Part 4 – Power5+ Environment and comparison to x86-64
**Basic Building Block: IBM p5-575 Server**

<table>
<thead>
<tr>
<th>p575+ System</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Processors</td>
<td>16 X 1.9 Ghz POWER5+</td>
</tr>
<tr>
<td>Memory</td>
<td>64 or 128 GB</td>
</tr>
<tr>
<td>Integrated Network</td>
<td>1 Gbit Ethernet, and InfiniBand</td>
</tr>
<tr>
<td>OS</td>
<td>Linux: SuSE SLES V9</td>
</tr>
<tr>
<td>Compilers</td>
<td>IBM xlf Fortran, c and C++</td>
</tr>
<tr>
<td>Application supported</td>
<td>Gaussian and other apps that need large Memory or SMP</td>
</tr>
</tbody>
</table>

**Installed at UKY**

8 p575 Systems
128 total processors
Linux on Power vs Linux on x86-64

- Linux on Power is PowerPC 64-bit Linux (ppc64)
- For the users, the commonly used commands are identical
- The software compiled on x86-64 will not run on ppc64 because of differences in the instruction set.
- i.e., all codes need to be recompiled using native XL compilers or GNU compilers
Symmetric Multi-Threading vs Hyper-Threading

- Hyper-threading in Intel is set on/off at the BIOS level
- SMT is set on/off at boot level for Linux on Power (note: on AIX SMT can be turned on/off dynamically without rebooting the system)
- If SMT is on, /proc/cpuinfo shows twice the number of PHYSICAL CPU
- System performance monitors, such as top, nmon, etc., show the LOGICAL CPU
- To Turn off SMT add the following line:
  ```
smt-enabled=off
  ```
  to the file /etc/yaboot.conf
When to use SMT:

- Two threads are similar in execution needs and CPU utilization is topping out
- There is lot of Random data access (i.e. CPU is waiting for data)
- Overall Through-put is more important than the through-put of an individual Thread

When not to use SMT:

- Cache size is important to performance of the application
- When one thread is slower, then this could degrade the performance of the other thread
Processor Utilization Resource Register (PURR)

- Traditional utilization metrics are misleading:
  - They think there are two physical processors (when simultaneous multi-threading is active)
- The number of dispatch cycles for each thread can be measured using a new register called the Processor Utilization Resource Register (PURR)
- Two PURR registers (one for each hardware thread)
- Units are the same as the timebase register
- Sum of the PURR values for both threads is equal to the timebase register
Power5+ XL Compiler

http://publib.boulder.ibm.com/infocenter/lnxpcomp/v8v101/index.jsp
### IBM Compiler Names

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Command</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>xlc</td>
<td>xlc and Visual Age C (vac)</td>
</tr>
<tr>
<td>C++</td>
<td>xIC</td>
<td>xIC and Visual Age C++ (vacpp)</td>
</tr>
<tr>
<td>FORTRAN</td>
<td>xlf, xlf90, xlf95</td>
<td>XL FORTRAN</td>
</tr>
</tbody>
</table>

There are a lot more, including fort77, cc99_128, xlc128_r7...
Quick Reference Page – Cheat Sheet

- Which Fortran compiler to use

<table>
<thead>
<tr>
<th>Language</th>
<th>Sequential</th>
<th>SMP</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran 77</td>
<td>xlf</td>
<td>xlf_r</td>
<td>mpxlf</td>
</tr>
<tr>
<td>Fortran 90</td>
<td>xlf90</td>
<td>xlf90_r</td>
<td>mpxlf90</td>
</tr>
<tr>
<td>Fortran 95</td>
<td>xlf95</td>
<td>xlf95_r</td>
<td>mpxlf95</td>
</tr>
</tbody>
</table>

- Compiler options for performance
  -O3 -qarch=pwr5 -qtune=pwr5 (use these at minimum)
  -hot (High order Transformation)
  -pg (profiling)
  -qstrict (no alter the semantics of a program)
  -qipa (inter procedural analysis)
## C Compiler Invocations

<table>
<thead>
<tr>
<th>Language</th>
<th>Sequential</th>
<th>Reentrant (for SMP)</th>
<th>Message Passing</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSI C++</td>
<td>xlC</td>
<td>xlC_r</td>
<td>mpCC</td>
</tr>
<tr>
<td>ANSI C</td>
<td>xlc</td>
<td>xlc_r</td>
<td>mpcc</td>
</tr>
<tr>
<td>Extended</td>
<td>cc</td>
<td>cc_r</td>
<td></td>
</tr>
</tbody>
</table>

