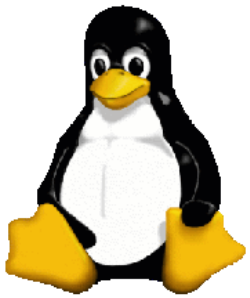
A new Linux scheduler to balance HPC applications

Roberto Gioiosa[1]

[1]Computer Science Division
Barcelona Supercomputing Center
roberto.gioiosa@bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación



Linux Kernel Hacking 2008

Outline

Outline

High Performance Computing

Very often High Performance Computing is achieved through Parallel Computing, a form of computing in which many instructions are carried out simultaneously.

Ideally, parallelizing a program in N parallel tasks reduces the execution time by N times. Unfortunately, the speed-up is limited by the Amdahl's law:

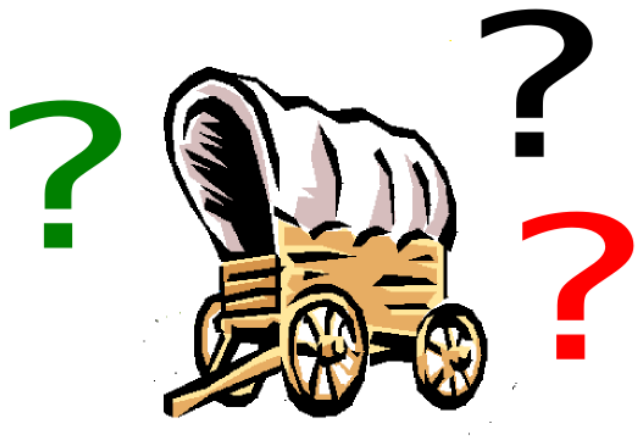
Amdahl's law

$$S = \frac{1}{(1 - P) + P/N}$$

where S is the speed-up, P the portion of the code that can be parallelized and N the number of concurrent processes.
If P=0 (S=1) there is no speed-up; if P=1 the speed-up is infinite.

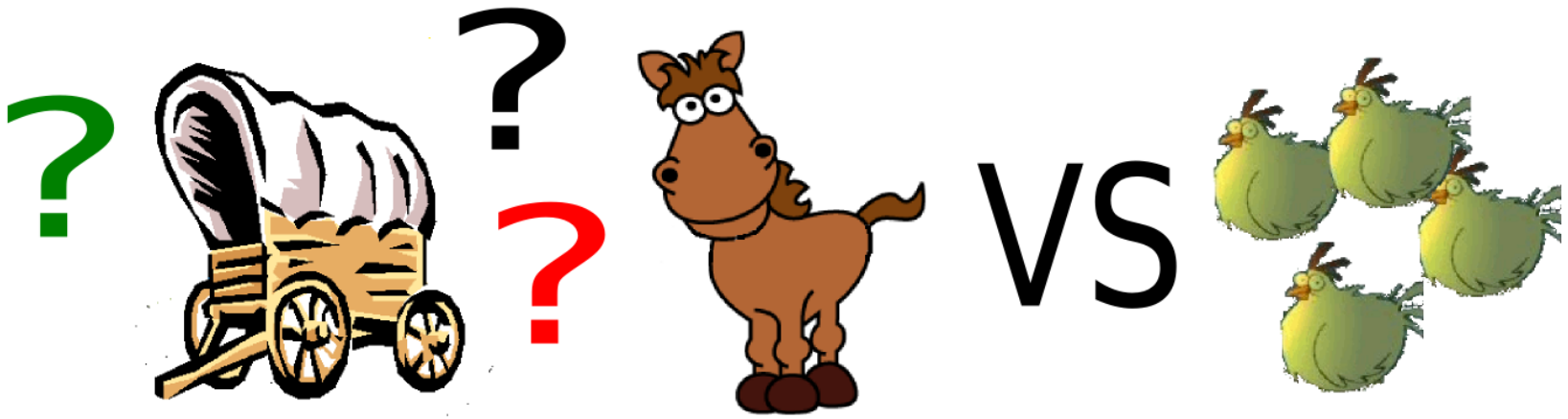
Putting in simple words...

Let's assume we have a wagon...



Putting in simple words...

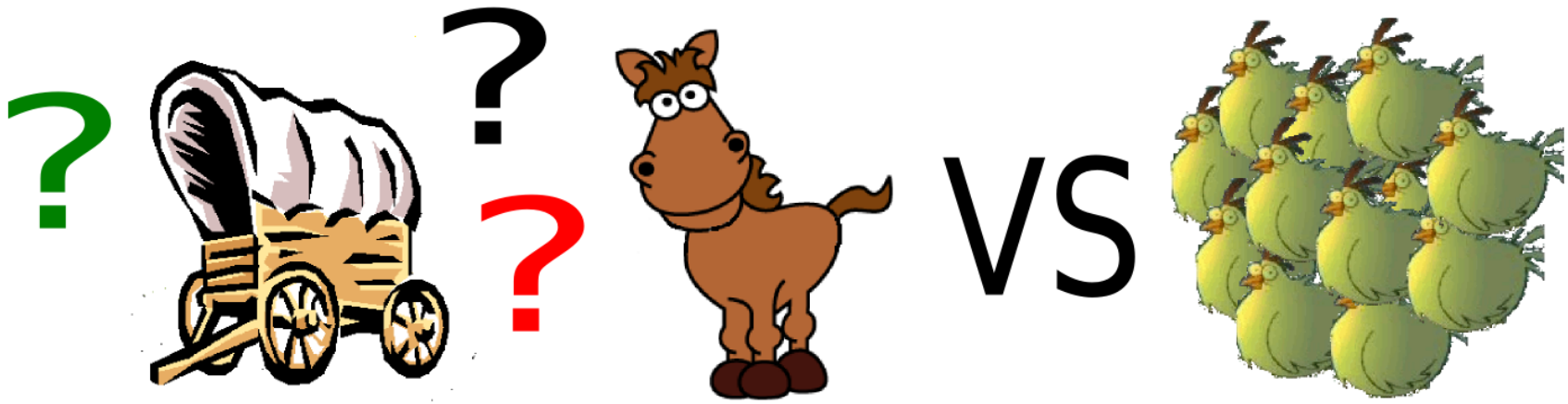
Let's assume we have a wagon...



Are N chickens more powerful than a horse?

Putting in simple words...

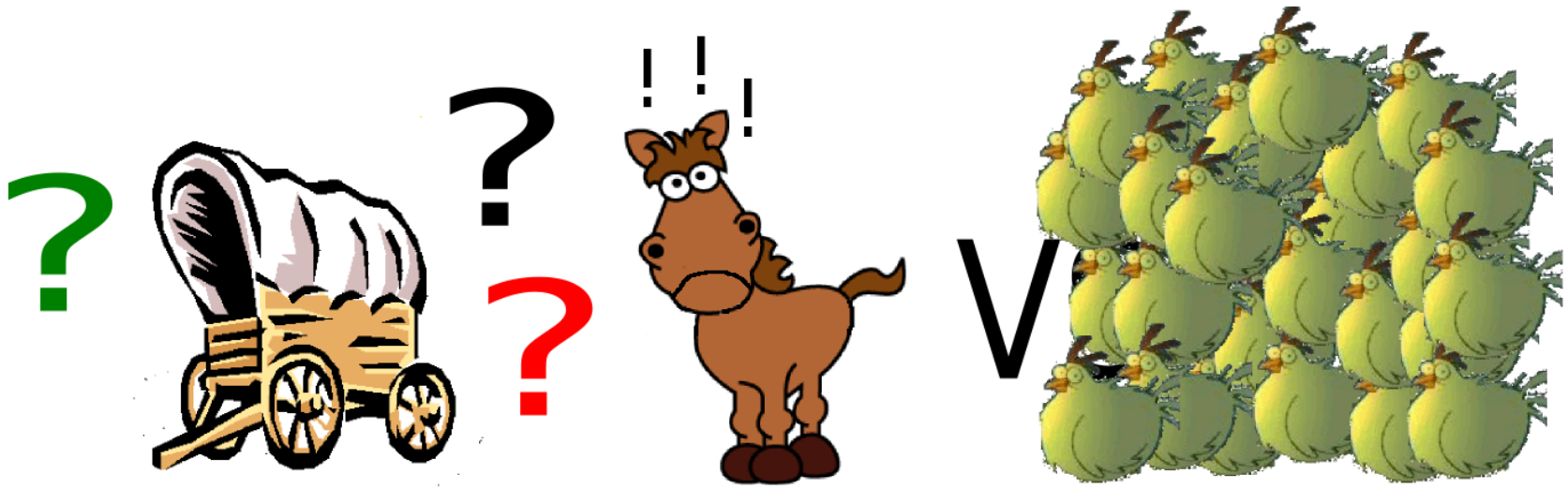
Let's assume we have a wagon...



