



Future of Ibis

Jason Maassen

Computer Systems Group
Department of Computer Science
VU University, Amsterdam, The Netherlands



Short Recap...

- JavaGAT can be used as a **master key** to access various resources.
- IPL+SmartSockets can be used as **glue** to combine resources into a single pool and run coupled codes.
- IbisDeploy can be used for easy deployment of such applications.



What is Missing ?

- Support for **heterogeneity** between resources:
 - Available cores, memory, storage, **GPUs**, etc.
 - Installed application, libraries, etc.
 - Data locality
- Support for **equi-kernels**:
 - Multiple **functionally equivalent** implementations of some algorithm with different resource requirements
 - Example: AMUSE



Next steps...

- Currently, we either assume all resources to be equal, or we assume the user/application knows which resource can run which code.
 - “Run code X on resource Y because it has a GPU”
- Can we come up with an simple and flexible way to express both **heterogeneity** (in all its forms) and **equi-kernels** ?



Constellation

- New system on the Ibis stack (research!)
 - Similar to the Pilot Job framework we saw earlier
 - ... but much more advanced!
- Offers task scheduling with application specific match-making
 - Ensures that each job is send to a resource that can actually execute it.



Constellation Model

- Application: set of **activities** (= Jobs)
 - Loosely coupled (communicate using **events**)
 - Size and complexity may vary
 - Sub-second sequential jobs to large parallel simulations that take hours
- Resources: set of **executors** (= Pilot Jobs)
 - Capable of running activities
 - May represent anything from a single core to entire cluster, a GPU, etc.



Constellation Model

Differences with pilot job model

- Individual executors may only be able to process a subset of the activities (**heterogeneity**)
- Executors may offer equivalent functionality but differ in implementation (**equi-kernels**)
- Running activities may submit other activities (**any resource is source of work**)



Constellation Model

- Each application consists of
 - A collection of many **distinct**, yet somehow related activities – a (possibly dynamic) DAGs of tasks.
 - A (possibly heterogeneous) set of executors, which **together** are capable of executing these tasks.
- It is now up to Constellation to ensure the correct mapping between activities and executors....

