Institute of Physics Publishing
doi:10.1088/1742-6596/32/1/007

A parallel in-time analysis system for Virgo.

F. Acernese[6], P. Amico[10], M. Alshourbagy[11], S. Aoudia[7], S. Avino[6], D. Babusci[4], G. Ballardin[2], F. Barone[6], L. Barsotti[11], M. Barsuglia[8], F. Beauville[1], S. Birindelli[11], M.A. Bizouard[8], C. Boccara[9], F. Bondu[7], L. Bosi[10], C. Bradaschia[11], S. Braccini[11], A. Brillet[7], V. Brisson[8], L. Brocco[12], D. Buskulic[1], E. Calloni[6], E. Campagna[3], F. Cavalier[8], R. Cavalieri[2], G. Cella[11], E. Chassande-Mottin[7], C. Corda[11], A.-C. Clapson[8], F. Cleva[7], J.-P. Coulon[7], E. Cuoco[2], V. Dattilo[2], M. Davier[8], R. De Rosa[6], L. Di Fiore[6], A. Di Virgilio[11], B. Dujardin[7], A. Eleuteri[6], D. Enard[2], I. Ferrante[11], F. Fidecaro[11], I. Fiori[11], R. Flaminio[1,2], J.-D. Fournier[7], O.Francois[2] S. Frasca[12], F. Frasconi[2,11], A. Freise[2], L. Gammaitoni[10], A. Gennai[11], A. Giazotto[11], G. Giordano[4], L. Giordano[6], R. Gouaty[1], D. Grosjean[1], G. Guidi[3], S. Hebri[2], H. Heitmann[7], P. Hello[8], L. Holloway[2], S. Karkar[1], S. Kreckelbergh[8], P. La Penna[2], N. Letendre[1], M. Lorenzini[3], V. Loriette[9], M. Loupias[2], G. Losurdo[3], J.-M. Mackowski[5], E. Majorana[12], C. N. Man[7], M. Mantovani[11], F. Marchesoni[10], F. Marion[1], J. Marque[2], F. Martelli[3], A. Masserot[1], M. Mazzoni[3], L. Milano[6], C. Moins[2], J. Moreau[9], N. Morgado[5], B. Mours[1], A. Pai[12], C. Palomba[12], F. Paoletti[2,11], S. Pardi[6], A. Pasqualetti[2], R. Passaquieti[11], D. Passuello[11], B. Perniola[3], F. Piergiovanni[3], L. Pinard[5], R. Poggiani[11], M. Punturo[10], P. Puppo[12], K. Qipiani[6], P. Rapagnani[12], V. Reita[9], A. Remillieux[5], F. Ricci[12], I. Ricciardi[6], P. Ruggi [2], G. Russo[6], S. Solimeno[6], A. Spallicci[7], R. Stanga[3], R. Taddei[2], M. Tonelli[11], A. Toncelli[11], E. Tournefier[1], F. Travasso[10], G. Vajente [11], D. Verkindt[1], F. Vetrano[3], A. Viceré[3], J.-Y. Vinet[7], H. Vocca[10], M. Yvert[1] and Z.Zhang[2]

[1]Laboratoire d'Annecy-le-Vieux de Physique des Particules, Annecy-le-Vieux, France;
[2]European Gravitational Observatory (EGO), Cascina (Pi), Italia;
[3]INFN, Sezione di Firenze/Urbino, Sesto Fiorentino, and/or Università di Firenze, and/or Università di Urbino, Italia;
[4]INFN, Laboratori Nazionali di Frascati, Frascati (Rm), Italia;
[5]LMA, Villeurbanne, Lyon, France;
[6]INFN, sezione di Napoli and/or Università di Napoli "Federico II" Complesso Universitario di Monte S.Angelo, and/or Università di Salerno, Fisciano (Sa), Italia;
[7]Departement Artemis – Observatoire de la Côte d'Azur, BP 42209 06304 Nice, Cedex 4, France;
[8]Laboratoire de l'Accélérateur Linéaire (LAL), IN2P3/CNRS-Univ. de Paris-Sud, Orsay, France;
[9]ESPCI, Paris, France;
[10]INFN, Sezione di Perugia and/or Università di Perugia, Perugia, Italia;
[11]INFN, Sezione di Pisa and/or Università di Pisa, Pisa, Italia;
[12]INFN, Sezione di Roma and/or Università "La Sapienza", Roma, Italia.

