Advanced Client Technology (A.C.T!) Centers

Best Practices—Session ONE
September 27, 2007

Sept 7, 2007

Advanced Client Technology (A.C.T!) Centers

HPC Best Practices

– Porting and debugging on System p

Xinghong He

HPC Benchmarking Support

Advanced Client Technology (A.C.T!) Centers

IBM Systems and Technology Group

Sept 7, 2007

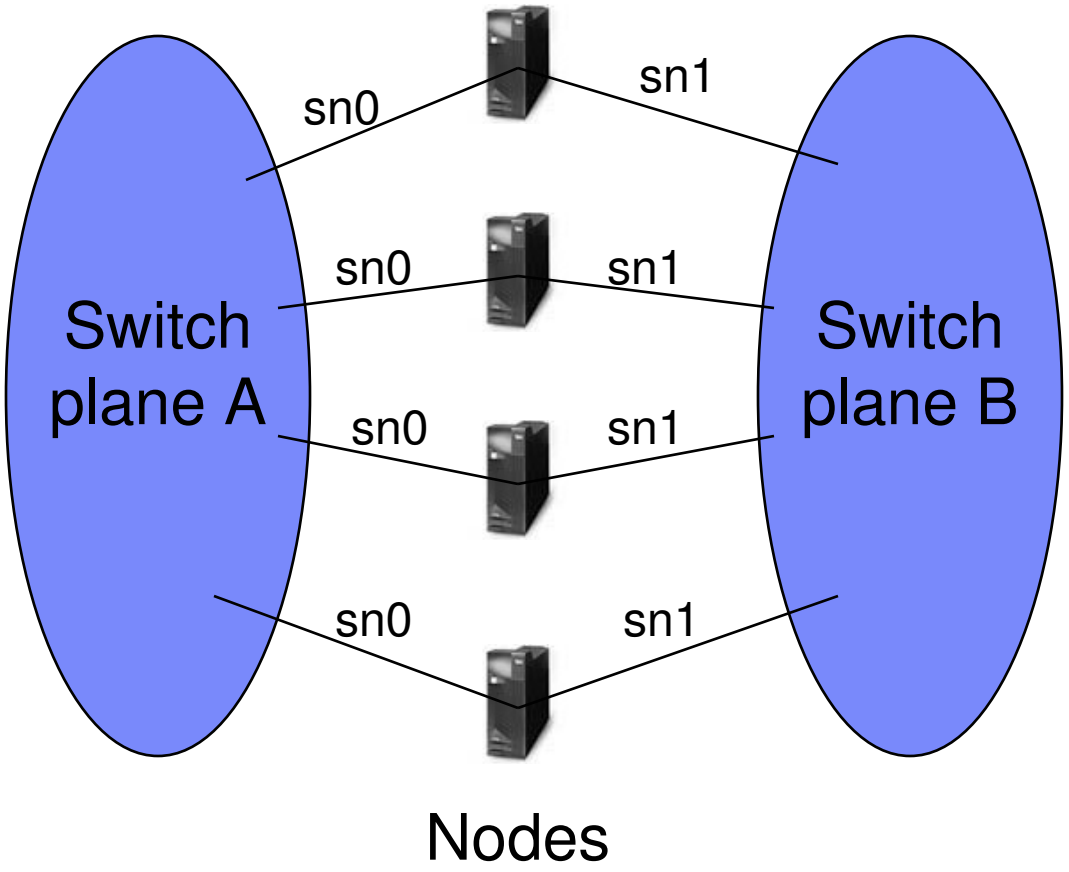
Agenda

- **System environment**
  - HW and SW environment for this presentation
- **Application information gathering**
  - What we need for a porting engagement
- **Porting/debugging with IBM compilers**
  - How to make use of compiler capabilities
- **BLAS, LAPACK, and ESSL libraries**
  - Their relationships regarding compatibility
  - Things to consider and check when migrating

System environment

- **HW: Power5+ p575 cluster with HPS**
  - 160 dual core compute nodes – 2560 cores in total
  - IBM 9118-575
  - 1.9GHz, 64KB L1I, 32KB L1D, 1920KB L2, 36MB L3
  - 64GB (1GBx64) DDR2 Memory
  - Two GPFS file systems
    - Two single core servers (8-way 2.2GHz)
    - Four dual core servers (16-way 1.9GHz)
  - Dual link, dual plane HPS

Dual plane dual link switch network



High Availability

(Fail over)

High Performance

(Message striping)