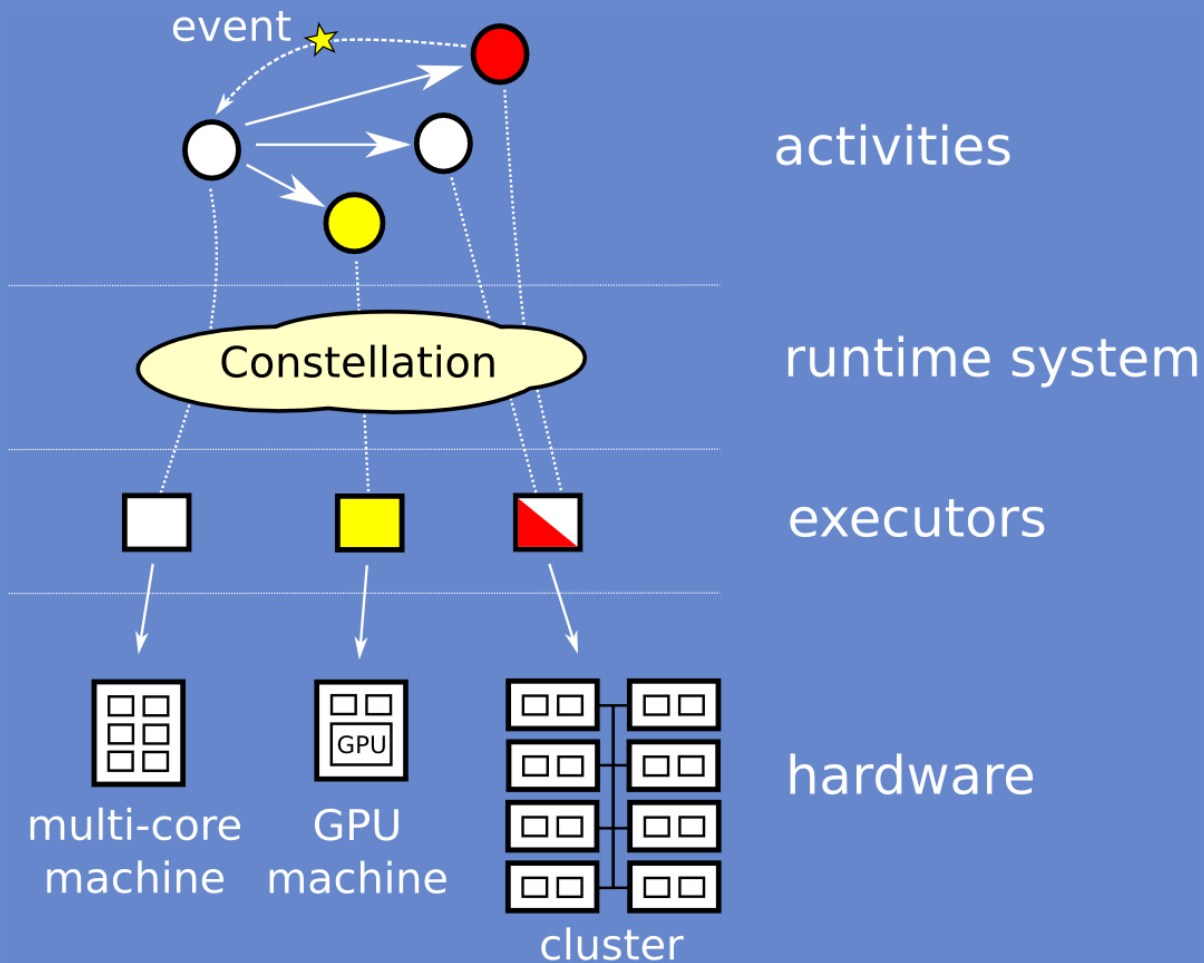


Constellation Model

- Both activities and executors can be tagged using a context:
 - Simple **application defined label** (= a string)
 - Defines which activities and executors are “**a match**”
 - **Application defined = extremely flexible**
- Constellation RTS performs
 - **match-making** (using these contexts)
 - **load-balancing** (using workstealing)

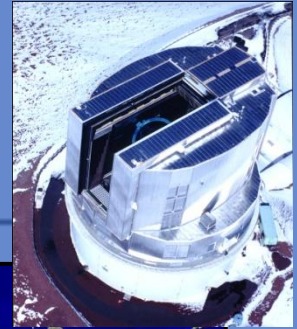


Constellation Example

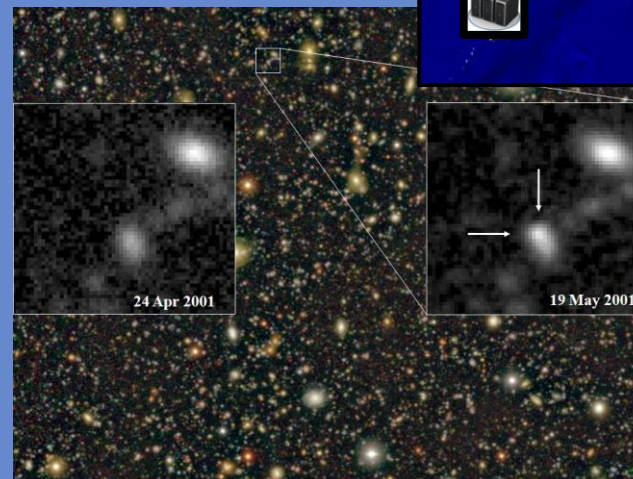
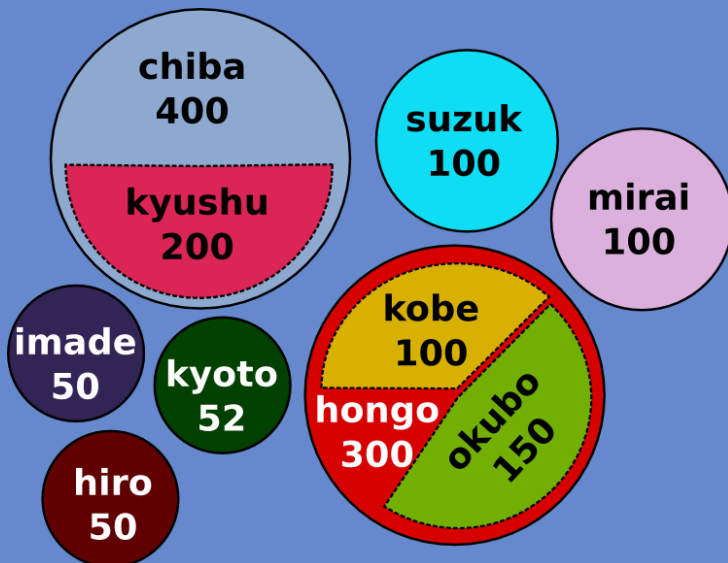
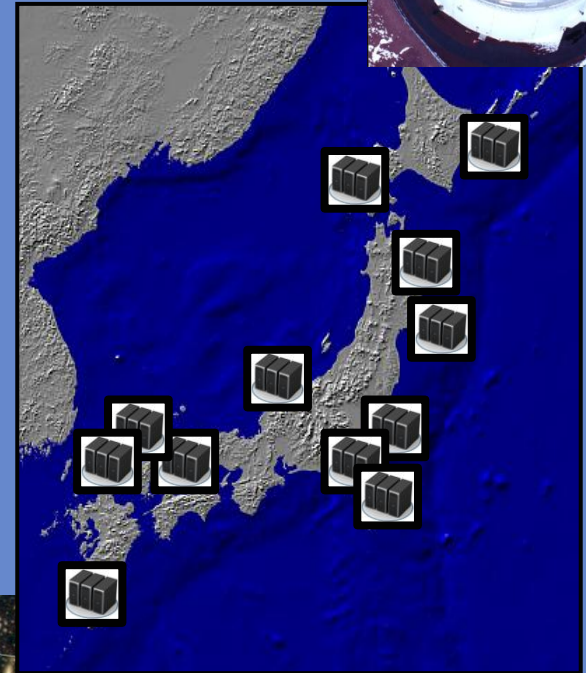