E-mail: leone.bosi@pg.infn.it

Abstract. The interferometric gravitational wave detector Virgo is currently completing its commissioning phase and it is close to start scientific observations. Among the signals to be searched for, those emitted

by coalescing binary systems are particularly promising and require a considerable computational effort to optimally search the parameter space. The Virgo collaboration has decided to implement an on-line analysis strategy capable of processing the interferometer data in-time. In this communication we present a component of the analysis pipeline, a parallel computing system based on the Message Passing Interface (MPI). We describe its capabilities, underlining its strength and flexibility, and we illustrate its relation with the other components of the pipeline. The on-line analysis chain, including the presented parallel system, has been run for the first time successfully during the Virgo commissioning run C5 in December 2nd to December 6th 2004[1].

1. Introduction

Virgo[1] is a detector built to observe gravitational waves coming from various sources. The neutron stars coalescing binaries are among the best candidates. To detect the signal emitted by them we plan to use the matched filter algorithm, comparing the data acquired with a bank of reference signal called templates; as described in [2] [3][4] this is an heavy computationally demanding technique. One of the main goal for Virgo is the realization of a reliable real time observation strategy in order to use the interferometer as a Gravitational Waves observatory. At this aim we carefully designed the computational strategy by addressing the size of the problem, the computational power required and the constrains due to the in-time condition. In the paper we present a particular solution implemented in Virgo; this is based on a parallel-distributed applications environment, the **Di**stributed **S**ignal **A**nalyzer (DiSA) [3], known as *Merlino*. This framework is composed by several processes communicating via MPI [5] with a SIMD[1] like logical architecture. Merlino distributes and controls user algorithms and data and is based on a Beowulf cluster of PCs.

This software device has been tested in the online chain during the Virgo commissioning engineering run C5 in the period 2nd-6th December 2004. There are also others in-time analysis code to perform coalescing binaries detection, see e.g. MBTA[6].

2. Hardware and software environment

At Virgo site, a Beowulf cluster, dedicated to the online detection activity is installed. This system is based on Dual Opteron Processor 2.2GHz model 248; Opteron processor provides very good performance on concurrent processing and IO memory operations [7]. The operating system environment and the software libraries installed are reported below:

- Linux Red Hat 3.4.x distribution
- Kernel 2.6.8 (64bit)
- gcc 3.4.1,
- MPI (lam-7.1.1),
- FFTW 3.0.1
- others Virgo packages

Merlino framework processes run on this cluster, using 20 nodes (40 processors).

2.1. Merlino main goals

Merlino is composed by six different processes types (Fig. 2), communicating through message passing interface primitives (MPI). The aim of this plan is to realize something analogous to a programmable and distributed digital signal processing.
The goals of this project can be summarized:

[1] simple instruction multiple data

(i) Realize a parallel signal analyzer, to optimize this type of problems: single data versus multiple data [2].

(ii) Optimized design to reduce data handling latency and gain performance: in-time analysis

(iii) Portability and scalability

(iv) Use of standard library

(v) General filtering algorithms (using dynamic library loaded at run-time)

(vi) General filters parameterization

(vii) Hide the parallel architecture to the user

(viii) Provide to the user a single control point to start/manage/stop the environment[3]

(ix) Provide to the user a simple way to customize the filters algorithms, using the plug-ins philosophy[4]

2.2. Merlino processes composition

Regarding the firsts versions [2] [3] [8] [9], Merlino (fig. 2) is now upgraded. There are two new processes, one to interface the VIRGO online, the **FdMPIServer**, and the other to improve the output data handling like results and internal messaging, the **MSGServer**.

