A Parallel Implementation of the Inspiral Search Algorithm using Triana

Dr David Churches, Prof B. S. Sathyaprakash
School of Physics and Astronomy
Cardiff University
David.Churches@astro.cf.ac.uk

Matthew Shields, Dr Ian Wang
Schools of Physics and Astonomy
and Computer Science
Cardiff University

Dr Ian Taylor
School of Computer Science
Cardiff University

Abstract

Black holes and gravitational waves are among the most fascinating predictions of Einstein's theory of General Relativity. Today we have indirect evidence for both but have directly observed neither. One of the most promising sources of gravitational waves for detection is the inspiralling compact binary system. This consists of a pair of dense, compact objects (either neutron stars or black holes) with masses of a few to a few tens of Solar masses, orbiting around each other with a period of minutes to hours. Such signals are hoped to be detected using a technique known as matched filtering. We present here an algorithm for performing the matched filtering inspiral search implemented as a workflow of connected Triana units. We use Triana's ability to distribute taskgraphs over a number of computers to parallelise the algorithm and compare it to a parallel implementation of the same algorithm implemented using MPI. Both implementations run on the same Beowulf cluster, allowing us to compare the trade off between the flexibility of Triana verses the speed of the MPI based code.

1 Introduction

A network of Interferometric Gravitational Wave Detectors is presently being constructed (the American LIGO [2], British–German GEO600 [1], French–Italian VIRGO [4], Japanese TAMA [3]). One of the most promising sources of gravitational waves for detection is the inspiralling compact binary system. This consists of a pair of dense, compact objects (either neutron stars or black holes) with masses of a few to a few tens of Solar masses, orbiting around each other with a period of minutes to hours. Such signals are hoped to be detected using a technique known as *matched filtering*.

In this technique, we have sufficient theoretical knowledge to construct the gravitational waveform produced by a given binary system. The problem is that the waveform shape is a function of the parameters of the binary (primarily the masses of the two compact objects), and we do not know *a priori* which binary waveforms may be present in our data. Therefore it is necessary to filter the detector data through a *bank* of waveform shapes, each

one characterised by different parameters. The template shape which most closely matches a waveform in the data will have parameters which are closest to the binary which actually produced the waveform.

Typically there may be several thousands of templates in a bank, and the data stretch may be several years long, sampled at 16KHz. Therefore the task is comutationally intensive, and is at present being carried out using the GEO++ software package [6] written in C++ and MPI on Beowulf clusters.

Triana [9] is an open source Problem Solving Environment (PSE), that is, a fully integrated computing environment for composing, compiling, and running applications in a number of problem areas [5]. Within Triana, there are several hundred tools for processing many types of data e.g. numerical data, signals, images and text documents.

Briefly, Triana is a graphical interactive environment that allows users to compose applications. A user creates a dataflow/workflow by dragging the set of desired units onto a workspace panel, and then interconnects these by dragging cables between them to specify the task-graph required. The user can then choose how their application will run i.e. serially or by distributing sections of the task-graph to other Triana services running on the Grid. Triana is a modular system that consists of a complex federated set of interacting components that gives Triana the flexibility it needs to be able to be applied to many different scenarios and at many different levels. For example, it can be used as a workflow engine for grid applications; for connecting data driven grid components and managing the workflow between them; or as a data analysis system for image, signal or text processing applications, allowing a scientist to quickly apply algorithms to data sets and view results. Triana was originally developed as a quick-look system for the forementioned GEO600 project but here we use Triana in a more sophisticated role, for analysing gravitational wave data by using the complex set of signal processing units described in the next section.

The Inspiral Search algorithm has been implemented in Triana using approximately fifty separate algorithmic components or *Units* connected to form a complicated work flow taskgraph (see Figure 1 below). We can use Triana's built in distribution mechanisms to parallelize the algorithm in a Single Program Multiple Data (SPMD) manner. Here, the sets of units that form the algorithm implementation are duplicated across a number of Triana Servers and the detector data is split into slices and passed to the distributed algorithms for processing.

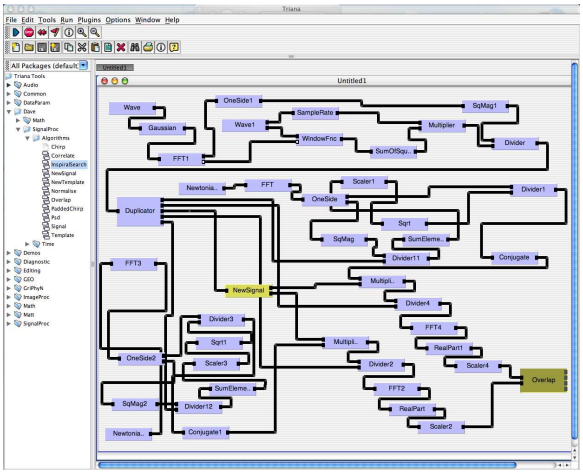

Figure 1: The Inspiral Search workflow constructed using Triana

The mechanism for distributing algorithms in Triana is discussed in more detail in the next section.

2 Triana and Distributing Algorithms

Triana can be used in two main modes – either as a stand alone application or as a distributed processing application. Algorithms such as the Inspiral Search can be developed and tested in the standalone mode by the scientist. Triana's rapid component drag and

drop mechanism allow ideas to be quickly implemented, tested with data, and amended or discarded. Traditional methods of scientific programming involve many interations of the "code, compile, execute" cycle. With Triana this is reduced to an "assemble, execute" cycle. Different filters or other algorithms can be easily replaced. Once the algorithm has been developed and tested in the standalone mode, the scientist can think about whether the alorgorithm is suitable for parallel execution.

