Les Cahiers du GERAD ISSN: 0711–2440

LibHSL: the ultimate collection for large-scale scientific computation

J. Fowkes, A. Lister, A. Montoison, D. Orban

G–2024–06

December 2024

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

LibHSL: the ultimate collection for large-scale scientific computation

Jaroslav Fowkes [a]

Andrew Lister [a]

Alexis Montoison [b]

Dominique Orban [b]

[a] *Scientific Computing Department, Rutherford Appleton Laboratory, Chilton, England*

[b] *GERAD and Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Montréal (Qc), Canada, H3T 2A7*

jaroslav.fowkes@stfc.ac.uk
andrew.lister@stfc.ac.uk
alexis.montoison@polymtl.ca
dominique.orban@gerad.ca

December 2024
Les Cahiers du GERAD
G–2024–06

Abstract : The Harwell Subroutine Library (HSL) is a renowned suite of efficient and robust numerical algorithms designed to tackle complex mathematical problems such as solving sparse linear systems or computing eigenvalues and eigenvectors of sparse matrices. LibHSL is a comprehensive collection of HSL packages that facilitates integration into C, Fortran or Julia projects. One of the notable features of LibHSL is its innovative compilation system based on Meson. The new build system not only significantly accelerates compilation, but also ensures portability across multiple operating systems. To further streamline the user experience, LibHSL provides archives of precompiled libraries as well as a package called HSL_jll.jl, specifically designed for the Julia ecosystem. This eliminates the need for users to compile anything on their local machines, offering ready-to-use functionality right away.

Keywords: HSL, Julia, Fortran, C, sparse linear algebra, Meson, cross-compilation, portability, performance

Résumé : La bibliothèque de sous-programmes Harwell (HSL) est une suite renommée de méthodes numériques efficaces et robustes conçus pour résoudre des problèmes mathématiques complexes tels que la résolution de systèmes linéaires creux ou le calcul de valeurs propres et de vecteurs propres pour les matrices creuses. LibHSL est la collection ultime des bibliothèques HSL et facilite leur intégration dans des projets en C, Fortran ou Julia. L'un des aspects notables de LibHSL est son système de compilation basé sur Meson. Ce nouveau système de compilation accélère considérablement la compilation tout en assurant la portabilité sur différents systèmes d'exploitation. Afin de rendre l'expérience utilisateur encore plus simple, LibHSL offre des archives contenant des bibliothèques dynamiques précompilées ainsi qu'un module HSL_jll.jl, spécialement conçu pour l'écosystème Julia. Les utilisateurs évitent ainsi la compilation sur leurs machines locales avec une bibliothèque prête à l'emploi.

Mots clés : HSL, Julia, Fortran, C, algèbre linéaire creuse, Meson, compilation multi-plateformes, portabilité, performance

---

Acknowledgements: Alexis Montoison is supported by an FRQNT grant and an excellence scholarship of the IVADO institute, and Dominique Orban is partially supported by an NSERC Discovery Grant.

This work began while Alexis Montoison was visiting Nick Gould, Jaroslav Fowkes and the Computational Mathematics Group at the STFC Rutherford Appleton Laboratory in December 2022 and July 2023. He expresses its appreciation to the HSL team for their invitation and their wonderful hospitality.

The authors would like to thank:

- Stefan Vigerske for the METIS adapter;
- Eli Schwartz for its advices with the Meson build system;
- Geoffroy Leconte, Paul Raynaud, and Oscar Dowson for testing early versions of HSL_jll.jl;
- Mosè Giordano, and Elliot Saba for their work on BinaryBuilder.jl, libblastrampoline and Yggdrasil.

1 Introduction

The HSL Mathematical Software Library, originally known as the Harwell Subroutine Library, is a valuable resource for researchers, scientists, and engineers working on numerical computations. It provides a comprehensive collection of robust, well-tested, and highly efficient mathematical routines, covering areas such as sparse linear or eigenvalue problems.

