

# Efficient Parallel Shell REDBASH

Georges-André Silber  
Centre de Recherche en Informatique  
Ecole des mines de Paris  
France

COSET-2, June 19, 2005, Cambridge, USA



Action Concertée Incitative  
[ACI]  
Globalisation des Ressources  
Informatiques et des Données  
[GRID]



MetaCC

# Outline

- Motivations
- RED (Remote Execution Daemon)
- REDBASH (Parallel BASH)
- Performance results
- Conclusion and Future Work

# Motivations

MetaCC and Video Encoding

# Motivation for RED

MetaCC: Remote and Transparent Compilation, Optimisation, and Execution of Applications on the GRID using Services

storage		
<b>C</b>	Identity	
<b>C</b>	Source code	<b>S</b>
<b>C</b>	Data	<b>S</b>
<b>C</b>	Analysis	<b>S</b>
<b>C</b>	Executable	<b>S</b>

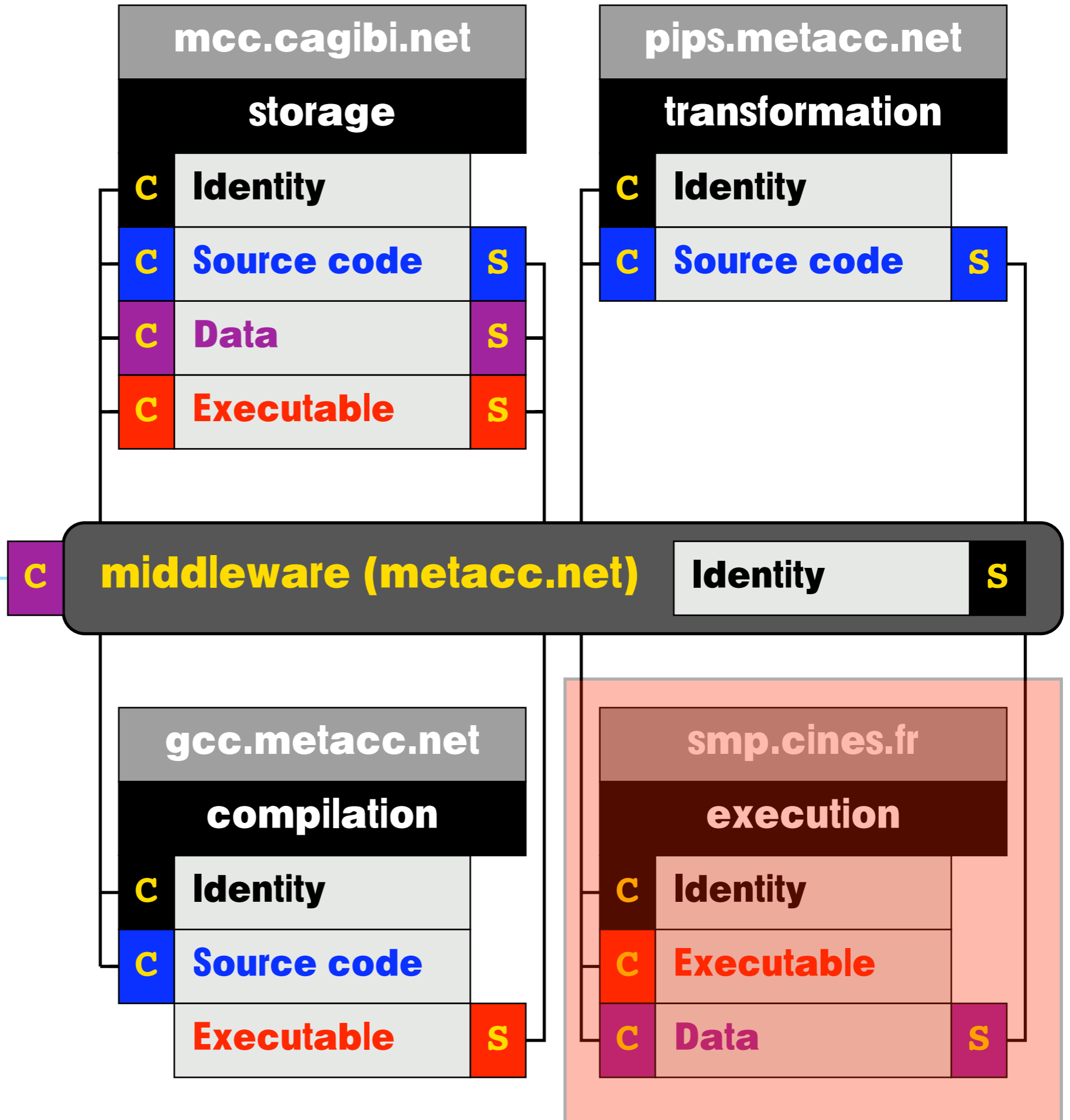
identification		
<b>C</b>	Data	
	Identity	<b>S</b>