Are N chickens more powerful than a horse? If N is large enough...

Putting in simple words...

Let's assume we have a wagon...



Are N chickens more powerful than a horse? If N is large enough...
Chickens might overwhelm the horse!!! [S. Filippone, CA 2000]

The Top500 list

Since 1993, the Top500 project has listed the 500 most powerful computer systems in the world.
The list is compiled by high-performance computer experts, computational scientists, manufacturers, and the Internet community.

The list is updated two times per year:

- International Supercomputing Conferences (ISC) in June (Germany)
- Supercomputing Conference (SC) in November (US)

The ‘best’ performance are measured as the performance of the LINPACK Benchmark (Jack Dongarra). LINPACK was chosen because it is widely used and performance numbers are available for almost all relevant systems.

The Nov07 Top500 list

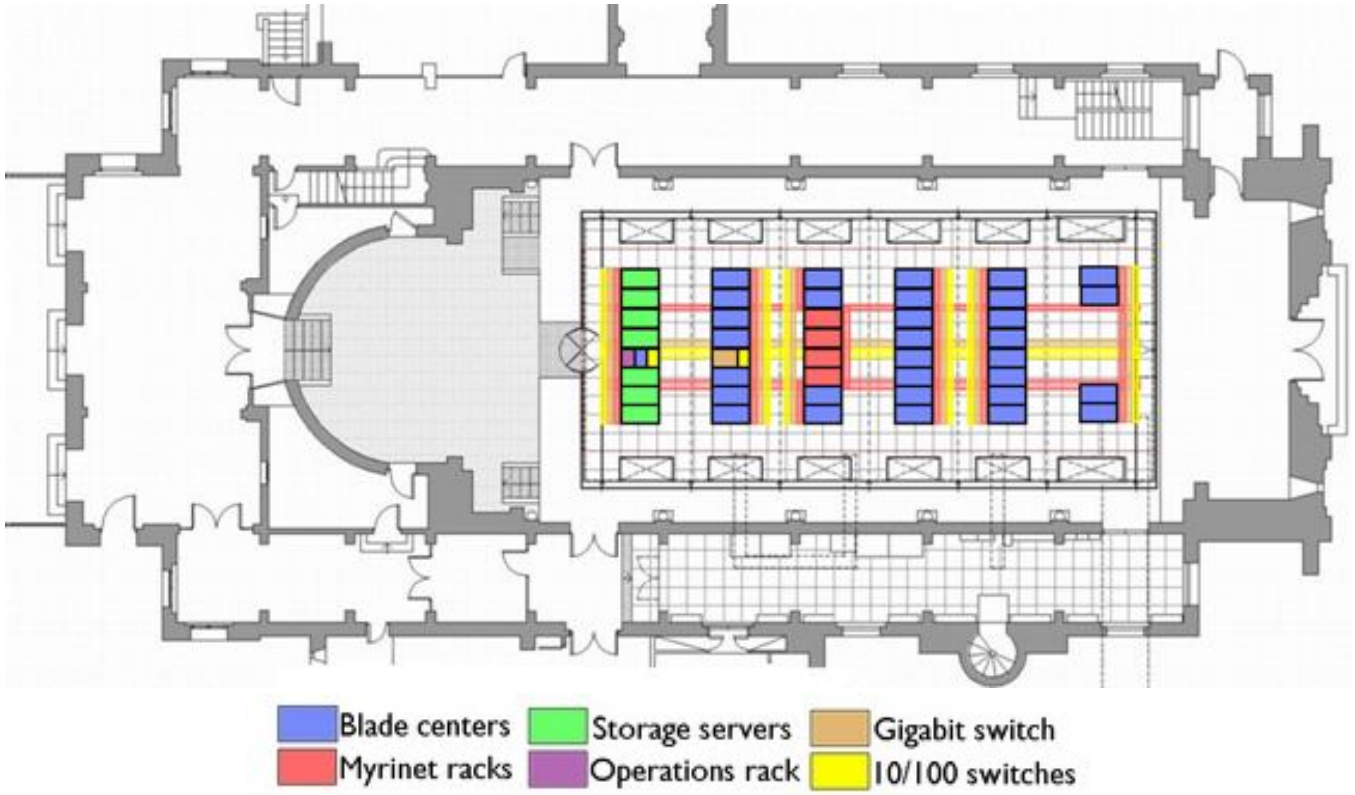
N	Site	Computer	CPUs	Rmax
1	LLNL	BlueGene/L	212992	478200
2	FZJ	BlueGene/P	65536	167300
3	NMCAC	SGI AltixICE8200 Xeon QCore 3GHz	14336	126900
4	TATA	EKA - HP Xeon 3GHz	14240	117900
5	Gov.	HP Xeon 2.66GHz	13728	102800
	...	...	...	...
13	BSC	MareNostrum - PPC970MP 2.3Ghz	10240	63830
	...	...	...	...
48	CINECA	BladeCenter Xeon DCore 3.0Ghz	5120	19910
	...	...	...	...
86	LANL	Roadrunner 2 - Opteron DCore	4608	14070
	...	...	...	...
91	LANL	ASCI Q - Alpha 1.25Ghz	8192	13880
	...	...	...	...
500	Sem C	HP Xeon 1.86Ghz	1344	5929.6

MareNostrum

MareNostrum is a supercomputer based on PowerPC 970MP processors, the architecture BladeCenter, a Linux system and a Myrinet interconnection.

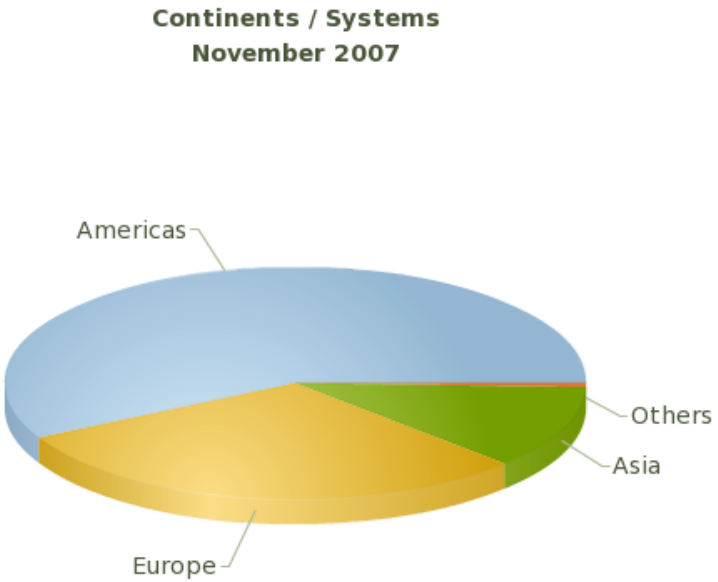
When it first entered the Top500 list, MareNostrum was ranked as the 4th supercomputer with about 40 TFLOPS. Today, the machine holds the 13th position (the 3rd in Europe) with about 64 TFLOPS.

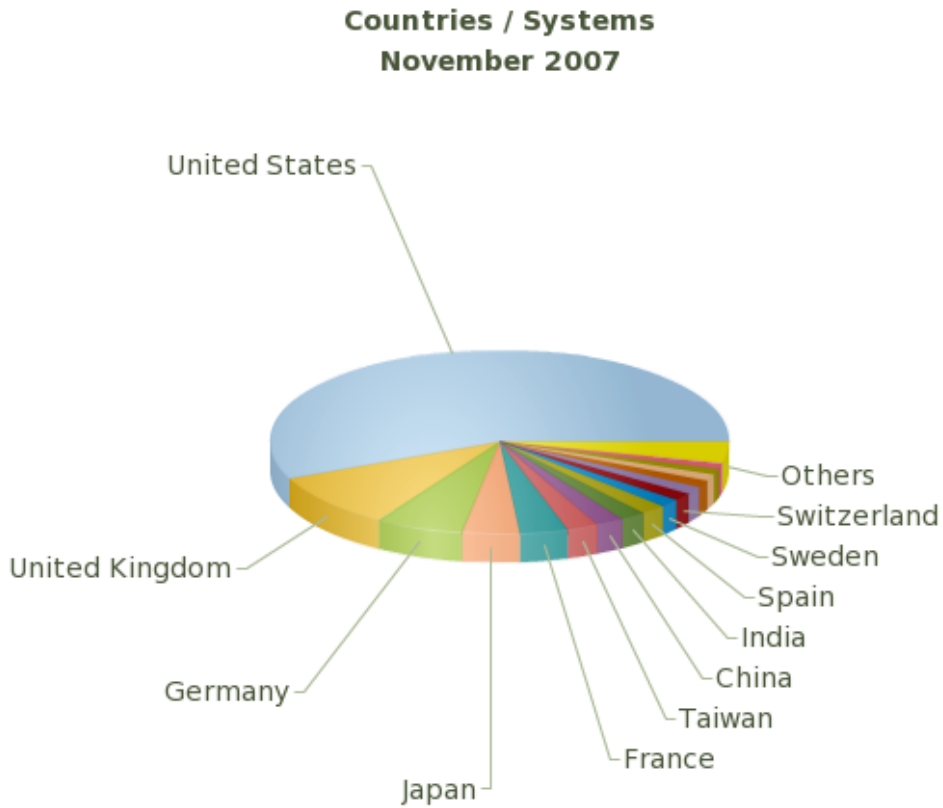
MareNostrum Architecture



- 94,21 TFLOPS PeakPerf
- 10240 IBM Power PC 970MP processors at 2.3 GHz
- 20 TB of main memory
- 280 + 90 TB of disk storage
- Myrinet and Gigabit Ethernet
- Linux OS

