MATLAB on HPC Systems

- What is MATLAB?
  MATLAB (= Matrix Laboratory) is a package that allows interactive and scripted numerical math. It is suitable for various fields of application in engineering and science.

License Terms and Usage Conditions

Licensing

Depending on your university/institute, different license models are applied (see table). According to the license conditions, LRZ is not allowed to provide access to installed MATLAB licenses to other Bavarian universities. Please contact us when you are interested in using MATLAB on LRZ systems on a medium-term or long-term scale. We will contact Mathworks in order to assess possible license models. Please also consult additional MATLAB licensing information.

<table>
<thead>
<tr>
<th>Customer</th>
<th>Licensing</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMU</td>
<td>The LRZ offers licenses for both MATLAB and Simulink including a large number of toolboxes and blocksets. Within the scope of the existing products, institutes may purchase licenses. Currently, modules not present in our price list cannot be purchased via LRZ. However, please report your needs by contacting our ServiceDesk. Then, it may be considered in the next centralized order.</td>
</tr>
<tr>
<td>TUM</td>
<td>Since October 1st, 2011 TUM university has its own contract on using a campus license. Since January 1st, 2012 the RBG (Rechnerbetriebsgruppe) of TUM – and not LRZ – is responsible for supply of licenses. For more details please visit: <a href="http://www.in.tum.de/rgb/beschaffung/rahmenvertraege/matlab.html">http://www.in.tum.de/rgb/beschaffung/rahmenvertraege/matlab.html</a></td>
</tr>
<tr>
<td>HM</td>
<td>Since 2017 the HM purchases its own licenses. Please consult the IT support of HM: Information for students Information for employees</td>
</tr>
</tbody>
</table>

Usage conditions on LRZ HPC systems

The MATLAB licenses purchased by LRZ are available on the LinuxCluster only, i. e., CoolMUC-2, CoolMUC-3, IvyMUC, housings and the Remote Visualization of the Linux Cluster.

Currently, MATLAB licenses purchased by LRZ may not be used on the supercomputers SuperMUC and SuperMUC-NG.

There is a limited number of MATLAB licenses on HPC systems at LRZ! Please consider that all entitled users share them! Please submit a reasonable number of simultaneous jobs. Furthermore, there is a limited number of licenses for MATLAB toolboxes which may differ from the number of MATLAB licenses.

Please consider the LRZ policy of using MATLAB on our systems:

- We encourage all MATLAB users to use SLURM batch jobs!
• MATLAB can be run interactively via GUI or command line. Running production jobs on login nodes using the interactive mode is not permitted!
• It is possible to run MATLAB in interactive mode (command line only) using an interactive SLURM job.
• On the condition that other users are not handicapped, it is allowed to run serial MATLAB on login nodes for small pre- or postprocessing steps or managing MPS jobs.

Support policy

By default LRZ provides the four latest Matlab releases and according support for issues, which are directly linked to the MATLAB installations and their configuration on the HPC systems at LRZ. For technical questions (such as: how to implement MATLAB code, problems with user-implemented code), you have several options:

• contact the MathWors Support or
• search for solutions or post new questions at the MATLAB Central.

TUM users may also consult the TUM MATLAB website to obtain more information on TUM's MATLAB licensing model and according technical support.

Both modules and installations of deprecated versions will be decommissioned. According to the support policy of Mathworks, we highly recommend to migrate Matlab codes to the latest release. Within the scope of the regularly updated LRZ software stack we provide the four latest MATLAB releases (see table). Releases might still exist in older versions of the Spack-based software stack (see details on LRZ software stack based on Spack modules) and might be usable. However, we only provide support for MATLAB releases of the newest software stack.