Some of these processes are present as single instances inside the environment, these are: FdMPIServer, MSGServer, **loader** and the **GroupManager**; others are present as multiple instances of the same process, like the **fbuild** and **fproc** processes.

Following we present a brief description of each:

- **GroupManager(GM)** process is the manager and scheduler of the environment. It serializes opportunely and in the right order the logic flow and processes and distributes the real-time user commands.
- **loader(LD)** process loads data and performs pre-processing activity and data conditioning like PSD estimation, whitening initialization, double whitening[10]. After that input data are sent to the GroupManager and to the fproc processes (fig. 2) for the processing.
- **fbuild** processes afterwards build filters data, starting from: filter generator code (loaded via plug-ins) and filters definition data(ASCII file), both written and defined by the user. Each filter[5] is transfered to the fproc process memory (fig. 1)[6]. In the environment each fbuild process is linked to an fproc process; this mean that the number of fbuild processes started in the Merlino framework is equal to the number of the fproc processes.
- **fproc** processes apply the filters stored in their memory to the incoming data from loader/GroupManager. At this aim they use the logic written in the user processing algorithms loaded from the plug-ins.
- **FdMPIServer** is a stand alone server. It interfaces the Cm (Communication manager) with MPI and permits to connect Merlino to the online chain. Cm is the data communication protocol used in Virgo to exchange messages via network between processes.
- **msgServer** is a stand alone server that gathers the outputs data coming from each fproc processes and performs some post processing operation like: sorting events in-time, clusterizing and packing in frame format. Hence, the results can be sent to the online chain through Cm communication.

[2] single data stream is processed using a bench of filters.

[3] e.g. the user effort to run Merlino with 1000 CPUs or 1 CPUs is the same

[4] To customize Merlino is possible to: change the merlino.cfg configuration file, wrap the user (serial) algorithms with a specific API and plug at run-time these into Merlino using the shared library. It is also possible to send run-time commands to control the framework.

[5] In the matched filter schema, the filters are the templates

[6] Filters are stored in single precision to gain space

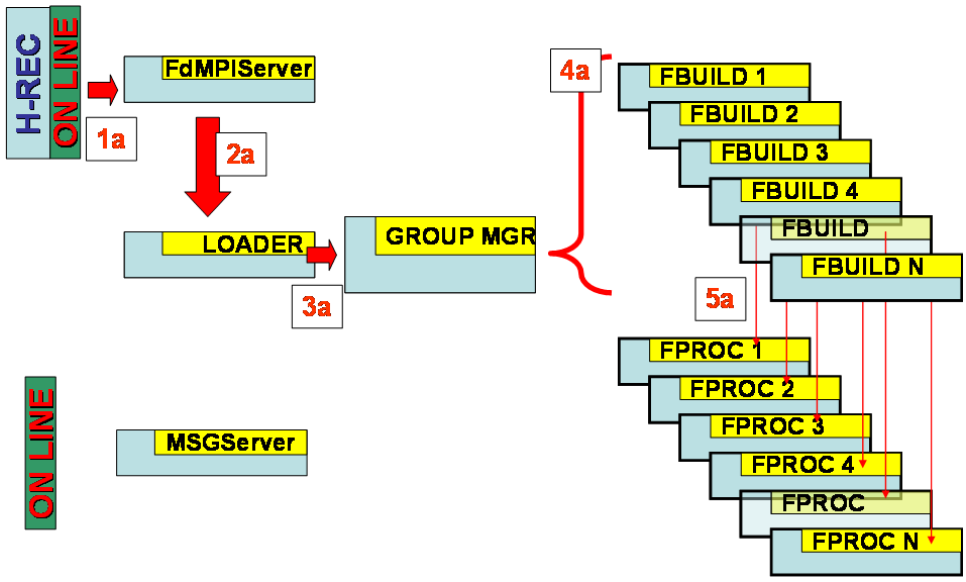

Figure 1. This figure shows the Merlino processes environment and the relation with the ONLINE chain. These are also the steps describing the generation phase

