PyAALib-JyAALib
Asynchronous Action Library Project

The Framework to Develop Massively Parallel Applications

Romeu Andre' PIERITZ
SciSoft Group - Exp. Division - ESRF

PyAALib-JyAALib
Asynchronous Action Library Project

What is it?

- ✓It is a simple code framework to develop massively parallel applications;
- ✓It provides a “Dynamic Parallel Application Skeleton”;
- ✓Any code can be added to the “application skeleton” as a full dynamic plug-in;
- ✓The framework is designed to be natively compiled in Java using Jython – JyAALib;

Why are we doing PyAALib-JyAALib?

- ✓ To Develop Parallel Online Data Analysis Code for Smart Experiments
- ✓ To Process Online Data in Parallel using different models to compare results
- ✓ To Process a Huge amount of Data in Parallel

Basic concept

ALAction

It is a complete atomic independent “thread control and exception system”

ALActionController

ALActionMethod

ALActionSetAsynchronous

ALActionProcess

ALActionSet

ALActionMethodTimer

ALActionSetSynchronous

ALActionCluster

ALPlugin

It is a multi-thread “Dynamic” code container

Basic concept – “Goodies”

ALApplication

It is a
Generic Parallel Application
“skeleton”

ALPluginFactory

ALXsdDataBinding

ALManagerProcess

ALManagerTest

ALCommandLine

ALCommandLineInterface

…others…

ALXmlRpcServer

It is the basis of the PyAALib
“GridRPC” – Grid Remote Procedure Call
technology

Basic concept

ALApplication

It is a
Generic Parallel Application
"skeleton"

- ✓ It is based on a dynamic parallel plug-in architecture;
- ✓ It is full operational from a single line of code - "fast developing";

What does that mean?

- ✓ it generates and controls the Asynchronous
  "log file, screen outputs and DEBUG flow control";
- ✓ It manages the "application resource file" = "application data persistence"
- ✓ It manages the multi-thread command line interpreter;
- ✓ It manages the "plugin factory engine":
  plug-in = search+load+compiling+import + object instance generation
- ✓ It manages the "thread pool";

A real example: “Hello World” in parallel

10,000…0000 hello’s … or more and more…

- ✓ 1 ALPlugin to print “hello world”;
- ✓ + 1 ALPlugin to define and control a “cluster” of “hello world”
- ✓ + 1 ALApplication to define and manage the software.

```
class PluginHelloWorld( ALPlugin ):

    def process( self, _oalObject = None ):
        oiPrintNumber = 10

        for i in range( 0, oiPrintNumber ):
            ALVerbose.screenID( self.getID(), "       [" + ALString(i) + "] Hello World " )
```

A real example: “Hello World” in parallel

10,000…0000 hello’s … or more and more…

- ✓ 1 ALPlugin to print “hello world”;
- ✓ + 1 ALPlugin to define and control a “cluster” of “hello world”
- ✓ + 1 ALApplication to define and manage the software.

```
class PluginHelloWorldController( ALPlugin ):

    def process( self, _oalObject = None ):
        oiTotalNumberAction = 1024
        oiClusterSize = 64

        oalActionCluster = ALActionCluster()
        oalActionCluster.setClusterSize( oiClusterSize )

        for i in range( 0, oiTotalNumberAction ):
            oalPlugin = ALApplication.getPluginObject( "PluginHelloWorld" )
            if(oalPlugin!=None):
                oalActionCluster.addAction( oalPlugin )

        # Execution
        oalActionCluster.execute()

        oalActionCluster.synchronize()
```