**Two C compilers:**
- C and C++
- C is a subset of C++
Fortran Compiler Invocations

<table>
<thead>
<tr>
<th>Language</th>
<th>Sequential</th>
<th>SMP</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran 77</td>
<td>xlf</td>
<td>xlf_r</td>
<td>mpxlf</td>
</tr>
<tr>
<td>Fortran 90</td>
<td>xlf90</td>
<td>xlf90_r</td>
<td>mpxlf90</td>
</tr>
<tr>
<td>Fortran 95</td>
<td>xlf95</td>
<td>xlf95_r</td>
<td>mpxlf95</td>
</tr>
</tbody>
</table>

One fortran compiler. Multiple invocations.
MATH LIBRARIES
Agenda

- Review: BLAS and LAPACK
- IBM ESSL Library
  - Engineering and Scientific Subroutine Library
- Review: PBLAS, BLACS, ScaLAPACK
- IBM PESSL Library
- MASS Library – libmass.a
  - Mathematical Acceleration SubSystem
Documentation

- [http://www.netlib.org/lapack/index.html](http://www.netlib.org/lapack/index.html)
- [http://www.netlib.org/blas/](http://www.netlib.org/blas/)
- [http://www.netlib.org/scalapack/](http://www.netlib.org/scalapack/)
### Libraries vs Parallel Enablement and Relationship with Others

<table>
<thead>
<tr>
<th>Libraries</th>
<th>Parallel Enablement and Relationship with Others</th>
</tr>
</thead>
</table>
| IBM Parallel ESSL | MPI enabled, independent of ESSL, BLACS and PBLAS are part of PESSL  
|                  | Same subroutine interface as ScaLAPACK when available                                                         |
| IBM ESSL-SMP     | Some of the subroutines in ESSL are multi-threaded.                                                            |
| IBM ESSL         | Serial version of ESSL, BLAS is part of ESSL  
|                  | Some interfaces are the same as LAPACK, others are different                                                   |
| ScaLAPACK        | MPI parallel enabled, Extension of LAPACK Project  
|                  | MPI calls are done through BLACS and PBLAS                                                                     |
| LAPACK-SMP       | Can be created by relink to BLAS-SMP, if available                                                             |
| LAPACK           | Serial version of LAPACK                                                                                      |
| BLACS            | Basic building block for ScaLAPACK, MPI enabled                                                                |
| PBLAS            | Basic building block for ScaLAPACK, MPI enabled                                                                |
| BLAS             | Basic building block of all the above libraries                                                                |
Example: using ScaLAPACK and ESSL


- Algorithms with an iterative matrix-diagonalization scheme (either conjugate gradient scheme, block Davidson scheme or a residual minimization scheme)

- Heavy use of FFT and scaLAPACK routines such as PDSYEVX (eigen value solver) or ORTHCH (Cholesky factorization)

- Library link in makefile:

```bash
SCALAPACK = -lscalapack -lblacsF77init -lblacs

ESSL = ../lib/libcci.a -lessl ../lib/liblapack.a

LIB = $(SCALAPACK) $(ESSL) ../lib/libmass.a
```
VASP Parallel Performance Comparison
Using ScaLAPACK, ESSL, LAPACK, MASS Libraries

Elapsed Time (second)

32 64 128 256

p575-1.9GHz
8CPU/node
**Review: BLAS**
(Basic Linear Algebraic Subroutines)

*Many computational problems boil down to linear algebraic computations.*

- **Level 1 BLAS – Vector-Scalar Operations**
  - DOT (inner product)
  - SUM (vector sum)
  - AXPBY (Scaled vector accumulation) \( y \leftarrow \alpha x + y \)
  - WAXPBY (Scaled vector addition)

- **Level 2 BLAS – Matrix-Scalar Operations**
  - GEMV (General matrix vector product) \( y \leftarrow \alpha x + \beta y \)
  - GBMV (Banded matrix vector product)
  - SYMV (Symmetric matrix vector product)
  - HEMV (Hermitian matrix vector product)
  - TRSV (Triangular solve)

- **Level 3 BLAS – Matrix-Matrix Operations**
  - GEMM (General matrix matrix product) \( C \leftarrow \alpha AB + \beta C \)
  - SYMM (Symmetric matrix matrix product)
  - HEMM (Hermitian matrix matrix product)
Factors that Affect HPC Applications Performance

- **Vectorization** of floating point operations
  - Natural in linear algebra represented by BLAS

- **Cache friendly** data movement
  - Block algorithm is very efficient for BLAS3 matrix operations, but not at all for BLAS1 operations.

- **Parallelism**
  - Loop-based SMP parallel can be implemented easily for BLAS operations, and so does MPI parallel.

