HPPS Course Project 2011/2012

POLITECNICO DI MILANO

Area: Operating Systems

Subject: Controlling system temperature with the idle cycles injection technique on BFS scheduler

Andrea Mambretti **andrea2.mambretti@mail.polimi.it**

WHY DOING THIS PROJECT?

The aim is simple:

Modify the Linux BFS scheduler in order to provide an easy way to keep under control temperature and/or throughput of a machine, using the idle cycles injection technique.

But, what is the idle injection technique?

It's the habit of putting into execution an idle cycle instead of the next valid process scheduled, with the intention of having the processor doing nothing.

Doing nothing requires less power consumpion than execute.

So we want to control the amount of this “nothing”, to reduce the warm in an overheated system.

IS HEAT A REAL PROBLEM?

For sure it is!!!!

Components, mainly enbedded ones like CPUs and GPUs, are really susceptible to malfunction if overheated.

In worst cases, they can also break.

For example Mambro's PC... He has to use his wallet to lift it in order to make funs working properly and not to burn himself touching the case!!!!

Working at higher frequencies and subjected to an increased workload (due to improvements such as ILP, TLP, SMT, etc.), today's CPUs have to face serious thermal problems.

And here we are with our project...

BFS, THE BRAIN FUCK SCHEDULER

Our kernel modification works on a specific scheduler, the BFS. This is an unofficial scheduler, written by Con Kolivas as an alternative to the CFS.

It uses one single system-wide runqueue, containing all non-running tasks. To choose next scheduled process, it implements an earliest effective virtual deadline policy.

A virtual deadline is the ideal time that any two tasks with the same niceness will have to wait before running on the CPU.

Each task has its own assigned.

The next scheduled process will be the one with the earliest virtual deadline.

(A typical BFS user...)

WHAT DO WE DO WITH OUR CODE?

Our goal is to provide a mechanism with which to control the number and frequency of idle cycles injected in the system.

In this way, we want to mantain the best working conditions in terms of heating, even if the machine is overloaded.

We mainly worked on the *schedule()* function, the one that chooses the next process to execute.

The modification is made for forcing an idle every X calls of the *schedule()* function, or every Y times that a specific process or thread is chosen for being executed.

This change is done before the *context_switch()* function is called.

YOU HAVE FUNCTIONS...

How do we do that?

We provide two functions...

The first one works globally on the machine and can be used to take under control the temperature an the throughput of the whole system.

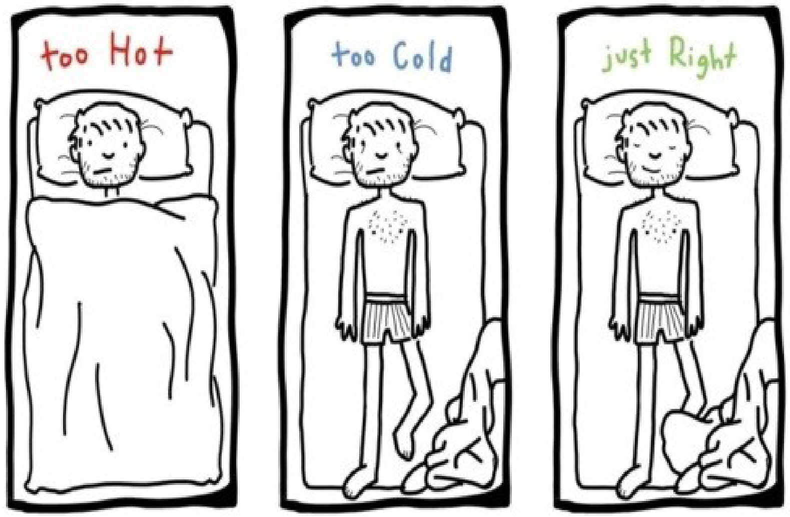
The second one works on a defined process/thread, using its pid (tid) for the identification.

With this second function, we are also able to control the throughput of that specific process/thread (even more than one at a time).

...AND INTERFACES!!!