A real example: “Hello World” in parallel

10,000…0000 hello’s … or more and more…

- ✓ 1 ALPlugin to print “hello world”;
- ✓ + 1 ALPlugin to define and control a “cluster” of “hello world”
- ✓ + 1 ALApplication to define and manage the software.
- ✓ Show Time – click here to play

```
from ALImportSystem      import *
from ALImportKernel      import *


if __name__ == '__main__':

    # Application Framework definition
    oalApplication = ALApplication( "HelloWorld-Parallel-Cluster", "2.0" )

    oalPlugin = oalApplication.getPluginObject( "PluginHelloWorldController" )
    if(oalPlugin!=None):
        oalApplication.connectExecute( oalPlugin.execute )

    # Execution = Main Thread
    oalApplication.execute()
```

A real example: “Hello World” in parallel

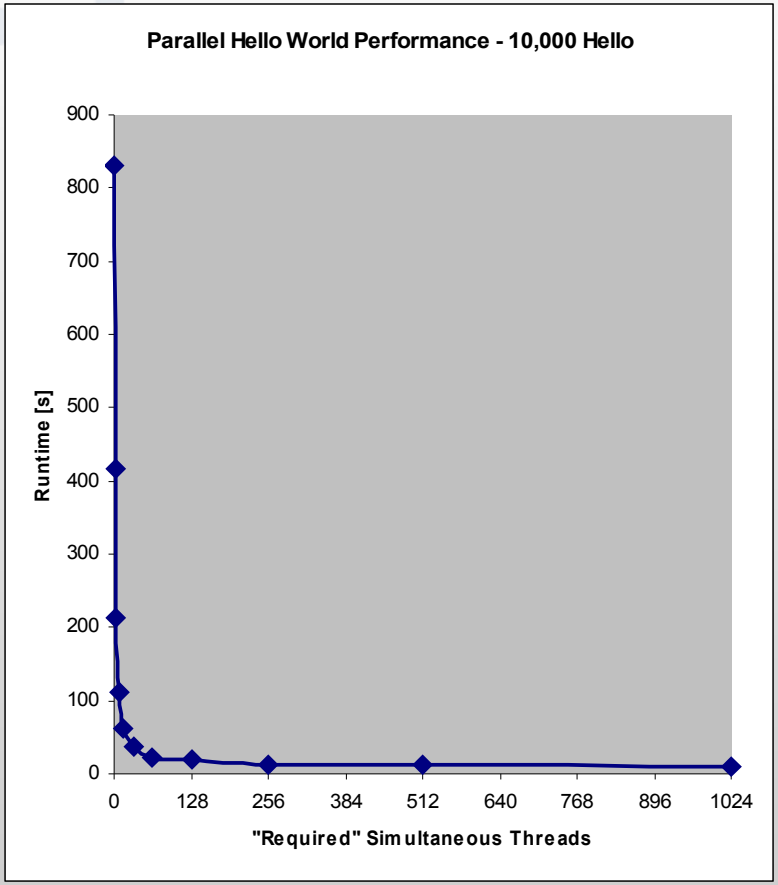
10,000…0000 hello’s … or more and more…

```
class PluginHelloWorldController( ALPlugin ):

    def process( self, _oalObject = None ):
        oiTotalNumberAction = 1024
        oiClusterSize = 64

        oalActionCluster = ALActionCluster()
        oalActionCluster.setClusterSize( oiClusterSize )

        for i in range( 0, oiTotalNumberAction ):
            oalPlugin = ALApplication.getPluginObject( "PluginHelloWorld" )
            if(oalPlugin!=None):
                oalActionCluster.addAction( oalPlugin )

        # Execution
        oalActionCluster.execute()

        oalActionCluster.synchronize()
```