transformation		
<b>C</b>	Identity	
<b>C</b>	Code source	<b>S</b>
<b>C</b>	Analysis	<b>S</b>

compilation		
<b>C</b>	Identity	
<b>C</b>	Code source	
<b>C</b>	Analysis	<b>S</b>
	Executable	<b>S</b>

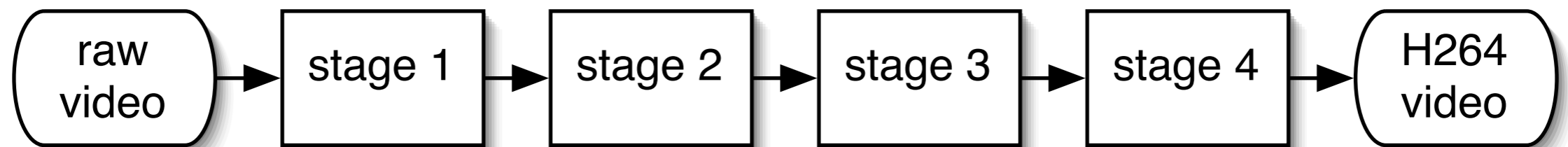
execution		
<b>C</b>	Identity	
<b>C</b>	Executable	
<b>C</b>	Data	<b>S</b>
	Analysis	<b>S</b>

RED



# Motivation for REDBASH

H.264 (MPEG4 AVC) video encoding



```
cat /dev/camera | s1 | s2 | s3 | s4 > video.h264
```

# Parallel execution on clusters

- Custom applications
  - sockets
  - PVM/MPI
- Single System Image Operating Systems
- Our approach is between those two worlds
- We work at the user level, without code modification

# RED

Remote Execution Daemon

# RED

## Remote Execution Daemon

- A simple daemon on each node of the cluster
- Implements a simple service with
  - sandbox (world) creation with an empty file system
  - file storage in this sandbox
  - remote program execution in this sandbox
- RED is a very simplified operating system
- Highly portable (Linux/FreeBSD/Darwin/Windows)

# RED model

RED is a RPC service

wkey = **getWorld** ()

(socket, jobid) = **createSource** (wkey, path)

(socket, jobid) = **createSink** (wkey, path)

jobid = **run** (wkey, cmd, insocket, outsocket, ...)