Country distribution

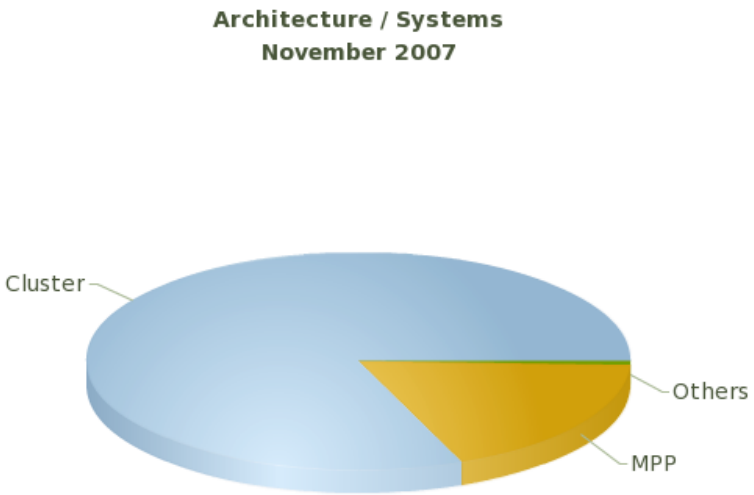


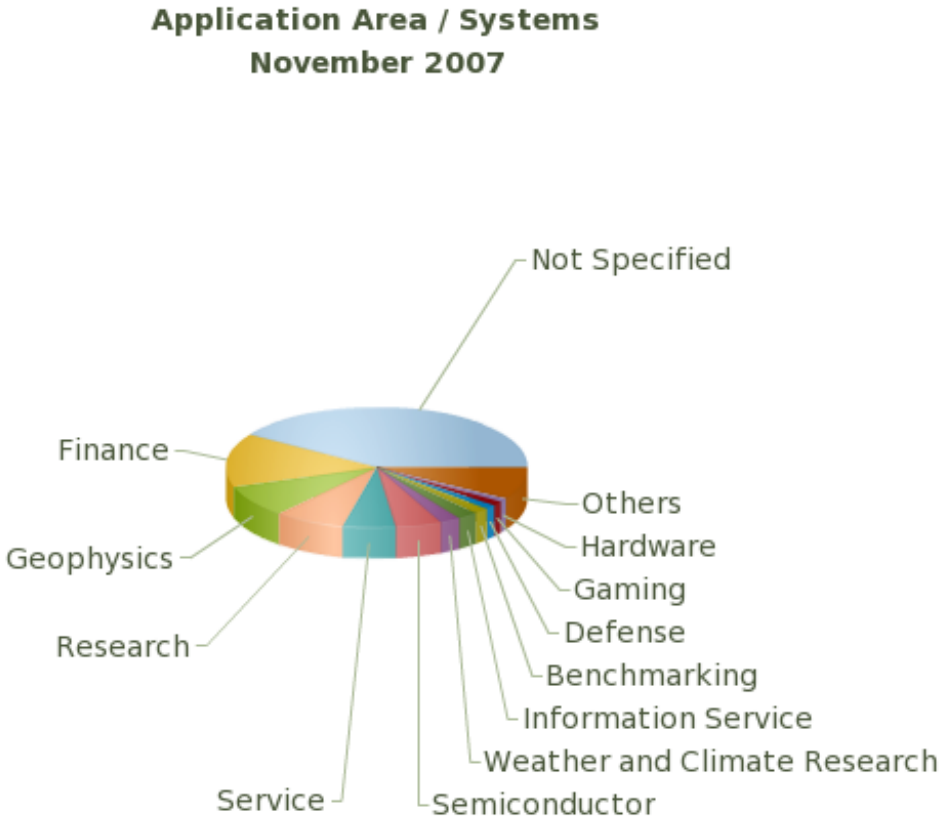


The most powerful supercomputers are in US, though Europe and Asia are buying more and more powerful machines. Italy is still behind the first group of European countries (Germany, UK, France, Spain,...)

What is a supercomputer used for?

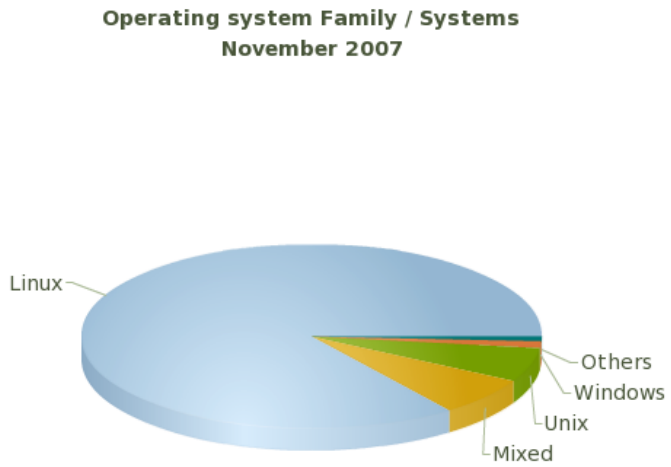
While supercomputers are used for several different goals, High Performance Computing clusters are the largest system architectures used nowadays.





Linux for HPC systems

Linux is the most used OS for HPC computers

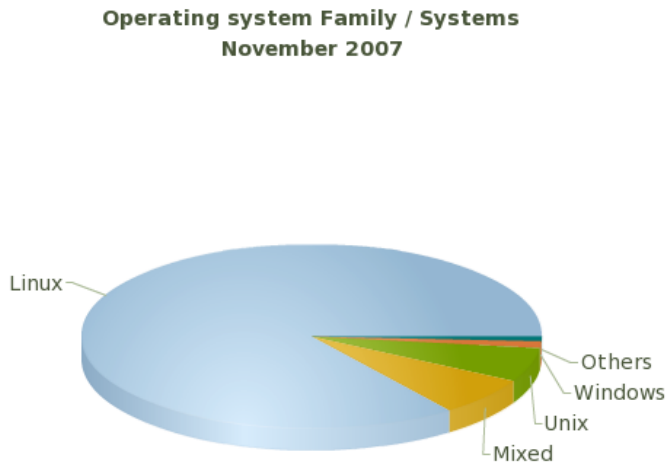


World Domination

"In the 1990s Linus Torvalds used to give a talk called World Domination 101 on the early steps he believed Linux would need to take to achieve world domination...

Linux for HPC systems

Linux is the most used OS for HPC computers

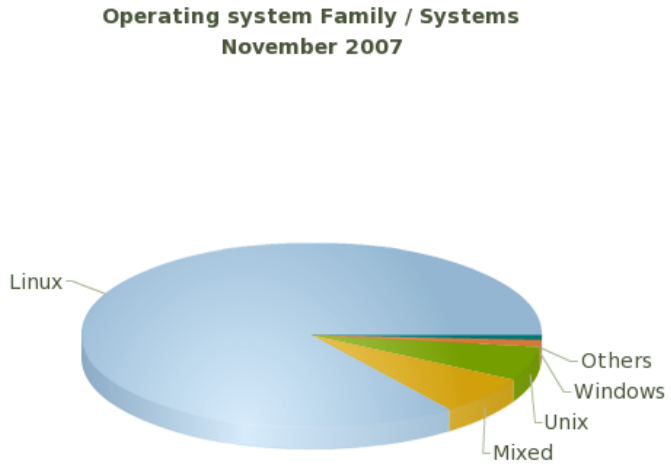


World Domination

"In the 1990s Linus Torvalds used to give a talk called World Domination 101 on the early steps he believed Linux would need to take to achieve world domination... fast"

Linux for HPC systems

Linux is the most used OS for HPC computers



World Domination

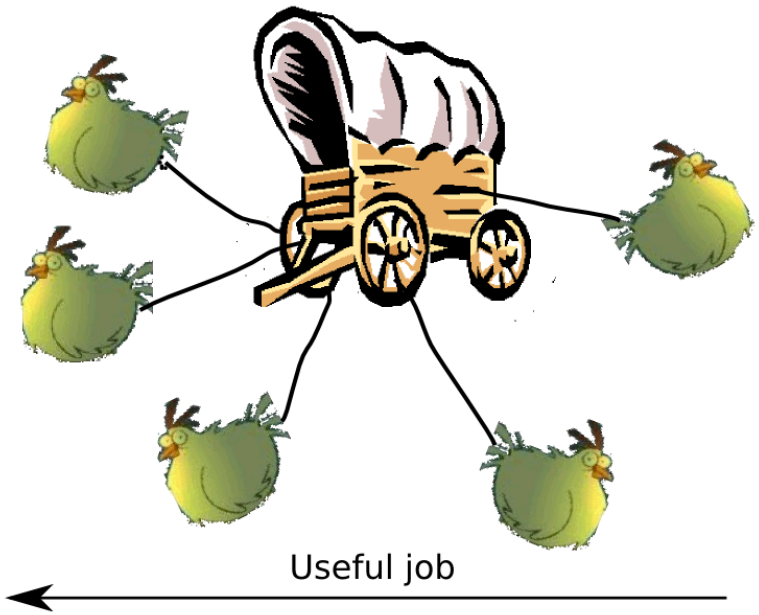
"In the 1990s Linus Torvalds used to give a talk called World Domination 101 on the early steps he believed Linux would need to take to achieve world domination... fast" ... we are on our way!