<table>
<thead>
<tr>
<th>MATLAB Release (Patch level)</th>
<th>MATLAB Compiler Runtime</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2019a</td>
<td>R2019a</td>
<td>recommended version</td>
</tr>
<tr>
<td>R2018b (Update 3)</td>
<td>R2018b</td>
<td></td>
</tr>
<tr>
<td>R2018a (Update 6)</td>
<td>R2018a</td>
<td></td>
</tr>
<tr>
<td>R2017b (Update 9)</td>
<td>R2017b</td>
<td>scheduled for decommission with installation of the next release</td>
</tr>
</tbody>
</table>

Getting Started

On Linux Cluster, you may list all available MATLAB versions:

```bash
> module avail matlab
```

If the desired MATLAB module is not listed, then list all available versions of the software stack, (re-)load the module of the software stack using the appropriate Spack version (cp. Spack releases) and check MATLAB modules again:

```bash
> module avail spack                           # List software stacks.
> module switch spack/release/<spack-version>  # Load official release.
> module switch spack/staging/<spack-version>  # Load unpublished "pre-release". Use at own risk. MATLAB MPS may not be configured (contact ServiceDesk for special needs)
> module avail matlab
```

MATLAB can be used by loading its module. Please consider, that the following command will load the default version. The default version is not fix. In general, it is set to the recent MATLAB release.

```bash
> module load matlab
```

We highly recommend to choose a particular – better: the newest – version, e. g.:

```bash
> module load matlab/R2018b-intel
```

Useful MATLAB commands

Please consider: All MATLAB commandline arguments are case-sensitive!

• Start MATLAB GUI:
Start MATLAB without desktop but allow GUI and graphics output:
```
> matlab -nodesktop
```

Start MATLAB without any GUI support:
```
> matlab -nodisplay
```

Some intrinsic MATLAB functions automatically exploit multithreading. MATLAB can be forced to disable this feature, which is important on login nodes:
```
> matlab -singleCompThread
```

Run a MATLAB script or function, e.g., `myfunc.m`:
```
> matlab -r myfunc
```

Interactive Jobs

Depending on the purpose, there are different possibilities to use MATLAB interactively:

- **MATLAB computations with focus on visualisation**: We recommend to use our Remote Visualisation System.
- **Pure MATLAB computations**: Interactive MATLAB sessions may be started on compute nodes of CoolMUC-2 or CoolMUC-3 by employing interactive Slurm jobs (see Slurm documentation).

Interactive Slurm jobs in a nutshell

Constraints

- Interactive jobs depend on the availability of compute resources. Matlab may not start immediately.
- Matlab will run on 1 compute node.
- The time limit is set to 2 hours (= maximum time for interactive Slurm sessions).

How to use it

1. Load desired MATLAB module.
2. For convenience, load the interactive-MATLAB module in order to use the loaded MATLAB version:
   ```
   > # use CoolMUC-2:
   > module load matlab-inter/coolmuc-2
   > # use CoolMUC-3:
   > module load matlab-inter/coolmuc-3
   ```
3. Start interactive MATLAB session. This command will implicitly submit an interactive Slurm job. Probably you have to wait if the interactive queue is fully occupied.
   ```
   > matlab-inter
   ```

Common Batch Jobs

Batch jobs are used for all MATLAB production runs. Usually, the resources, consumed by MATLAB applications, are limited to a single CPU core (pure serial job) or a single compute node (parallel job involving either multithreading or the Parallel Computing Toolbox [PCT]). Following table lists job examples for various cases which can be used on CoolMUC-2.