Supernova Detection

2008 Data Challenge Winner

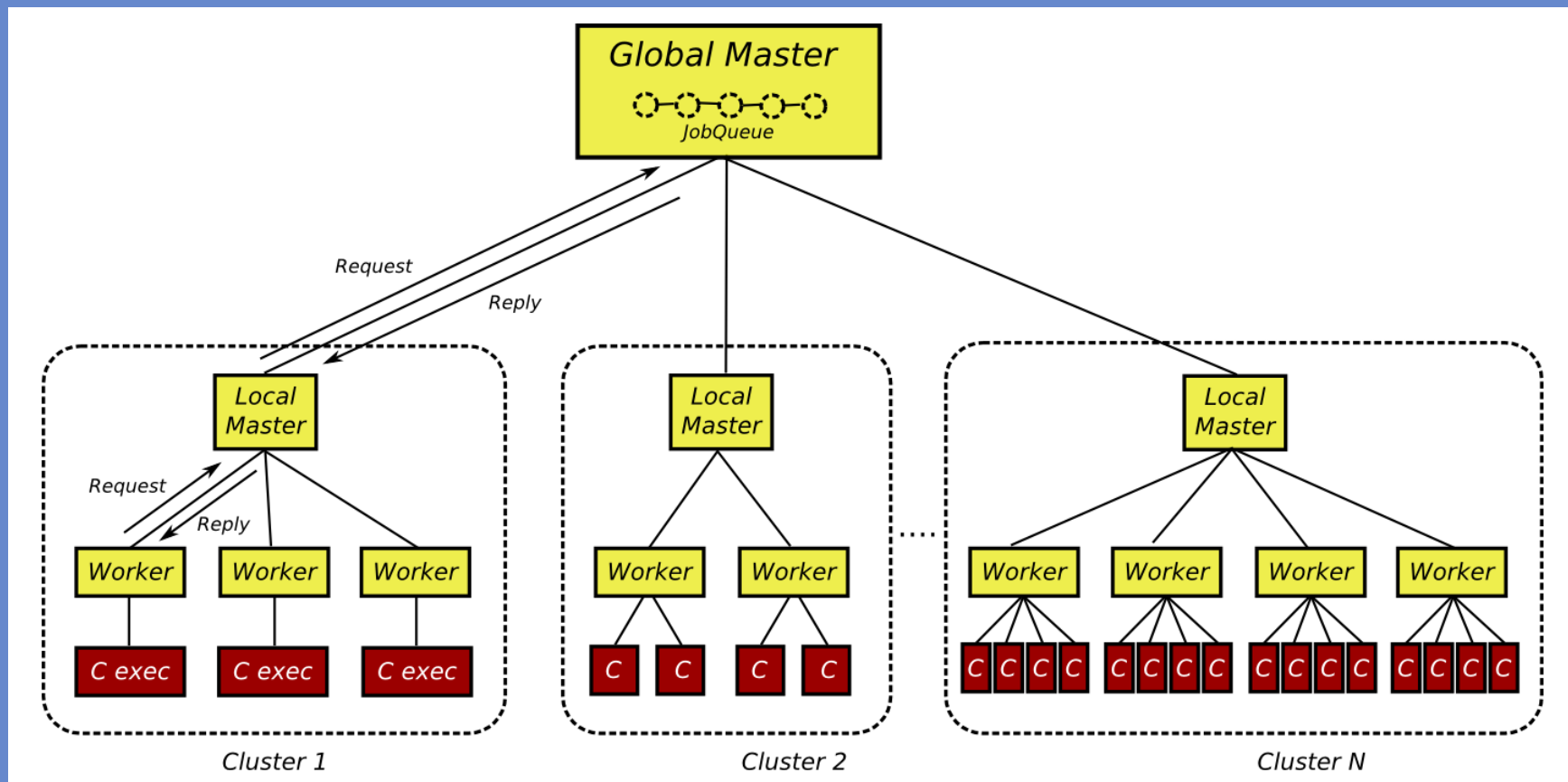


- Analyse 1052 image pairs
 - 10~12 clusters (1162 cores max)
 - Partitioned / replicated data
 - Varying resolution



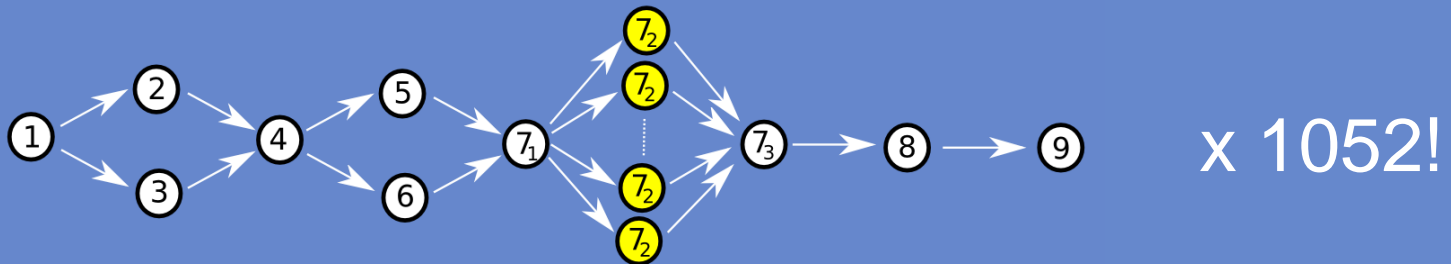