System environment

- **SW**
  - AIX 5300-05-05
  - XLF 10.1.0.3, VACPP 8.0.0.12
  - ESSL 4.2.0.4, PESSL 3.2.0.0
  - POE 4.3.0.4
  - LL 3.4.0.3
  - PPE.XPROFILER 4.1.0.0
  - CSM 1.5.1.2
  - GPFS 3.1.0.9
  - PMAPI 5.3.0.50

Application information gathering

Must have

- Source code
- Makefiles, build instructions and/or scripts
- Run scripts and/or instructions
- Sample input data
- Correctness criteria if not clear from output files

Nice to have

- Ported systems (HW, OS, compilers etc)
- Application docs on porting and tuning
- Sample output files

Porting/debugging with IBM compilers

- Addressing space: 32-bit or 64-bit
- Sizes of basic data types: i4 or i8, r4 or r8, pointers
- Precisions of constants
- Pre-processing and Fortran suffix rules
- Name mangling – mixed-language applications
- Array bound checking
- Un-initialized variables handling
- Compiler differences in dealing with language extensions (non-standard or not-specified)
- Run-time environments

Addressing space: 32-bit or 64-bit

- **Not to be confused with**
  - Processor family (p3/p4/p5/p6 are all 64-bit)
  - OS Kernel (32-bit kernel vs 64-bit kernel)
  - Floating point number precision (single precision vs double precision)
- **Controlled by options –q32, –q64, and env OBJECT_MODE**
  - Default is 32 bit
  - Option -q overrides env OBJECT_MODE
  - must use the same mode for both compiling and linking
  - Prefer OBJECT_MODE in many cases
- **No need for separate lib/ and lib64/**
  - AIX allows 32-bit and 64-bit objects coexist in the same archive

Addressing space: 32-bit or 64-bit

- **My app needs large memory >2GB**
  - Use –q64 without –bmaxdata and –bmaxstack options
  - history: 32-bit default data and stack are small
- **My app does not build with –q64**
  - Set OBJECT_MODE=64 and try again
  - This takes care of library operations ar, ranlib, nm etc
- **My new 64-bit app has run-time error (segmentation fault etc)**
  - Check pointers (64 bits), integers (32 bits), long (64 bits)

Addressing space: 32-bit or 64-bit

- **To check if an executable is 32-bit or 64-bit**
  - `$ file a.out`
- **Some commands (not –q32|–q64)**
  - `ld [-b32|-b64]`
  - `ar [-X32|-X64|-X32_64]`
  - `nm [-X32|-X64|-X32_64]`
  - `ranlib [-X32|-X64|-X32_64]`
  - `dump [-X32|-X64|-X32_64]`

Sizes in bytes of basic C data types

Data types	32-bit					64-bit				
	GNU	PSC	Intel	PGI	XLC	GNU	PSC	Intel	PGI	XLC
char	1	1	1	1	1	1	1	1	1	1
short	2	2	2	2	2	2	2	2	2	2
int	4	4	4	4	4	4	4	4	4	4
long	4	4	4	4	4	8	8	8	8	8
long long	8	8	8	8	8	8	8	8	8	8
float	4	4	4	4	4	4	4	4	4	4
double	8	8	8	8	8	8	8	8	8	8
long double	12	12	12	8	8	16	16	16	8	8
pointer	4	4	4	4	4	8	8	8	8	8

Default sizes could be changed by compiler options.

Watch for int, long, long double, pointer, and Fortran integers.

XLF options for data type sizes

- **-q32 - pointers, loc() are 4 bytes**
- **-q64 - pointers, loc() are 8 bytes**
- **-qintsize={2|4|8}**
  - Affect default integer and logical
  - Intrinsic functions are supported
- **-qdpc[=e] | -qnodpc**
  - double precision constant
  - dbl_var=**1.000000002000**

    this "2" will be lost without –qdpc
  - Even worse case: call sub(a,1.0)
- **-qrealsize={4|8}**
  - Similar to –qintsize option, affect default sizes
  - -qrealsize=8 overrides –qdpc
  - -qrealsize=8 promotes REAL
- **-qautodouble={none|dbl4|dbl4pad|dbl8|dbl8pad|dbl|dblpad}**
  - Mostly dbl4 or dbl4pad
  - Overrides –qrealsize
  - dbl4 promotes REAL, REAL*4
  - use also at link stage

Pay particular attention to external libraries such as ESSL, PESSL, LAPACK, MPI

Pre-processing and Fortran suffix rules

- **Pre-processing**
  - Option -D is for debugging purpose ==-qdlines
  - Use -WF,-D*macro1*,-D*macro2*,…
    - No space in between
    - **Option –d** can be used to save the processed source file (**Warning**: Do a list of files before using this option to prevent file overwriting)
  - Add -qsuffix=cpp=*suffix* if source files are not .F90 or .F
  - File “a.F90” implies Fortran 90 code and “a.F” implies Fortran 77 code, not just free format and fixed format.