Please, use the serial queue of the Linux Cluster for those job types.
<table>
<thead>
<tr>
<th>Slurm job script</th>
<th>Matlab script</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Serial batch job</td>
</tr>
</tbody>
</table>
Parallel job using multithreading
#!/bin/bash
#SBATCH -o ./out/matlab_job.%j.%N.out
#SBATCH -e ./out/matlab_job.%j.%N.err
#SBATCH -D ./
#SBATCH -J matlab_threading_batch_job
#SBATCH --export=NONE
#SBATCH --nodes=1
#SBATCH --tasks-per-node=1
#SBATCH --cpus-per-task=14
#SBATCH --clusters=serial
#SBATCH --partition=serial_mpp2
#SBATCH --time=00:30:00
#SBATCH --mail-type=all
#SBATCH --mail-user=my_email_address@my_domain

module load matlab/R2018b-intel

# Example: matrix-matrix multiplication C = A*B
#          with A of size NROWA x NCOLA and
#          B of size NROWB x NCOLB
NROWA=1000
NCOLA=2000
NROWB=2000
NCOLB=5000

# Run MATLAB
# => Using option -r don't add file extension .m
to the function call!
# => MATLAB command line arguments are case-sensitive!
matlab -nodisplay \ 
-r "matmul_mthread([NROWA NCOLA], [NROWB NCOLB]);"
% Work

fprintf('Hello from MATLAB process PID=%d running on node %s!
',...
feature('getpid'),...
strtrim(evalc('system('''hostname''');')));

% generate well-defined matrices
NA = prod(size_A);
NB = prod(size_B);
A = reshape( linspace( 1,NA, NA), size_A);%
B = reshape( linspace(NB, 1, NB), size_B);

% compute
tic;
C = A*B;
comptime = toc;
fprintf('parallel computation (multithreading) of matrix-matrix product:
');
fprintf('	number of threads = %d
', nw);
fprintf('	time = %.2f s
', comptime);

function [Cglob, comptime] = matmul_pct(size_A, size_B)

% MATLAB EXAMPLE: PARALLEL HELLO WORLD USING PCT TOOLBOX
% -> matrix-matrix multiplication C = A*B
% INPUT
% size_A, size_B ... 2-element row vectors defining sizes of A and B
% OUTPUT
% C ................. result
% comptime .......... computation time (matrix product only)
%
% Check input
%
if nargin~=2
    error('Invalid number of input arguments!');
end
if size_A(2)~=size_B(1)
    error(sprintf('Dimension mismatch of A (%d columns) and B (%d rows)!''...
        size_A(2), size_B(1)));
end
#!/bin/bash
#SBATCH -o ./out/matlab_job.%j.%N.out
#SBATCH -e ./out/matlab_job.%j.%N.err
#SBATCH -D ./
#SBATCH -J matlab_pct_batch_job
#SBATCH --get-user-env
#SBATCH --export=NONE
#SBATCH --nodes=1
#SBATCH --tasks-per-node=4
#SBATCH --cpus-per-task=1
#SBATCH --clusters=serial
#SBATCH --partition=serial_mpp2
#SBATCH --time=00:30:00
#SBATCH --mail-type=all
#SBATCH --mail-user=my_email_address@my_domain

# IMPORTANT
# Default settings of Intel MPI module may disrupt
# functionality of Parallel-Computing-Toolbox!
# Do one of the following solutions:
# (1) Unload module mpi.intel:
module rm mpi.intel
# (2) If Intel MPI module is mandatory, add next line
# export KMP_AFFINITY=granularity=thread,none
module load matlab/R2018b-intel

# Example: matrix-matrix multiplication C = A*B
# with A of size NROWA x NCOLA and
# B of size NROWB x NCOLB
NROWA=1000
NCOLA=2000
NROWB=2000
NCOLB=5000

# Run MATLAB
# => Using option -r don't add file extension .m
to the function call!
# => MATLAB commandline arguments are case-sensitive!
m Watson r 0 display 1 -singleCompThread \ -r "matmul_pct([$NROWA $NCOLA], [$NROWB $NCOLB]);"

% Verify that no parallel pool is initialized by creating/deleting a dummy pool
%
if ~isempty(gcp('nocreate'))
    poolobj = gcp('nocreate');
    delete(poolobj);
end