- **Portability** between hardware platforms is also a requirement for a successful software solution

BLAS was written to provide high performance with all the above features
BLAS – key to the success of LAPACK

- **BLAS: the building blocks of LAPACK**
  - Well established standard in the industry
  - Efficient implementation provided by h/w or ISV vendors
  - Low level interface between LAPACK and hardware architecture

- **BLAS3:**
  - Used as much as possible in LAPACK code
  - Meet the challenge of instruction vectorization and cache-friendly data movement, delivering near peak performance
  - “adequate” for parallelism

- **BLAS1**
  - Widely used in previous generation EISPACK and LINPACK
  - Data movement not cache-friendly
  - Used in LAPACK for convenience

- **BLAS2**
  - Improved data movement, but not enough.

For more information: http://www.netlib.org/lapack/
Review: LAPACK

- It provides routines for solving
  - systems of simultaneous linear equations,
  - least-squares solutions of linear systems of equations,
  - eigenvalue problems, and singular value problems.

- It also provides
  - matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur),
  - related computations such as reordering of the Schur factorizations and estimating condition numbers.

- It handles dense and banded matrices, but not general sparse matrices.

- In all areas, similar functionality is provided for [S], [D], [C], [Z].

For more information: http://www.netlib.org/lapack/
LAPACK – Performance Considerations

- High performance considerations
  - Take advantages of multi-layered memory hierarchies of modern computer systems
  - Make the widely used EISPACK and LINPACK libraries run efficiently on shared-memory vector and parallel processors.
  - Reorganize the algorithms to use block matrix operations in the innermost loops. Example: matrix multiplication.
  - Computation performed by calls to BLAS3 library as much as possible – coarse granularity than BLAS1 used in EISPACK and LINPACK

- Requires highly optimized block matrix operations be already implemented.
LAPACK – Routine Structure

- **Driver routines:**
  - to solve standard types of “complete” math problems
  - i.e. (1) a system of linear equations, (2) compute eigenvalues of a real symmetric matrix

- **Computational routines:**
  - to perform a distinct computational tasks
  - i.e. an LU factorization, (2) reduction of a real symmetric matrix to tridiagonal form,

- **Auxiliary routines:**
  - to perform a certain subtask, common low-level computation, extension to BLAS, etc
LAPACK – Driver Routines

- Driver routines are divided into
  - Simple driver (name ending –SV)
  - Expert driver (name ending –SVX)

<table>
<thead>
<tr>
<th>Complete Math problems covered</th>
<th>Problem types covered</th>
<th>Short-Precision Routines</th>
<th>Long-Precision Routines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear equations</td>
<td>11</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>Linear least squares problems</td>
<td>1</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Eigenvalue and singular value problems</td>
<td>6</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>Generalized eigenvalue problems</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>19</td>
<td>74</td>
<td>74</td>
</tr>
</tbody>
</table>
## LAPACK – Computational Routines

<table>
<thead>
<tr>
<th>Computational tasks covered</th>
<th>Problem types covered</th>
<th>Short-Precision Routines</th>
<th>Long-Precision Routines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear equations</td>
<td>17</td>
<td>124</td>
<td>124</td>
</tr>
<tr>
<td>Symmetric eigen problem</td>
<td>7</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>Nonsymmetric eigen problem</td>
<td>4</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>Singular value decomposition</td>
<td>3</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Generalized symmetric definite eigen problems</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>33</strong></td>
<td><strong>182</strong></td>
<td><strong>182</strong></td>
</tr>
</tbody>
</table>
SMP LAPACK? Yes!

- LAPACK was originally designed for serial computations
  - Can be easily extended to run multithreaded on an SMP system

- Replace the BLAS library with SMP-BLAS - If the platform hardware vendor provides such SMP version, then LAPACK becomes SMP-LAPACK

- See ScaLAPACK(!) user guide for more info
  - Similar ideas were used to design ScaLAPACK!
IBM ESSL Library

- Introduction
Introducing ESSL – Engineering and Scientific Subroutine Libraries

- **Runtime library of high performance mathematical subroutines, Supporting**
  - Servers: pSeries, JS20/21 and Blue Gene
  - Processors: POWER3,4,5, PPC970, 440,
  - OS: AIX, Linux
  - Languages: Fortran, C, C++
  - 32-bit and 64-bit, meet ANSI/IEEE standard

- **Latest releases:**
  - V4.2 for AIX5.2, AIX5.3 (for pSeries and JS20/21)
  - V4.2.3 for Linux on POWER (for JS20/21 and Blue Gene)

- **3 Libraries:**
  - ESSL Serial Library
  - ESSL SMP Library
  - ESSL Blue Gene Library

- **The MPI flavor is called Parallel ESSL and will be discussed separately.**
# ESSL Libraries Installed at /opt/ibmmath

