Les Cahiers du GERAD ISSN: 0711–2440

`JuliaHSL.jl`: the ultimate collection for large scale scientific computation

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

G–2023–XX

May 2023

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

`JuliaHSL.jl`: the ultimate collection for large scale scientific computation

Jaroslav Fowkes [b]

Andrew Lister [b]

Alexis Montoison [a]

Dominique Orban [a]

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

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

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

May 2023
Les Cahiers du GERAD
G–2023–XX

Abstract : This paper presents `JuliaHSL`.

Keywords: Julia, numerical linear algebra

Résumé : Cet article présente `JuliaHSL`.

Mots clés : Julia, algèbre linéaire numérique

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. He expresses its appreciations to the HSL team for their invitation and their wonderful hospitality.

The authors would like to thank Stefan Vigerske for the METIS adapter, 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

JuliaHSL is the ultimate sparse linear algebra collection for large-scale scientific computing. It contains more than 155 HSL packages and aims to facilitate the use of HSL in Julia. JuliaHSL also integrates easily into Fortran and C projects such as GALAHAD.

JuliaHSL focuses on ease of use for users on all platforms by using a Meson build system. Meson enables the distribution of prebuilt binaries for the package using BinaryBuilder.jl. Additionally, JuliaHSL supports either METIS 5 or the older METIS 4.

This new package provides the source code of the included HSL packages as well as a Julia package named HSL_jll.jl. HSL_jll.jl is a pre-built version of JuliaHSL to be readily used in the Julia ecosystem. Once HSL_jll.jl is installed, the shared library that contains the C and Fortran routines of the HSL packages can be called in Julia and the HSL wrappers provided in the Julia interface HSL.jl are functional. HSL_jll.jl also provides an easy way to use the HSL linear solvers MA27, MA57, MA77, MA86 and MA97 within the IPOPT.jl interface to the IPOPT nonlinear optimization solver.

Two versions of HSL_jll.jl are available. 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.

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

2 Building JuliaHSL 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

```
meson setup builddir [options...]
CC=icc FC=ifort meson setup builddir [options...]
```

creates the build directory `builddir` and populates it in preparation for a build. Non-default compilers can be selected by setting the `CC` and `FC` shell variables. See this table for supported compilers.

2.3 Compilation

```
# Meson
meson compile -C builddir

# Ninja
ninja -C builddir
```

compiles JuliaHSL in the `builddir` folder.

option	default value	description
-Dmodules	true	install the Fortran module files
-Dlibmetis_version	5	version of the METIS library
-Dlibblas	blas	name of a BLAS library
-Dliblapack	lapack	name of a LAPACK library
-Dlibmetis	metis	name of a METIS library
-Dlibmpi	mpifort	name of a MPI library
-Dlibblas_path	[ ]	location of the BLAS library
-Dliblapack_path	[ ]	location of the LAPACK library
-Dlibmetis_path	[ ]	location of the METIS library
-Dlibmpi_path	[ ]	location of the MPI library
-Dlibmetis_include	[ ]	location of the METIS headers
-Dlibmpi_include	[ ]	location of the MPI headers

Table 1: Options supported by JuliaHSL

2.4 Installation

```
# Meson
meson install -C builddir

# Ninja
ninja -C builddir install
```

installs the library, headers and Fortran modules in `C:/` on Windows, and `usr/local` otherwise. This can be changed by passing the command line argument `--prefix` to Meson during configure time.

```
meson setup builddir --prefix=~/juliahsl
```

2.5 Cross-compilation

JuliaHSL 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 JuliaHSL are available in the file `build_tarballs.jl`.

3 Use JuliaHSL in C and Fortran projects

Once JuliaHSL 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 HSL_jll.jl-XXXX.YY.ZZ.zip
xattr -d com.apple.quarantine HSL_jll.jl-XXXX.YY.ZZ.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. JuliaHSL 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 or Ipopt.jl.

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

# Load the LP64 version of Intel MKL
import LinearAlgebra, MKL_jll
LinearAlgebra.BLAS.lbt_forward(MKL_jll.libmkl_rt_path)
```

We can verify what backends are loaded with `LinearAlgebra.BLAS.lbt_get_config()`. The LP64 and ILP64 versions of Intel MKL can also be loaded with the Julia package MKL.jl and a `using MKL`. In the future, an option to use BLIS and Apple Accelerate LP64 versions of BLAS and LAPACK will be available with the BLISBLAS.jl and AppleAccelerate.jl packages. The user can also use their own LP64 version of BLAS and LAPACK.

4.3 How to use HSL_jll.jl in Julia packages

A version 1.0 of HSL_jll.jl based on this dummy JuliaHSL 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 `JULIAHSL_isfunctional` that returns the boolean `false`. The real version also exports this C function and returns the boolean `true`.

```
using HSL_jll
```

```
function JULIAHSL_isfunctional()
    @ccall libhsl.JULIAHSL_isfunctional()::Bool
end

bool = JULIAHSL_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 Bug reports and discussions

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.