% Start parallel pool
%
% Get number of workers depending on job type. Start parallel pool via "local"
% cluster object.
% obtain number of tasks from Slurm environment variables
cluster = getenv('SLURM_CLUSTER_NAME');
if strcmp(cluster, 'inter')
    nw = str2num(getenv('SLURM_JOB_CPUS_PER_NODE'));
elseif strcmp(cluster, 'mpp2') || ...
    strncmp(cluster, 'mpp3') || ...
    strncmp(cluster, 'ivymuc')
    % batch job
    nw = str2num(getenv('SLURM_NTASKS'));
else
    % default
    nw = 1;
end

% disallow Threading
if maxNumCompThreads > 1
    maxNumCompThreads(1);
    warning('MultiThreading: number of threads has been set to 1!');
end

% create a local cluster object
pc = parcluster('local');
% set number of workers
pc.NumWorkers = nw;
% set the JobStorageLocation to SCRATCH (default: HOME - not recommended)
pc.JobStorageLocation = strcat(getenv('PTMP'));
% start the parallel pool
poolobj = parpool(pc, nw);

% Work
%
snd
fprintf('Hello from MATLAB process PID=%d running on node %s!\n',...
    feature('getpid'),...
    getenv('HOSTNAME'));
end

% generate well-defined matrices
NA = prod(size_A);
NB = prod(size_B);
A = reshape( linspace( 1,NA, NA ), size_A );
B = reshape( linspace(NB, 1, NB), size_B );

% distribute data to workers and do parallel computation
spmd
    Aloc = codistributed(A, codistributor2dbc([nw 1]));
   Bloc = codistributed(B, codistributor2dbc([1 nw]));
    tic;
    Cloc = Aloc*Bloc;
    t = toc;
end
comptime = max(cell2mat(t(:)));
fprintf('parallel computation (MPI) of matrix-matrix product:
');
fprintf('	number of tasks (MATLAB workers) = %d
',nw);
fprintf('	time = %.2f s
', comptime);
Cglob = gather(Cloc);

MATLAB Parallel Server (MPS)

Please note: This MATLAB product is available
- to particular TUM users only,
- on LRZ systems for MATLAB releases R2018a and later.

The MPS (fka MATLAB Distributed Computing Server (MDCS) in R2018b and older releases) extends the functionality of the Parallel Computing Toolbox by allowing parallel jobs across multiple compute nodes. Currently, a MPS license is available, which allows the use of up to 32 Matlab workers in a single parallel job. MPS jobs are handled as common parallel jobs using the Slurm queueing system in the background. MPS jobs can be submitted from both the login nodes of the Linux Cluster and the user's remote computer (e.g., laptop, desktop PC). Hereafter, we briefly describe both ways. Please also consult the MPS User Guide for use on CoolMUC-2 for detailed and MATLAB Parallel Server for general information.

Submit MPS job on a login node of the Linux Cluster

In order to run your parallel MATLAB code exploiting PCT + MPS, you have to follow following steps:

<table>
<thead>
<tr>
<th>Step</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Start MATLAB on one of the Linux Cluster login nodes</td>
<td></td>
</tr>
<tr>
<td>2. Job configuration for CoolMUC-2</td>
<td>Run cluster configuration. This step is mandatory. Otherwise, MATLAB will use its default cluster settings ('local' cluster) which will not work! Both name of the cluster (e.g. serial) and the name of the partition/queue (e.g. serial_mpp2) have to be passed to configCluster().</td>
</tr>
</tbody>
</table>

```matlab
>> configCluster(cluster_name, cluster_queue);
>> ch = parcluster;
```
Create a cluster object and return the cluster object handle.
Define job parameters (members of the cluster object). MPS will translate all settings to the according Slurm flags (needed by the sbatch command, see documentation of the Slurm Workload Manager at LRZ). Please consider, that only most important Slurm flags are provided by the cluster object. Following parameters can/must be adjusted by the user.

Further Slurm flags can be added to the cluster object as a space-separated string using the field AdditionalSubmitArgs.

