Intel Performance Libraries (MKL, TBB, IPP, DAAL)

This document gives a short introduction to the usage of highly optimized scientific library routines on the Linux-based HPC systems at LRZ.

- Version, Platforms and Licensing
- Usage
  - Special versions of the performance libraries
  - Linking with the MKL
  - Fortran 90 modules and C interfacing
  - Compiling and Linking with the TBB
  - Compiling and Linking with the IPP
  - Multi-Threading in MKL
- Troubleshooting and Feedback
  - Undefined symbol: _MKL_SERV_lsame (or so):
- Documentation
  - Links on the Intel web site
  - TBB examples and documentation

Version, Platforms and Licensing

LRZ has licensed the performance libraries from Intel for use on the LRZ HPC systems (Linux Cluster and National Supercomputing System). These libraries are comprised of

- the Math Kernel Libraries (MKL), containing well-optimized implementations of the BLAS and LAPACK interfaces, sparse solvers, support for interval arithmetic, FFT routines and other functionality.
- ScalAPACK and distributed FFT implementations for various MPI flavours.
- Threading Building Blocks (TBB), which enable the C++ programmer to relatively easily integrate (shared-memory) parallel capability into her/his code. In particular, this package has support for scalable threaded containers (note that the STL is typically not thread-safe).
- the Integrated Performance Primitives (IPP) containing highly optimized primitive operations used for digital filtering, audio and image processing etc.
- the Data Analytics Library (DAAL)

The presently installed versions are described in the following table.

<table>
<thead>
<tr>
<th>Product</th>
<th>Versions available</th>
</tr>
</thead>
<tbody>
<tr>
<td>MKL</td>
<td>2017, 2018</td>
</tr>
<tr>
<td>TBB</td>
<td>2017, 2018</td>
</tr>
<tr>
<td>IPP</td>
<td>2017, 2018</td>
</tr>
<tr>
<td>DAAL</td>
<td>2017, 2018</td>
</tr>
</tbody>
</table>

Special versions of the performance libraries

- **ILP64**: For some of the MKL functionality (BLAS, LAPACK, ScalAPACK, FFT), an 8-Byte Integer version is available. Please load the module with the _i8 appended to the version number. Note that it is not possible to mix 4-Byte and 8-Byte calls.
- **serial MKL**: A purely serial version of the MKL which can be used if suppression of multi-threading in the threaded version is not feasible or leads to problems. Please load the module with the _s appended to the version number.

These special MKL versions are not available for all versions or all platforms; issuing the command `module avail mkl` will reveal what special versions are available.

Usage

Before using the MKL please load the environment module `mkl`:

```
module load mkl
```

Before using the TBB, load the module `tbb`

```
module load tbb
```

Before using the IPP, load the module `ipp`

```
module load ipp
```

Before using the DAAL, load the module `daal`

```
module load daal
```

Loading an appropriate module is also **required** before running a program that uses shared libraries from the above packages.
Linking with the MKL

The mkl environment module provides environment variables which can be used for handling the compilation and linkage process. If you need optimized BLAS; LAPACK or other routines provided by the MKL, please provide the library location when linking your executable:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Linking prescription</th>
</tr>
</thead>
<tbody>
<tr>
<td>static linkage</td>
<td>ifort -parallel -o myprog.exe myprog.o mysub1.o ... $MKL_LIB</td>
</tr>
<tr>
<td>dynamic linkage</td>
<td>ifort -parallel -o myprog.exe myprog.o mysub1.o ... $MKL_SHLIB</td>
</tr>
</tbody>
</table>

Fortran 90 modules and C interfacing

The MKL contains functionality encapsulated within Fortran 90 modules (the DFTI API). In this case, it is necessary to write an appropriate module reference into the Fortran source code. In the case of DFTI, this would for example be a line of the form

```
use mkl_dfti
```

When compiling your code, you then also need to add the include path for the module information file:

```
ifort -c -o foo.o ... $MKL_INC foo.f90
```

The analogous procedure applies for C interfaces, again illustrated for the DFTI example: Specify

```
#include <mkl_dfti.h>
```

and compile with

```
icc -c -o cfoo.o ... $MKL_INC cfoo.c
```

Please check the MKL documentation as well as the directory `$MKL_BASE/include` for available modules and include files.