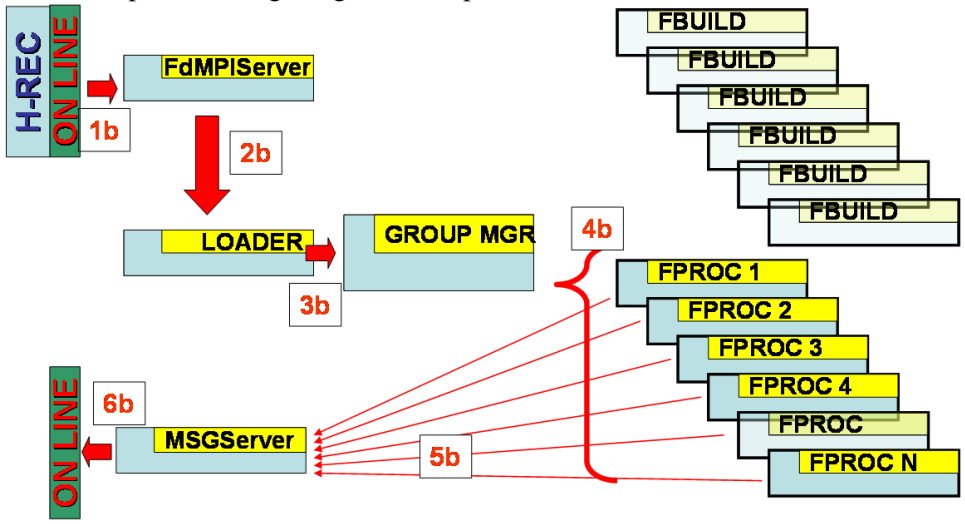

Figure 2. This figure shows the steps, describing the generation phase. Merlino read data from the online, using the FdMPIServer process. LD process perform pre-processing and data conditioning on input data, GM sends these data to the fproc processes with a broadcast communication. Results are sent to the msgServer.

2.3. Merlino scheduler: generation phase and processing phase

There are two main phases defined inside Merlino, the generation of the filters and the effective data processing. [7]

The **generation phase** is naively described in fig. 1. It is possible to highlight 5 main steps:

1a FdMPIServer loads data for the online chain after the h-reconstruction [11][12]. If necessary data

[7] This is more complex, because each phase is like a macro operation, composed by micro operations or sub-tasks. Each of these can be called autonomously also at run-time. Follow that the sequence of the micro-operation define each phase, because logically ordered to produce the required logic behavior.

are downsampled and Butterworth filtered.

2a LD loads data from FdMPIServer process. It performs some pre-processing activity, one of these is the power spectrum (PSD) estimation (AR model)[10].

3a PSD and others informations are distributed to the fbuild and fproc processes.

4a GM sends the filters definition data to the fbuilds; there, these data are processed with the generator plug-ins in order to build the filters.

5a Filters data are sent to the fproc processes memory.

The generation phase ends when all the filters defined by the user in an ASCII table are generated.

When merlino uses the coalescing binaries detection plug-ins, it generates and normalizes the templates and produces and stores the χ^2 [13] bands information for each of them.

The **processing phase** is naively described in fig. 2. It is possible to highlight 6 main steps:

1b FdMPIServer loads data for the online chain after the h-reconstruction. If necessary data are downsampled and Butterworth filtered.

2b LD loads data from FdMPIServer process. It performs pre-processing activity, such as Double Whitening filtering.

3b GM asynchronously loads data from LD, and packs some information about the data for the next step.

4b GM sends broadcasts data to the fproc processes. fproc processes elaborate the incoming data with the filters stored in memory, using the user processing algorithm loaded with the plug-ins.