An user, or even the machine itself, is provided with two interfaces.

Working both through the *procfs* file, they can be used to control the injection of idle cycles.



While a human user can choose him or herself the idle's amount, the Monitor Program we wrote as an example has a simple heuristic that changes it basing on the current temperature (the higher it is, the greater is the number of cycles injected) or on the throughput (in case of the third test, we'll see it).

LET'S TALK ABOUT TESTS

We made 3 demonstrations:

1. Global Temperature Management
2. Throughput Management of Specific Threads
3. Global Throughput Management

The aim of each one is explained after.

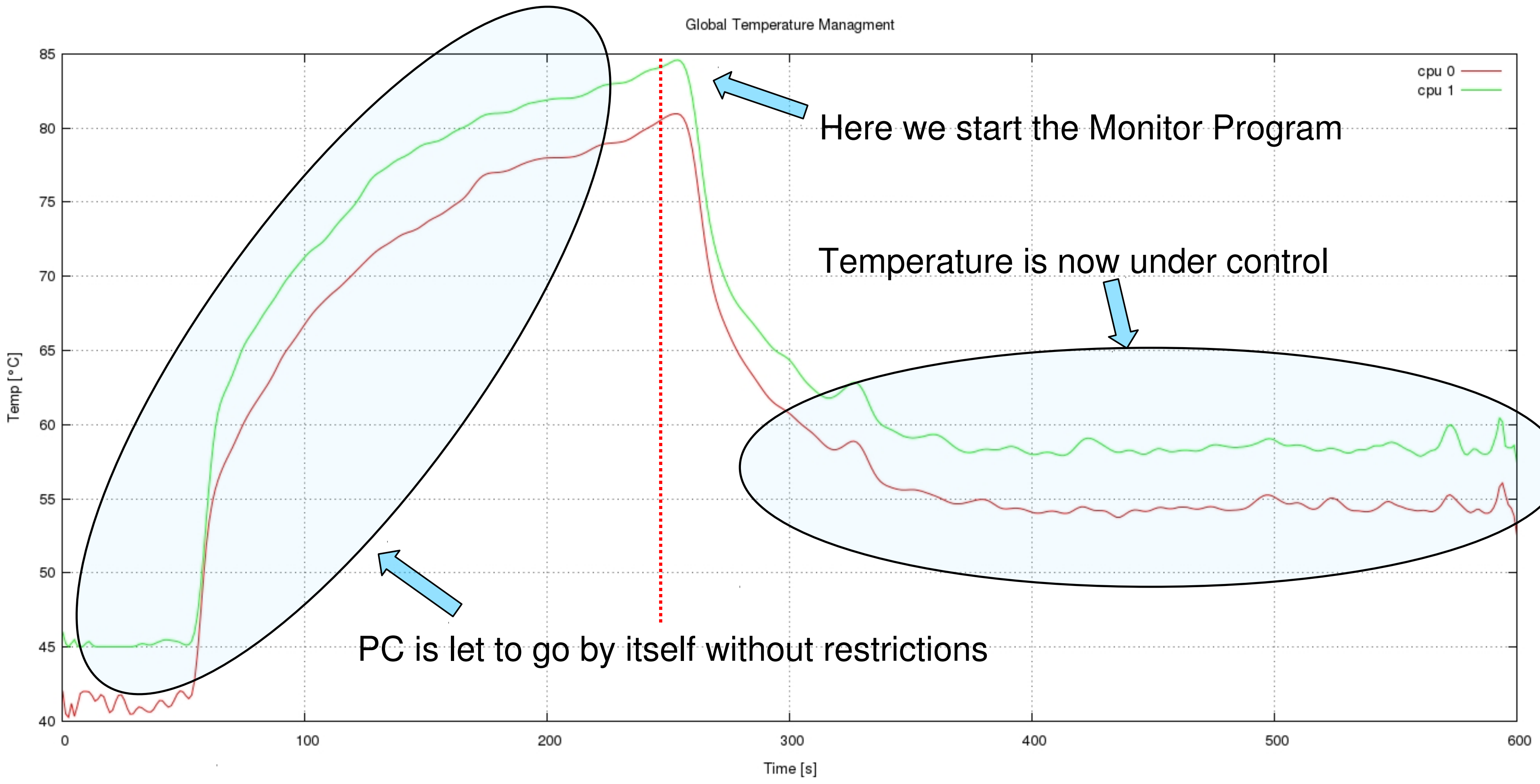
The testing environment is a Dell Laptop xps m1330 version with an Intel Core2 Duo T7100.

