Running BoSSS Benchmarks

Florian Kummer  
Chair of Fluid Dynamics / TU Darmstadt  
kummer@fdy.tu-darmstadt.de  
Phone: 0049 (0)6151 16 5387

06sep10

# Introduction

* This document covers primary the Unix/Linux – specific issues of BoSSS. The Windows-Version can be run directly from the provided binaries, and only requires that Microsoft-MPI is installed.
* For getting best performance with mono, we strongly recommend to use the LLVM version of mono (http://www.mono-project.com/Mono\_LLVM); E.G, an optimized GEMM can profit more than a factor of 2 from LLVM;
* The C#-part of BoSSS is binary platform independent. No recompilation, for any platform should be necessary or is recommended
* The actual version requires the following sizes of the C-compiler: sizeof(int)=sizeof(MPI\_Fint) = 4;
* The Unix/Linux version of BoSSS is still preliminary and missing support for ParMETIS (which is not recommended for the benchmark examples, because the numerical gird is designed in a fashion that is perfectly balanced between MPI processes); also, no support for sparse solvers hypre or Aztec or PARDISO is included;

# Compiling / Running the BoSSS Navier – Stokes - solver

As mentioned above, only the C-Code parts (called the native part) of BoSSS need to be compiled.

## Compiling the native part on Unix-Systems

Requirements for compiling the native code : OpenMPI or MPICH2 with mpicc and mpif77, BLAS, LAPACK

**(Step 1)** Unzip BoSSS-Source.zip; subsequently, the newly created directory will be refffered to as $BoSSS\_ROOT.

**(Step 2 - optional)** execute the shell script $BOSSS\_ROOT/setvars-release.sh form directory $BOSSS\_ROOT; Remark: this step will append $BOSSS\_ROOT/bin/Release to the current path and start a child bash-process in the current console, to support some of the management tools like bcl.exe (see below).

**(Step 3)** Compile the native Code: enter directory  
$BOSSS\_ROOT/src/public/L0-native/src/Platform\_native/Platform\_Native  
and execute shell script   
compile.sh; (Attention: check the line endings: if DOS convention (CR+LF) convert to   
On success, this should produce the file   
$BOSSS\_ROOT/bin/native/unix-current/libPlatform\_Native.so

**Remarks (on linking):** libPlatform\_Native.so is linked using mpif77 (and not mpicc, as one would expect), because BoSSS uses the FORTRAN bindings of MPI and BLAS/LAPACK.

**Remarks (on BLAS and LAPACK):** The expected name mangling for FORTRAN BLAS and LAPACK functions is small-letters with trailing underscore, e.g. the function DDOT has the signature ddot\_(int\*, double\*, int\*, double\*, int\*) in C (the FORTRAN version, not the CBLAS – version!); This is known to work with Atlas on Ubuntu Linux and with Intel MKL on Windows. For another name mangling, one must modify the files  
BoSSS\_LAPACK.c, BoSSS\_BLAS.c, and compile.sh,  
all in the same directory; The necessary changes should be self-describing.

## Running BoSSS

Requirements: a running mono version (lowest tested version was 2.4.2.3), or a MS windows system;

**Remarks (on native libraries):** On Unix-like systems, the native library is called libPlatform\_Native.so. How BoSSS (in general, Mono-applications) search for this library depends on the implementation of the dlopen() – implementation of the operation system (see man dlopen). (See troubleshooting!).  
On Windows the native library is called Platform\_Native.dll; A “Bootstrapping”-routine will copy the correct format (32 or 64 Bit) from subdirectories ‘amd64’ or ‘x86’ of the application directory to the application directory. Unlike Linux, a Windows application first searches for a shared library in the applications directory.

