A Computational test facility for distributed analysis of gravitational wave signals

P. Amico, **L. Bosi**, C. Cattuto, L. Gammaitoni, F. Travasso, **M.Punturo**, H. Vocca

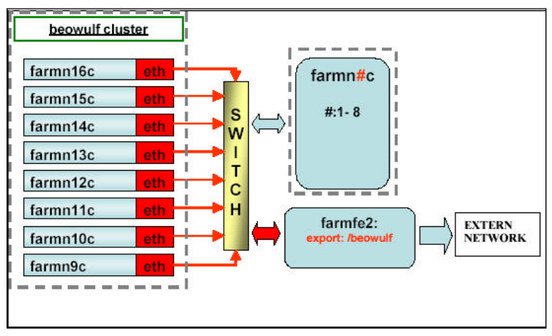
Istituto Nazionale di Fisica Nucleare – Sezione di Perugia, Italy
Virgo Experiment

The Problem

The gravitational signal coming from a system of two coalescing NS stars is the best candidate for a detection in a terrestrial GW interferometric detector. Assuming that the signal can be described by its 2nd order PN approximation, it is possible to implement a Wiener (or matched) filtering strategy for the detection.
The good seismic attenuation in the Virgo detector allows to take in account the low frequency part [1] of the signal coming from such as stellar system, relatively far from the coalescence. This possibility improves the detection probability in Virgo, but dramatically increases the computational needs. In fact, to apply the Wiener detection strategy, with only a 3% of SNR loss, it is necessary to build a large grid of templates (more than 45000) where, for the lightest pairs of stars, there are templates of more than 100 seconds long [2][3].
Since matching against the 45000 templates must be performed in time with respect to the data stream coming from the interferometer, a considerable computational power is needed . Hence, a distributed computing philosophy, based on a beowulf (http://www.beowulf.org) cluster, was adopted.

Hardware and Software platform

A first facility was realized in Perugia in the past using a cluster of dual Pentium III processor. The system consists of ten 866Hz rack mounted dual processor machines that operate as computational nodes, plus two machines (Atlon and P4 based) that operate as fileservers. The nodes have a fast-Ethernet interconnect.
An upgraded version of this machine has been recently configured at the Virgo site, deploying DUAL-Xeon (1.7 GHz) nodes with a Gigabit Ethernet interconnect.

