Comparison of Cloud and Local HPC approach for MUSCLE-based Multiscale Simulations

Katarzyna Rycerz(1,2), Marcin Nowak(1), Paweł Pierzchała(1), Eryk Ciepiela(2), Daniel Harężlak(2) and Marian Bubak (1,2,3)

(1) AGH University of Science and Technology, Department of Computer Science, Krakow, Poland
(2) ACC CYFRONET AGH, Krakow, Poland
(3) Informatics Institute, University of Amsterdam, The Netherlands

DMC Workshop

5 Dec 2011

http://dice.cyfronet.pl

The Mapper project receives funding from the EC's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° RI-261507.

Plan

- Multiscale Application Requirements
- Motivation and Goals
- Background tools
  - Multiscale Couping Library and Environment (MUSCLE)
  - GridSpace Virtual labolatory
- Environment supporting execution of MUSCLE based application on HPC and Cloud resources using GridSpace
- Performance results and comparison

Multiscale Applications Requirements

- Focus on multiscale applications that can be described as a set of independent modules of two types:
  - simulating certain phenomena in certain time or space scale (scaleful)
    - usually computationally intensive and require HPC resources,
  - converting data from one scaleful module to another
    - usually do not have demanding computational requirements
    - to avoid additional communication, they often required to be executed ”close” to the scaleful modules they are connecting
- focus on peer to peer type of computation
  - application modules are executed concurrently
  - exchange data in usually asynchronous fashion
- Examples from VPH, hydrology, fusion fields of science

Goals

- Fulfill multiscale application requirements by integrating solutions from
  - multiscale computing environments (MUSCLE)
  - virtual experiment frameworks (GridSpace)
  - Infrastructures (local HPC and Cloud)
- Compare Local HPC and Cloud approaches:
  - performance of setting up and running multiscale application
  - ease of usage

Background

Ongoing research in supporting composition of multiscale simulations from single scale models on various levels

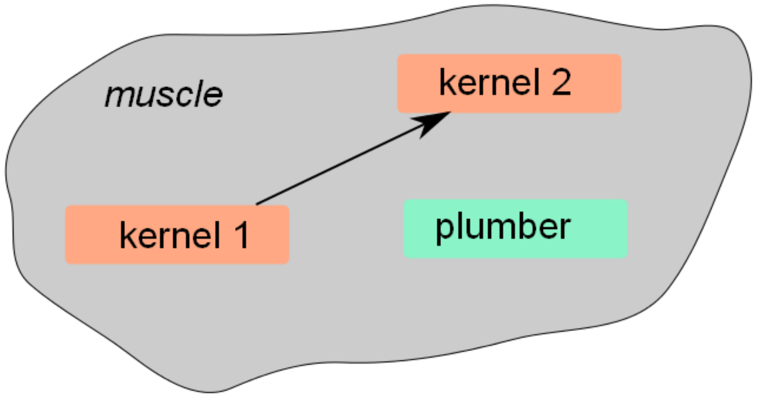
- Description languages
  - Multiscale Modelling Language (MML)
- Dedicated multiscale environments
  - Multiscale Coupling Library and Environment (MUSCLE)
  - Model Coupling Toolkit (MCT)
  - Astrophysical Multi-Scale Environment (AMUSE)
- Efforts of exploiting European Grid e-Infrastructures and clouds:
  - Euforia http://www.euforia-project.eu/
  - MAPPER http://www.mapper-project.eu/
  - UrbanFlood http://urbanflood.eu/
  - VPH-Share http://vph-share.org/

MUSCLE

- Connects tightly coupled simulation modules (called kernels)
- kernels are executed concurrently
- actor-based concurrency model: asynchronous sending, synchronous receiving
- kernels communicate directly using unidirectional pipes (conduits) connecting the entrances with exits.
  - connections defined by external configuration mechanism in a ruby script called CxA file
  - controlled by a so-called plumber
- each conduit can have a custom filter for additional transformation/conversion of data