There are pre-defined parameters which may not be changed:

- job output: logfiles for output (*.out) and error (*.err) are created automatically

MPS will store both results of user code (MATLAB's "mat" file format) and job output to the file system. By default the HOME directory is used. Due to performance and capacity reasons, we highly recommend to use the SCRATCH partition.

**NOTE:** Depending on the usecase, the output might exceed the maximum size of a mat file. The job will finish successfully. However, the data will be lost. Hence, we also recommend that the user code directly writes all data to the file system.

Submit job, which will run the user code 'myfunction.m', by calling the batch function as a member of the cluster object. The input/output arguments are as follows:

**Input:**
@myfunction ... reference to myfunction
n_arg_out ..... number of output arguments of myfunction
arg_in_# ..... list of input arguments of myfunction
'Pool', np .... key-value pair with size of parallel pool = number of workers

**Output:**
job ........... job object providing all job information and member functions to control the job

**IMPORTANT:**
The setting of np workers will result in the allocation of np+1 tasks, because MPS requires an additional management task!
Example: The job uses 14 tasks per node and 28 workers in total. Including an additional task a third compute node with only one task will be involved into the job. That results in inefficient resource usage and probably longer waiting times!
Submit MPS job on remote computer

This feature is still under development.

Licensing and troubleshooting

The following bash command performs a MPS license check and can be used in two cases:

- You may check, whether you are allowed to use the MPS.
- The output might be helpful in troubleshooting of licensing issues. Please attach it to incident tickets submitted to the support.

If MATLAB prints the list of installed toolboxes then everything is fine. Otherwise, you will receive an error message.

> matlab -dmlworker -nodisplay -r "ver,quit"

The number of MPS licenses is limited. Slurm will automatically queue MPS jobs unless licenses are available (please consider: our Slurm regulations are applicable and there is no guaranty that jobs start immediately). You may check the availability of MPS licenses via following bash command on one of the login nodes:

> scontrol show licenses --clusters=mpp2 | grep -A1 "MATLAB_Distrib_Comp_Engine"

For troubleshooting purposes you may also run a license check using the license command (as shown below) within a MATLAB session. If it returns "1" then the MPS license exists.

>> license('checkout','distrib_computing_toolbox')

MPS Examples

The following table shows two examples using either spmd environment or parfor loop. For convenience, the MATLAB file "job_config.m" summarizes all configuration steps and submits the job to Slurm. Start MATLAB and run job_config, for example:

>> myfunction = 'myfunction_spmd';
>> % or
>> myfunction = 'myfunction_parfor';
>>
>> cluster_name = 'serial';
>> cluster_queue = 'serial_mpp2';
>> walltime = '00:30:00';
>> tasks_per_node = 28;
>> num_worker = 5;
>>
>> [job,ch] = job_config(myfunction, cluster_segment, cluster_queue, walltime, tasks_per_node, num_worker);

<table>
<thead>
<tr>
<th>Configuration script</th>
<th>Implementation of user-defined function</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_config.m</td>
<td>spmd example</td>
</tr>
</tbody>
</table>
function [job,ch] = job_config(example, ...  
cluster_name, ...  
cluster_queue, ...  
walltime, ...  
tasks_per_node, ...  
num_worker)

% MATLAB MDCS EXAMPLE
% -> configuration script to initialize MDCS and submit job
% input:
% example .................... name of user function without extension ".m"
% cluster_name ............... name of cluster, e.g.: serial (refers to the HPC machine, e.g. CoolMUC-2)
% cluster_queue .............. name of queue /partition, e.g.: serial_mpp2
% walltime, tasks_per_node ... equivalent to Slurm parameters
% num_worker .................. number of MDCS workers
% return values:
% job ........................ job handle
% ch ........................ cluster object handle

% Step 1: cluster configuration
configCluster(cluster_segment, cluster_queue);