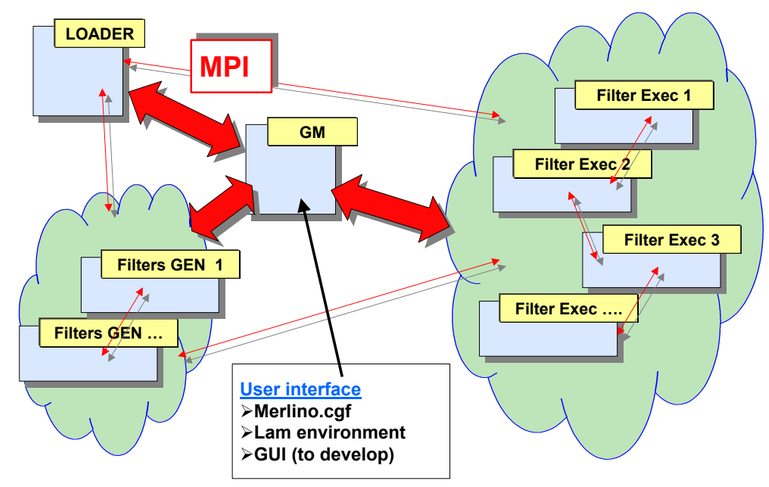


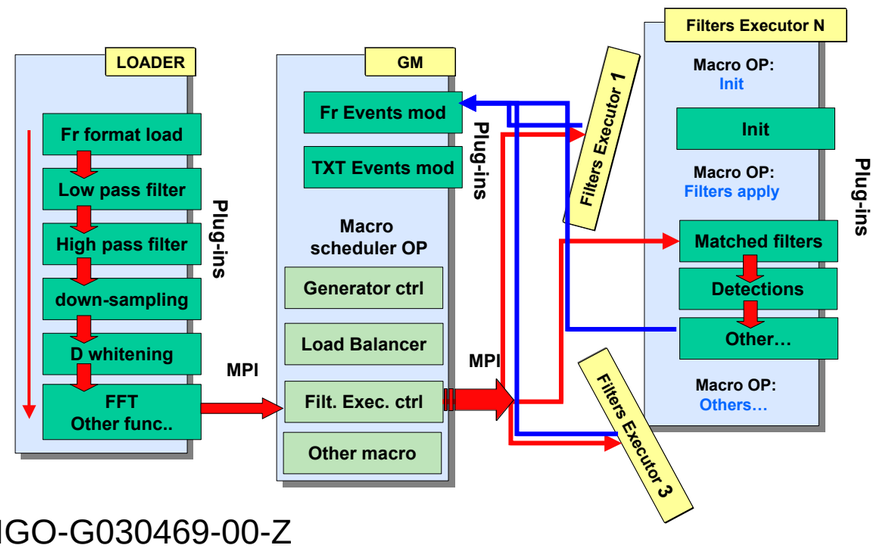
The environment in the new facility is similar to the Perugia scheme:
•GNU/Linux OS (RedHat-based)
•GNU compiler toolchain (gcc-3.2)
•LAM-MPI 6.5.9
•FFTW 2.1.3 - 3.0
•Siglib 6, Fr v6r07, Frv v4r02,…

- **Root filesystem over NFS, read-only mounted**
  - ***. root filesystem based on a customized RedHat distribution***
  - ***. root filesystem exported and managed using the standard RedHat tools***
- **Single nodes image**
  - ***. Single node image stored on the server (farmn9)***
  - ***. Any change of the images is automaticall visible to the cluster nodes***
- **NFS-shared /beowulf /virgoApp /virgoDev filesystems**
  - ***. /beowulf contains the system libraries for the Beowulf, like lam-mpi, fftw3***
  - ***. /virgoApp , /virgoDev hold the standard Virgo software distribution***
- **NIS authentication with VIRGO standard username/passwd**

The matching engine

The detection software environment (Merlino) is composed of four main processes communicating through MPI primitives, in a master-slave configuration:
•**Controller**: one process for the whole cluster (currently linked in the group manager)
•**Loader**: one process for the whole cluster
•**Group Manager**: one for each active master in the cluster
•**Filter Executor**: one or more processes for each GM



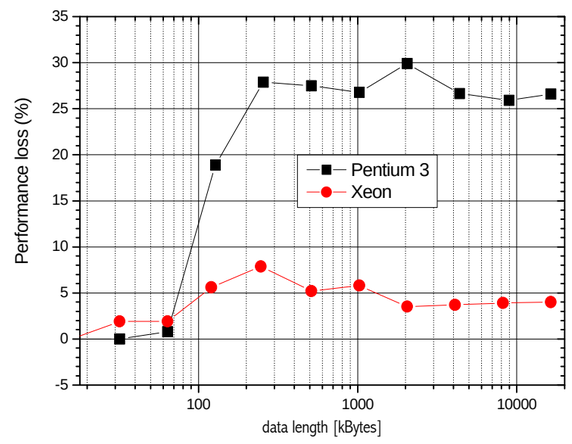


Performances of the system

All the templates of the grid are kept (in the frequency-domain representation) in memory after the generation. To evaluate the difference in performance of the old and new clusters it is necessary to discriminate the different contributions to the CPU load. The matched filter can be seen as composed of three main steps:
•Array product
•Inverse real Fourier transform
•Detection of the largest peak in an array of data
In the following table, the total computation time and the fractional contributions of the different sub-tasks are reported:

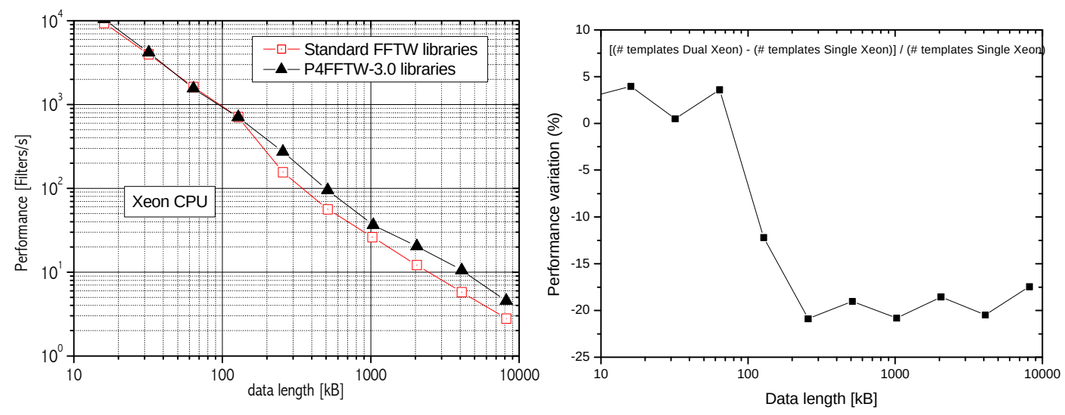
	Total time (ms)	Array product	rFFT	Max. detection
PIII	238	13.4%	75.3%	11.3%
Xeon	181	4.1%	94.8%	1.1%
Xeon (FFTW-GEL)	106	8.0%	88.0%	4.0%

The anomalous weight of the rFFT algorithm in the Xeon architecture is due to the lack of specific Xeon optimizations in the FFTW 2.1.3 code. With the P4 specific version of that library the CPU load sharing between the different part of the matching filter is recuperated.
Dual CPU nodes, while affording higher density, can have memory bottleneck issues:

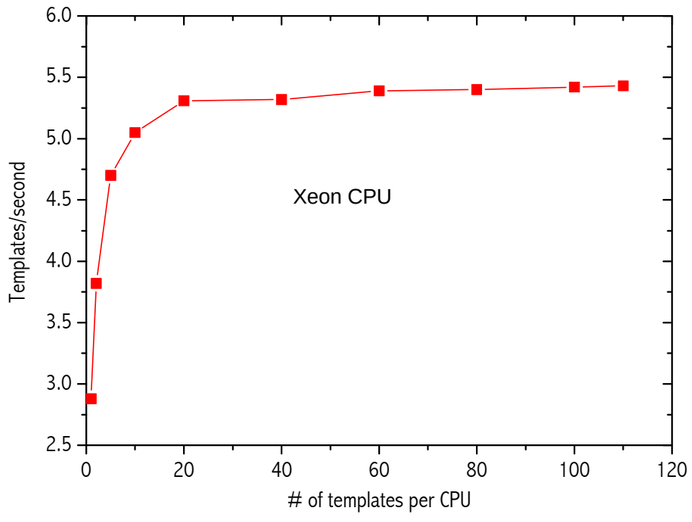


Concurrent memory access by the two CPUs places high demands on the memory bus, effectively starving the older Pentium-III architecture. Xeon-based nodes, on the contrary, are able to gracefully handle the load.

The number of matching filters that a Xeon machine is able to compute vs the dimension of the data buffer is reported in these figures. Using 100s templates and taking in account the buffer grown due to the zero-padding technique, the expected dimension of the buffer (2kHz sampling rate) is 4MB (corresponding to 256s). Then the measured performances are about 5 templates/second per CPU (10 using FFTW-3.0)



If the system should be used in a “in time” philosophy, the computational window to perform all our 45000 filters, is about 150s (256s of each data buffer minus 100s of the overlap between buffers); in this time interval each CPU is able to evaluate 825 templates (in single CPU configuration) or 726 templates (in double CPU configuration) using the older version of FFTW. To keep in memory such number of templates each CPU should have available 5GB of memory. Despite the smaller amount of memory available in the small scale facility, the system is able to reach the plateau of 5 templates per second expected using the FTTW 2.1.3:



Hence, in Virgo, to perform an *in time* analysis of the coalescing binaries signal, with PN2 approximation of the templates, neglecting any spin effect, starting from about 30Hz, a beowulf cluster of about 60 P4 machines is required.

References

- M.Punturo, “The VIRGO sensitivity curve”, VIR-NOT-PER-1390-51, Virgo internal note (2001)
- P. Canitrot et al., “Computational costs for coalescing binaries detection in VIRGO using matched filters”, VIR-NOT-PIS-1390-149, Virgo internal note (2000).
- P. Amico et al., Computer Physics Communications 153 (2003), 179-189