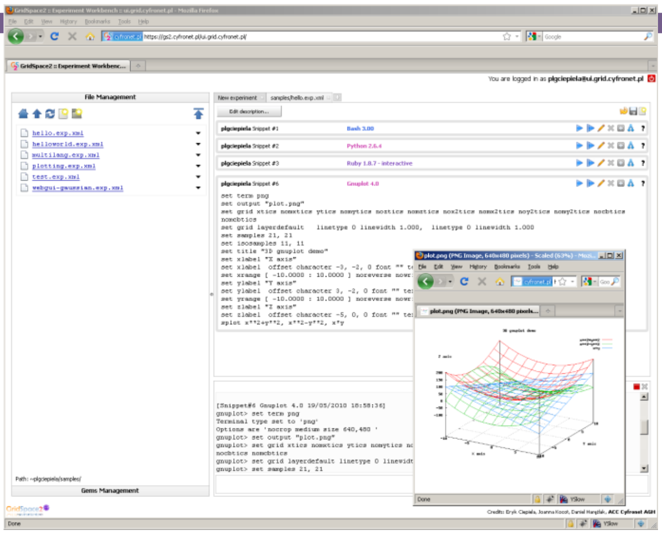
CxA file

```
cs.attach(kernel1 => kernel2) {
  tie(entrance, exit)
}
```



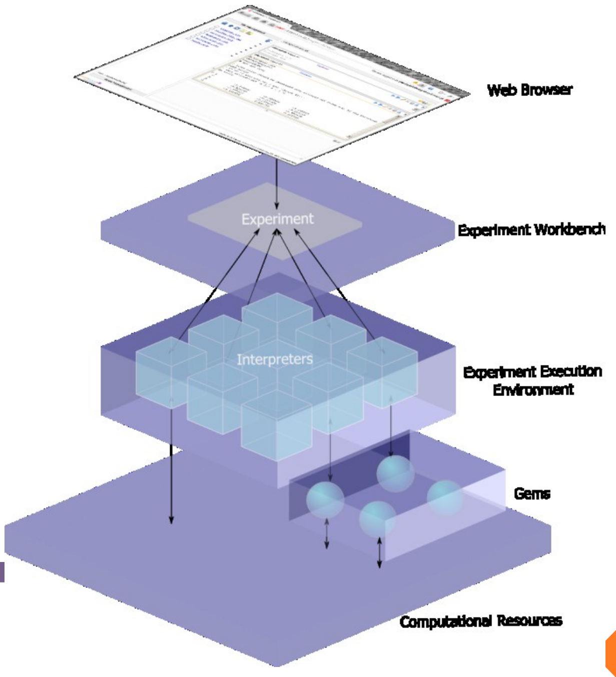
GridSpace

- Easy access using Web browser
- Experiment workbench
  - Constructing experiment plans from code snippets (Ruby, Python, MatLab, Gnuplot etc.)
  - Interactively run experiments
- Experiment Execution Environment
  - Multiple interpreters
  - Access to libraries, programs and services
- Access to cluster, grid and cloud
- Experience
  - **Virolab** project
  - **PL-Grid** NGI
  - **MAPPER** project


You are logged in as plgciepiela@ui.grid.cyfronet.pl
File Management
hello.exp.xml
helloworld.exp.xml
multilang.exp.xml
plotting.exp.xml
test.exp.xml
webgui-gaussian.exp.xml
Gems Management
New experiment
samples/hello.exp.xml
Edit description...
plgciepiela Snippet #1 Bash 3.00
plgciepiela Snippet #2 Python 2.6.4
plgciepiela Snippet #3 Ruby 1.8.7 - interactive
plgciepiela Snippet #6 Gnuplot 4.0