5b The results generated by fproc processes are sent to the msgServer

6b msgServer gathers the output data, post processes and saves them

2.4. Merlino and the overlap-add method

To speed-up the convolution/correlation computation Merlino arranges data inside the framework in order to use the *overlap-add* procedure with the FFT correlation(if activated) [14] [15].

The overlap-add method decomposes the signal into simple components, processes each of the components in some useful way, and recombines the processed components into the final signal overlapping the results on each step.

This method permits not only to remove the problem of spurious wrap-around and periodicity effects [15] after an FFT convolution/correlation, but uses these effects to process the data like a continuous data stream. Thus we don't loose computed data segments and computation time.

2.5. Coalescing Binaries detection strategy

In the following a brief description of the strategies and methods implemented to perform the coalescing binaries signals detection will be presented.

In Fig. 3 are shown the logical steps involved inside the fproc processes with the coalescing binaries(CB) plug-in. This uses the matched filter algorithm to detect the signals and use the theoretical gravitational waves signals (templates) as filters. Each fproc process matches all the filters(templates), loaded in its memory, with the data coming from the (GM); the output is called correlator output. When the correlator output is higher than a defined threshold[8], fproc executes a χ^2 reconstruction over several frequency bands. After that, the event is returned to the msgServer. χ^2 can be used online as a second level trigger, comparing its value to another threshold to decide if send or not an event to the msgServer.

[8] Threshold can change at run-time

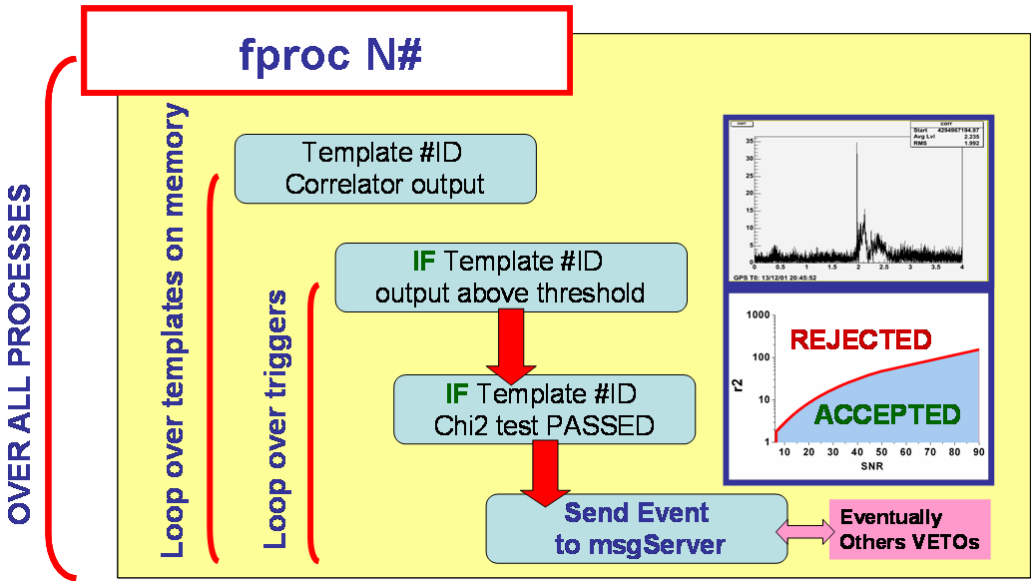

Figure 3. Coalescing binaries detection plug-ins involve the following operations inside the fproc processes. fproc processes loop the matched filter algorithms and χ^2 (two plots in the inset) over all the assigned templates. For each template, when the matched filter output is higher than a defined threshold, fproc executes χ^2 reconstruction over several frequency bands. This compares its value with another threshold and then decides if to accept or reject the event. If an event is accepted, it is sent to the msgServer.