% Step 2: job configuration
ch = parcluster;
jobdir = fullfile(getenv('SCRATCH'),  
'MdcsDataLocation/coolmuc/R2018b');
if ~exist(jobdir)
    mkdir(jobdir);
end
ch.JobStorageLocation = jobdir;
ch.AdditionalProperties.WallTime = walltime;
ch.AdditionalProperties.TasksPerNode = tasks_per_node;
ch.saveProfile;

% Step 3: job submission to Slurm

myfunction_spmd.m

function [nlabs,comptime,Cref,Cfin] = myfunction_spmd

% MATLAB EXAMPLE: PARALLEL HELLO WORLD USING PCT TOOLS
% -> matrix-matrix multiplication C
% return values:
% nlabs ...... total number of workers (just FYI and used by data distribution
% comptime ... time needed for multiplication
% Cref ...... reference result obtained from serial computation
% Cfin ...... result obtained from parallel computation

% Input

% exemplary matrices
SIZE_A = [2000 100];
SIZE_B = [100 8000];
A = zeros(SIZE_A);
B = zeros(SIZE_B);

for n=1:SIZE_A(1)
    A(n,:) = linspace(1,n, SIZE_A(2));
end
for n=1:SIZE_B(1)
    B(n,:) = linspace(1,n, SIZE_B(2));
end

% reference result
Cref = A*B;

% Manage parallel pool
spmd
    nl = numlabs;
end
nlabs = nl(:);

disallow Threading
maxNumCompThreads(1);

% Parallel work
% Command:
%   job = ch.batch(@myfunction, n_arg_out, [arg_in_1, ..., arg_in_n], 'Pool', np)
% Input:
%   @myfunc ...... reference to user-defined function
%   myfunc.m
%   n_arg_out .... number of expected output arguments
%   {arg_in_1, ..., arg_in_n} ... list of input arguments of myfunction
%   'Pool', np ... key-value-pair: define size of pool (number of workers)
% Help via Matlab commands:
%   help batch
%   doc batch

fhandle = eval(sprintf('@%s', example));
job = ch.batch(fhandle, 4, {}, 'Pool', num_worker);

parfor example

myfunction_parfor.m

function [nlabs,comptime,Cref,C] = myfunc_parfor
  % MATLAB EXAMPLE: PARALLEL HELLO WORLD USING PCT TOOLBOX
  % -> matrix-matrix multiplication C
  % return values:
  % nlabs ...... total number of workers (just FYI and used by data distribution
  % functions)
  % comptime ... time needed for multiplication
  % Cref ....... reference result obtained from serial computation
  % Cfin ....... result obtained from parallel computation

  % exemplary matrices
  SIZE_A = [2000 100];
  SIZE_B = [100 8000];
  A = zeros(SIZE_A);
  B = zeros(SIZE_B);

  spmd
    % distribute data to all workers
    Ad = codistributed(A, codistributor2dbc([nlabs 1]));
    Bd = codistributed(B, codistributor2dbc([1 nlabs]));
    Cd = zeros(SIZE_C, codistributor2dbc([1 nlabs]));

    % timing
    tic;
    Cd = Ad*Bd;
    t = toc;
  end

  % collect data from all workers => final result
  Cfin = gather(Cd);
  comptime = t;
for n=1:SIZE_A(1)
    A(n,:) = linspace(1,n, SIZE_A(2));
end
for n=1:SIZE_B(1)
    B(n,:) = linspace(1,n, SIZE_B(2));
end
% reference result
Cref = A*B;

% Manage parallel pool

% Parallel work
SIZE_C = [SIZE_A(1) SIZE_B(2)];
C = zeros(SIZE_C);
% timing
tic;
% parallel environment
parfor n=1:SIZE_A(1)
    % compute
    C(n,:) = A(n,:)*B;
end
comptime = toc;
MATLAB Compiler Runtime (MCR)

The purpose of the MATLAB Compiler Runtime is to run MATLAB standalone applications on systems without a MATLAB installation. The standalone application is compiled by the MATLAB Compiler which requires a licensed version of MATLAB. In contrast, the standalone application should be executable on any system providing a MCR installation, and without the need of a MATLAB license. The MATLAB version used for compilation has to match the MCR version at runtime! On LRZ systems we provide all MCR versions which correspond to the installed MATLAB versions. Using examples from the section “Common Batch Jobs”, exemplary basic workflows are shown in the following subsection.