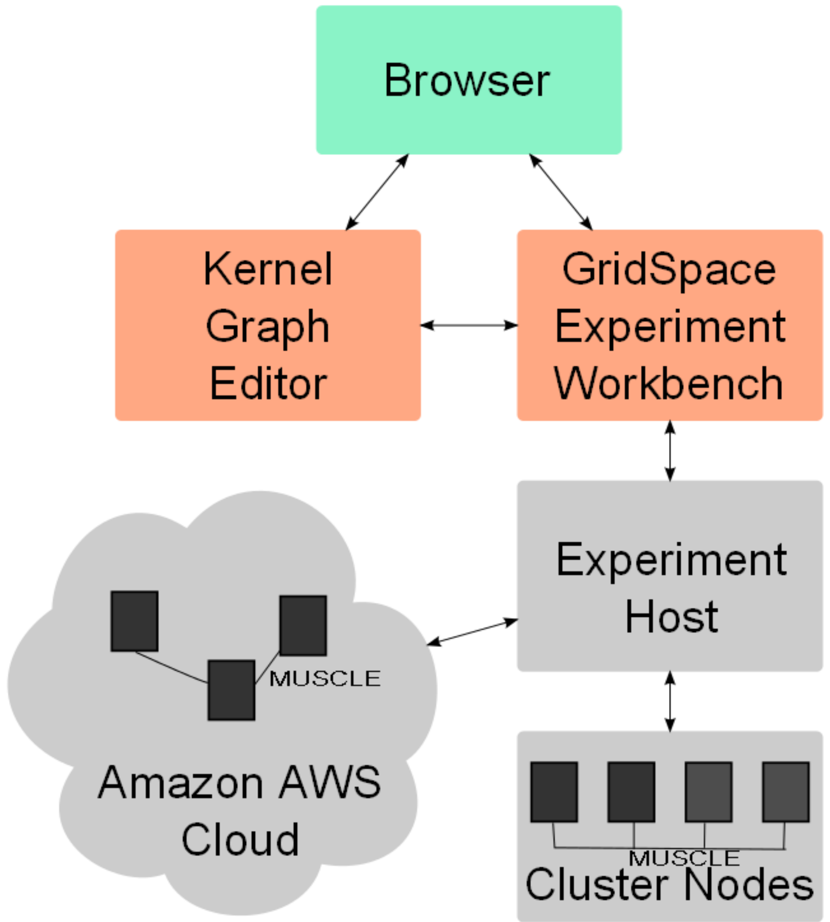
```
set term png
set output "plot.png"
set grid xtics nomxtics ytics nomytics noztics nomztics nox2tics nomx2tics noy2tics nomy2tics nocbtics
nomcbtics
set grid layerdefault   linetype 0 linewidth 1.000,  linetype 0 linewidth 1.000
set samples 21, 21
set isosamples 11, 11
set title "3D gnuplot demo"
set xlabel "X axis"
set xlabel  offset character -3, -2, 0 font "" te
set xrange [ -10.0000 : 10.0000 ] noreverse nowr
set ylabel "Y axis"
set ylabel  offset character 3, -2, 0 font "" te
set yrange [ -10.0000 : 10.0000 ] noreverse nowr
set zlabel "Z axis"
set zlabel  offset character -5, 0, 0 font "" te
splot x**2+y**2, x**2-y**2, x*y
```

```
[Snippet#6 Gnuplot 4.0 19/05/2010 18:58:36]
gnuplot> set term png
Terminal type set to 'png'
Options are 'nocrop medium size 640,480 '
gnuplot> set output "plot.png"
gnuplot> set grid xtics nomxtics ytics nomytics nc
nocbtics nomcbtics
gnuplot> set grid layerdefault linetype 0 linewidt
gnuplot> set samples 21, 21
```



Proposed Environment Architecture

- User logs to chosen access machine (Experiment Host) using GridSpace Experiment Workbench
- Actual connection is done using SSH
- User creates or loads CxA connection scheme as experiment snippet
- CxA scheme is parsed and sent to Kernel Graph Editor that displays connections.
- Kernel Graph Editor aids the user in joining kernels in groups that should be executed at the same host.
- Depending on user preference the MUSCLE application is performed on HPC Cluster or AWS Amazon Cloud.