**(Step 4)** copy native libraries (if running on a Unix – system, those which were created under step 2) to the $BOSSS\_ROOT/bin/Release – directory; This can be done either manually, or by executing  
# bcl.exe nativelib-inst

bcl.exe itself is a .NET – program too, located in $BOSSS\_ROOT/bin/Release, and therefore the command listed above requires a shell that passes \*.exe – files to the mono – runtime (like the bash in Ubuntu Linux); If such a shell is not supported, the command must be executed as

mono $BOSSS\_ROOT/bin/Release/bcl.exe nativelib-inst

Add the directory with the generated library to LD\_LIBRARY\_PATH:

export LD\_LIBRARY\_PATH=$BOSSS\_ROOT/bin/Release:$LD\_LIBRARY\_PATH

Note that, for security reasons, the LD\_LIBRARY\_PATH – variable is ignored on some Linux systems, see Troubleshooting; For testing, whether the library and its dependencies are loaded correctly, the command

ldd $BOSSS\_ROOT/bin/Release/libPlatform\_Native.so

may be used. If any library is missing, its directory should be added to LD\_LIBRARY\_PATH, e.g. by:

export LD\_LIBRARY\_PATH=/usr/lib64/openmpi/lib:$LD\_LIBRARY\_PATH.

**(Step 5)** Unpack the Database’s for the three different test cases (nse2-coarse.zip, nse2-med.zip, nse2-fine.zip) and the database for the system test (“BoSSS runs or runs not”). Subsequently, these directories are referred to as $DATABASE\_ROOT.

**(Step 6)** Modify the control files: the path to the databases must be entered to the control files.

Within the files control-\*.xml, the ‘path’ – attribute of the /BoSSSControl/Base/dboptions element must point to the corresponding database directory $DATABASE\_ROOT.

**(Step 7)** Run the application:

mpiexec *mpi-params* mono *mono-params* $BOSSS\_ROOT/bin/Release/NSE2.exe control-coarse.xml

Here, *mono-params* could be e.g. –llvm and *mpi-params* could be –n 64 to specify the number of MPI-processes.

## Evaluating Performance

For each run of the NSE2.exe – application, a subdirectory with a GUID-identifier name in the sessions-directory of the database is created, e.g. $DATABASE\_ROOT/sessions/8420abc8-42bd-48d3-ae89-f1ac86500bdc; This directory contains so-called trace files, trace.*n*.txt, where *n* denotes the MPI-process rank that give information about the runtime of individual methods (aka. Functions, subroutines).

We recommend

grep RunSolverOneStep trace.0.txt

to examine the runtime of one complete solver step and

grep PerformTransport trace.0.txt

grep PerformViscous trace.0.txt

grep PoissonSolver trace.0.txt

to find the runtime of the three major sub-steps; These times should be the result of the benchmark.

## Troubleshooting

**Unresolved native libraries (DllNotFoundExcpetion)**: The most common problem when starting a BoSSS application (NSE2.exe) on another machine is that the native libraries cannot be found. Here, it depends on the implementation of dlopen on the used system.

Form <http://www.mono-project.com/Interop_with_Native_Libraries>:

From the **dlopen**(3) man page, the necessary shared libraries needed by the program are searched for in the following order:

1. A colon-separated list of directories in the user's LD\_LIBRARY\_PATH environment variable. This is a frequently-used way to allow native shared libraries to be found by a CLI program.
2. The list of libraries cached in /etc/ld.so.cache. /etc/ld.so.cache is created by editing /etc/ld.so.conf and running **ldconfig**(8). Editing /etc/ld.so.conf is the preferred way to search additional directories, as opposed to using LD\_LIBRARY\_PATH, as this is more secure (it's more difficult to get a trojan library into /etc/ld.so.cache than it is to insert it into LD\_LIBRARY\_PATH).
3. /lib, followed by /usr/lib.

On Ubuntu Linux we had success when we placed libPlatform\_Native.so …

* … in the directory from which we called NSE2.exe (not the directory where NSE2.exe is stored!!!)
* … in the /usr/lib – directory.

We failed (on Ubuntu Linux) when we placed the file into the application directory and called NSE2.exe from another directory. Also, because of security reasons, Ubuntu ignores the LD\_LIBRARY\_PATH – variable, so the 1st option will not work.

**Extended Error Information:** If you get an error message, you can activate debug logging by setting an environment variable: export MONO\_LOG\_LEVEL=debug