Pre-processing and Fortran suffix rules

- **XL Fortran suffix rules**
  - No need to add –qsuffix for files of the suffixes: .f .f90 .F .F90 New
  - It's the combinations, not the compiler commands (xlf, xlf90), nor the file suffixes, which determine the file type (see table on the right).
  - Fixed or free format is implied
  - Behavior can be changed by explicit options such as –qfixed, -qfree, -qsave, -qnosave etc.

Fortran 90 and 77 codes as determined by combinations of commands and file suffixes.

	xlf	**xlf90**
a.f	F77	F90
a.f90	F90	F90
a.F	F77	F90
a.F90	F90	F90

Fortran name mangling

- Fortran compilers often change some symbols (functions, subroutines, common blocks etc) internally.
- Different vendors (compilers) do this differently – no standard.
- Not an issue if app is not mixed-language (C, Fortran)
- Could be a source of headache for mixed-language apps involving multiple packages (FFTW, netCDF, HDF, MPICH etc).
- Typical error

```
xlf flush.f  #flush.f contains call flush(12)
ld: 0711-317 ERROR: Undefined symbol: .flush
```

Fortran name mangling - **default**

Compilers	suba	subb_	Default option
g77/pathf90	suba_	subb___	[-funderscoring -fsecond-underscore]
gfortran	suba_	subb__	[-funderscoring -fno-second-underscore]
ifort	suba_	subb__	[-us -assume no2underscores]
pgf77	suba_	subb__	[-Munderscoring -Mnosecond_underscore]
xlf	suba	subb_	[-qnoextname]

Fortran name mangling – **no underscore**

Compilers	**suba**	**subb_**	**Options**
g77/pathf90	suba	subb_	-fno-underscoring
gfortran	suba	subb_	-fno-underscoring
ifort	suba	subb_	-nus
pgf77	suba	subb_	-Mnounderscoring
xlf	suba	subb_	default

Fortran name mangling – **one underscore**

Compilers	**suba**	**subb_**	**Options**
g77/pathf90	suba_	subb__	-funderscoring -fno-second-underscore
gfortran	suba_	subb__	default
ifort	suba_	subb__	default
pgf77	suba_	subb__	default
xlf	suba_	subb__	-qextname

Fortran name mangling – **one and two underscores**

Compilers	**suba**	**subb_**	**Options**
g77/pathf90	suba_	subb___	default
gfortran	suba_	subb___	-funderscoring -fsecond-underscore
ifort	suba_	subb___	-us -assume 2underscores
pgf77	suba_	subb___	-Munderscoring -Msecond_underscore]
xlf	suba	subb___	N/A, use -brename

XLF options handling names

- **-qextname[=*name1*[:*name2*…]] | -qnoextname**
  - *name1*→ *name1_*, *name2*→*name2_*,…
  - -qextname without a name list will convert all
  - **compile time**, no dots involved
  - -qextern=*name* is totally different. (external vs. intrinsic)
- **-brename:*old_name,new_name***
  - one pair per option
  - often starts with a dot→*.old_name,.new_name*
  - no space around
  - **link time**
- **-qextname and -brename can be combined**

Inter language calls (C and Fortran)

- **XLF converts symbols to lower case by default**
  - Use option –U or directive @PROCESS=MIXED to preserve case
  - Intrinsic functions must be in lower case if –U is used
- **XLF always call-by-reference while C can do call-by-reference and call-by-value**
  - Use %VAL, %REF to change it
  - Example: iptr = malloc(%VAL(n))

XLF name mangling example

```
$ cat  a.f
      call sub
      end
      subroutine suba
      print *, 'suba called'
      end
      subroutine sub_
      print *, 'sub_ called'
      end
```

- $ xlf a.f
  - will fail because no sub
- $ xlf -qextname a.f
  - will also fail because both sub and sub_ get an underscore
- $ xlf -qextname=sub a.f
  - will call sub_
- $ xlf -brename:.sub,.suba a.f
  - will call suba