Problems with parallel computing

Let's suppose we have enough chickens to overwhelm a horse, i.e., the sum of the computing power of each processor in the cluster is greater than the computing power of a mainframe:

Problems with parallel computing

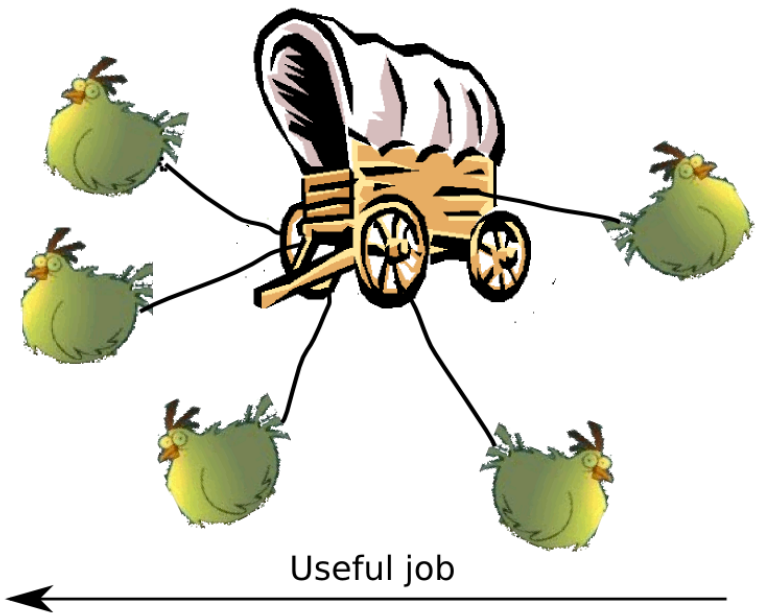
Let's suppose we have enough chickens to overwhelm a horse, i.e., the sum of the computing power of each processor in the cluster is greater than the computing power of a mainframe:

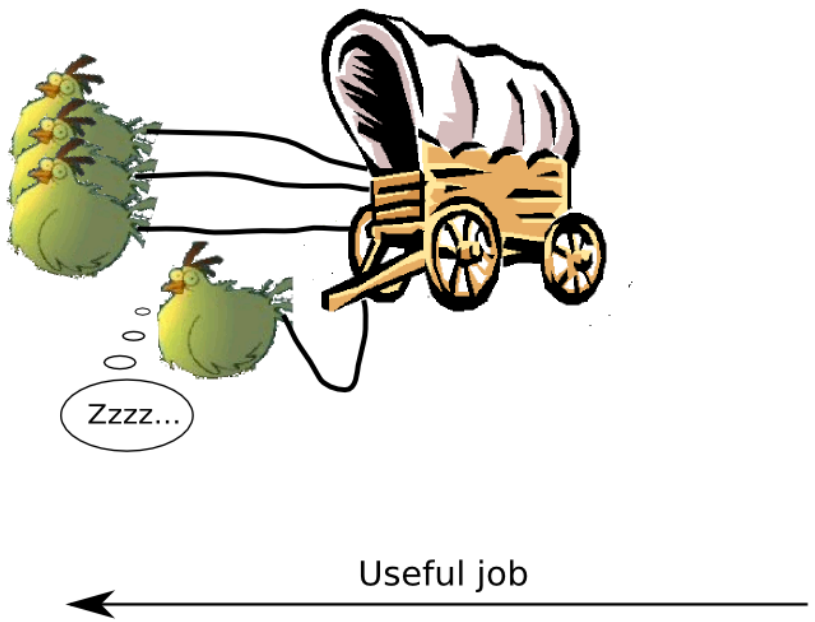


Can we make them going to the same direction?

Problems with parallel computing

Let's suppose we have enough chickens to overwhelm a horse, i.e., the sum of the computing power of each processor in the cluster is greater than the computing power of a mainframe:





Can we make them going to the same direction?

And going to the same direction with the same speed?

Outline

HPC applications

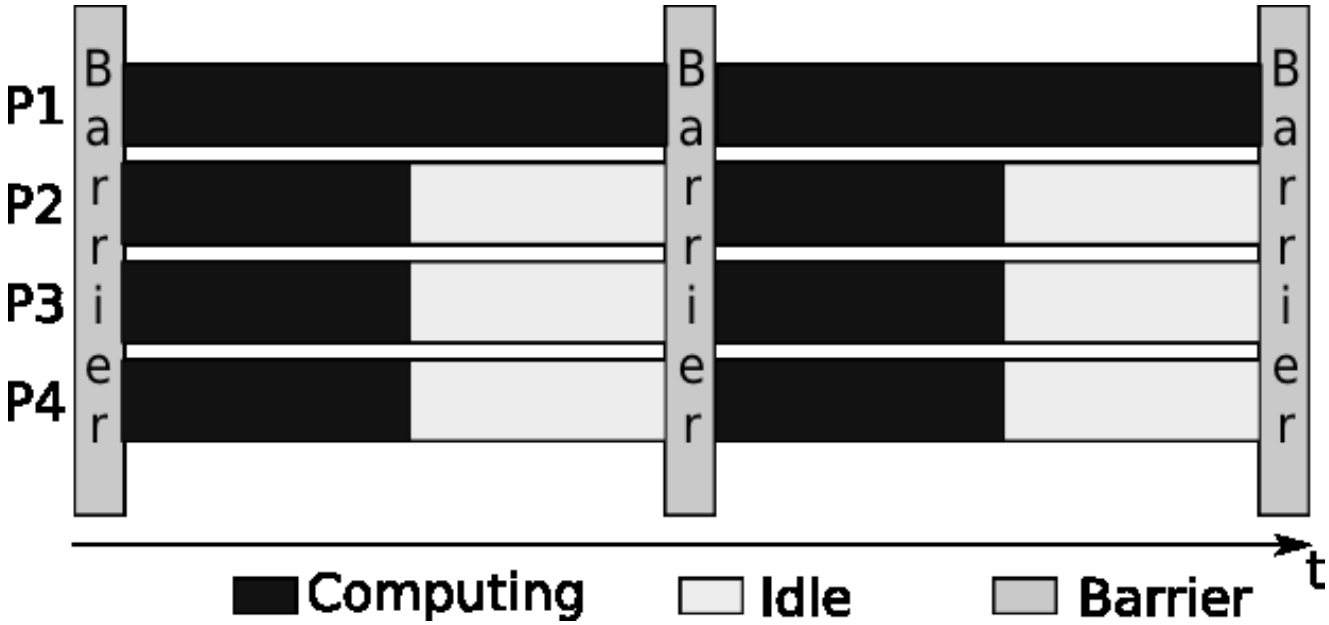
High Performance Computing (HPC) applications are usually *Single Process-Multiple Data* (SPMD).

- In SPMD applications, all the processes execute the same code on different data sets and use synchronization primitives to coordinate their work.
- Since the processes execute the same code, they are supposed to reach their synchronization points roughly at the same time.

However, some HPC applications suffer from load-imbalance, i.e. the execution time of the processes in the parallel application is not the same.

Load-imbalance in HPC applications

If a process runs for longer than the others belonging to the same application, all the other processes have to wait for that process to complete its execution.



During this time the CPUs of the waiting processes are idle, resulting in a significant loss of performance and waste of resources.

Causes for load imbalance

The causes for load-imbalance can be internal or external to the application (likely both):

Intrinsic The causes are internal to the application's code, input set or both.