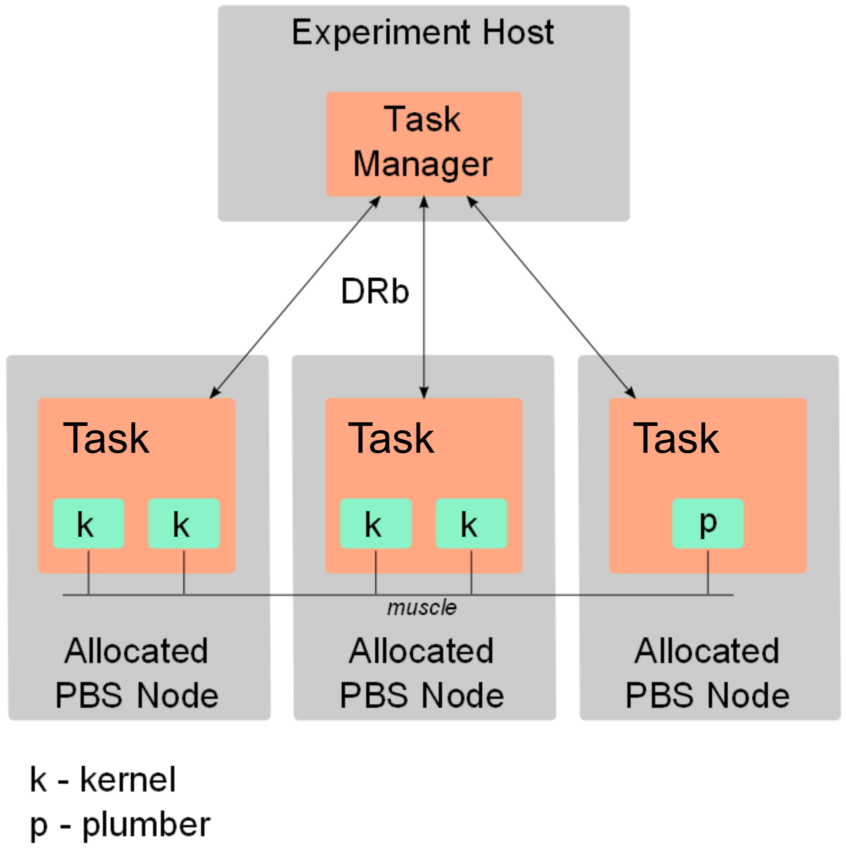


Automatic MUSCLE application distribution

- plain legacy MUSCLE software requires manual start of kernels and plumber on each computing node
- to support automatic control we applied a general Master - Slave architecture
  - Master distributes computational tasks, supports synchronization and standard output/error gathering.
  - Slaves start actual kernels and plumber and redirect its standard output and error streams.
- once started by Slaves, actual kernels communicate with each other using MUSCLE.
- solution used for both types of infrastructures (local HPC and Cloud)