**destroyWorld** (wkey)

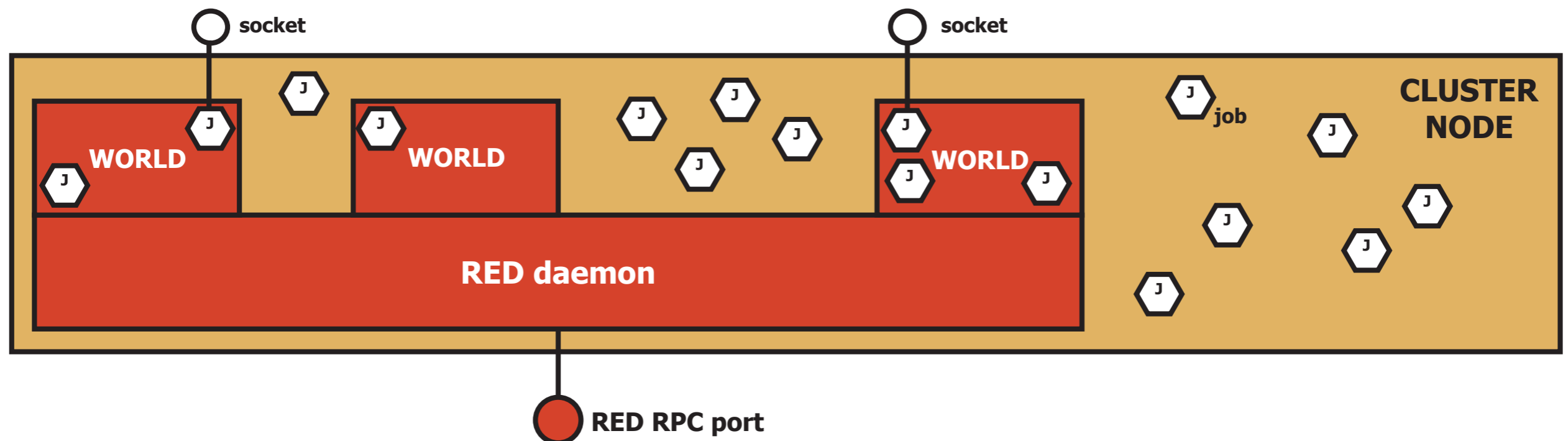
*/\* Create sandbox \*/*

*/\* Source creation \*/*

*/\* Create a sink \*/*

*/\* Run command \*/*

*/\* Destroy sandbox \*/*



# RED implementation

REQUEST                      RESPONSE

	<b>RED RPC</b>	Plain, XML-RPC, SOAP	<b>RED SPACE</b>
	<b>RED library</b>	Implementation using POSIX calls	
<b>SOCKETS</b>	<b>libc</b>	GNU libc, FreeBSD libc	<b>USER SPACE</b>
↔	<b>Kernel</b>	Linux, FreeBSD, Darwin, Windows	<b>KERNEL SPACE</b>
	<b>Hardware</b>	ia86, ia64, ia86-64, PowerPC	<b>HARDWARE</b>
	<b>Network</b>	Ethernet, Myrinet	

# Other RED methods

`props = worldProps (world)`

*/\* Get list of world properties \*/*

`pVal = getProp (world, prop)`

*/\* Get world property value \*/*

`setProp (world, prop, pVal)`

*/\* Set world property value \*/*

`filelist = getFileList (world, path)`

*/\* Get list of file properties \*/*

`makeDirectory (world, path)`

*/\* Create a directory \*/*

`setPerms (world, path, rp, wp, xp)`

*/\* Set file permissions \*/*

`removeFiles (world, path)`

*/\* Remove files recursively \*/*

`sendSignal (world, jobid, posixsignal)`

*/\* Sends a signal to a job \*/*

`jobidList = jobsList (world)`

*/\* Get the jobs list \*/*

`jobprops = jobProps (world, jobid)`

*/\* Get job properties \*/*

# REDBASH

A Remote BASH using RED

# Unix shell

- Some parallelism exists in Unix shells

- Pipeline

```
task1 | task2 | task3
```

- Parallel loop

- Operators

```
| < > &
```

```
for i in $LIST  
do  
  task &  
done  
wait
```

# Parallel Shell

- Modified version of BASH, called REDBASH
- Shell level (user level) actual parallelism
- The user can explicitly or implicitly launch commands on cluster nodes
- Shell manages redirections
  - | (pipe)
  - > (write to a file)
  - < (read from a file)

# REDBASH

- The user execute commands locally or
  - on a sandbox on a specific host
  - on a sandbox on a host choosed by the shell
- The shell takes care of executable file transfers
- The shell takes care of redirections and pipes
- No need for a global, shared file system

# Local shell, remote hosts

- When shell starts, he builds a list of available RED nodes
- Creation of a world on each RED
  - The world is empty at the beginning
  - It is filled on-demand when commands are executed

# Simple shell commands

- Simple shell command: sequence of words separated by blanks, terminated by a control operator: '|', '&&', '&', ...
- The first word of a command can be prefixed by a sequence of characters of the form **node:**, where **node** is the name of a RED host.

```
node3:/usr/bin/command a b c
```

# Redirections

- Special case for filenames that are used in redirections. User can prefix them with a node's name.

```
node3:command < node1:f1 > node2:f2
```

```
cat < node2:f2 > localfile
```

# Pipeline

- If at least one command of a pipe is a remote execution, the pipe becomes a direct network connection.

```
producer | node1:task1 | node2:task2 | consumer
```

# New parameter

- We add a new parameter **\$:** to the shell that contains the name of a RED.
- Two consecutive uses of **\$:** not necessary give the same result.

```
RH=$:  
$RH:command
```

```
RESULTHOST=$:  
producer | $::task1 | $::task2 > $RESULTHOST
```

# New variable

- The variable **REDHOSTS** contains the list of available RED

```
for i in $REDHOSTS
do
  $i:task &
done
wait
```

# Performance results

This is a simulation !