- Input set
- Domain density
- Data exchanging

Extrinsic External factors not under the control of the application

- OS noise
- User daemons
- Network topology

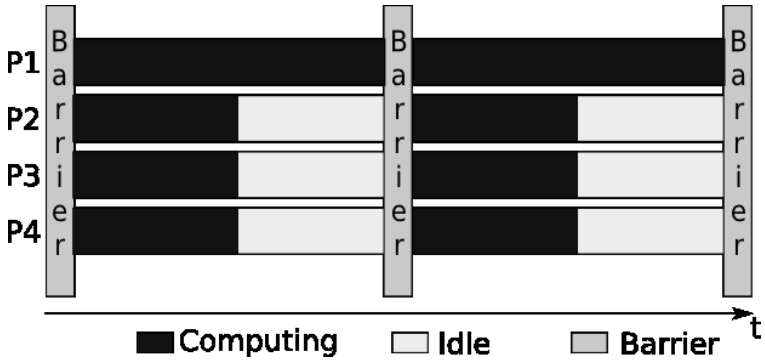
Summary of the problem

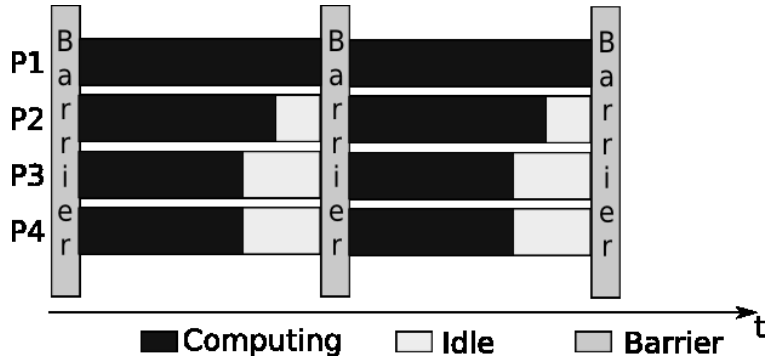
- Though not easy, an expert programmer could reduce the intrinsic imbalance in the application by tailoring the algorithm for a particular system and/or balancing the input.
- These operations should be repeated for each systems and data input set
- Extrinsic imbalance factors are neither under the control of the application nor of the programmer

A mechanism that aims to solve the imbalance of an application should be transparent to the user

Hardware resource allocation

Our solution for balancing HPC applications consists of assigning more hardware resources to the most computing-intensive processes..





Giving this process more hardware resource shall decrease its execution time and the execution time of the whole MPI application.

It is not for free: the performance of the task running with less resources will decrease!

Run time hardware resource allocation

The proposed solution is very simple and does not make any assumptions on the kind of application, the programming model or the input set used.

The only assumption regards the underneath processor, which must provide a shared resource control mechanism.

The approach is dynamic in the sense that the amount of resources assigned to a process can be set at run time.

The IBM POWER5 processor

The idea is general and can be applied to any processors that allows the software to control the hardware resource allocation.

IBM POWER5 processors provide this kind of mechanism with the *hardware thread priority management*.

- Each context in a core has a *hardware thread priority*: as the hardware thread priority of a context increases the amount of hardware resources assigned to that context increases too.
- Other MT processors like the IBM POWER6 or the Cell processor are capable of dynamically allocating resources to the contexts at run time.

The hardware thread priority mechanism

Each thread can set its priority using an `or-nop` instruction:

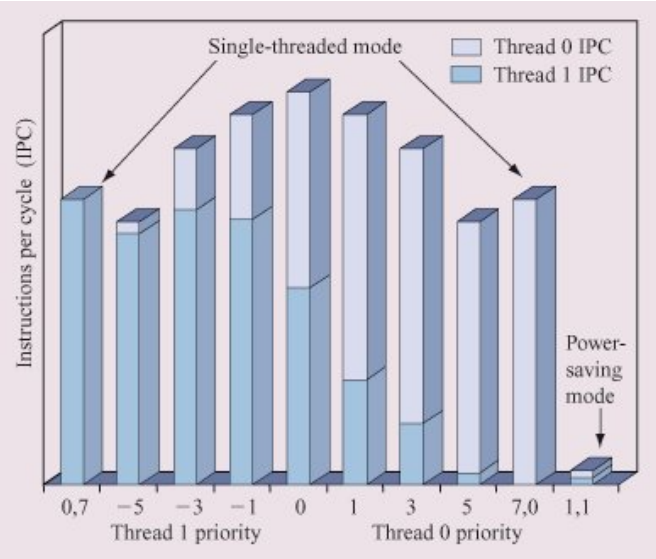
Priority	Priority level	Privilege level	or-nop inst.
0	Thread shut off	Hypervisor	-
1	Very low	Supervisor	`or 31,31,31`
2	Low	User	`or 1,1,1`
3	Medium-Low	User	`or 6,6,6`
4	Medium	User	`or 2,2,2`
5	Medium-high	Supervisor	`or 5,5,5`
6	High	Supervisor	`or 3,3,3`
7	Very high	Hypervisor	`or 7,7,7`

However, the processor core assigns the hardware resources according to the difference of the priorities of the two threads.

Thread priority implementation

The core processor assigns resources to each thread by decoding more instructions from one context than from the other.

The number of decode cycles assigned to each thread depends on its hardware priority: every R decode cycles, and it is computed as $R = 2^{|X-Y|+1}$



Prior. diff.	R	A	B
0	2	1	1
1	4	3	1
2	8	7	1
3	16	15	1
4	32	31	1

* R.N. Kalla et al., IBM Journal of Research and Development: POWER5 System Microarchitecture

Thread priority in Linux

The Linux 2.6.24 kernel only exploits hardware priorities in a limited number of cases:

1. The processor is spinning for a lock in kernel mode. In this case the priority of the spinning process is reduced.
2. A CPU is waiting in kernel mode for some operations to complete (for example with `smp_call_function()`). In this case the priority of the CPU is decreased until the operation completes.
3. The kernel is running the idle process. In this case the kernel reduces the priority of the idle CPU and, eventually, put the core in Single Thread (ST) mode.

The HMT patch

In order to check that our approach can be used for balancing the HPC application, we had to modify the original kernel code for two reasons:

1. Every time the CPU receives an interrupt, the interrupt handler sets the priority back to MEDIUM (4)
2. Only few hardware priorities (2 (LOW), 3 (MEDIUM-LOW) and 4 (MEDIUM)) can be set at user mode level.

To set priority `N` for process `<PID>` a user shall simply write to a `proc` file:

```
echo N > /proc/<PID>/hmt_priority
```

Experimental environment

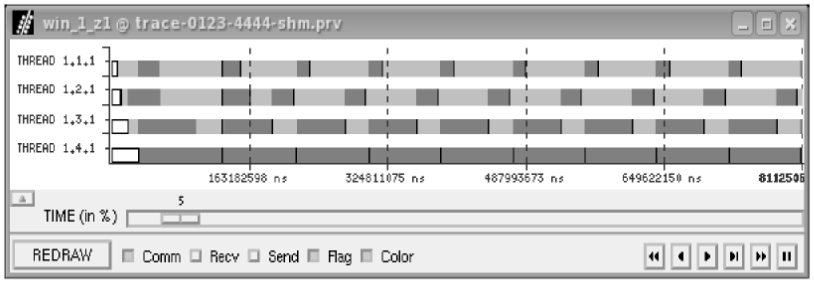
We tested our load-balancing solution on a IBM OpenPOWER 710 system equipped with an IBM POWER5 processor (2 cores, 2 contexts per-core).

We ran on this machine a modified version of the Linux 2.6.19.2 kernel that allows us to exploit priority levels from 1 to 6.

We performed our tests on three kind of applications:

1. MetBench: A microbenchmark suite developed at BSC (not in this presentation)
2. BT-MZ from the NAS MPI benchmark suite
3. SIESTA: A real application commonly used by BSC's researchers.

