** - resolution in `SYSTEM_CLOCK` intrinsic
- ❑ **-q[no]essl** - using ESSL for Fortran 90 intrinsics



Fortran External Names

- Rename system functions (ld Options): **-brename**
 - **-brename:.dttime,.dttime_**
- Use "**-qextname**" to add "_" to name

```
xlf -qextname ...
```

```
...
```

```
call flush( )
```

```
call dttime( )
```

```
or
```

```
...
```

```
call flush_()
```

```
call dttime_( )
```



XLF Environment variables

- Temporary Files
 - TMPDIR points to the directory used by the compiler for its own working files. 'scratch' files will also go in this directory
 - **export TMPDIR=/user/scratch**

- Libraries:
 - LIBPATH gives the search paths for libraries (runtime)
 - **export LIBPATH=my_directories:others:/usr/lib**

- Configuration file: /etc/xlf.cfg

- Default variables in: /etc/environment



XLF Runtime variables

- **XLFRTEOPTS**
 - XLFRTEOPTS=namelist={old|new}
 - XLFRTEOPTS=buffering={enable|disable_all...}
 -

- **XLSMPOPTS** (OpenMP control runtime for automatic parallelization)
 - schedule, parthds, stack, spins, yield



Fortran: XLF Optimization

Fortran Directives for Power4

□ Prefetch directives

■ **PREFETCH_BY_LOAD**

- generates a load byte and zero (lbz) instruction. Useful for data that may be loaded or stored later (avoid store misses)
- inserts extra load instructions in L1 which can penalize reused data in L1
- do i=1,n
!IBM* PREFETCH_BY_LOAD(x(i+17))
 x(i) = s
enddo

■ **PREFETCH_FOR_LOAD, PREFETCH_FOR_STORE**

- generates a cache line touch instruction (dcbt and dcbtst). Cause a cache line to be loaded into L1.
- do not have to wait for the cache line to be loaded for completion (as PREFETCH_FOR_LOAD)



Fortran Directives for Power4

- Loop-related directives
 - INDEPENDENT
 - indicates that the iterations of the loop can be performed in any order
 - UNROLL(n)
 - indicates that compiler may unroll following loop to depth n (see next page)
 - CNCALL
 - indicates that no dependencies between iterations exist for procedures called
 - PERMUTATION
 - would be used where the integer array was being used to index another way

Outer Loop Unroll

```
DO I = 1, N
  DO J = 1, N
    s = s + X(J)*A(J,I)
  END DO
END DO
```

```
DO I = 1, N, 4
  DO J = 1, N
    s = s + X(J)*A(J,I+0)
      + X(J)*A(J,I+1)
      + X(J)*A(J,I+2)
      + X(J)*A(J,I+3)
  END DO
END DO
```

Better use of hardware resources



Fortran Directives for Power4

- Cache and other directives
 - **CACHE_ZERO**
 - **!IBM* CACHE_ZERO(x(i))**
 - generates a dcbz instruction that zeros a cache line without needing to first load the cache line from the memory
 - could be very efficient for initialization array.
 - modifies the whole cache line
 - it is essential to check the location of the element with
 - **MOD (LOC(x(i)), 128)**
 - **LIGHT_SYNC**
 - synchronization multiple processors without waiting for a confirmation from each processor.
 - improves the performance
 - used to ensure that a thread does not access the value until an other thread has updated.

Exemple: Stream for Power4

Copy loop

- **Standard version** for copy

```
do j=1,n
    c(j) = a(j)
Enddo
```

- **Tuned version:** CACHE_ZERO directive + unrolling 3 for prefetching

```
NP = (n-AHEAD)/(3*16); N2 = 16*NP
```

```
do i=1,N2-AHEAD,16
```

```
!IBM* CACHE_ZERO ( c(j+AHEAD) )
```

```
!IBM* CACHE_ZERO ( c(j+N2+AHEAD) )
```

```
!IBM* CACHE_ZERO ( c(j+2*N2+AHEAD) )
```

```
do i=j,j+15
```

```
c(i) = a(i)
```

```
c(i+N2) = a(i+N2)
```

```
c(i+2*N2) = a(i+2*N2)
```

```
enddo
```

```
enddo ...
```

Exemple: Stream for Power4

- Performances in Mbytes/second (XLF 7.1.1.2 -qarch=pwr3)

	Based (4 KB pages)	Based (16 MB pages)	tuned (4 KB)	tuned (16 MB pages)
copy	1825	2132	3530	6400
scale	1818	2132	3320	6293
add	2136	3345	3158	6712
triad	2153	3347	3158	6676



Fortran: XLF Parallelization

- Automatic Parallelization:
 - **-qsmp=auto:**
 - parallelize loops
 - **-qsmp -qreport= smplist:**
 - produce a parallelization report

- User Parallelization:
 - **-qsmp=omp:**
 - Understand OpenMP directives and parallelize only OpenMP constructions