[http://www.cri.ensmp.fr/people/silber/metacc/red/  
coset02.tar.gz](http://www.cri.ensmp.fr/people/silber/metacc/red/coset02.tar.gz)

# Application

- A pipeline, where  $x$  is the number of packets of 1024 bytes produced by **producer** and consumed by **consumer**.
- Each **task**
  - Reads a packet of 1024 bytes from stdin
  - Iterates  $w$  times over it doing 2 arithmetic operations on each byte
  - Writes the packet on stdout

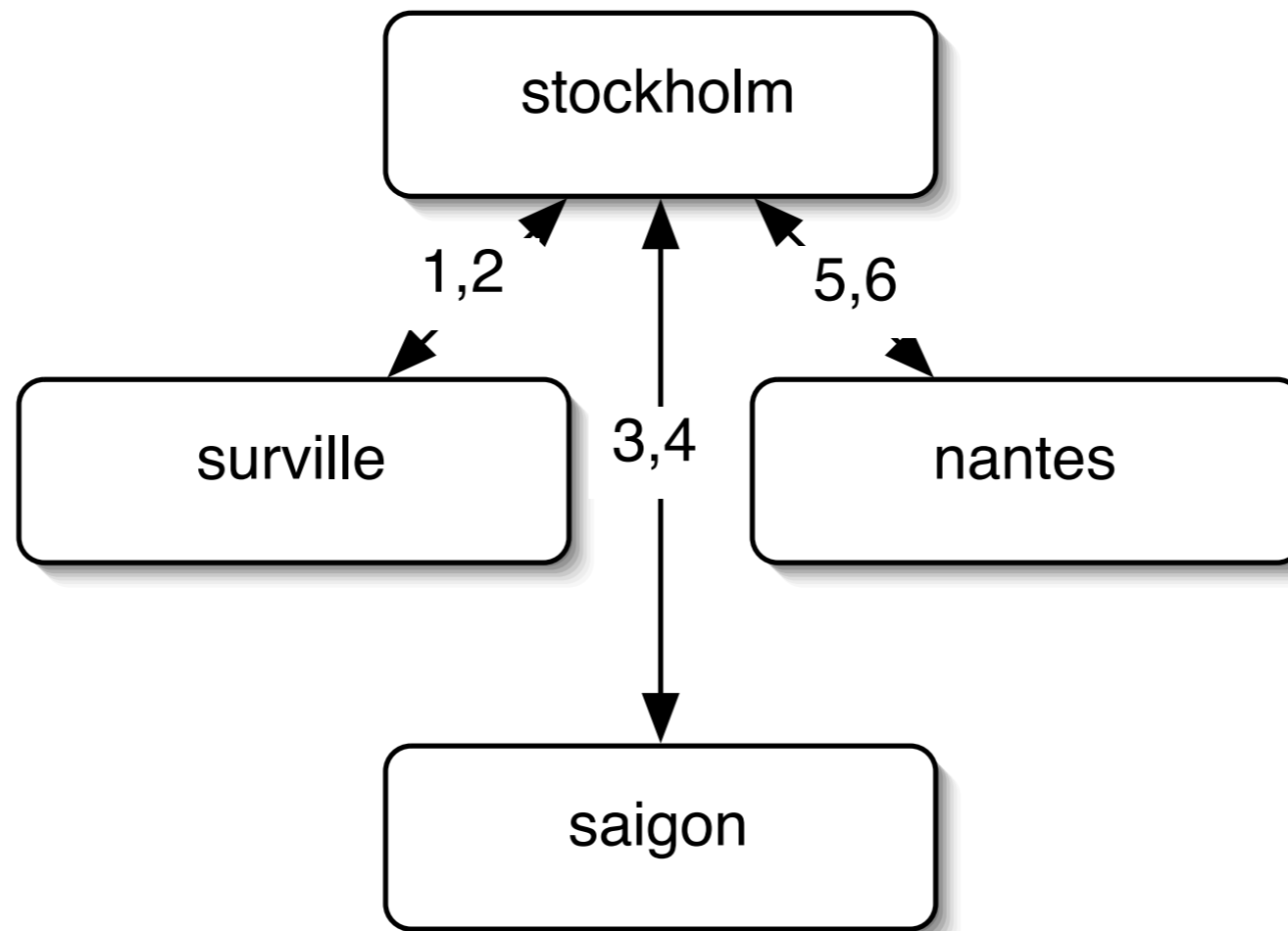
producer  $x$  | task  $w$  | task  $w$  | task  $w$  | consumer  $x$