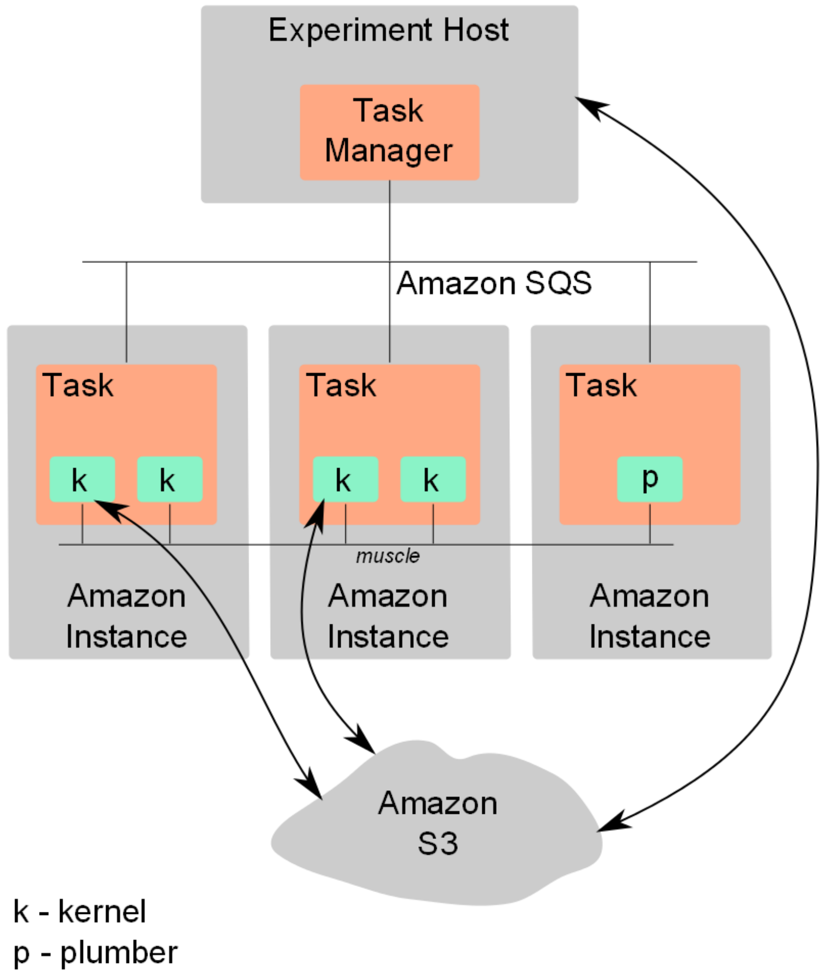
Setting up MUSCLE application local HPC

- Portable Batch System (PBS) local management system for allocating resources (pbsdsh tool)
- Distributed Ruby (DRb) for communication between master and slaves.
- http://www.youtube.com/watch?v=3S9-kljyXIw



Setting up muscle application Amazon AWS cloud

- standard cloud mechanisms for launching virtual instances
  - Amazon EC2 Ruby API
  - one instance for one group of kernels
  - preconfigured Amazon Machine Image (AMI) with preconfigured MUSCLE instalation
    - Light kernel implementations (jars) can be fetched from S3
    - Heavy kernel implementations should be installed on AMI beforehand
- Amazon SQS for communication between master and slaves
- Amazon S3 for storing results