LibHSL is a collection of more than 160 HSL packages. It aims to facilitate the use of HSL in Julia [1] as well as Fortran and C projects such as GALAHAD [2]. LibHSL focuses on ease of use for users on all platforms by using the Meson build system. Meson enables the distribution of prebuilt binaries for the package using BinaryBuilder.jl. Additionally, LibHSL supports either METIS 5 or the older METIS 4 [3].

This new package provides the source code of the included HSL packages, prebuilt binaries, and a Julia package named HSL_jll.jl. HSL_jll.jl is a pre-built version of LibHSL to be readily used in the Julia ecosystem. Once installed, HSL_jll.jl allows calling the shared library, containing the C and Fortran routines of the HSL packages, directly from Julia. The HSL wrappers provided in the Julia interface HSL.jl become fully functional. HSL_jll.jl also offers a convenient way to employ HSL linear solvers MA27, MA57, HSL_MA77, HSL_MA86 and HSL_MA97 within the Ipopt.jl interface for the IPOPT [5] nonlinear optimization solver.

Two versions of HSL_jll.jl are included. One precompiled with OpenBLAS that requires at least Julia 1.6 and the other version precompiled with libblastrampoline (LBT) that requires at least Julia 1.9. LBT allows one to dynamically switch the BLAS and LAPACK backends between e.g. OpenBLAS, BLIS, Intel MKL or Apple Accelerate. HSL_jll.jl is precompiled for various operating systems (Windows, Mac, Linux, FreeBSD) and architectures (x64, arm64, ppc64).

For information on how to use each HSL routine available through this package please see the relevant documentation on the HSL website. Where C interfaces are available, the C documentation should be used.

2 Building LibHSL with Meson

2.1 Meson

To download and install Meson, we refer the user to this guide. By default Meson uses the Ninja build system. After configuration, it is equivalent to build the package or install it directly with Ninja.

2.2 Configuration

To configure the compilation for your system, first create and initialise a build directory. For examples in this document, we will use `builddir` for our build directory

```
meson setup builddir [options...]
```

To choose an alternative install directory you can pass the `--prefix` argument. Non-default compilers can be selected by setting the `CC` and `FC` shell variables.

```
CC=icc FC=ifort meson setup builddir --prefix=~/libhsl [options...]
```

See this table for supported compilers. The compilation of libHSL has been tested with the Fortran compilers `gfortran`, `ifort`, `ifx` and `nagfor`. Note that the version 1.2.0 of Meson is required for using `ifx` on Linux.

The list of all options supported by libHSL can be displayed with

```
meson configure
```

This command can also be used from the build directory to change options, if required, after running `meson setup`.

To test the installation using our examples after building, configure with the `-Dexamples=true` option.

Table 1: Project options supported by libHSL

Options	Default value	Description
modules	true	Install Fortran modules
examples	false	Generate the examples
libmetis_version	5	Version of the METIS library
libblas	blas	BLAS library against which to link
liblapack	lapack	LAPACK library against which to link
libmetis	metis	METIS library against which to link
libmpi	mpifort	MPI library against which to link
libblas_path	∅	Additional directory to search for the BLAS library
liblapack_path	∅	Additional directory to search for the LAPACK library
libmetis_path	∅	Additional directory to search for the METIS library
libmpi_path	∅	Additional directory to search for the MPI library
libmetis_include	∅	Additional directory to search for the METIS header files
libmpi_include	∅	Additional directory to search for the MPI header files

2.3 Compilation

After configuration, the source can be compiled in the build directory with

```
# Meson
meson compile -C builddir
```

2.4 Installation

Finally, the library, headers, and Fortran modules can be installed to the specified path with

```
# Meson
meson install -C builddir
```

This installs the library, headers, and Fortran modules in the folder specified by `--prefix`, if the argument is used. If the argument `--prefix` is not used, the default installation folder is `C:/` on Windows, and `usr/local` otherwise.

2.5 Running Examples

If LibHSL is configured to build the example programs (`-Dexamples=true` in the setup call), compiled versions of these will be available in your install directory as well as a utility program `run_example` to help run them. By default, example programs are for the Fortran interface. Programs postfixed with `_c` are examples written in C.