Debugging with XLF

- **Tools are a great help for debugging**
  - idebug, totalview, dbx, gdb
- **XLF compiler can do some work very easily and effectively**
  - Name mangling
  - Sizes of basic data types
  - Array bound checking
  - Un-initialized variables and arrays
  - Floating point exception catching

Array bound checking with XLF

- **Compile code with `-C -g` options**
  - **`-C`** turns on checking at compile time and run-time
  - Some errors are caught at compile time, some run-time
  - **`-g`** helps with line number of the error (debugger), could be omitted or replaced by `-qlinedebug`.
- **Run the code**
- **Run dbx or gdb to find the line of error**
- **Problem: legacy code dummy argument arrays**

```
subroutine suba (n, a)
dimension a(1)      ! actually meant for a(n)
```

Array bound checking with XLF - Example

```
      dimension a(3)
      do i = 0, 4
         a(i) = i
      end do
      print *, a
d     print *, a(0), a(4)
      end
```

Compile time or run-time?
It may change.

This line will be caught at run-time if compiled with –C -g

This line will be caught at compile time with options –C –g -qdlines

Array bound checking with XLF - Example

```
$ xlf  -C -g bound.f
$ a.out
Trace/BPT trap(coredump)


$ dbx a.out
Type 'help' for help.
[using memory image in core]
reading symbolic information ...


Trace/BPT trap in _main at line 3
   3            a(i) = i
```

```
$ xlf -qdlines -C -g bound.f
"bound.f", line 6.16: 1516-023 (S)
   Subscript is out of bounds.
"bound.f", line 6.22: 1516-023 (S)
   Subscript is out of bounds.
** _main   === End of Compilation 1
   ===
1501-511  Compilation failed for file
   bound.f.
```

Array bound checking and dummy arguments

```
subroutine aa(n,a)
dimension a(1)
do i = 1, n
   print *, a(i)
enddo
end
```

This line will core dump if array bound checking is turned on (xlf -C).

Undesirable, but ...any other way except changing the code a(1) to a(n) ?

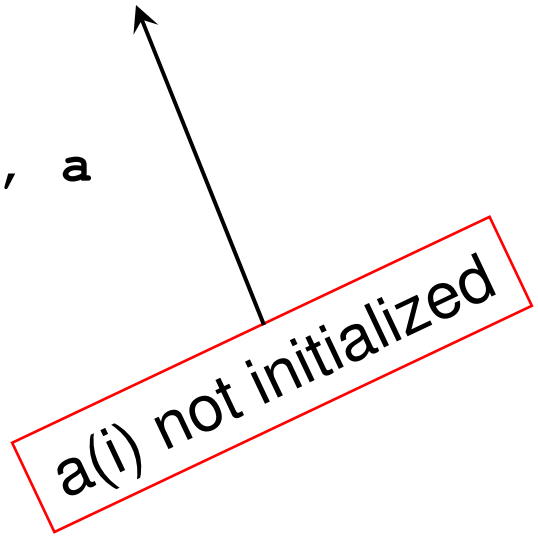
Find un-initialized variables

1. **Compile with `-g -qnosave -qinitauto=FF -qflttrap=nanq`**
   - Every option has its role
     - -qsave will always set –qinitauto=0
     - -qflttrap=nanq will trap the error, not letting go with nanq
   - Will catch (cause core dump) un-initialized REAL*4 and REAL*8, including COMPLEX
   - Replace FF by 7FBFFFFF to catch REAL*4 only
2. **Run the code followed by dbx or gdb**

-qinitauto=0 may be desirable in most cases

Un-initialized variables - Example

```
real a(3)
do i = 1, 3
   a(i) = a(i) + 100
enddo
print *, a
end
```




```
$ xlf90 -g -qinitauto=FF -
  qflttrap=nanq tmp.f
$ ./a.out
```

Trace/BPT trap(coredump)

```
$ dbx a.out
```

…

Trace/BPT trap in _main at line 3

3 a(i) = a(i) + 100

Fortran90 derived data types - example

```
subroutine sub(myname)
type name
!sequence
character(20) lastname
character(10) firstname
character(1) initial
end type name
type (name) myname
print *, myname
end subroutine
```

- XLF (v10.1 and v11.1) requires the explicit sequence statement – Reason: dummy argument.
- All the following do not require it. No warning message too.
  - gfortran 4.0.2
  - Intel ifort 9.1
  - Pathscale pathf90 2.5
  - PGI pgf90 6.2.5

More floating point exception handlings

- **-qflttrap=ov:und:inv:zero:nanq:en**
  - The last suboption “en” is needed. Otherwise the user needs to implement exception handler
  - -qflttrap alone means