2008 Solution

Hierarchical Pilot Job Framework



Early Experiments

- Application is a DAG / Workflow



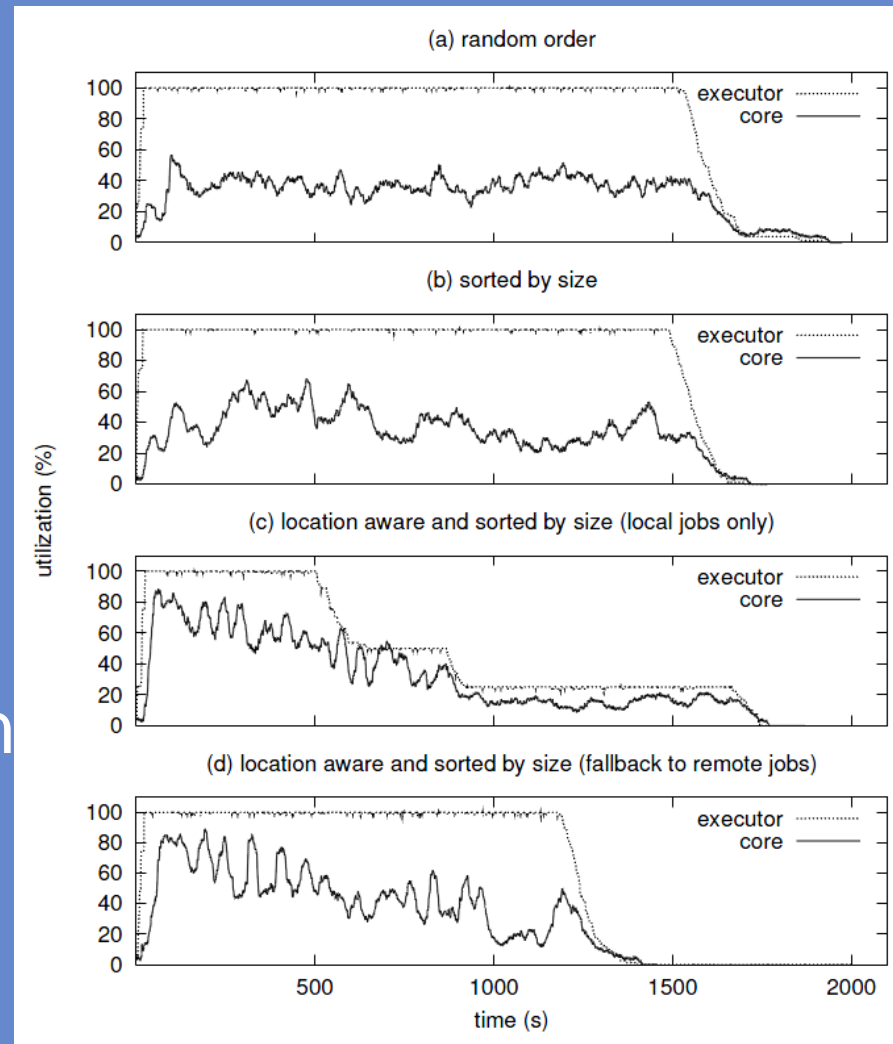
- Test Constellation in 3 different scenarios:
 - Data locality
 - Executor granularity
 - Heterogeneity