Purely serial MCR job

<table>
<thead>
<tr>
<th>Step</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1. Create folder structure of example</strong></td>
<td></td>
</tr>
<tr>
<td>mcr_serial_job</td>
<td>-&gt; base path</td>
</tr>
<tr>
<td>bin</td>
<td>-&gt; path for executable file</td>
</tr>
<tr>
<td>mcr_build.m</td>
<td>-&gt; build script running MATLAB Compiler</td>
</tr>
<tr>
<td>mcrjob.cmuc2.slurm</td>
<td>-&gt; Slurm script to run MCR job</td>
</tr>
<tr>
<td>src</td>
<td>-&gt; application sources: m-files</td>
</tr>
<tr>
<td>dep1</td>
<td>-&gt; dependencies</td>
</tr>
<tr>
<td>matmul_serial.m</td>
<td>-&gt; serial matrix-matrix multiplication</td>
</tr>
<tr>
<td>dep2</td>
<td>-&gt; dependencies</td>
</tr>
<tr>
<td>plot_save_matrix.m</td>
<td>-&gt; plot result</td>
</tr>
<tr>
<td>mcr_run.m</td>
<td>-&gt; run script of application (&quot;main&quot; function)</td>
</tr>
</tbody>
</table>

| **2. Prepare application sources** | |
| mcr_run.m | * "main" function which will be compiled by MATLAB Compiler |
| | * can be adjusted to any usecase |
| | * main purposes: |
| | * set MCR cache to SCRATCH (otherwise, it uses HOME which is not recommended) |
| | * displaying debug information |
| | * calling application code |
function mcr_run(size_A_str, size_B_str)

% MATLAB MCR EXAMPLE: RUNTIME CODE
% This function will be compiled by Matlab Compiler. This function
% summarizes some configuration steps, e.g. defining dependencies
% calls the Matlab function implemented by the user
% This function calls matmul_mthread.m
% (matrix-matrix-multiplication with threading support) as dependency 1
% and a plotting function as dependency 2.
% %
% % Set MCR cache path to SCRATCH. Don't use HOME directory!
% setenv('MCR_CACHE_ROOT', getenv('SCRATCH'));
% %
% % Show some information, also useful for troubleshooting
% disp(['MATLAB temporary directory = ' tempdir]);
disp(['MCR_CACHE_ROOT = ' getenv('MCR_CACHE_ROOT')]);
disp(['MCR root dir = ' ctfroot]);
disp(['Hostname = ' strtrim(evalc('!hostname'))]);
% % Run user code.
% % get sizes of input matrices for computation of C = A*B from commandline arguments
size_A = str2num(size_A_str);
size_B = str2num(size_B_str);
[C, comptime] = matmul_serial(size_A, size_B);
plot_save_matrix(C, 'matmul_result');
fprintf('computation time = %.4f.
', comptime);
matmul_serial.m

- dependencies of application "mcr_run.m"
- matmul_serial.m: similar to example shown in section "Common Batch Jobs"
- plot_save_matrix.m: auxiliary plotting function
function [C, comptime] = matmul_serial(size_A, size_B)

%% MATLAB EXAMPLE: SERIAL HELLO WORLD
% \rightarrow matrix-matrix multiplication C = A*B
%%
%% INPUT
% size_A, size_B ... 2-element row vectors defining sizes of A and B
%% OUTPUT
% C ................. result
% comptime .......... computation time (matrix product only)
%%
% Check input
%%
if nargin\textasciitilde{}2
    error('Invalid number of input arguments!');
end
if size_A(2)\textasciitilde{}size_B(1)
    error(sprintf('Dimension mismatch of A (%d columns) and B (%d rows)!',...
        size_A(2), size_B(1)));
end