    -qflttrap=inv:inex:ov:und:zero

Run-time environments

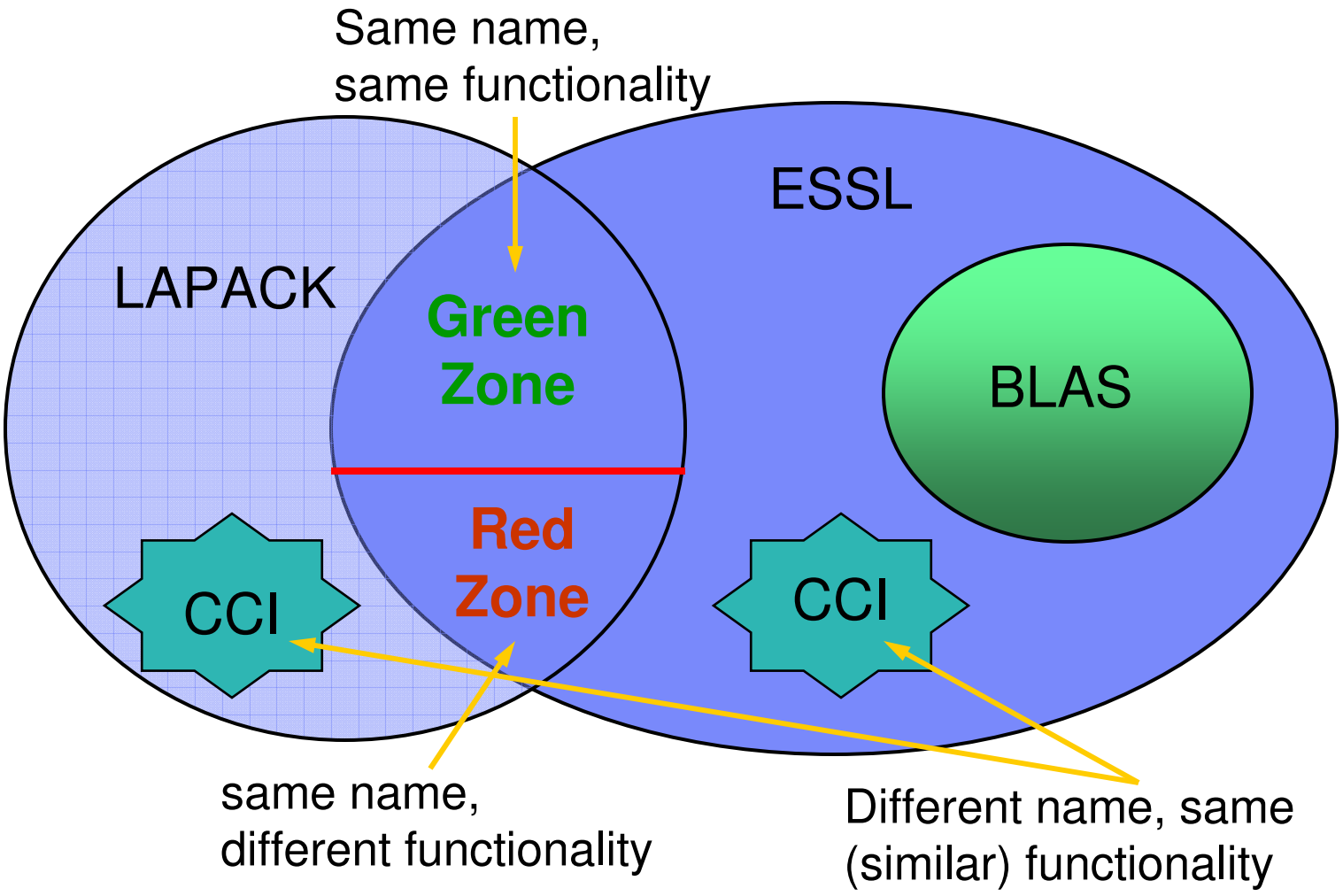
- **LIBPATH**
  - for shared libs, not normally used
- **TMPDIR**
  - for scratch files
  - default is /tmp
- **PDFDIR**
  - if doing PDF runs

- **XLFRTEOPTS**
  - XLF run-time
- **XLSMPOPTS**
  - for SMP code
- **OMP_NUM_THREADS**
  - for OpenMP code
  - Recommend to set to 1 in .cshrc file to avoid surprise