To keep the example code simple, data is read from `stdin` in each program. If the example requires an input, the data can be found in the install directory under `examples/Fortran/data` or `examples/C/data`. The expected output is also available in the install directory under `examples/Fortran/output` or `examples/C/output`.

To run an example on linux after installing with `--prefix=<installdir>`, the command would look like:

```
cd <installdir>/examples/Fortran
./hsl_ma97ds < data/hsl_ma97ds.data > output.txt
```

In order to simplify this and make it available to non-linux users the following is equivalent

```
cd <installdir>/examples
./run_example Fortran/hsl_ma97ds output.txt Fortran/data/hsl_ma97ds.data
```

`run_example` supports 1, 2, or 3 arguments:

- `run_example <example_program>` runs the example with no inputs and prints all outputs to stdout,
- `run_example <example_program> <output_file>` runs the example with no inputs and writes all outputs to `output_file`,
- `run_example <example_program> <output_file> <input_file>` runs the example with input from `input_file` and writes all outputs to `output_file`.

2.6 Cross-compilation

LibHSL can be easily cross-compiled with the Julia package BinaryBuilder.jl. We provide the script `build_tarballs.jl` to easily regenerate an HSL_jll.jl with different options or dependencies, such as METIS 4 instead of METIS 5. It can also be used to generate a static library `libhsl.a` for an application outside of Julia. Commands to cross-compile LibHSL are available in the file `build_tarballs.jl`. Note that the fortran compiler used to cross-compile LibHSL is `gfortran`.

3 Use LibHSL in C and Fortran projects

Once LibHSL is compiled, you can link it to your C and Fortran projects with

```
# C
-I${prefix}/include -L${prefix} -lhsl

# Fortran
-I${prefix}/modules -I${prefix}/include -L${prefix} -lhsl
```

By default Meson generates a shared library but, if you prefer to do static linking, the Meson's option `--default-library=static` can be used to generate a static library `libhsl.a`.

4 Use HSL_jll.jl in the Julia ecosystem

4.1 Installation of the HSL_jll.jl package

To install the package HSL_jll.jl, you only need the following commands in the Julia REPL:

```
julia> ]
pkg> dev path_to_hsl_jll
```

We recommend using the version precompiled with libblastrampoline_jll.jl if you use Julia 1.9 or future stable releases. The version precompiled with OpenBLAS32_jll.jl can be used in the long-term support (LTS) release Julia 1.6. Due to security policy of the macOS operating system, the Mac users may need to remove the quarantine before extracting.

```
xattr -d com.apple.quarantine openblas_HSL_jll.jl-2023.11.7.zip
xattr -d com.apple.quarantine openblas_HSL_jll.jl-2023.11.7.tar.gz
xattr -d com.apple.quarantine lbt_HSL_jll.jl-2023.11.7.zip
xattr -d com.apple.quarantine lbt_HSL_jll.jl-2023.11.7.tar.gz
```

4.2 Load an LP64 BLAS and LAPACK backend

By default, Julia ships with only an ILP64 version of OpenBLAS as BLAS and LAPACK backend. ILP64 libraries use the 64-bit integer type (necessary for indexing large arrays, with more than $2^{31}-1$ elements), whereas the LP64 libraries index arrays with the 32-bit integer type. LibHSL can only be linked with LP64 versions of BLAS and LAPACK as HSL uses 32-bit integer index arrays. Thus, one version of HSL_jll.jl is precompiled with OpenBLAS32_jll.jl and automatically loads an LP64 version of OpenBLAS with a `using HSL_jll`. For the version precompiled with libblastrampoline_jll.jl, the user needs to manually load one LP64 BLAS and LAPACK backend except if HSL_jll.jl is used with HSL.jl [4] or Ipopt.jl.

```
# Load the LP64 version of OpenBLAS
import LinearAlgebra, OpenBLAS32_jll
LinearAlgebra.BLAS.lbt_forward(OpenBLAS32_jll.libopenblas_path)

# Load LP64 and ILP64 versions of Intel MKL
using MKL

# Load LP64 and ILP64 versions of Apple Accelerate
using AppleAccelerate
```

We can verify what backends are loaded with `LinearAlgebra.BLAS.lbt_get_config()`. The user can also use their own LP64 version of BLAS and LAPACK. Note that LP64 and ILP64 versions can coexist if they have different symbols, such as a suffix `_64` in the naming conventions.