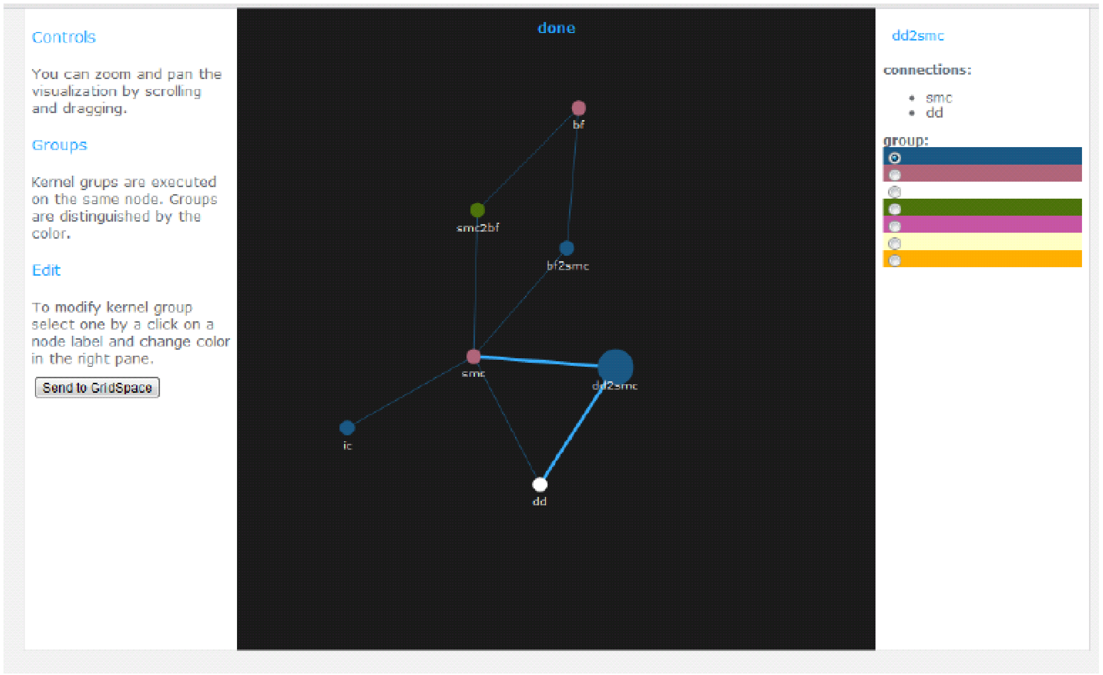


Local HPC and Cloud approach - comparision

Local HPC	**Cloud**
requires hardware investment	pay for what you use
often requires contact with administrator for additional installation of packages	a user has administrative access to a virtual instance
variable PBS waiting time depending on number of other users' jobs	constant and predictable virtual instances booting time
using DRb requires setting up point to point socket connections, hosts and ports have to be explicitly known, firewall issues	using SQS more convenient: high level API to shared message queue, communicating entities are not visible to each other, well known port
Shared file system – no extra staging data needed	Staging input/output data is needed

Kernel Graph Editor

- used to group kernels that should be executed on the same node
- renders connection scheme in a user web browser and sends the final grouping back to GridSpace
- communication with GridSpace is done by WebGUI tool (GS gem)
- information about kernel groups in JavaScript Object Notation (JSON) format sent with HTTP/POST.
- Kernel Graph Editor server is implemented in Ruby (Sinatra framework)
- Client is written in JavaScript (library InfoVis)


Controls

You can zoom and pan the visualization by scrolling and dragging.

Groups

Kernel grups are executed on the same node. Groups are distinguished by the color.

Edit

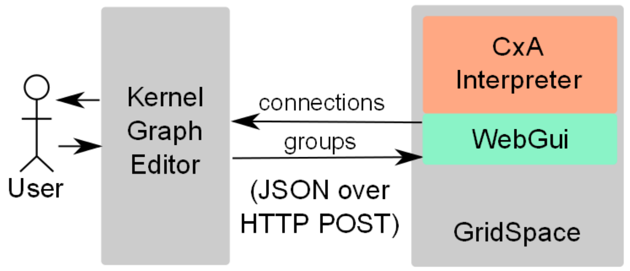
To modify kernel group select one by a click on a node label and change color in the right pane.

dd2smc

connections:

