# Practical:
# Porting applications to the GILDA grid

**Slides based on Vladimir Dimitrov's work, IPP-BAS**

**Application from Gabor Hermann, MTA SZTAKI**

www.eu-egee.org

Information Society

# Contents

- Introduction
- **Practical:** Preparing and submitting a job starting from a non-grid application.
- **Talk:** Discussing common problems and obstacles of porting applications to a Grid while awaiting the job results.
- **Practical:** Retrieving and inspecting the job results.
- Final remarks.

# Introduction

## The main goal:

- **To port and execute an existing non-grid application to the Grid.**

  (Currently we use GILDA Grid.)

Some sources define this process commonly as "**gridification".** There are many useful and "single-processor" or "single-machine" applications which need gridification.

# Basic tasks while porting applications to the Grid

1. **Developing a non-grid application (or inheriting and updating a legacy one)**
2. **Executing, Testing and Debugging the application on a single machine**
3. **Constructing the grid suite**
    1. Write JDL (Job Description Language) files,
    2. Modify / extend executables
        - *Write auxiliary scripts or components that interact with grid services*
    3. Store input/output data files on storages;
4. **Start the gridified application on the Grid;**
5. **Execute, Test and Debug the application;**
6. ***IF* something goes wrong**
   *THEN GOTO* 3 (or 2);
7. **Scale up the application to the production level**
    - Larger input files
    - Larger parameter set
    - More grid resources

# The practical

- **Goal: The application called *MatrixDemo* will be ported and executed in GILDA grid environment.**

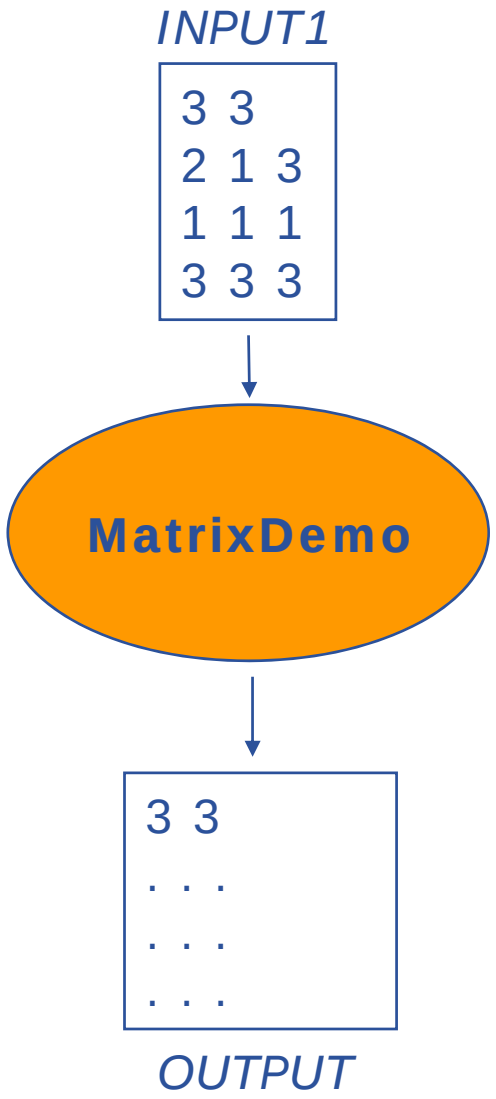
***MatrixDemo* is written in C programming language.**

Many Grids, especially GILDA and EGEE Grid middleware (gLite) are based on Globus Toolkit, (http://www.globus.org/).

**The Globus Toolkit is written in C, so porting the C or C++ programs is easy … probably.**

# MatrixDemo program

- C code
- Reads matrix from a file called INPUT1
- Writes inverted matrix to a file called OUTPUT
- Requires command line parameters: I V
- `./MatrixDemo I V`



# *MatrixDemo* program (continued)

- **Prerequisites:**

✓ **File *MatrixDemo.c*** – the source code of the program.

✓ **File *INPUT1*** – it contains a sample input matrix

✓ **A standard C compiler and linker.** In this case we will use GNU C (gcc) – already installed.

✓ **File *MatrixDemo.jdl*** – a prepared JDL (Job Description Language) file.

# *MatrixDemo*: compilation

- **Step:**

1. **Log on to the GILDA user interface using SSH (Secure shell) from your Desktop.**
   (The user input is given in red color.)

   Hostname: **`glite-tutor.ct.infn.it`**

   login as: **`budapest`*`XX`***

   (where *XX* is your number)

   Password: **`GridBUD`*`XX`***

   (where *XX* is your number)

# *MatrixDemo*: getting the prerequisites

- **Step:**

2. **Download the prerequisites stored in a zipped file *MatrixDemo.zip* with the following command:**

```
wget http://vgd.acad.bg/MatrixDemo.zip
```

**Unzip the archive in your current directory with the command:**

```
unzip MatrixDemo.zip
```

(This will create a subdirectory *matrix* with all of the prerequisite files inside.)

**Change the current directory:**

```
cd matrix
```

# *MatrixDemo*: compilation

- **Step:**

3. **Compile and link the program using GNU C compiler / linker:**

```
gcc -o MatrixDemo MatrixDemo.c
```

This will create an executable file *MatrixDemo*.