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>libessl.a,</td>
<td>Serial</td>
</tr>
<tr>
<td>libessl_r.a</td>
<td>Serial (multithread safe)</td>
</tr>
<tr>
<td>libesslsmp.a</td>
<td>Multithreaded (SMP)</td>
</tr>
<tr>
<td>libpessl.a</td>
<td>MPI</td>
</tr>
<tr>
<td>libpesslsmp.a</td>
<td>SMP+MPI</td>
</tr>
</tbody>
</table>

PESSL will be discussed separately
Interface to Fortran, C and C++

- ESSL follows standard Fortran calling conventions
- Must run in the Fortran run-time environment
- When called from other language, all aspect of Fortran conventions much be used
  - Including linkage conventions and the data conventions
  - Array order must be consistent with Fortran array ordering techniques
## Compile/Link ESSL in Fortran and C

<table>
<thead>
<tr>
<th></th>
<th>Fortran Command and options</th>
<th>C Command and Options</th>
<th>Link with ESSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMP</td>
<td><code>xlf_r -O -qnosave</code></td>
<td><code>cc_r -O</code></td>
<td><code>-lesslSmp</code></td>
</tr>
<tr>
<td>SMP</td>
<td><code>xlf_r -O -qnosave -q64</code></td>
<td><code>cc_r -O -q64</code></td>
<td><code>-lesslSmp</code></td>
</tr>
<tr>
<td>Serial</td>
<td><code>xlf_r -O -qnosave</code></td>
<td><code>cc_r -O</code></td>
<td><code>-lessl</code></td>
</tr>
<tr>
<td>Serial</td>
<td><code>xlf_r -O -qnosave -q64</code></td>
<td><code>cc_r -O -q64</code></td>
<td><code>-lessl</code></td>
</tr>
<tr>
<td>Serial</td>
<td><code>xlf -O</code></td>
<td><code>cc_r -O</code></td>
<td><code>-lessl</code></td>
</tr>
<tr>
<td>Serial</td>
<td><code>xlf_r -O -q64</code></td>
<td><code>cc_r -O -q64</code></td>
<td><code>-lessl</code></td>
</tr>
</tbody>
</table>

For more info on `-qnosave`, see XLF Compiler Reference Guide
ESSL Accuracy Features

- For most math operations, there are 4-5 implementations:
  - [s] for Short-precision real.
  - [d] for long-precision real
  - [c] for short-precision complex
  - [z] for long-precision complex
  - [i] for integer

- **Examples:**
  - SAXPY, DAXPY, CAXPY, ZAXPY for \((Ax + y)\) calculation
  - SGEMM, DGEMM, CGEMM, ZGEMM for matrix multiplication

- Support ANSI/IEEE 32-bit and 64-bit binary floating-point format, and 32-bit integer.
### ESSL Coverage: 9 Areas + Utilities

<table>
<thead>
<tr>
<th>Area of Computation</th>
<th>Integer Subroutines</th>
<th>Short-Precision Subroutine</th>
<th>Long-Precision Subroutine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Algebra Subprograms</td>
<td>0</td>
<td>74</td>
<td>77</td>
</tr>
<tr>
<td>Matrix Operations</td>
<td>0</td>
<td>25</td>
<td>26</td>
</tr>
<tr>
<td>Linear Algebraic Equations</td>
<td>0</td>
<td>74</td>
<td>76</td>
</tr>
<tr>
<td>Eigensystem Analysis</td>
<td>0</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>FFT, Convolution, etc</td>
<td>0</td>
<td>31</td>
<td>19</td>
</tr>
<tr>
<td>Sorting and Searching</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Interpolation</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Numerical Quadrature</td>
<td>0</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Random Number Generation</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Utilities</td>
<td>8</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td><strong>Total ESSL Subroutines</strong></td>
<td><strong>13</strong></td>
<td><strong>240</strong></td>
<td><strong>253</strong></td>
</tr>
</tbody>
</table>

**Total ESSL Subroutines = 494**
## BLAS Subroutines in ESSL

<table>
<thead>
<tr>
<th>Math Operations</th>
<th>Example subroutine name &amp; (Math Operation)</th>
<th>Number of Math Op’tions supported</th>
<th>Number of BLAS subprograms In ESSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1 BLAS: Vector-Scalar Operations</td>
<td>SAXPY $(y \leftarrow \alpha x + y)$</td>
<td>10</td>
<td>44</td>
</tr>
<tr>
<td>Level 2 BLAS: Matrix-Vector Operations</td>
<td>SGEMV $(y \leftarrow \alpha Ax + \beta y)$</td>
<td>11</td>
<td>66</td>
</tr>
<tr>
<td>Level 3 BLAS: Matrix-Matrix Operations</td>
<td>SGEMM $(C \leftarrow \alpha AB + \beta C)$</td>
<td>6</td>
<td>30</td>
</tr>
<tr>
<td>Total</td>
<td>- -</td>
<td>27</td>
<td>140</td>
</tr>
</tbody>
</table>

Standard BLAS covers more math operations
Multithreaded ESSL SMP Subroutines

- ESSL contains 3 libraries
  - ESSL Serial Library
  - ESSL SMP Library
  - ESSL Blue Gene Library

- ESSL SMP Library is thread-safe, and ~1/2 of the subroutines are multithread enabled.