The template bank is presently produced using the grid generator described in the reference [16] [17]; this provides an optimized template bank distribution using the stars masses range, detector noise PSD, frequency cut-in and the minimal match as input parameters. The gravitational waves templates are generated with the inspiral Virgo software package [18]. It provides the post-Newtonian and Effective One Body approximation to produce the required waveforms. Usually to analyze data, we use the $0th$ order post-Newtonian approximation in amplitude, and $2nd$ order post-Newtonian approximation in phase [19].

In orer to follow the online, Merlino controls the trigger rate and the time spent on each step, because the computation must end inside the time window established by the online demands; this window is long as the input data length.

The nonstationary and/or nongaussianity of the input data changes the matched filters trigger rate. This can produce a delay in the computation. To reduce this effect there are many possible ways: the threshold can change, following this rate; another strategy is to remove, when needed, the most computational demanding function, the χ^2, in order to store only all the triggers coming from the matched filter algorithms.

Another method that will be implemented and tested soon is to use a double layer templates grid. The idea is to have two layered grids. e.g. these joined together provide a 98% minimal match coverage of the parameters space and they are used in normal detector noise condition. If it happens that for a certain time the interferometer is noisier, an higher trigger rate is produced. In this case one layer can be removed in order to have less point to process (e.g.95%). This smaller grid is used until normal noise conditions are restored.

It is important also study how to manage changes of the template grid due to the Virgo sensitivity changes. In fact we need to maintain the minimal match coverage nearly constant, without losing performance. An interesting strategy will be implemented and tested soon: use a reference grid slowly updated, and follow the detector sensitivity, adding only few templates on the grid. These are placed in

grid regions where minimal match coverage is decreased due to Virgo sensitivity changes.

Triggers handling is one of the most critical problem, because the fproc processes produce many data that must be re-united and elaborated in-time. msgServer is the dedicated process studied to do that. It gathers the outputs data coming from each fproc processes and performs some post processing operations. It is obvious that the stress over this component depend on the number of fproc processes connected and more on the trigger rate. The msgServer operations are:

- sorting events, this operation is necessary in the following steps. The list of events stored in the msgServer memory is initially not ordered. This is due to the fproc processes communication, because they sends events autonomously and asynchronously.
- clusterization, reduce the number events clusterizing them in time.
- events reconstruction, reconstruct the informations about the physical events: GPS time, distance, masses and others.
- post-processing, Here is possible to introduce others post-processing operations as other veto algorithms [20] [21].
- packing, the events are packed in frame format or ASCII format and stored.

To gain in performance, we wrote msgServer as a multi-thread asynchronous MPI communicating process, and each operation has been optimized. Moreover it is possible to define more than one msgServer inside the same Merlino environment, in order to distribute the computational load [9]. The results can be sent to the online chain through Cm communication or stored on files.

3. Merlino and the C5 engineering run

Before describing Merlino test done during the C5 run, it is important to introduce the concept of **in-time factor**. This is the relevant parameter describing the performance of the in-time analysis. The in-time factor is the ratio between the time length of data and the time needed to process it.

3.1. The online engineering test

C5 has been a technical test to verify the Merlino on-line connections processes and to stress the code. During the Virgo Commissioning run C5 Merlino runs continuously for 7 days (Fig. 4) [10] and no malfunctioning was noticed. The run parameters used in this test were: grid size of **10481** templates (in order to stress the system, we set the minimal match to produce this high templates number. The ammount of used memory is equivalent to 150GB of RAM, allocated over 34CPUs), a starting frequency of 40Hz, sampling frequency of $4kHz$, χ^2 algorithm disabled and threshold of 6.5;

During the run the mean value of the in-time factor was **1.8**; in this configuration and with the actual detector noise, Merlino was able to online process data up to 1.8 time faster than the acquisition rate.