Scenario 1

Data Locality

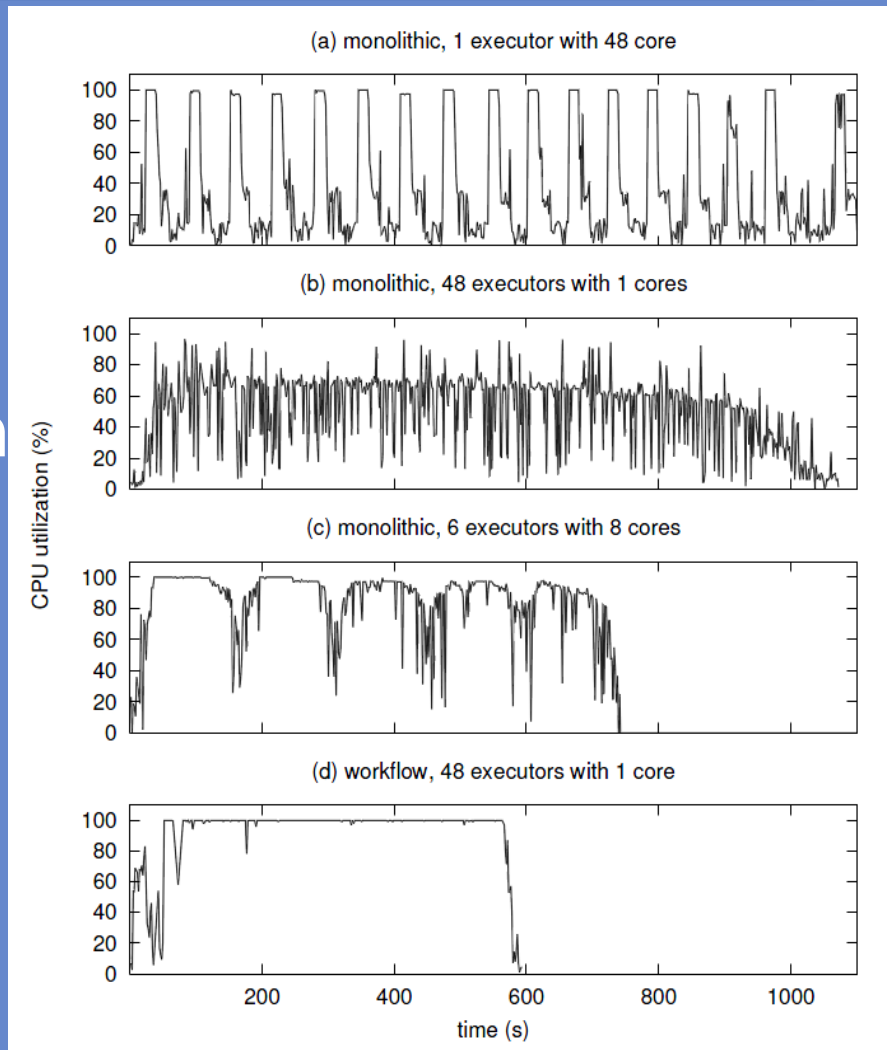
- Data distributed over 4 clusters (DAS3/4)
 - Activity: entire application
 - Executor: complete node
- Use context to express data locality
 - Locality aware task farming
- No change in application
 - Use constellation wrapper
 - Adapt context to tune application