- To use libesslsmp.a, simply re-link it to your existing applications, no code change needed.
  - Experiment is often needed in order to find optimal SMP performance.

- If you decide to develop your own multithreaded program, you can use thread-safe ESSL Serial Library libessl_r.a

- Be sure to distinguish libesslsmp.a from libessl_r.a
### SMP ESSL vs. Serial ESSL

<table>
<thead>
<tr>
<th>ESSL Area of Computation</th>
<th>Serial version</th>
<th>Multithreaded version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Algebra Subprograms</td>
<td>151</td>
<td>116</td>
</tr>
<tr>
<td>Matrix Operations</td>
<td>51</td>
<td>46</td>
</tr>
<tr>
<td>Linear Algebraic Equations</td>
<td>150</td>
<td>72</td>
</tr>
<tr>
<td>Eigensystem Analysis</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>FFT, convolution, etc</td>
<td>50</td>
<td>24</td>
</tr>
<tr>
<td>Sorting and Searching</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>Interpolation</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>Numerical Quadrature</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>Random Number Generation</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Utilities</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>Total ESSL Subroutines</td>
<td>494</td>
<td>258</td>
</tr>
</tbody>
</table>

To use `libesslsmmp.a`, simply re-link it to your existing applications, no code change needed.
ESSL vs. LAPACK

- ESSL has 495 subroutines
- LAPACK has 512 driver & computational routines
- They are not a subset of each other
- LAPACK
  - handles only dense and banded matrix
  - SMP parallel done within BLAS
- ESSL
  - also handles sparse matrix
  - SMP parallel done above BLAS
- There are many other differences
## ESSL vs. LAPACK

<table>
<thead>
<tr>
<th>Math Operation</th>
<th>ESSL subroutines that conform to name &amp; functionality in LAPACK</th>
<th>ESSL subroutines that have same name but different functionality in LAPACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense Linear Algebraic Equations</td>
<td>52</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>(total=113)</td>
<td></td>
</tr>
<tr>
<td>Linear Least Squares</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>(Total=8)</td>
<td></td>
</tr>
<tr>
<td>Eigensystem Analysis</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(total=16)</td>
</tr>
</tbody>
</table>
Call Conversion Interface (CCI) from LAPACK

- LAPACK provides an Call Conversion Interface (CCI) that substitutes a call to an ESSL subroutine in place of an LAPACK subroutine whenever an ESSL subroutine provides either functional or near-functional equivalence.

- CCI allows LAPACK users to obtain the optimized performance of ESSL for an additional subset of LAPACK subroutines.

For details, see Dongarra, J. J.; Kolatis M. October 1994. "Call Conversion Interface (CCI) for LAPACK/ESSL." LAPACK Working Note 82  http://www.netlib.org/lapack/lawn78/lawn82.ps
Use of LAPACK, CCI, and ESSL

- **First try linking with ESSL**
  - `$LDR` … -l essl

- **Second, link with LAPACK first, followed by ESSL:**
  - `$LDR` … -l lapack -l essl (give CCI interface a chance)
  - This leads to 4 possible cases

1. Functionality exists only in LAPACK
   - Use LAPACK object

2. Subroutine has same name and argument
   - Use LAPACK object

3. Same name, different argument
   - Use LAPACK object

4. Different names, same functionality
   - Use ESSL through CCI
Planning Your Program for ESSL

- Select data type: [S], [D], [C], [Z], [I]
- Select data structure: sparse, dense, banded, etc
- Select performance and accuracy
  - For some subroutines
- Select name of your subroutines to avoid conflict with ESSL
- Your data set up
  - Scalar
  - Array, and Array data alignment
  - Storage mode and conversion between modes
  - Setting up ESSL calling sequences
  - Using auxiliary storage in ESSL
Guidelines: Getting Best Performance from ESSL

- Use BLAS and matrix operations in the order of
  - Matrix-matrix, matrix-vector, vector-scalar
- Choose the subroutines that do multiple computations, i.e., SNAXPY, not SAXPY.
- Use a stride of 1 for the data in your computation
- Avoid using $2^N$ (128, 256, etc) as the leading dimension of an array (to avoid cache thrusting)
- For small problems, avoid using a large leading dimension for your matrix (to improve cache line usage)
- Align arrays on double word boundaries
IBM ESSL Library