# Experimental environment

- Four computers
  - **stockholm**: Pentium 4, 2.4GHz 2GB
  - **saigon**: Pentium 4, 2.4 GHz 2GB
  - **nantes**: Pentium 4, 3.0 GHz 2GB
  - **surville**: 2xOpteron 244, 8GB
- Interconnected by a 100Mbit/s Ethernet Switch

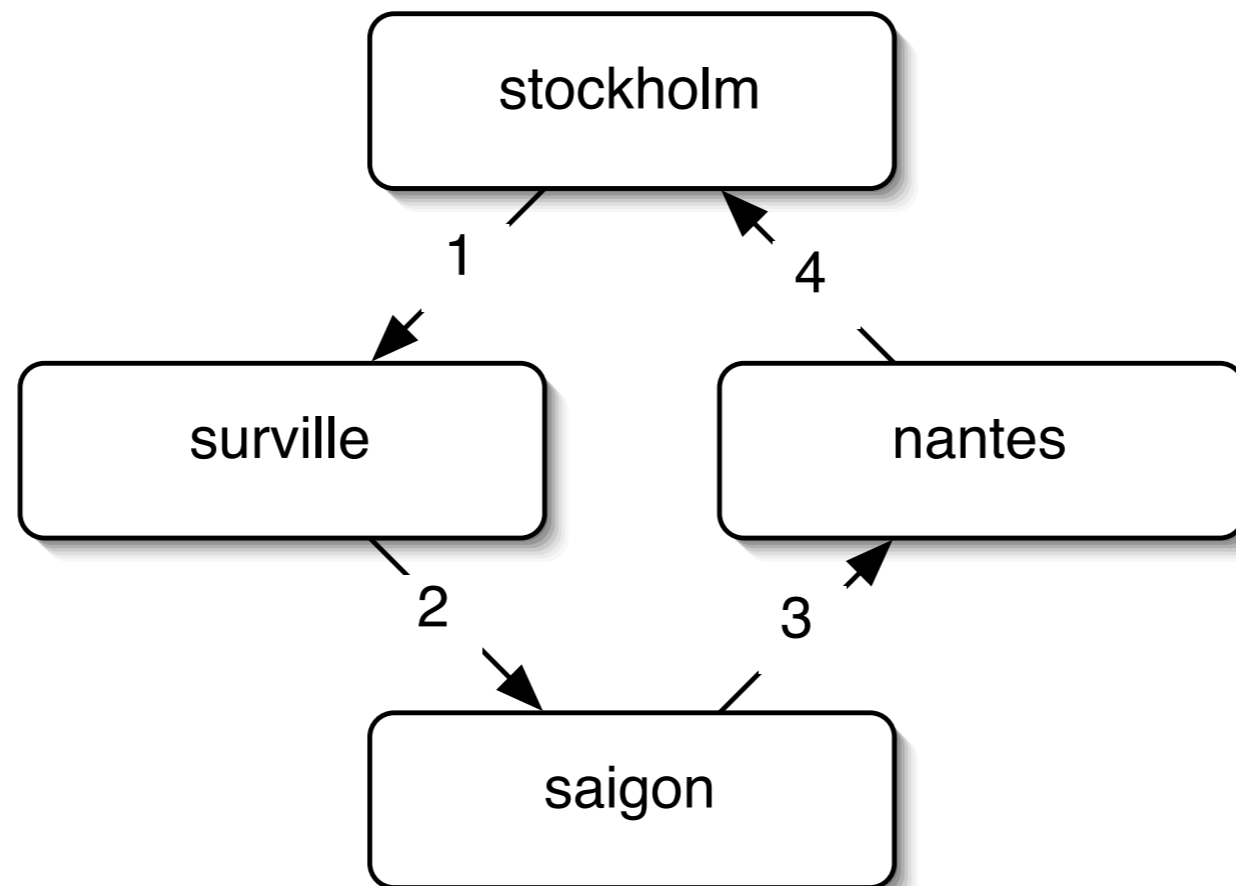
# Remote execution

```
producer x | (rsh surveille task w)  
           | (rsh saigon task w)  
           | (rsh nantes task w)  
           | consumer x
```