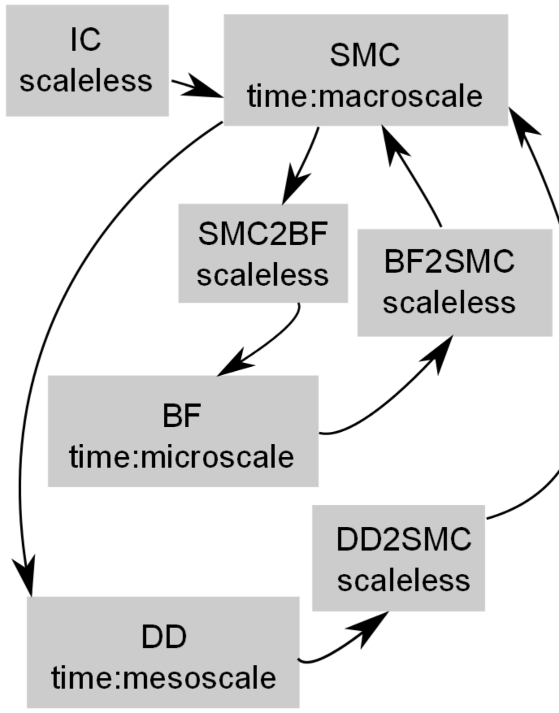
- smc
- dd

group:



Use Case – in-stent restenosis application

- simulates treating of recurrent stenosis of artery after surgical correction.
- 2D version
- implemented with MUSCLE
- kernels containing scale
  - blood flow (BF)
  - simulation of muscle cells (SMC)
  - drug diffusion (DD)
- scaleless kernels (mappers)
- kernel calculating Initial Conditions (IC)



Performance Results

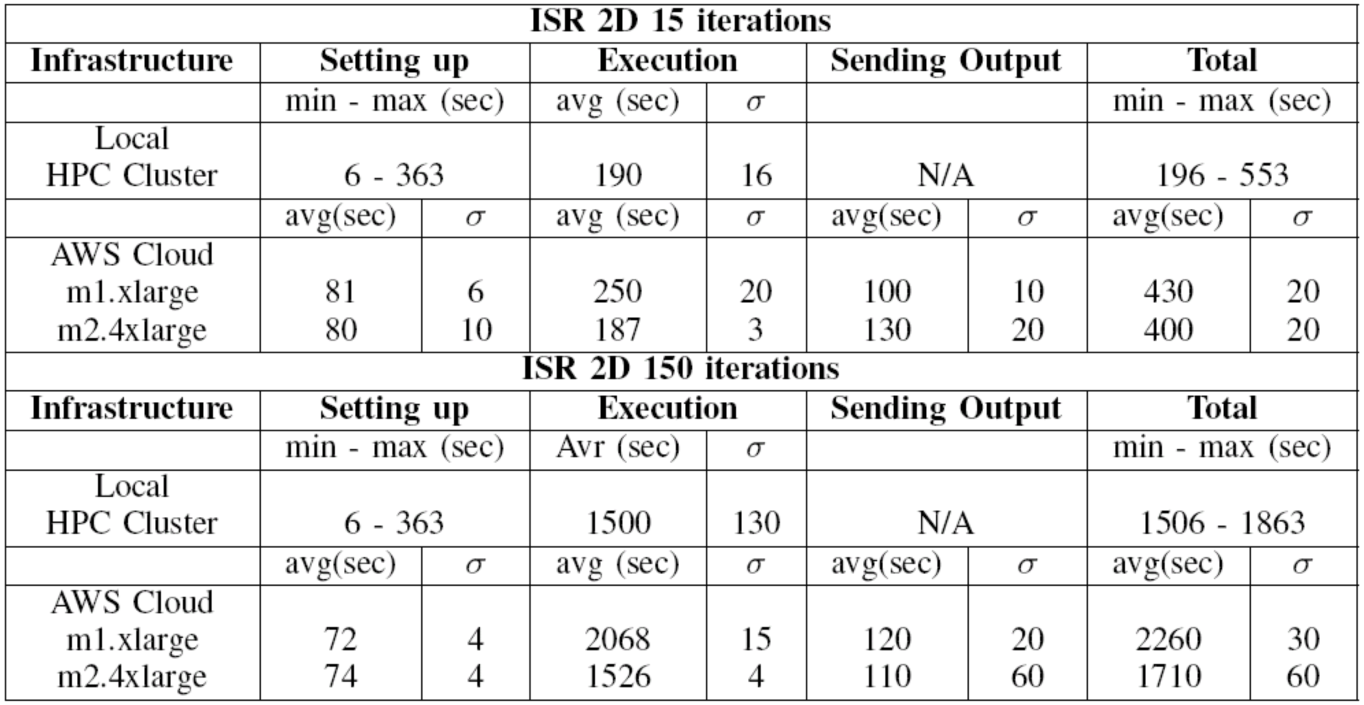
- As full application is running 3 days – we present tests for a partial execution (for 15 and 150 number of iterations)
- Local HPC : the HP Cluster Platform 3000 BL 2x220 (CPUs Intel Xeon 2260 MHz connected with Infiniband) hosted at ACC Cyfronet, Krakow (no 88 on the Nov 2011 Top 500 list)
- Two types of instances in Amazon cloud (1 EC2 Compute Unit = CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor)
  - High-CPU Extra Large (m1.xlarge) Instances with 7 GB of memory, 20 EC2 Compute Units (8 virtual cores with 2.5 EC2 Compute Units each), 1690 GB of local instance storage, 64-bit platform
  - Cluster Compute Quadruple Extra Large (m2.4xlarge) Instances 23 GB memory, 33.5 EC2 Compute Units, 1690 GB of local instance storage, 64-bit platform, 10 Gigabit Ethernet