- Introduction
- Dive deeper into ESSL
High Performance Techniques Used in ESSL

- Block algorithm to maximize cache and TLB hit
- Access data contiguously (stride-1)
- Maximize available floating-point register usage
- Minimize paging
- Fully utilizing the dual floating-point execution units, when applicable
- Utilize hardware data-prefetching when applicable
- In most cases, ESSL will not benefit from SMT features
Introducing 1st of the 9 Areas of Computation Linear Algebra

- 4 sub-areas of linear algebra subprograms:
  - Vector-scalar (incl. subset of BLAS 1)
  - Sparse vector-scalar (incl. subset of BLAS 1 extensions)
  - Matrix-vector (incl. subset of BLAS 2)
  - Spares matrix-vector

- Some of them were designed in accordance with the level 1 and 2 BLAS de facto standard
Introducing 2nd area of Computations Matrix Operation

- Four sub-areas of coverage:
  - Dense linear algebraic equations (some correspond to BLAS2, 3 and LAPACK)
  - Banded linear algebraic equations
  - Sparse linear algebraic equations
  - Linear least squares (some correspond to LAPACK)

- Some of them were designed in accordance with the level 2 and 3 BLAS and LAPACK de facto standard.
Introducing 3\textsuperscript{rd} Area of Computations Linear Algebraic Equation

- Compute matrix +, -, X, +x, transpose, etc
- Almost \( \frac{1}{2} \) of the subprograms were designed in accordance with the level 3 BLAS de facto standard
Introducing 4th Area of Computations Eigensystem Analysis
Subprograms

- Compute eigenvalues and/or eigenvectors
- Used adaptations of EISPACK routines
- Have the same subroutine name as an existing LAPACK routine,
  - Warning: the calling-sequence arguments and functionality are different
Introducing the 5th Area of Computation
Fourier Transforms, Convolutions and Correlations, and Related Computations

- Three sub-areas:
  - Fourier Transform (mixed-radix transformation in 1, 2 and 3 dimensions)
  - Convolutions, and Correlations (Fourier- or direct-method)
  - Related Computations (for general signal processing apps)
Introducing the 6th, 7th and 8th Area of Computation

- **Sorting and Searching**
  - Operates on integer, short-precision, and long-precision.
  - Sort with or without index designations
  - Search using binary or sequential search

- **Interpolation**
  - Polynomial interpolation
  - Local polynomial interpolation
  - Both 1D or 2D cubic spline interpolation

- **Numerical Quadrature**
  - Gaussian quadrature methods for integration
Next Topic: ScaLAPACK
“Message-passing version of LAPACK”

ScaLAPACK
A Software Library for Linear Algebra Computations on Distributed-Memory Computers

AVAILABLE SOFTWARE:
- Dense, Band, and Triangular Linear Systems
- general
- symmetric positive definite
- Full-Rank Linear Least Squares
- Standard and Generalized
- Orthogonal Factorizations
- Eigenproblems
  - SEP: Symmetric Eigenproblem
  - NEP: Non-Hermitian Eigenproblem
  - GSEP: Generalized Symmetric Eigenproblem
- SY

Prototype Codes
- HPL (High-Performance Linpack) Reference Code
- ScaLAPACK

DOCUMENTATION:
- ScaLAPACK User’s Guide

Future Work
- Out-of-core Eigensolvers
- Divide and Conquer routines
- C++ and Java Interfaces

Commercial Use
ScaLAPACK has been incorporated into the following software packages:
- NAS Numerical Library
- NEC Parallel ESSL
- SGI Scientific Library
  - and is being integrated into the VNL/NESSL Numerical Library, as well as software libraries for Fujitsu, HP/Compaq, Hitachi, and NEC.

http://www.netlib.org/scalapack/

© 2007 IBM Corporation
ScaLAPACK - Introduction

- A continuation of the LAPACK project,
- Covers a subset of LAPACK routine,
- Share very similar interface with LAPACK
- Written in a Single-Program-Multiple-Data style using explicit message passing for inter-processor communication.
- Assumes matrices are laid out in a two-dimensional block cyclic decomposition.
- Goals
  - Efficiency, Scalability, Reliability, Portability, Flexibility, and Ease of Use.
- Covers dense and band matrices, not general sparse matrices
- Written in Fortran77 with a few in C
- Latest version V1.7.4, released May 25, 2006
  - [http://www.netlib.org/scalapack/](http://www.netlib.org/scalapack/)
  - Source code available
Introducing PBLAS
Parallel Basic Linear Algebra Subprograms

- A message-passing parallel version of BLAS
  - Perform message-passing on MIMD systems
  - Routine Interface is as similar to BLAS as possible
  - Enable ScaLAPACK to be similar or nearly identical to the analogous LAPACK code
  - Hope PBLAS will provide a distributed memory standard, as the BLAS have provided as a shared memory standard

- Supporting 3 levels of PBLAS, just like BLAS

- For more information, see
Introducing BLACS
Basic Linear Algebra Communication Subprograms

- **Goals:** to create a linear algebra oriented message passing interface that may be implemented efficiently and uniformly across a large range of distributed memory platforms.
- **the communication layer of** [ScaLAPACK](http://www.netlib.org/blacs/BLACS/QRef.html).
- **Key ideas in the BLACS include:**
  - Standard interface,
  - Process grid and scoped operations,
  - Contexts,
  - Array-based communication,
  - ID-less communication.