1 Action (sequential) = 12 hello /s [820s]
~80 Action (parallel) = ~787 hello /s [13s]

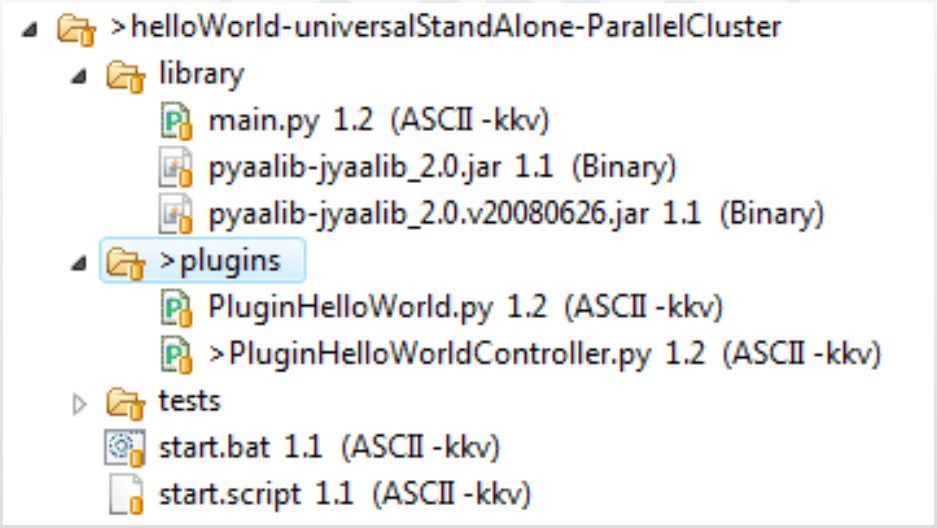
In parallel = ~ 65 Times Faster !!!

The Hardware is the limit !!!!



Standalone Generic Application Folder Structure

- ✓ It defines a Simplified folder structure = ALApplication skeleton and PyAALib-JyAALib "JAR" file.



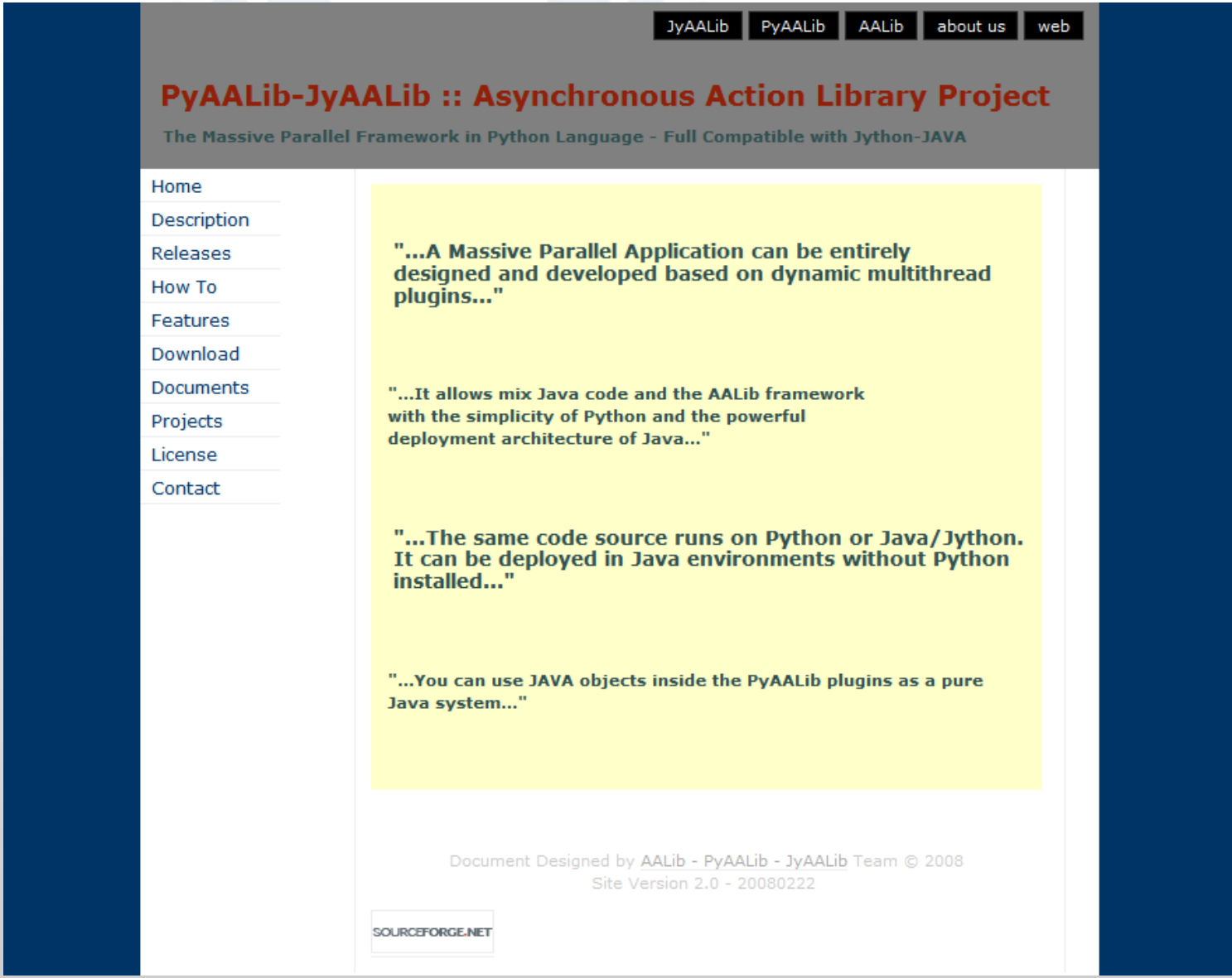
…Simplified Application Deployment and Maintenance…