We installed Gentoo on it, with a patched kernel v3.2.6 for running the HRM System and BFS scheduler.

We used HRM to see which is the throughput of our threads and the lm_sensors Module to monitor the temperatures.

FIRST TEST – Global Temperature Management

Here we want to show how a simple monitor can manage the temperature of an overloaded laptop. Result is below:



UPON THE FIRST TEST...

The test was made while the machine was running both processes and threads.

We first let the machine work by itself and then we started our Monitor Program.

We established that:

- Without any control the temperature rises
- When the Monitor starts, we have a sudden decrease of heat, really strong in the very first seconds (due to the higher temperature that implies a greater number of idles injected)
- We empirically saw that the system temperature tends to stabilize at about 55-60°C

Go ahead with the second test...

MEANING OF THE SECOND TEST

Now we want to prove that our system is able to reduce the throughput of a specific thread, without affecting other thread’s execution (we chose threads instead of processes for no reason, nothing would have changed).

We set a process running two threads, one on each core. Thread1 on CPU0, Thread2 on CPU1. After a while, we start injecting some idle cycles on Thread2.

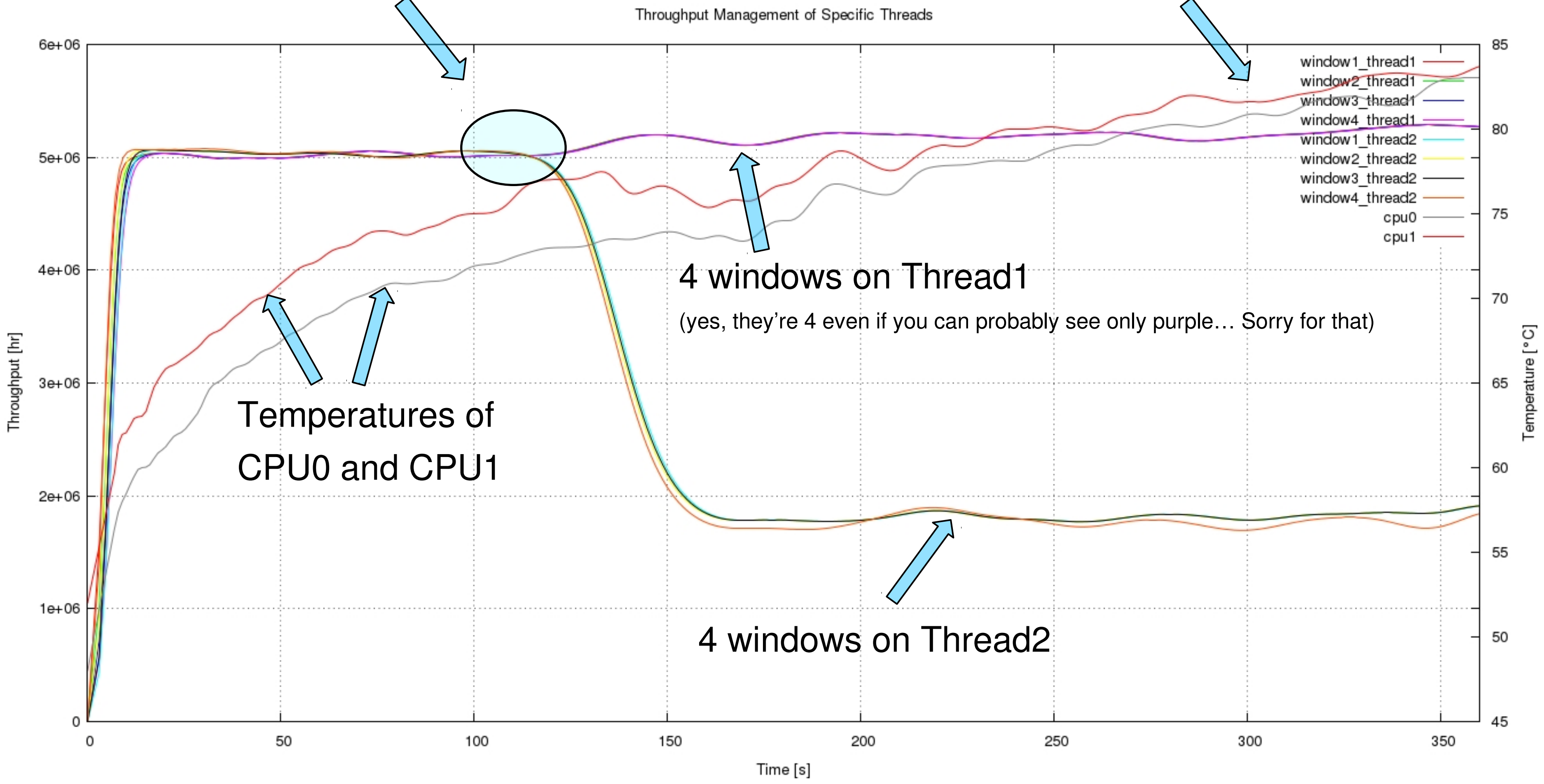
For this test-case we used the HRM System, properly modified, which attaches 4 windows to each thread to monitor their throughput.