3.2. The off-line engineering test

The C5 off-line analysis [12] has been performed with 6677 templates, a starting frequency of 60Hz, sampling frequency of $4kHz$, minimal match of 98%, mass range $[0.9-10]M\odot$. Applying the matched filter alone, with 34 CPU the in-time factor is **4** (Fig. 5); instead adding the χ^2 test (15 bands) it decreases down to 3. The χ^2 is computed only on the correlators events over threshold, in this way the computation time follows the trigger rate. If the noise is quite stable this strategy provides good performance. It is possible to compute the χ^2 information over the whole correlators, using a different algorithm implementation. With this implementation the loss of performances is inversely correlated and independent to the number of bands.

[9] A single process can produce a time latency (bottleneck) in the computation and reduce the effective in-time window available

[10] it stopped once, because of an hardware crash of a cluster node

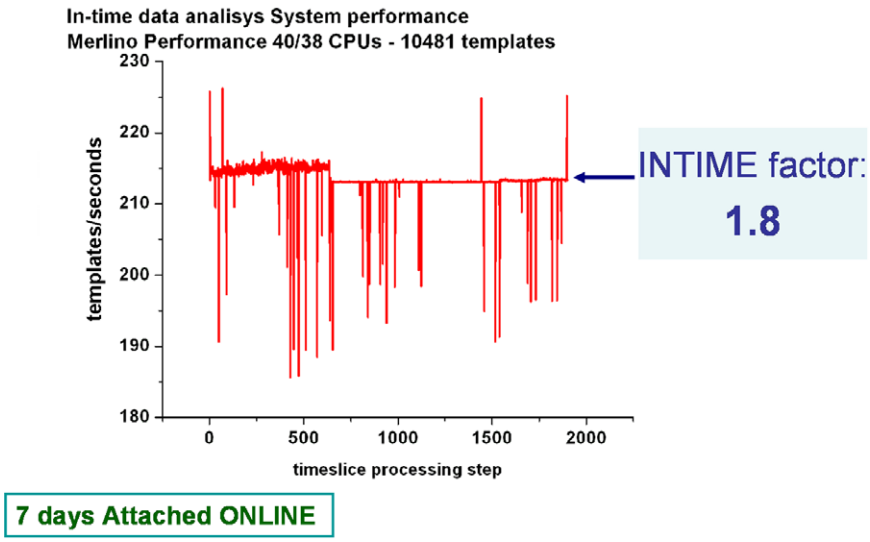

Figure 4. This plot shows Merlino performance during the Commissioning run C5. A single timeslice processing step is $\sim 300s$. In the vertical axis there is the number of templates per second processed by Merlino. It is visible the hardware crash of one cluster node at step 600.

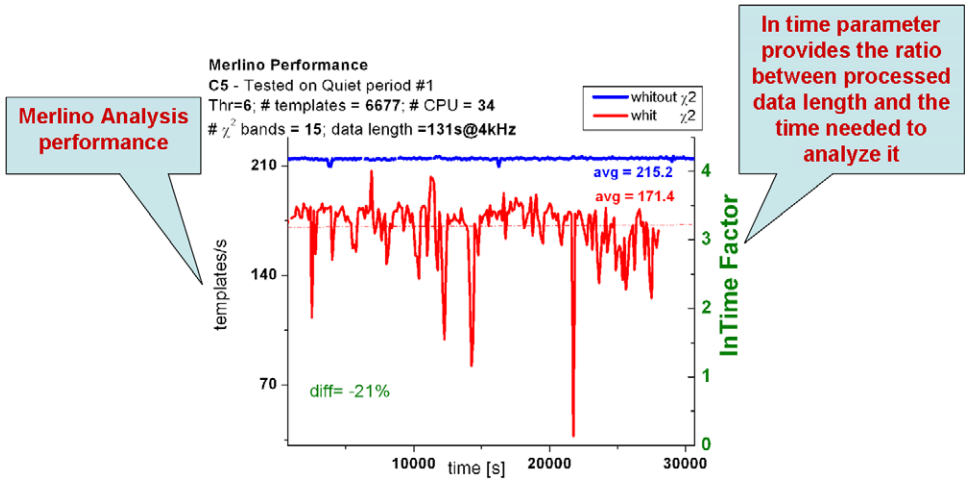

Figure 5. Applying the matched filter alone and 34 CPU, the in-time factor is **4**, instead with the χ^2 test enabled with 15 bands, we are able to be up to 3 time faster than the data acquisition. The performance difference is about 21%. The overall Merlino performances are 215.2 template/s in the first case and 171.4 template/s in the second case; These numbers are the number of templates processed by the Merlino per second.