%% Work
%%
% Hello message from compute node
fprintf('Hello from MATLAB process PID=%d running on node %s\n',...
    feature('getpid'),...
    strtrim(evalc('system (''hostname'');')));

% generate well-defined matrices
NA = prod(size_A);
NB = prod(size_B);
A = reshape(linspace(1,NA,NA), size_A);
B = reshape(linspace(NB,1,NB), size_B);

% compute
 tic;
 C = A*B;
 comptime = toc;
 fprintf('serial computation of matrix-matrix product:\n');
 fprintf('%2f s\n', comptime);
plot_save_matrix.m

function plot_save_matrix(A, fname)

% Plot matrix A and save figure to file fname.

fig = figure('visible','off');
imagesc(A);
saveas(gcf, [fname '.png']);

3. Check build script

mcr_build.m

function mcr_build(srcpath, mainfunc, threadingopt)

% MATLAB MCR EXAMPLE: BUILD CODE
% This function calls Matlab Compiler to build an executable from user code.
% INPUT
% srcpath ........ base path to sources (m-files)
% mainfunc ....... name of main M-function to be compiled
% -> file name of function only (no path)
% -> without extension ".m"
% -> according m-file must be located in srcpath
% threadingopt ... set multithreading mode
% -> disable: '-singleCompThread'
% -> enable: ''
% (leave string empty)

% Construct basic compiler command.

cmd = ['mcc' ...
'  -m' ...
'  -R -nodisplay' ...
'  -R ' threadingopt ...
'  -o ' mainfunc ...

This is a generic build script which can be used for any use case. Here, it is not necessary to modify it.
4. Compile application

If you work on the Linux Cluster, please load the desired MATLAB module and then call this command to compile. It is allowed to run this compilation procedure on the login nodes.

In this example, we compile a purely serial code with multithreading being disabled.

5. Prepare Slurm batch script and run application

First panel:
- show available MCR modules on LRZ systems

Second panel:
- Job script to run application on CoolMUC-2
- Remember: The MATLAB version used for compilation has to match the MCR version at runtime!
APPNAME=./bin/mcr_run.exe
APPPATH=$HOME/MATLAB/EXAMPLES/doku.lrz.de/MCR/serial_job

SIZE_MATRIX_A="[2000 1000]"
SIZE_MATRIX_B="[1000 8000]"

# Default settings of module mpi.intel cause compatibility problems with Matlab
# in terms of disrupting functionality of Parallel-Computing-Toolbox!
# Solution: Reset environment variable KMP_AFFINITY
export KMP_AFFINITY=granularity=thread,none

# Load Matlab Compiler Runtime
module rm matlab
module load matlab-mcr/R2018b

# check
echo "MCR root path = $MCRROOT"
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:${MCR_ROOT}/runtime/glnxa64
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:${MCR_ROOT}/bin/glnxa64
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:${MCR_ROOT}/sys/os/glnxa64
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:${MCR_ROOT}/sys/opengl/lib/glnxa64

# some Matlab releases use environment variable TMP for temporary directory
export TMP=$SCRATCH

# RUN APPLICATION
cd $APPPATH
mpiexec -n 1 ./bin/mcr_run.exe "${SIZE_MATRIX_A}" "${SIZE_MATRIX_B}"

Parallel MCR job
Parallel batch jobs (job-farming)
Content coming soon.
Links and other contact options

- MathWorks Support
- MATLAB Central

address of MathWorks Germany:

The MathWorks GmbH
Adalierostr. 45
D - 85737 Ismaning

Tel.: 089 - 45235-6700
Fax : 089 - 45235-6710