4.3 How to use HSL_jll.jl in Julia packages

A version 2.0 of HSL_jll.jl based on this dummy LibHSL was precompiled with Yggdrasil. Therefore HSL_jll.jl is a registered Julia package and can be added as a dependency of any Julia package. The dummy version only has one C function `LIBHSL_isfunctional` that returns the boolean `false`. The real version also exports this C function and returns the boolean `true`.

```
using HSL_jll

function LIBHSL_isfunctional()
    @ccall libhsl.LIBHSL_isfunctional()::Bool
end

bool = LIBHSL_isfunctional()
```

This function makes it possible to determine if we have a dummy version or not and consequently to call the C and Fortran routines of the genuine version.

The package HSL.jl provides wrappers for all HSL packages with a C interface or those written in Fortran 77. Wrappers are Julia functions that call C and Fortran routines with the macro `@ccall`. Thus, the combination of HSL.jl and HSL_jll.jl allows one to abstract away from the low-level C and Fortran languages and use the majority of HSL packages as if they were packages developed in Julia. Note that HSL.jl also loads the LP64 version of OpenBLAS automatically for the user if needed.

For the Julia interface Ipopt.jl, HSL_jll.jl must be used directly because it interacts with the Ipopt_jll.jl package under the hood.

```
using JuMP
import Ipopt, HSL_jll

# An optimization problem
model = Model(Ipopt.Optimizer)
@variable(model, x)
```

```
@objective(model, Min, (x - 2)^2)

# Load the HSL solvers
set_attribute(model, "hsllib", HSL_jll.libhsl_path)

# Use the linear solver MA57
set_attribute(model, "linear_solver", "ma57")
optimize!(model)

# Use the linear solver MA97
set_attribute(model, "linear_solver", "ma97")
optimize!(model)
```

The available HSL linear solvers are `"ma27"`, `"ma57"`, `"ma77"`, `"ma86"` and `"ma97"`. If no LP64 BLAS and LAPACK backend is loaded, Ipopt.jl loads OpenBLAS32_jll.jl automatically just like HSL.jl.

5 Future improvements

5.1 Support for ILP64 interfaces of BLAS and LAPACK

Currently, libHSL only supports LP64 versions of BLAS and LAPACK. A planned enhancement for the future is to add support for ILP64 versions of the library. This upgrade would enable the resolution of even larger-scale problems while simplifying the integration of libHSL into optimization solvers or differential equation solvers compiled with 64-bit integers. Extending to ILP64 would broaden the applicability of libHSL, meeting the growing demands of scientific and engineering applications.

5.2 GPU-accelerated linear solvers

Another anticipated improvement involves adding support for GPU versions of BLAS and LAPACK, such as CUBLAS for NVIDIA GPUs or rocBLAS for AMD GPUs. The integration of these GPU libraries into the linear solvers could significantly accelerate computations. However, this modification poses complex challenges, including efficient data transfers between the CPU and GPU.

6 Bug reports

If you think you have found a bug, feel free to open an issue. Useful suggestions and requests can also be opened as issues. If you would like to ask a question not suitable for a bug report, feel free to start a discussion.

References

[1] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A Fresh Approach to Numerical Computing. SIAM Rev., 59(1):65–98, 2017.

[2] N. I. Gould, D. Orban, and P. L. Toint. GALAHAD, a library of thread-safe fortran 90 packages for large-scale nonlinear optimization. ACM Trans. Math. Software, 29(4):353–372, 2003.

[3] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput., 20(1):359–392, 1998.

[4] A. Montoison, D. Orban, and contributors. HSL.jl: A Julia interface to the HSL mathematical software library, March 2021.

[5] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math. Program., 106(1):25–57, 2006.