Scenario 2

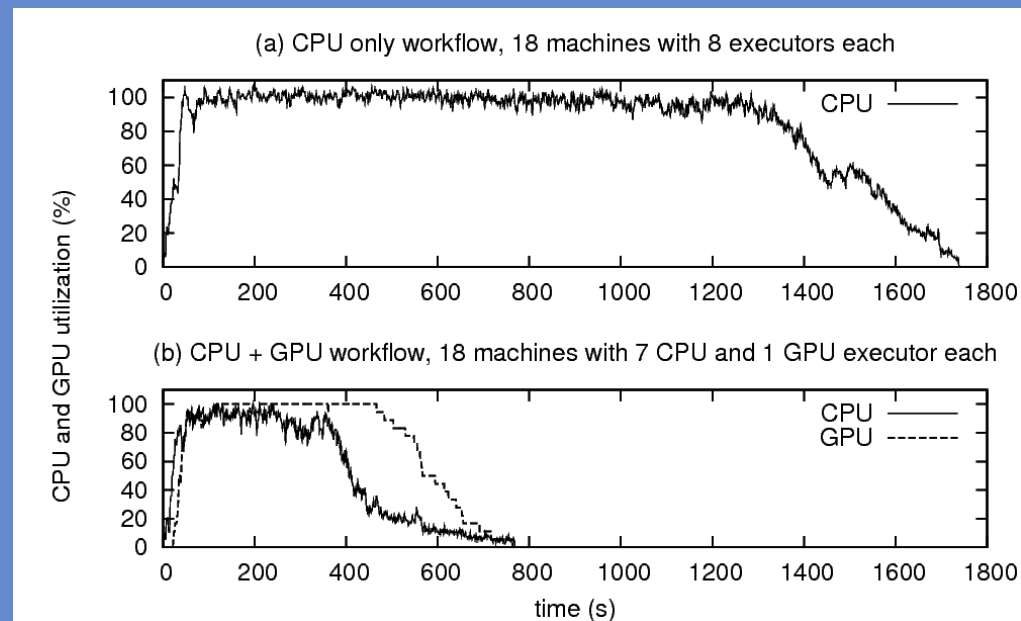
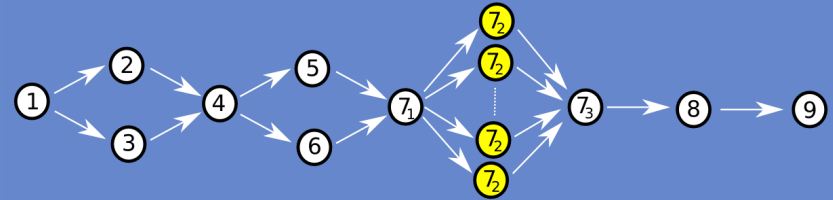
Executor Granularity

- Single 48 core machine
 - Activity: entire application (a-c)
single task (d)
 - Executor: [n]-cores
- No change in application for experiment (a-c)
 - Only change executor config.
- Completely ported application in (d)
 - Significant performance gain!



Scenario 3: Heterogeneous System

- 18 node GPU cluster
 - 8 cores + 1 GPU per node
 - Activity: single task
 - Executor: 1 core (top)
1 core or 1 GPU (bot.)
- Replaced activity 7.2 with GPU version.
 - Label activities and executors accordingly
 - Constellation takes care of rest!
 - Significant performance gain.



Conclusions

- Initial experiments show that Constellation works well for a wide range of hardware configurations
- Constellation offers a highly configurable model
 - Easy to extend and reconfigure applications
 - Allows integration of specialized accelerator codes
- Suitable basis for a Jungle Computing model ?



Future Work

Easy ones

- More applications on wider range of hardware
- Implement on top of Constellation:
 - Domain specific languages
 - Phyxis-DT (successor of Joris)
 - User-friendly workflow models
 - Existing programming models (Satin)
 - Applications (SAT)



Future Work

Interesting ones!

- Integration of executor deployment into model
 - Currently it is up to the user to deploy the right type and right amount of executors
- Can we automate this ?
 - User has to provide the **correct executors** and a set of **available resources**
 - Constellation then **dynamically** selects the **right resources** and **the right number of resources** and performs the deployment



Zorilla 2.0

Wild ideas...

- Merge the existing P2P middleware with the SmartSockets overlay and the resource selection mechanism needed by Constellation ?
- A single solution for many open questions ?