Performance Results

- Local HPC
  - setting up: waiting in a PBS queue and dispatching tasks using DRb
- Cloud
  - setting up: creation of SQS queues, sending input sandbox to S3, booting instances, dispatching tasks by SQS queue and fetching input sandbox by tasks.
  - sending output time (to S3) measured additionally
  - amount of output :1MB (for 15 iterations) and 3MB (for 150 iterations)
- Both:
  - the execution: actual application execution (including MUSCLE environment start-up)
  - Kernel communication with MUSCLE – 10 MB per iteration

Performance Results

- Cloud resources is more predictable comparing to batch queue system
- Execution time is comparable on both infrastructures, especially when using Quadruple Extra Large instances dedicated for HPC applications
- Cloud solutions require additional time for data staging



ISR 2D 15 iterations								
Infrastructure	**Setting up**		**Execution**		**Sending Output**		**Total**	
	min - max (sec)		avg (sec)	σ			min - max (sec)	
Local HPC Cluster	6 - 363		190	16	N/A		196 - 553	
	avg(sec)	σ	avg (sec)	σ	avg(sec)	σ	avg(sec)	σ
AWS Cloud								
m1.xlarge	81	6	250	20	100	10	430	20
m2.4xlarge	80	10	187	3	130	20	400	20
ISR 2D 150 iterations								
Infrastructure	**Setting up**		**Execution**		**Sending Output**		**Total**	
	min - max (sec)		Avr (sec)	σ			min - max (sec)	
Local HPC Cluster	6 - 363		1500	130	N/A		1506 - 1863	
	avg(sec)	σ	avg (sec)	σ	avg(sec)	σ	avg(sec)	σ
AWS Cloud								
m1.xlarge	72	4	2068	15	120	20	2260	30
m2.4xlarge	74	4	1526	4	110	60	1710	60

Summary

- Comparison of the two approaches for using computing infrastructure for MUSCLE-based multiscale applications:
  - local HPC cluster
  - Amazon AWS Cloud
- Both types of infrastructures were integrated with GridSpace
- Kernel Graph Editor for kernels grouping
- The preliminary results have shown that setting up multiscale application in a Cloud environment is comparable to its submission on a classical PBS-based HPC cluster

Thank you !

- Special thanks to Alfons Hoekstra, Joris Borgdorff and Eric Lorenz from UvA for discussions on ISR2D, CxA and MUSCLE
- Thanks to Maciej Malawski and Jan Meizner (Cyfronet) for discussions about cloud computing.
- partially supported by the MAPPER project – grant agreement no 261507, 7FP UE
- Access to the Amazon EC2 cloud was supported by an AWS in Education grant

- See more on **http://dice.cyfronet.pl**