BT-MZ: Case A

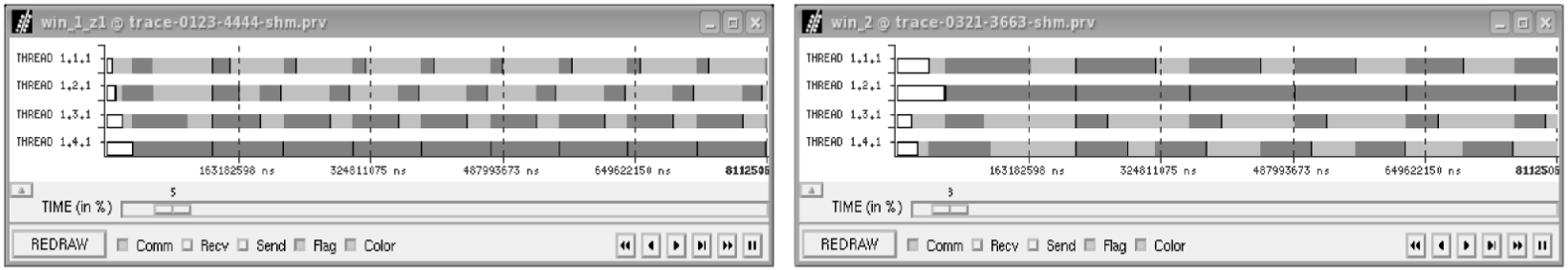


Computing Waiting

Proc	Core	P	Comp %	Sync %
P1	1	4	17.63	82.32
P2	1	4	28.91	71.02
P3	2	4	66.47	33.4
P4	2	4	99.72	0.09

Test	Imb %	Exec. Time
A	82.23	81.64s

BT-MZ: Case B

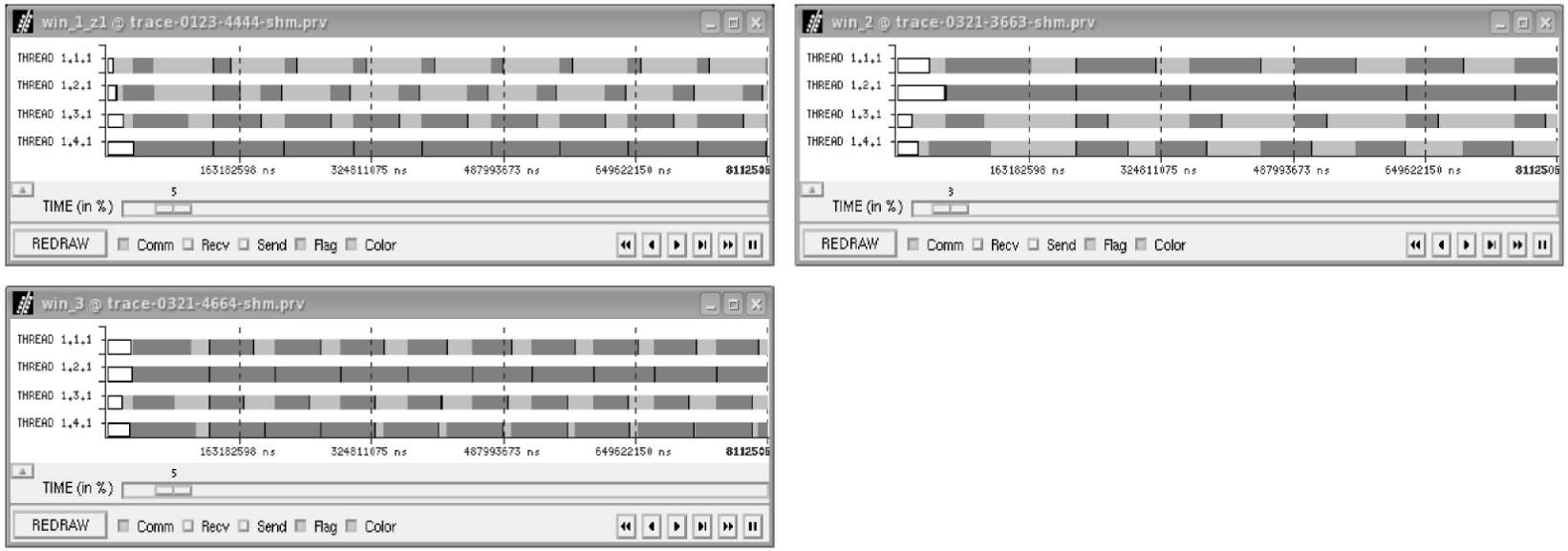


Computing Waiting

Proc	Core	P	Comp %	Sync %
P1	1	3	52.33	47.49
P2	2	3	99.64	0.14
P3	2	6	28.87	71.07
P4	1	6	46.26	53.65

Test	Imb %	Exec. Time
A	82.23	81.64s
B	70.93	127.91s

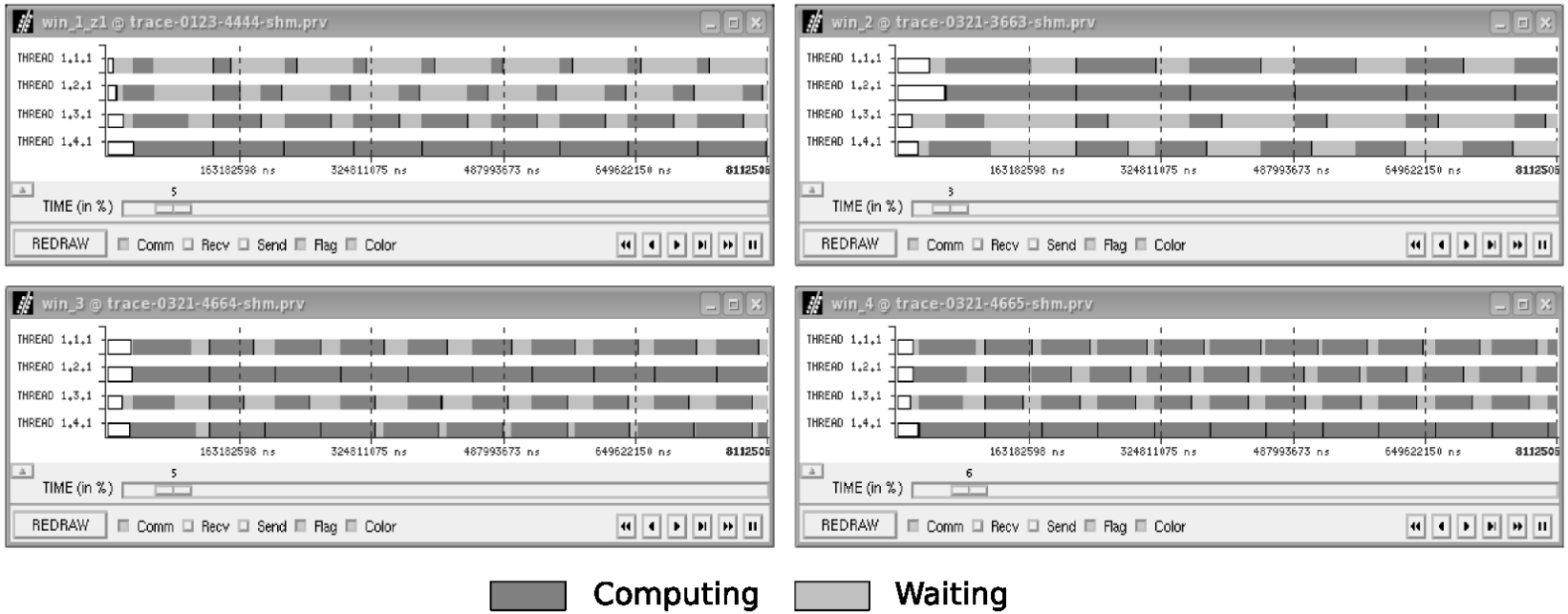
BT-MZ: Case C



Proc	Core	P	Comp %	Sync %
P1	1	4	65.32	34.48
P2	2	4	99.68	0.12
P3	2	6	53.78	46.11
P4	1	6	85.88	14.44

Test	Imb %	Exec. Time
A	82.23	81.64s
B	70.93	127.91s
C	45.99	75.62s

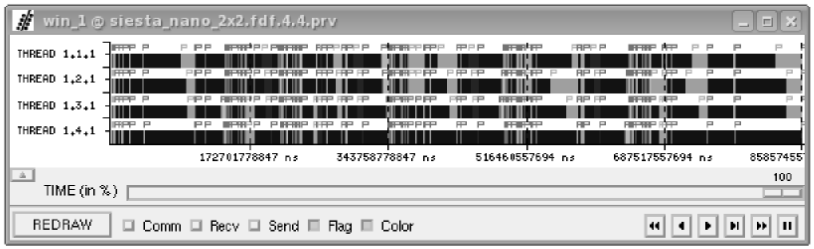
BT-MZ: Case D



Proc	Core	P	Comp %	Sync %
P1	1	4	82.73	17.10
P2	2	4	73.68	26.17
P3	2	5	66.40	33.47
P4	1	6	99.72	0.09

Test	Imb %	Exec. Time
A	82.23	81.64s
B	70.93	127.91s
C	45.99	75.62s
D	33.38	66.88s