4. Conclusions

In this paper we have presented the Distributed Signal Analyzer (DiSA) also know as Merlino project. It is one of the on-line code in the Virgo pipeline. Now it is employed for searching inspiralling binaries events using matched filter techniques and χ^2 test, but due to the plug-ins design, it can be programmed to run also others vetoes and detection algorithms. The C5 run allowed to perform some online and offline tests on the Merlino and on-line code [12].

C5 run noise was particularly challenging respect those found during the previous run C4 [22], with significant nonstationarity. This has evidenced that more work is required in order to improve the way

algorithms adapt to the noise level, handle the templates bank upgrade, and tune the veto procedures; very promising results have been obtained and useful lessons for the future upgrades have been learned.

References

[1] S Braccini et al. for the Virgo gollaboration *2005 Status of Virgo*, these proceedings.
[2] P Amico, L Bosi, C Cattuto,L Gammaitoni, M Punturo, F Travasso, H Vocca, (2004) Class.Quant.Grav. 21 S847 S851.
[3] P Amico, L Bosi, C Cattuto, L Gammaitoni, F Marchesoni, M Punturo, F Travasso, H Vocca, (2003) Comp. Phys. Comm. **153**, 179.
[4] Owen B. J., Sathyaprakash B. S., (1999) Phys.Rev. D60 022002
[5] W Gropp, E Lusk, A Skjellum *Using MPI - Portable Parallel Programming with message passing interface* MIT Press
[6] F Marion et al. 2003 "Multi-band search of coalescing binaries applied to Virgo CITF data", Proceedings of the Rencontres de Moriond.
[7] *http://www.devx.com/amd/Article/21979*
[8] Acernese F et al. for the Virgo gollaboration, (2004) Class.Quant.Grav. 21, S709:S716
[9] Acernese F et al. for the Virgo gollaboration, Class.Quant.Grav. submitted
[10] E.Cuoco et al. (2004) Class.Quant.Grav.21:S801-S806
[11] Fabrice Beauville Thesis *Prélude à l'analyse des données du détecteur Virgo: De l'étalonnage à la recherche de coalescences binaires* http://democrite.in2p3.fr/view_by_stamp.php?label=LAPP&langue=fr&action_todo=view&id=tel-00010297&version=1#
[12] A Vicerè for the Virgo gollaboration *The status of coalescing binaries search code in Virgo, and the analysis of C5 data*, these proceedings
[13] B Allen, (2005) Physical Review D 71 062001
[14] Steven W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing* (1997) California Technical Publishing ISBN 0-9660176-3-3
[15] W H Press, B P Flannery, S A Teukolsky and W T Vetterling, *Numerical Recipes in C*, 2nd edition, Cambridge University Press (1992).
[16] F Beauville et al, (2003) Class.Quant.Grav. 20, S789.
[17] F Beauville et al, (2005) Class.Quant.Grav. 22 4285-4309.
[18] Bosi L, Buskulic D, Cella G, Cokelaer T, Guidi G M, Viceré A (2004) *The inspiral library user manual*.
[19] L. Blanchet, B.R.Iyer, C.M.Will and A.G.Wiseman, Class. Quantum Grav. **13**, 575 (1996).
[20] Shawhan P, Ochsen E (2004) Class.Quant.Grav. 21, S1757-S1765.
[21] Guidi G M (2004) Class.Quant.Grav. 21, S1767-S1774.
[22] Acernese F et al. (the Virgo collaboration) (2005) Class.Quant.Grav. 22, S1139-S1148.