No Python or Jython Installed on the target system !!!!

= only a JAVA machine !!!!!!

Documentation
+
Distribution

http://jyaalib.sourceforge.net

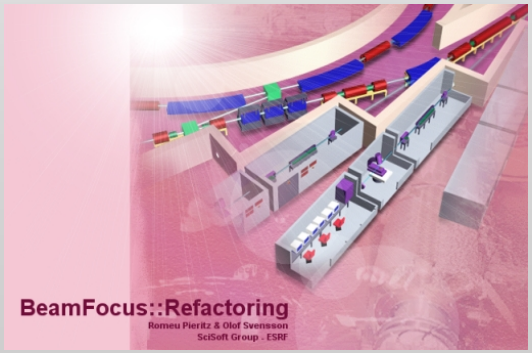


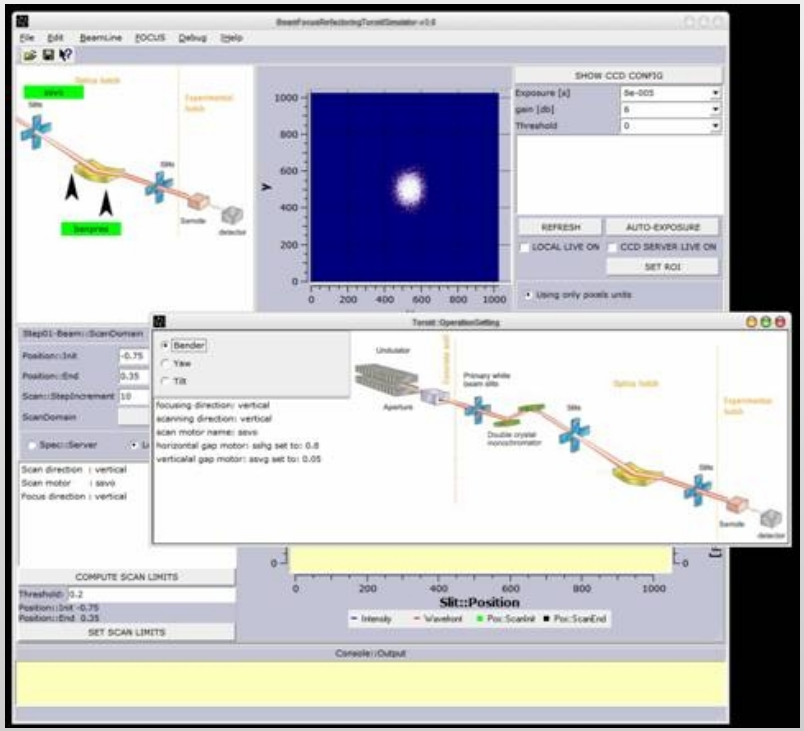
Projects using PyAALib-JyAAlib?