SIESTA: Case A

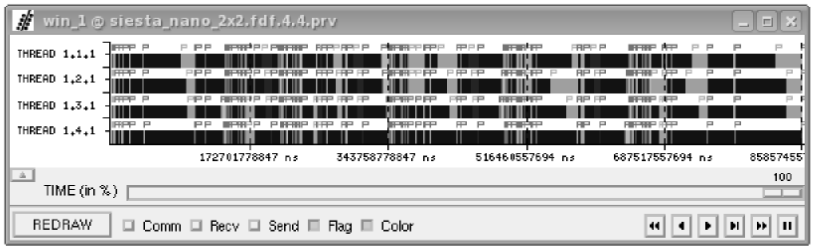


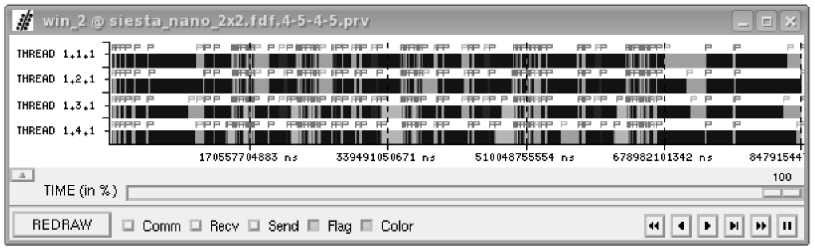
Computing Waiting

Proc	Core	P	Comp %	Sync %
P1	1	4	75.94	15.42
P2	1	4	75.24	18.11
P3	2	4	82.08	10.71
P4	2	4	93.47	3.18

Test	Imb %	Exec. Time
A	14.43	858.57s

SIESTA: Case B



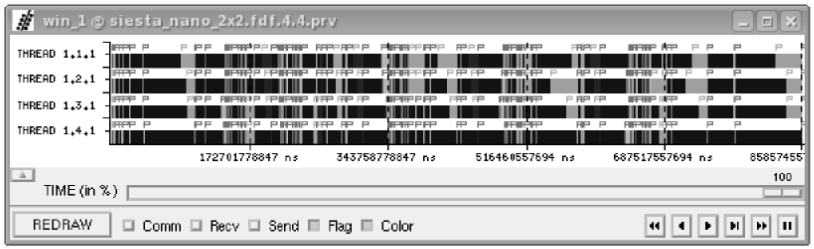


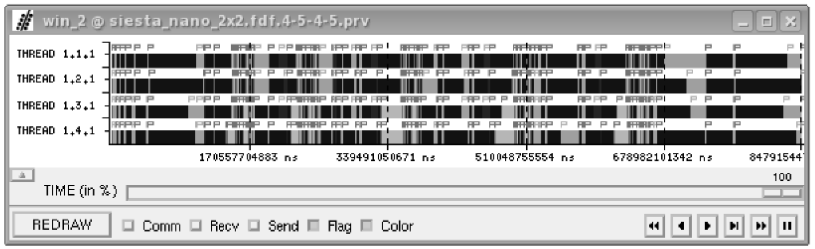
Computing Waiting

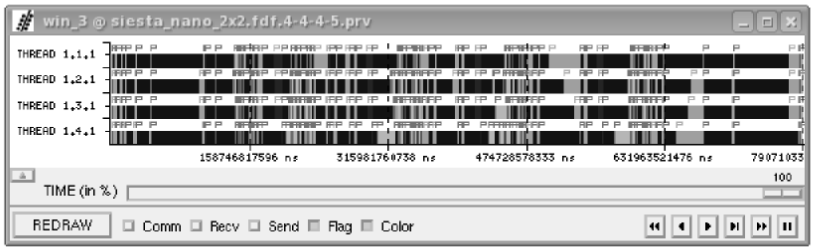
Proc	Core	P	Comp %	Sync %
P1	2	4	79.57	14.67
P2	1	4	87.06	10.15
P3	1	5	72.04	12.69
P4	2	5	77.73	8.68

Test	Imb %	Exec. Time
A	14.43	858.57s
B	5.99	847.91s

SIESTA: Case C







Computing Waiting

Proc	Core	P	Comp %	Sync %
P1	2	4	83.04	10.59
P2	1	4	79.66	10.52
P3	1	4	80.78	9.41
P4	2	5	78.74	9.13

Test	Imb %	Exec. Time
A	14.43	858.57s
B	5.99	847.91s
C	1.46	789.20s

SIESTA: Case D

Proc	Core	P	Comp %	Sync %
P1	2	4	90.76	5.60
P2	1	4	65.74	22.25
P3	1	4	68.08	19.36
P4	2	6	63.95	18.10

Test	Imb %	Exec. Time
A	14.43	858.57s
B	5.99	847.91s
C	1.46	789.20s
D	16.64	976.35s

Outline

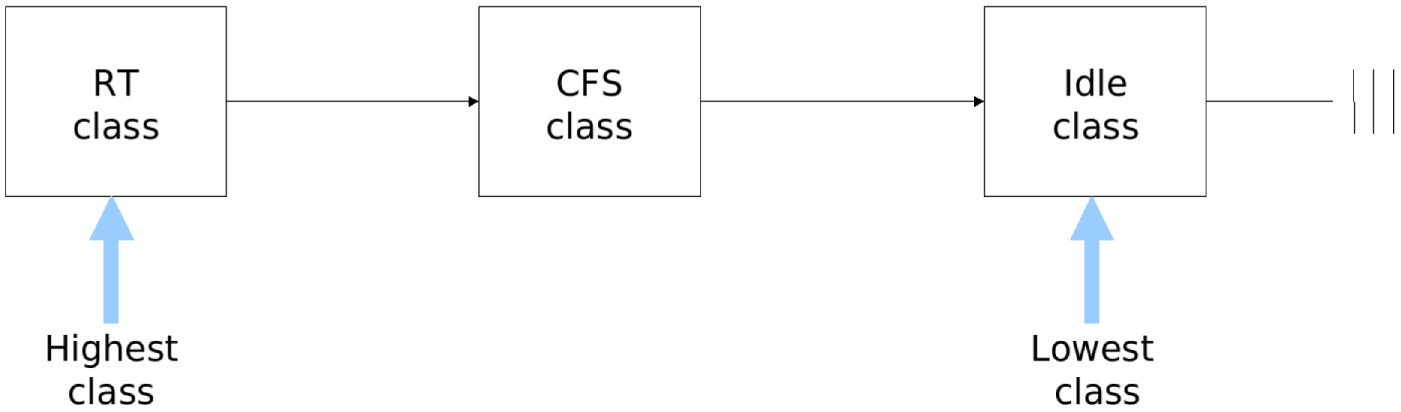
The new Linux Scheduler Framework

Linux 2.6.23 introduces new scheduling features:

- A new scheduler for normal processes (the Complete Fair Scheduler)
- A new Scheduler Framework based on the concept of a Core Scheduler and one Scheduling Class for each process class (RT, normal, idle).

Each Scheduling Class implements a different scheduler algorithm and handles some scheduling policy

Scheduling classes



Scheduling Classes are linked together in a simple list:

- Every processor has its own list
- Every class contains a list of runnable processes assigned to a CPU
- The order of the list is important because it implicitly prioritize the Scheduling Classes

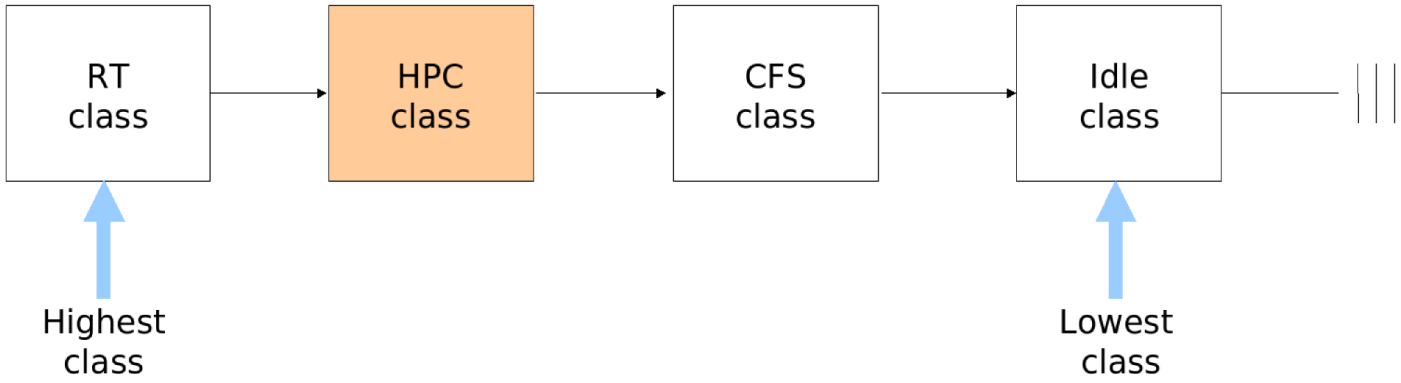
The Scheduler Core

The scheduler core:

1. Starts from the highest priority class (real time class)
2. Search for a runnable process in that class
   - This step depends on the scheduler algorithm of the specific class, thus the Scheduler Core calls the `next_task()` method of the scheduling class.
   - The `next_task()` returns the task descriptor of the next task to run or NULL if there are no runnable tasks in the class
3. If there are no runnable task in the class, the Scheduler Core moves to the next class
4. The Scheduler Core repeats these steps until it finds a runnable task to assign the CPU to.