Run-time environments

- **Little endian to big endian**
  - export XLFRTEOPTS=ufmt_littleendian=*list_of_units*
  - *list_of_units* is a coma-separated list of Fortran I/O units
  - example: -7,11,13,15-20,70-
  - UNFORMATTED files only
  - Handles REAL*4, REAL*8 appropriately, not REAL*16 data, not derived type data.
- **I/O buffering**
  - export XLFRTEOPTS=buffering={enable|disable_preconn|disable_all}
  - preconn I/O units are 0, 5, 6
  - Performance (enable) and mixed language apps (disable)
- **NAMELIST**
  - XLFRTEOPTS=namelist={new|old}

BLAS, LAPACK, ESSL



LAPACK – CCI – ESSL

- **CCI stands for Call Conversion Interface**
  - A collection of wrappers of LAPACK routines using ESSL.
  - Help LAPACK users to use ESSL without modifying their source codes.
  - Available from netlib.
  - Latest version 1.2, 2000-12-07.
  - Out dated. Only 1, out of 30, is not in the Green zone (ESSL 4.2).
  - ***There are still candidates for CCI.***

Call Conversion Interface (CCI 1.2), netlib

cgetrf.f	dgetrs.f	dpptri.f	**spbtrf.f**	stptri.f
cgetrs.f	dpbtrf.f	dtptri.f	spotrf.f	strtri.f
cpotrf.f	dpotrf.f	dtrtri.f	spotri.f	zgetrf.f
cpotrs.f	dpotri.f	sgetrf.f	spotrs.f	zgetrs.f
dgetrf.f	dpotrs.f	sgetri.f	spptrf.f	zpotrf.f
dgetri.f	dpptrf.f	sgetrs.f	spptri.f	zpotrs.f

All are now directly available from ESSL 4.2 except spbtrf.f

LAPACK3.1.1 routines included in ESSL4.2

Single	Double	Single	Double	Single	Double
SGESV CGESV	DGESV ZGESV	SGETRF CGETRF	DGETRF ZGETRF	SGETRS CGETRS	DGETRS ZGETRS
SPPSV CPPSV	DPPSV ZPPSV	SGETRI CGETRI	DGETRI ZGETRI		DGEQRF
SPOSV CPOSV	DPOSV ZPOSV	SPOTRI CPOTRI SPPTRI	DPOTRI ZPOTRI DPPTRI		DGELS
SPOTRF CPOTRF SPPTRF CPPTRF	DPOTRF ZPOTRF DPPTRF ZPPTRF	SPOTRS CPOTRS SPPTRS CPPTRS	DPOTRS ZPOTRS DPPTRS ZPPTRS	STRTRI STPTRI CTRTRI CTPTRI	DTRTRI DTPTRI ZTRTRI ZTPTRI

For these routines, always link ESSL before LAPACK for performance.

Name Conflicting in ESSL4.2 and LAPACK3.1.1

Single	Double
SGEEV CGEEV	DGEEV ZGEEV
SPPEV CHPEV	DSPEV ZHPEV
SSPSV CHPSV	DSPSV ZHPSV
SGEGV	DGEGV
SSYGV	DSYGV

Alert !

- **These routines have the same name in ESSL and LAPACK, but they function differently.**
- **In case any of them are used, link order matters for correctness !**

LAPACK routines to ESSL - Unofficial

LAPACK wrapper	Calls ESSL	Note
dsyev	**dspev**	Eigenvalues and, optionally, the eigenvectors of a real symmetric matrix. **dspev is in Red Zone.**
dsyevx	**dspev**	
zheev	**zhpev**	Eigenvalues and, optionally, the eigenvectors of a complex hermitian matrix**. zhpev is in Red Zone.**

Link the wrappers with ESSL, not LAPACK

LINPACK routines to ESSL - Unofficial

LINPACK wrapper	Calls ESSL	Note
cgefa	**cgetrf**	General matrix factorization. **SMP, in Green Zone.**
dgefa	**dgetrf**	
cgesl	**cgetrs**	General matrix multiple right-hand side solve. **SMP, in Green Zone.**
dgesl	**dgetrs**	

Summary

- **Understand compiler differences**
- **Make best use of compiler capabilities**
  - `-WF,-D`*`macro`* `-g -q32 -q64 -qextname -qextern -qintsize -qdpc -qrealsize -qautodbl -qintlog -qinitauto -qflttrap -brename -C`
- **Set XLFRTEOPTS properly**
- **Understand relationships among ESSL, LAPACK, BLAS**
  - Green Zone, Red Zone, CCI concept