- ✓ i) DRank: Crystal Data Ranking Module in the DNA Package – BioXHIT EU Project
- ✓ ii) BeamFocus: BeamLine Assisted Focusing Application
- ✓ iii) EDFExplorer: ESRF Data Format converter to HDF5 File container
- ✓ iv) NEW European Collaboration EDNA – "Enhanced DNA".

dna

EDNA
(temporary logo)





Comparison:

PyAALib-JyAALib
x
Eclipse

	PyAALib - JyAALib	Eclipse
Framework Size	Compact	Extreme large
Kernel Design Style	“Multi Task”	“Event Drive”
Plug-in Structure	YES	YES
Plug-in Factory Import Scheme	YES	YES
Native Parallel Plug-in	YES	NO
Number of Files to define a Plug-in	1	4
Any code can be a Plug-in	YES	NO
Work on Python	YES	NO
Work on Java	YES	YES
Mix Python + Java in same code source	YES	NO
Server Code Development	YES	NO
Client Development	YES	YES
Rich Client Development	YES	YES
GUI Development	NO	YES
Standalone Application Deployment	YES	YES
Compatible with other IDE (Interface Development Environment)	YES	NO
License	Totally Free (Open Source FreeBSD License Scheme)	Proprietary (EPL – Eclipse Public License) – “Consortium Style”
Open Source	YES	YES

Conclusion

✓We have a smart open source framework based on dynamic plug-ins for fast development of massively parallel code;

✓ We have a large experience in the design, development and maintenance of complex concurrent systems;

✓ We have international collaborations investing in that kind of parallel technology;

The Future:

✓ A new MX Group project based on PyAALib-JyAALib driving a "massively parallel processing workflow" will start in July 2008;

More information:

✓Call me for a coffee…

Office Phone: 2637 – central building – room 113-117

Acknowledgements

- Sine Larsen, Sean McSweeney and Claudio Ferrero
- BioXHIT Project Funds
- DNA Collaboration
- EDNA Collaboration
- ESRF MX Group and ID14
- ESRF BLISS Group
- ESRF Experimental Division
- ID15 – ID17 – ID19 Beamlines

Contact and References

- ESRF SciSoft Group – http://www.esrf.eu/UsersAndScience/Experiments/TBS/SciSoft
- BioXHIT European Project – http://www.bioxhit.org
- DNA Collaboration – http://www.dna.ac.uk
- EDNA Collaboration – http://www.edna-site.org
- AALib Framework – http://aalib.sourceforge.net
- AALib Python Version – http://pyaalib.sourceforge.net
- AALib Python-Jython-JAVA Version – http://jyaalib.sourceforge.net
- BeamFocus:Refactoring Application – http://beamfocus.sourceforge.net

Example:
Dynamic Plugin - multi threading - Intense disk IO processing

- Processing a Huge amount of Data for Tomography
  - Example of Scalability of the code using the Class "ALActionCluster" to transpose 128x128x128 volume;
    - One line of code to define and control the cluster size;
    - Using 259 threads (buffer size is 64 slices = 256 threads + 3 control threads);
    - The OS and the hardware are the limiting factor;

- Machine: Intel Single Core - Hyper Thread Architecture

Nom du modèle	Fréquence	Front Side Bus	Date de sortie
Pentium M 770	2133 MHz	533 MT/s	19 Janvier 2005

Minimum Memory Required
Only on Disk – intense IO operations

In parallel = ~ 7 Times Faster !!!

The Hardware is the limit !!!!