In Triana, parallelising algorithms can be as simple as a few mouse clicks. As part of the pan-European GridLab [8] project, Triana is built on top of a middleware independent communication API, called the GridLab GAT. The full release version of the GridLab GAT will provide a consistent API to Grid services, such as resource brokering and monitoring. However, at present, Triana is based on a peer-to-peer subset of the GAT-API that we call the GAP Interface. The GAP Interface provides method calls for discovering other running Triana services, creating communication pipes and sending or receiving messages through those pipes. In Triana these distributed pipes are synonymous with connections between two Triana components. We have developed several GAP implementations, including a JXTA [7] binding; PSPS, a socket based peer to peer implementation developed in Cardiff; and a web services binding.

Distribution of executable code in Triana is based around "aggregate" or "group" components. The group component consists of one or more other components and is distributed according to a distribution policy. Currently we have implemented two distribution policies, but this set can be extended: the first is a pipeline parallelisation where the components are split and distributed over the available processing nodes, execution passes from one node to the next in the workflow; the second is a task farming or SPMD implementation where the group component is duplicated and a copy distributed to all available nodes, the data set is split, passed to a node for process and the results passed back.

3 Inspiral Search Algorithm

As mentioned above, one takes the gravitational wave detector output and looks within it for signals which have a particular shape. This shape is called a *template*, and it is constructed using our theoretical knowledge about relativistic binary systems. In actual fact, each template is composed of two waveforms which are shifted in phase with respect to each other by $\pi/2$ radians. The shape of the template waveform is determined by its family of parameters, and the two most important parameters which characterise a binary system are the masses of the compact objects. We denote the family of parameters by $\vec{\lambda}$.

To look for a template in a data stretch one performs a *correlation*. This is achieved by taking the fourier transform of both the template and the data, multiplying them together, then taking the inverse fourier transform. We denote the correlation of the data with the zero-phase template by $C_0(\tau, \vec{\lambda})$ and that with the $\pi/2$ phase template as $C_{\pi/2}(\tau, \vec{\lambda})$. Since the shape we are looking for could occur at any point in the data stretch, we must shift the template along the data stretch, and this shift is what is denoted by τ.

The two correlations are combined as follows,

$$X(\tau, \vec{\lambda}) = \sqrt{C_0^2(\tau, \vec{\lambda}) + C_{\pi/2}^2(\tau, \vec{\lambda})} \ , \quad (1)$$

and the final result is obtained by maximising over this array. The maximum occurs for the value of the shift τ which gives the best match between the template and the data.

This calculation must be performed for several thousands of templates, each differing in their parameters $\vec{\lambda}$. Such a collection of templates is called a *bank*, and it must be chosen

so that whatever the parameters of any real signal in the data, at least one of the templates in the bank has a close enough match to produce a detection.

When we run our algorithm, we use pre–existing C code to calculate the bank parameters and write them to a text file. We read from the file one line at a time, generate the appropriate pair of waveforms, calculate the correlations $C_0(\tau, \vec{\lambda})$ and $C_{\pi/2}(\tau, \vec{\lambda})$, combine them as shown in equation 1, then maximise over the resulting array. This yields a simgle number, which may be written into a file or submitted to an external database.

An advantage of Triana over the C++ implementation is that in Triana the two independent correlations are performed in separate threads, whereas in the C++ code they are performed sequentially. Therefore although Java may run more slowly than C or C++, significant gains may still be achieved, especially when executed on a dual processor machine.

4 Results and Conclusions

The Inspiral Search was construced in standalone mode but has been successfully run in parallel. As mentioned earlier, making the algorithm run in parallel requires no additional effort by the algorithm developer apart from a knowledge of the domain data and how to perform the data decomposition. In this case the Inspiral Search data is independent time series data and so the decomposition just involves splitting the data into discrete segments. The fact that the data segments are independent of one another means that the inspiral search is a data analysis problem which is very suitable for this form of parallel processing. Timing comparisons with the MPI C++ code are pending.

The development time needed to build the network shown in Figure 1 was of the order of several man weeks, as opposed to a timescale of months for the MPI version. Apart from the unit which generates the template waveforms, all other units were taken from pre–existing Triana toolboxes. This illustrates the major advantages of Triana; code reuse, rapid prototyping of ideas and extremely quick design/execute cycles. Java code cannot compete with a native compiled language and MPI in terms of speed, but we hope to illustrate that we can drastically reduce development time while incurring only a modest reduction in execution speed.

If this implementation of the Inspiral Search algorithm compares favourably in terms of speed with the MPI C++ version there is a strong possibility that the full realtime inspiral search of GEO data will be carried out using Triana.

References

[1] The GEO600 Project http://www.geo600.uni-hannover.de/.

[2] The LIGO Project http://www.ligo.caltech.edu/.

[3] The TAMA Project http://tamago.mtk.nao.ac.jp/.

[4] The VIRGO Project http://www.virgo.infn.it/.

[5] E. Gallopoulos, E. N. Houstis, and J. R. Rice. Computer as Thinker/Doer :Problem-Solving Environments for Computational Science. *IEEE Computational Science and Engineering*, 1(2), 1994.

[6] GEO++. http://www.astro.cf.ac.uk/pub/r.balasubramanian/geo++/index.htm.

[7] Project JXTA. http://www.jxta.org.

[8] The GridLab Project. http://GridLab.org.

[9] The Triana-Grid Group. Triana Software Environment: http://www.triana.co.uk and http://www.trianacode.org.