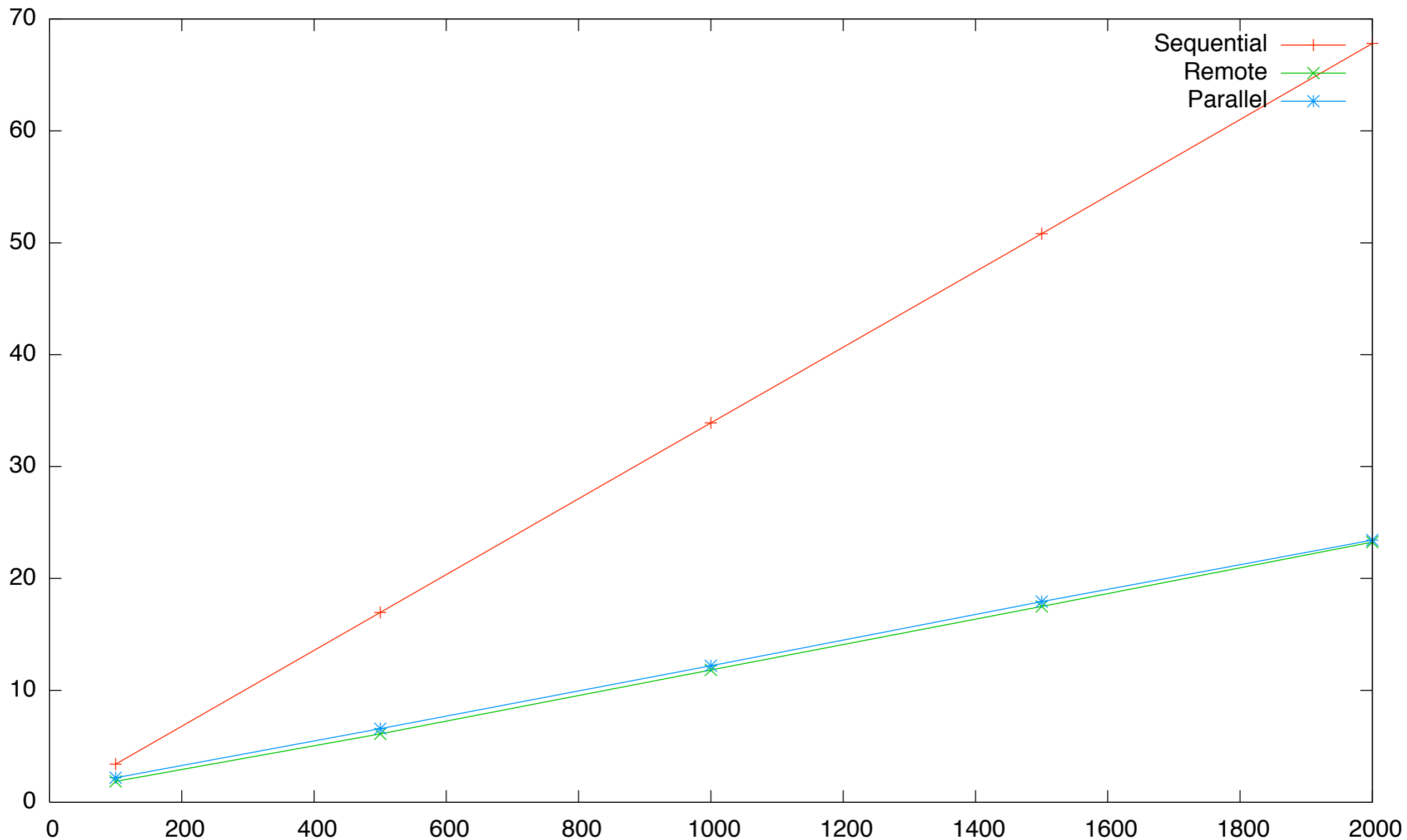
# Parallel execution

```
producer x | surveillance:task w  
           | saigon:task w  
           | nantes:task w  
           | consumer x
```