- **For more info, see:**
  - [http://www.netlib.org/blacs/BLACS/QRef.html](http://www.netlib.org/blacs/BLACS/QRef.html)
ScaLAPACK User’s Guide

Published in 1997
Available on line:  http://www.netlib.org/scalapack/slug/index.html
ScaLAPACK Efficiency & Portability

- **Design strategy:**
  - Perform computation by calls to PBLAS
  - PBLAS perform global computation by relying on BLAS for local computation and the BLACS for communication
  - BLAS and BLACS form a low-level interface between ScaLAPACK and different machine architectures
  - Use block-partitioned algorithms \(\rightarrow\) cache friendly

- **Requirement:**
  - BLAS and BLACS be treated as standard, and implemented efficiently by hardware vendors
Software Components

These components are synchronous parallel routines, and Arguments include matrices and vectors are distributed across multiple processors

The components are called on a single processor, with arguments stored on single processors only
ScalAPACK Structure

- **3 Broad Category of Routines - similar to LAPACK**
  - Driver routines – each solves a complete problem. i.e. a system of linear equations
    - Linear Equations
    - Linear Least Squares Problems
    - Standard Eigenvalue and Singular value Problems
    - Generalized Symmetric Definite Eigenproblems
  - Computational routines – each performs distinct computational task, i.e. an LU factorization.
  - Auxiliary routines – each performs a subtask or some commonly required low-level computations
Four Basic Steps to Call a ScaLAPACK Routine

1. Initialize the process grid
   • A call to the ScaLAPACK TOOLS routine SL_INIT initializes a Pr × Pc
     (denoted NPROW and NPCOL) process grid by using a row-major
     ordering of the processes.

2. Distribute the matrix on the process grid
   • All global matrices must be distributed on the process grid prior to the
     invocation of a ScaLAPACK routine. It is the user's responsibility to
     perform this data distribution.

3. Call ScaLAPACK routine
   • All ScaLAPACK routines assume that the data has been distributed on
     the process grid prior to the invocation of the routine.

4. Release the process grid
   • All ScaLAPACK routines assume that the data has been distributed on
     the process grid prior to the invocation of the routine.

For a hands-on practice example:

http://www.netlib.org/scalapack/slug/node26.html#SECTION0421000000000000
To Run a ScALAPACK Example

- **Required software libraries**
  - Lib scalapack.a
  - Lib pblas.a
  - Lib blacs.a
  - Lib blas.a
  - Lib mpi.a
Shifting Topic: IBM Parallel ESSL

- **Introduction**

- Parallel ESSL is not a simple subset of ESSL
  - It has a separate release version number
  - It has its own user guide – also 1000+ pages
  - It does not have any implementation for [c], and only ~10 subroutines for [s]
  - The manual never used abbreviation “PESSL”
    - We use PESSL here to save space on the slides.

- PESSL and ESSL will be compared in the next ~10 slides
Introducing Parallel ESSL – a “cousin” of ESSL

- A Scalable math subroutine library that supports parallel processing apps, using the SPMD (Single Program Multiple Data) programming model and MPI library

- **Supporting**
  - Servers: pSeries (with Federation or colony switch), JS20/21 (with Myrinet switch),
  - Processors: POWER3,4,5, PPC970, 440,
  - OS: AIX, Linux
  - Languages: Fortran, C, C++
  - 32-bit and 64-bit, meet ANSI/IEEE standard

- **Latest releases:**
  - V3.2 for AIX5.2, AIX5.3 (for pSeries and JS20/21)
  - V3.2 for Linux on POWER (for JS20/21)
How Parallel ESSL Works

- PESSL can be called by applications of SPMD programming model
  - Each process works only on a portion of the global data

- Application must distribute global data structure across all processes before calling PESSL subroutines
  - This will avoid bottleneck of initial scatter or ending gather operations. Input/output are also distributed among multiple processes

- PESSL includes the BLACS for communication,
- PESSL uses ESSL subroutines for computation.
  - ESSL is a requirement for PESSL
## PESSL & ESSL Libraries Installed at /opt/ibmmath/