What follows is the graph that we obtained…

SECOND TEST – Thrughput Management of Threads

Here we start idling Thread2 on CPU1

Getting closer



4 windows on Thread1

(yes, they're 4 even if you can probably see only purple… Sorry for that)

Temperatures of CPU0 and CPU1

4 windows on Thread2

UPON THIS SECOND TEST...

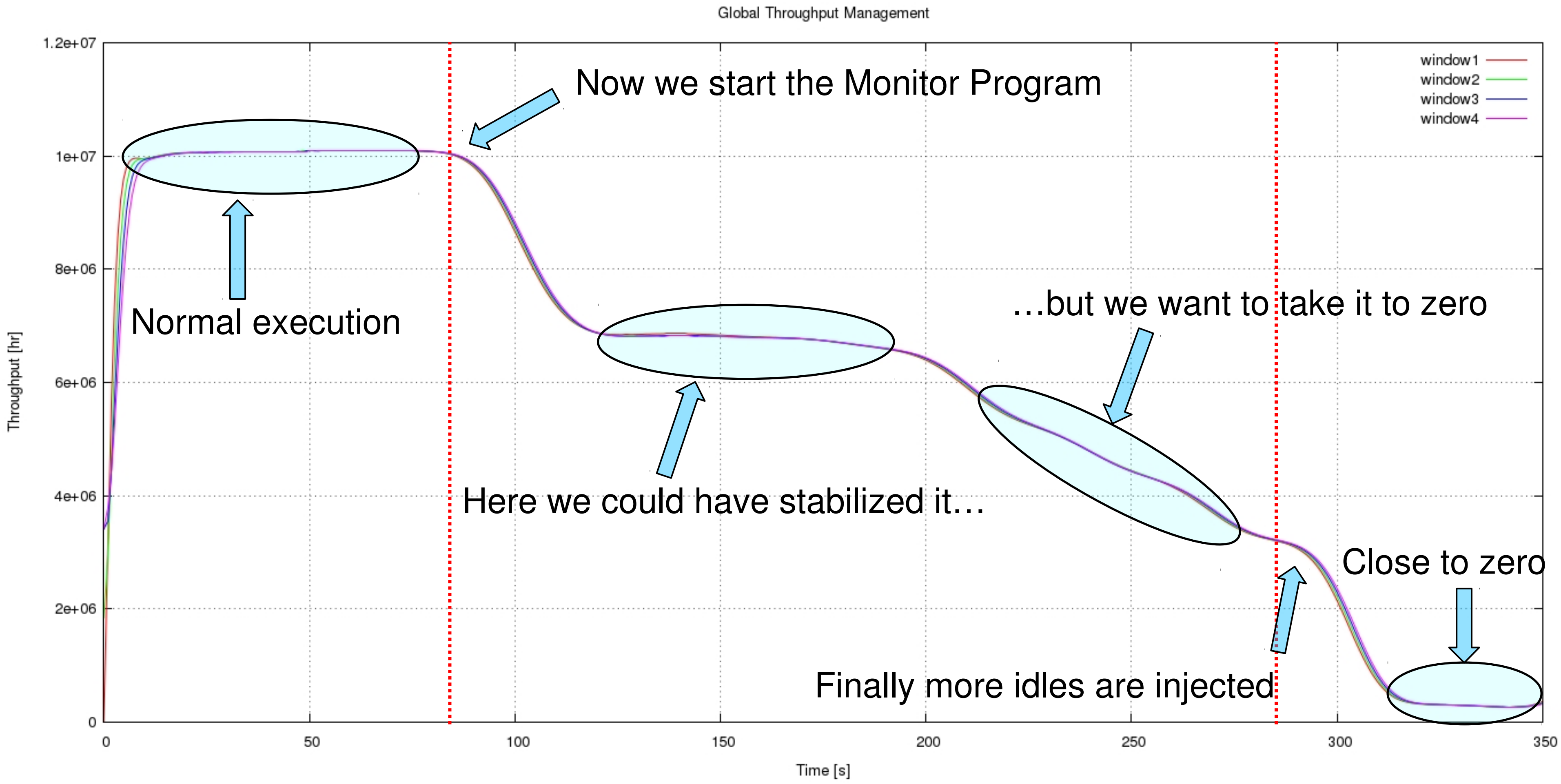
So we realized that:

- If a thread is not touched, its throughput will stabilize at a point
- With our system we can actually inject idles on a single thread, changing its performances
- If we idle a thread, others won't be affected and keep executing as usual
- Idling just one thread on a specific core, we saw a little variation in its CPU's temperature, which became closer to the other CPU's one

Go ahead with the last test…

THIRD TEST – Global Throughput Management

Lastly, we want to verify that we can significantly influence the whole system throughput, with a Monitor Program. Here's the graph:



UPON THIS THIRD TEST...

The Monitor Program used is like the one of the first test, but with a significant modification: it always injects idles basing on the current machine performance, within a range. But this time, we decided to increase the number of idle cycles executed if the throughput is low, to show that we can bring it near zero.

Results state that:

- If no Monitor Program is running, the throughput is stable
- If we start to inject, the system performance will be affected, proportionally to the number of idle cycles executed
- We are able to inject so much idles to even block a machine

POSSIBLE FUTURE WORKS

Those results gave us some ideas for new improvements:

- Developing a Monitor which takes into account the number of processes/threads running, not only temperature or throughput
- Taking into account the process namespace for a finer grain (problems with openvz)
- Providing the possibilty to idle a whole single core and influence only its temperature and throughput (better in machines with many core on different chips, to see an actual change in terms of heating)
- Implementing a way to keep a core/process/thread throughput in a range, calculated dinamically from the current system state
- Adapting our system to work with other scheduler versions

SPECIAL THANKS

For having helped us doing this project, we'll give a special thank to:

- Con Kolivas, that wrote the BFS…

(Without you we would have been able to do nothing)

- Our tutor Eng. Davide Basilio Bartolini

(who provided us the HRM System and Helped us during the whole project)

- NECST Lab users

(for supporting us morally and physically)

(we apologize for any grammatical mistake made in this presentation…)

THE END!!!

Q&A???