**Look at the directory contents:**

```
ls -l
```

# *MatrixDemo*: testing as a non-grid application

- **Step:**

4. **Invert the matrix stored in *INPUT1* file with the following command:**

```
./MatrixDemo I V
```

**Look at the content of the input file *INPUT1*:**

```
more INPUT1
```

**Look at the content of the output file *OUTPUT*:**

```
more OUTPUT
```

**And you may examine the source code:**

```
more MatrixDemo.c
```

# gLite: entering the Grid!

- **Step:**

5. **Login to the GILDA Grid:**

```
voms-proxy-init --voms gilda
```

This will ask for the passphrase which is BUDAPEST for all users.

**Check the proxy status with:**

```
voms-proxy-info
```

# gLite: Checking the job requirements

- **Step:**

**6. Investigate the abilities to run the job among the Grid-sites with gilda VO support:**

```
edg-job-list-match MatrixDemo.jdl
```

This command will produce a listing with all of the Grid Computing elements together with jobmanager queues that fulfill the requirements of our job.

# gLite: Submitting the job to GILDA Grid

- **Steps:**

7. **Execute the following command:**

```
edg-job-submit -o MatrixDemo.id MatrixDemo.jdl
```

This will submit the job and will store its unique identifier in a file called *MatrixDemo.id.* You may look at that file.

8. **Monitor the job status with:**

```
edg-job-status -i MatrixDemo.id
```

Execute this command several times until **“Done (Success)”** status.

*PLEASE STOP AT THIS POINT. TALK CONTINUES…*

# Some theory: Common problems and obstacles

- The candidate-applications for porting usually are huge and complex.
- Some of them use low-level network functions and/or parallel execution features of a specific non-grid environment.
- Usage of non-standard or proprietary communication protocols.
- The complete source code might not be available, might not be well documented or its “out-of-host” usage is  restricted by a license agreement.

# Common obstacles (continued)

- The application might be written in many different programming languages – C, C++, C#, Java, FORTRAN etc. or even mixture of them.
- Applications may depend on third-party libraries or executables which are not available by default on some Grid worker nodes.
- Some application features could cause unintentional violation of Grid Acceptable Use Policies (Grid AUP).
- Furthermore, the application can have hidden security weakness which will be very dangerous in case of remote Grid job execution.

# Common obstacles (continued)

- Some applications are pre-compiled or optimized for using on a machine with particular processor(s) only – Intel, AMD, in 32-bit or 64-bit mode, etc. But the Grid is heterogeneous!
- The application may contain serious  bugs  which have never been  detected while running in a non-grid environment.
- Finally, the formal procedure for accepting a new application to be ported to a Grid for production or even experimental purpose is not simple.

**Therefore, the porting of an arbitrary application to Grid could be very long, difficult and expensive process!**

# Support for application gridification

- **SZTAKI provides *Grid Application Support Centre***
- **Open to any grid community**
- **Support cycle:**
  - *Contact phase:* provide us with input – fill out and return the Application Description Template
  - *Pre-selection, preliminary analysis and planning phases:* SZTAKI creates a generic “how-to” document – guide for the gridification
  - *Prototyping, testing, execution phases:* the gridified version is created and exposed on a production VO with our and EGEE NA4 help
  - *Dissemination and feedback:* let the whole grid community benefit from our experiences and achievements!
- **More information**
  - www.lpds.sztaki.hu/gasuc

# Practical (continued): retrieving the job results

- **Step:**

9. **Execute the following command:**

```
edg-job-get-output -i MatrixDemo.id -dir ./
```

This will retrieve the Output sandbox files and will store them into a local directory with a strange name under the current directory. Directory name will be something like *sofia01_aJiesiAtu96H09XASy_j_Q.*

Enter the output directory and look at the files named *OUTPUT* and *std.out*

```
more OUTPUT
more std.out
```

# *MatrixDemo*: the JDL-file

- **Step:**

**10. Look to the supplied *MatrixDemo.jdl* file:**

```
more MatrixDemo.jdl
```

The *MatrixDemo.jdl* contents:

```
[
        VirtualOrganisation = "gilda";
        Executable = "MatrixDemo";
        JobType = "Normal";
        Arguments = "I V";
        StdOutput = "std.out";
        StdError = "std.err";
        InputSandbox = {
                "MatrixDemo",
                "INPUT1"
                };
        OutputSandbox = {
                "std.out",
                "std.err",
                "OUTPUT"
        }
]
```

# *MatrixDemo*: preparing the JDL-file (continued)

- **Short explanation of some important JDL-attributes:**
  - **`VirtualOrganisation`** – this points to our training VO (gilda);
  - **`Executable`** – sets the name of the executable file;
  - **`Arguments`** – command line arguments of the program;
  - **`StdOutput, StdError`** - files for storing the standard output and error messages output;
  - **`InputSandbox`** – input files needed by the program, including the executable;
  - **`OutputSandbox`** – output files which will be written during the execution, including standard output and standard error output;

# If you have time…

- **Modify the JDL and submit the job in a way that it will produce multiplication of the two matrices stored in *INPUT1* and *INPUT2* files.**

- **Hint: try `./MatrixDemo M V`**

**Questions?**