Compiling and Linking with the TBB

This package can only be used for C++ code; for compilation a command of the form

```
icpc -c -o cfoo.o ... $TBB_INC cfoo.cpp
```

is required. Linkage is only possible against shared libraries:

```
icpc -o myprog.exe ... main.o cfoo.o ... $TBB_SHLIB
```

The following environment variables are provided for debugging and the scalable memory allocator:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Linking prescription</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBB_SHLIB</td>
<td>TBB library for top performance</td>
</tr>
<tr>
<td>TBB_SHLIB_MALLOC</td>
<td>optimized scalable memory allocator library</td>
</tr>
<tr>
<td>TBB_SHLIB_DEBUG</td>
<td>debug version of TBB library; build source with -DTBB_DO_ASSUME=1</td>
</tr>
<tr>
<td>TBB_SHLIB_MALLOC_DEBUG</td>
<td>debug version of scalable memory allocator library; build source with -DTBB_DO_ASSUME=1</td>
</tr>
</tbody>
</table>

Compiling and Linking with the IPP

The ipp environment module provides environment variables in a similar manner as for mkl. Note that a Fortran interface is not available; hence you need to write a C interop (or !DEC$ directive) based interface block yourself if you need to call IPP routines from Fortran. For C, the compilation command is

```
icc -c -o cfoo.o ... $IPP_INC cfoo.c
```

This presupposes that you have inserted appropriate #include entries into your source.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Linking prescription</th>
</tr>
</thead>
<tbody>
<tr>
<td>static linkage</td>
<td>icc -o myprog.exe myprog.o mysub1.o ... $IPP_LIB</td>
</tr>
<tr>
<td>dynamic linkage</td>
<td>icc -o myprog.exe myprog.o mysub1.o ... $IPP_SHLIB</td>
</tr>
</tbody>
</table>

Multi-Threading in MKL

The MKL can make use of shared memory parallelism; by default only a single thread is used. If you wish to use multiprocessing, there are the following possibilities:
• If you wish to use OpenMP in your own program, but the MKL calls should run single-threaded, please perform the settings

```bash
export OMP_NUM_THREADS=8   # example to set up for execution with 8 threads
export MKL_SERIAL=yes
```

If you use the Intel compilers, setting MKL_SERIAL will not be necessary since in this case the MKL will automatically detect whether it is called from within a parallel region.

If a serial MKL version is loaded (see above), this will also enforce single threaded execution.

• If MKL should run multi-threaded, please perform the settings

```bash
export OMP_NUM_THREADS=8   # example to set up for execution with 8 threads
unset MKL_SERIAL
```

• If your application uses its own (non-OpenMP) threading, it is recommended that MKL calls run single-threaded:

```bash
export MKL_SERIAL=yes
```

Troubleshooting and Feedback

If you find problems with any of the libraries please contact the Service Desk. Here are a few remarks on how to solve certain known problems:

**Undefined symbol: _MKL_SERV_isame (or so):**

This may happen if LRZ changes the default library version to a newer release, and the dynamically linked binary cannot cope. Binary compatibility is apparently not always fully supported. Here are your options:

1. Re-link your executable with the present default MKL version
2. Do a `module switch mkl mkl/x.y`, which presupposes that you know the version x.y you originally used
3. Use the static library in the first place

Documentation

**Links on the Intel web site**

- MKL documentation
- IPP documentation
- TBB documentation
- DAAL documentation

**TBB examples and documentation**

When the tbb module is loaded, some example codes are available below `$TBB_BASE/examples`. PDF and HTML documentation is available in the folder `$TBB_DOC`. Since the TBB are also available as open source, a lot of information is available at the [threadingbuildingblocks](https://www.threadingbuildingblocks.org) web site.