<table>
<thead>
<tr>
<th>Library</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial</td>
<td>libessl.a,</td>
</tr>
<tr>
<td>Serial (multithread safe)</td>
<td>libessl_r.a</td>
</tr>
<tr>
<td>Multithreaded (SMP)</td>
<td>libesslsmp.a</td>
</tr>
<tr>
<td>MPI</td>
<td>libpessl.a</td>
</tr>
<tr>
<td>SMP+MPI</td>
<td>libpesslsmp.a</td>
</tr>
</tbody>
</table>

This page is identical to the page used earlier for ESSL
Interface to Fortran, C and C++

- PESSL follows standard Fortran calling conventions
- Must run in the Fortran run-time environment
- When called from other language, all aspect of Fortran conventions much be used
  - Including linkage conventions and the data conventions
  - Array order must be consistent with Fortran array ordering techniques

The requirement is identical to that for ESSL V4.2 for AIX
Coverage of Parallel ESSL Subroutines

- Level 2 Parallel Basic Linear Algebra Subprograms (PBLAS) *
- Level 3 PBLAS *
- Linear Algebraic Equations
- Eigensystem Analysis and Singular Value Analysis
- Fourier Transforms *
- Random Number Generation

* ESSL has broader coverage for these category of applications
## PBLAS Subroutines in PESSL

<table>
<thead>
<tr>
<th>Math Operations</th>
<th>Example subroutine name &amp; (Math Operation)</th>
<th>Number of Math Op’tions supported</th>
<th>Number of PBLAS subprograms In PESSL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Level 1 PBLAS:</strong> Vector-Scalar Operations</td>
<td>SAXPY ((y \leftarrow \alpha x + y))</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Level 2 PBLAS:</strong> Matrix-Vector Operations</td>
<td>SGEMV ((y \leftarrow \alpha Ax + \beta y))</td>
<td>7</td>
<td>15</td>
</tr>
<tr>
<td><strong>Level 3 PBLAS:</strong> Matrix-Matrix Operations</td>
<td>SGEMM ((C \leftarrow \alpha AB + \beta C))</td>
<td>7</td>
<td>18</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>- -</td>
<td>14</td>
<td>33</td>
</tr>
</tbody>
</table>

*Standard PBLAS covers more math operations*
## Serial ESSL, SMP ESSL and Parallel ESSL

<table>
<thead>
<tr>
<th>ESSL Area of Computation</th>
<th>Serial version</th>
<th>Multithreaded version</th>
<th>Parallel version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Algebra Subprograms</td>
<td>151</td>
<td>116</td>
<td>15</td>
</tr>
<tr>
<td>Matrix Operations</td>
<td>51</td>
<td>46</td>
<td>18</td>
</tr>
<tr>
<td>Linear Algebraic Equations</td>
<td>150</td>
<td>72</td>
<td>56*</td>
</tr>
<tr>
<td>Eigensystem Analysis</td>
<td>16</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>FFT, Convolution, etc</td>
<td>50</td>
<td>24</td>
<td>12</td>
</tr>
<tr>
<td>Sorting and Searching</td>
<td>15</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Interpolation</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Numerical Quadrature</td>
<td>12</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Random Number Generation</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Utilities</td>
<td>11</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td><strong>Total Subroutines</strong></td>
<td><strong>494</strong></td>
<td><strong>258</strong></td>
<td><strong>131</strong></td>
</tr>
</tbody>
</table>

*including subroutines specifically implemented in F77 and F90, which is not identified in such manor in ESSL*
Migrating from ScaLAPACK to Parallel ESSL

- Parallel ESSL uses compatible calling sequences with ScaLAPACK V1.5
Mathematical Acceleration SubSystem (MASS)

- Accelerated set of frequently used math intrinsic functions
- Available for AIX, Linux (inc. Power and BG/L)
- Languages:
  - Fortran, C & C++
- 32 and 64-bit
- Scalar and vector versions
- Shipped with XL compiler product

For more information:  http://www-306.ibm.com/software/awdtools/mass/
Use of libmass.a

- Automatically used if code is compiled with the following options:
  - -qhot -qnostrict
  - -qhot -O3
  - -qsmp -qnostrict
  - -qsmp -O3
  - -O4
  - -O5

- Specify explicit link
  `xlf progf.f -o progf -lmass -lmassv`

For more information
Libmass.a Performance

For more information and update
Vector MASS Library

- **Examples**
  - call vexp(y,x,n)
  - call vdiv(x,y,z,n)
  - call vsincos(x,y,z,n)
  - call atan2(x,y,z,n)
  - call vdsign(x,y,z,n)

- **Recoding IS necessary**

- **C call interface: Call by reference (only)**

- **Linking:**
  - [xlf,xlc] ... -l massv

For more information on consistency of libmassv.a