Notice that the Scheduler Core cannot fail in its research, for the idle class has always one runnable process: the idle process

HPCSched



We introduced in the Linux scheduler framework a new Scheduling Class called HPC class

- It is supposed to handle HPC applications with a scheduler specifically tailored for improving HPC metrics (performance, power, etc.)
- At this stage, the HPC runs properly an HPC application and tries to load balance (when it is necessary) the application using the POWER5 features
- In the current schema, HPC applications have higher priority than normal processes

HPCSched components

There are 3 independent components in the HPC scheduler

1. Policy: the scheduler algorithm itself that selects the next task to be executed
2. Metrics and Heuristics: a formula that translates the information provided by the metrics to increment/decrement of the task priority
3. Applying mechanism: the functions required to read/set the POWER5 thread priority (architecture dependent). This components comes from the previous HMT patch

Every component is almost independent from the others and mainly architecture independent (only the functions required to read/set the thread priority are architecture-dependent).

HPCSched scheduling policy

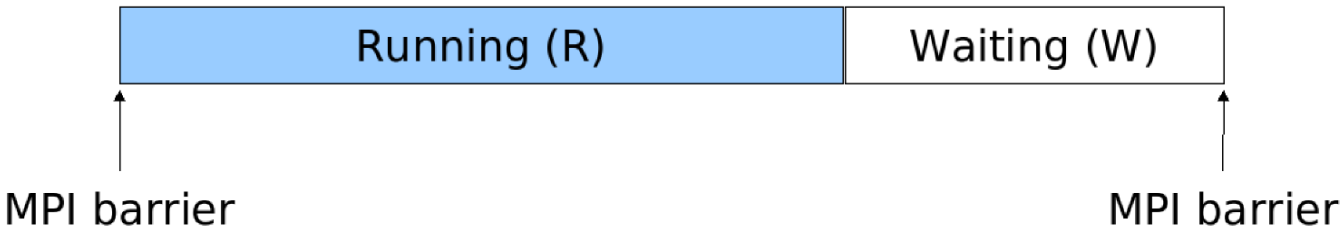
The scheduler policy is the algorithm used by the HPC scheduler to select the next task to run from the run-queue list
We introduced a new scheduler policy (`SCHED_PC`)

1. This policy represents HPC class applications
2. The new policy can be set by means of a `sched_setscheduler()` system call

We assume the following hypothesis:

1. There is usually one HPC process per CPU (safely, few processes)
2. An HPC process either computes or wait for an message from its neighbours

HPCSched balancing heuristics

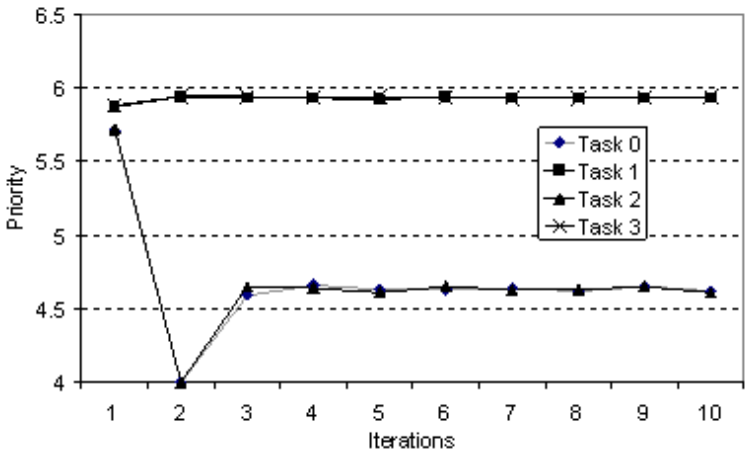


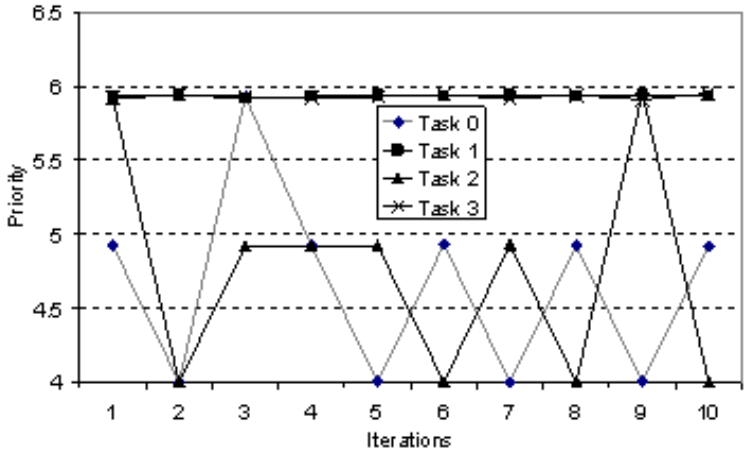
For each MPI iteration, we compute the running and waiting time. These information are used by the balancing heuristic. We tried two heuristics:

Uniform This heuristic uses the global utilization ratio of a task in a HPC application

Adaptive This heuristic uses the “recent history” of a task

Preliminary tests: MetBench

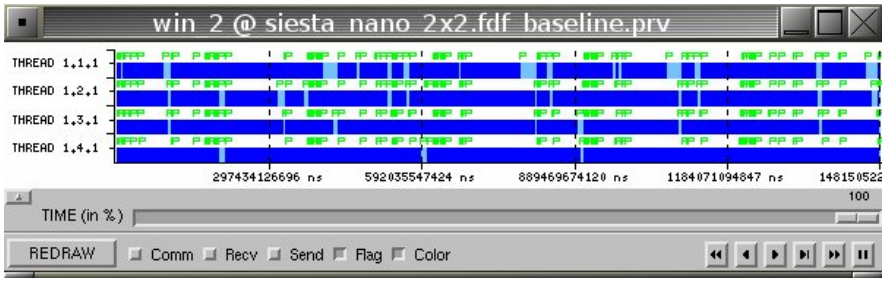


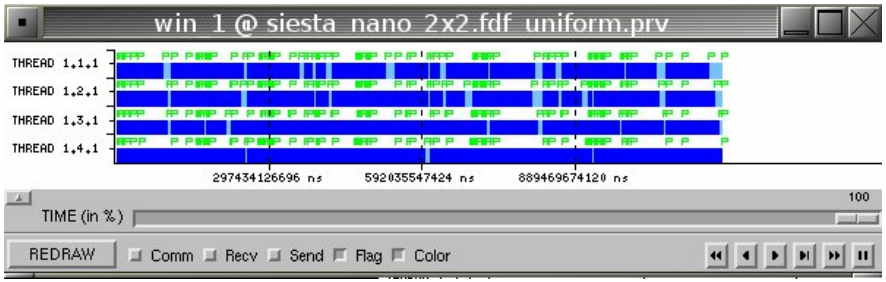


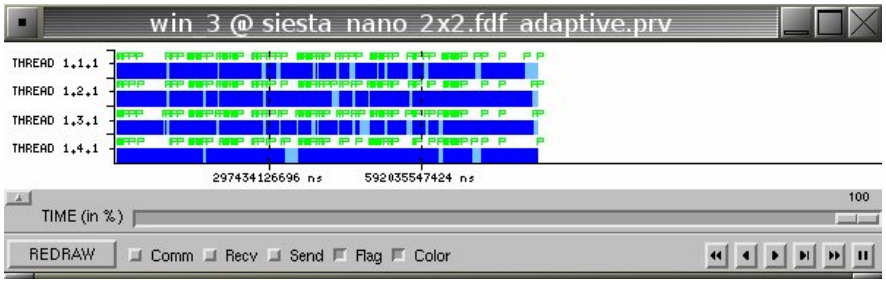
Both the Uniform and Adaptive heuristics work more or less well with MetBench:

- Imbalance reduced from 74.37% to 32.95%
- Execution time reduced from 86.05s to 75.47s (about 12% of improvement)
- The Uniform heuristic quickly finds the stable state and remains there
- The adaptive heuristic oscillates between two different priorities

Preliminary tests: SIESTA







Neither the Uniform nor the Adaptive heuristic is able to significantly reduce the imbalance.

Yet there is a big performance improvement:

- Uniform: 20.6%
- Adaptive: 44.67%

Outline

Conclusion

- Supercomputers are becoming “common” in many academic and industrial institutions
- Linux is the main OS used for Supercomputing even if it is not designed for HPC
- Some applications are imbalanced behavior (because of internal or external reasons). This kind of problem is usually “solved” by hand
- Linux offers a great opportunity to customize the OS for HPC and take full advantage of the underneath hardware
- In this study we showed how allowing software to control the amount of shared resources assigned to each task may improve the performance of HPC applications.

Questions?

roberto.gioiosa@bsc.es, gioiosa@sprg.uniroma2.it
http://www.sprg.uniroma2.it/home/gioiosa/