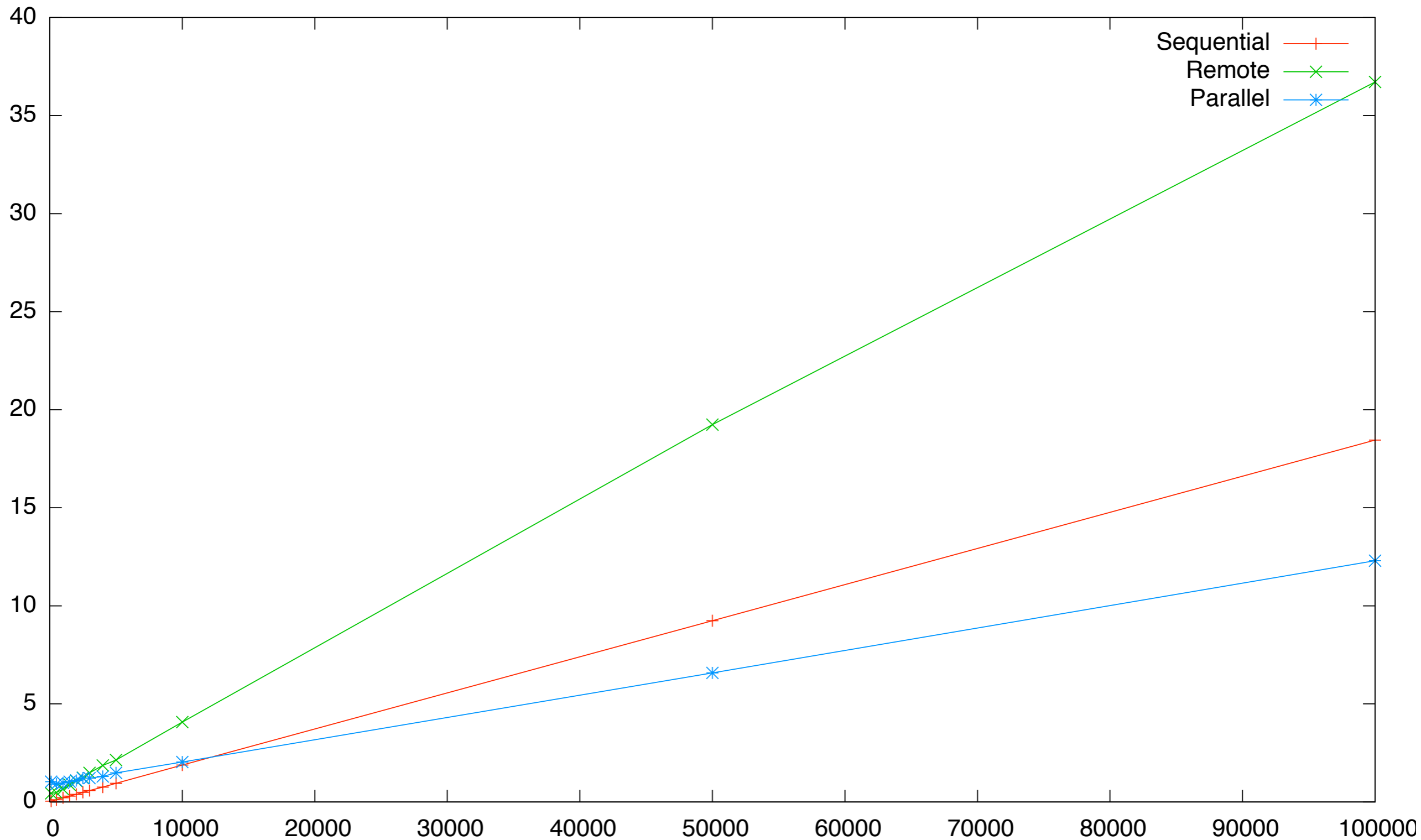
# CPU-bound

w=1024 (1048576 iterations per packet)



# IO-bound

w=2 (2048 iterations per packet)



# Conclusion and Future work

- A new syntax to express explicit or implicit parallelism on clusters with Unix shell constructs.
- The specification of a very simple remote operating system.
- Significant performance results.
- No modification of existing applications.
- Future work: heterogeneous clusters (Grid), persistence?, fault-tolerance?, REDBOX.