# Parallel Horace – design ideas

## User interaction.

1. A user expected to continue to use Horace as he does now, i.e. typing or scripting Horace commands.
2. If appropriate powerful machine or parallel cluster is available to a user, user would enable parallel capabilities by issuing a simple command. (e.g. **hpc on**), which spawns execution of the time consuming operations in parallel.
3. The parallel Horace’s features are configured using Herbert configuration classes and available for advanced users to configure and fine-tune. A simplified default version of the framework should run without any users configuring.
4. User may or may not have access to parallel computing toolbox and distributed Matlab server. The presence to these resources improves user’s computational capabilities, but the basic access to parallel Horace resources expected to be independent on these resources.
5. The commands which may take long time and will benefit from parallelization are **gen\_sqw cut\_sqw**, **unit** and **binary operations**, **tobyfit.fit** ,**sqw\_eval** and **symmetrise**. The description of the dataflows, occurring in these algorithms is provided in the Appendix 1

## Technical facts, constrains and opportunities.

1. All Horace algorithms benefiting from parallelization can be summarized by the following pseudocode:

*while* condition(Data):

*For* i=1:N\_iterations

Results(i) = Do\_processing(Data,i)

*End*

condition,summary =

reduce\_results\_process\_condition(Data,Results)

*end* (condition)

*return* summary

where expensive operations of interest on Data are in fact the operations over the pixels of one or the group of sqw objects. The sqw objects expected not fit the memory. The *Do\_processing* operation can be efficiently executed independently on each pixel or small group of pixels located im memory or retrieved from a file system, **common for all computational nodes**, while *reduce\_results\_process\_condition* request interprocess communications. The condition and summary variables can easy fit the memory of a single node and exchanged through MPI communications.

The sufficient way for parallelizing such job would be division of the Data in the *For* loop among MPI workers and the usage of MPI communications to gather Results and condition on a head MPI node.

1. A MPI job can always and only be executed in the form:

>> mpiexec n\_workers the\_mpi\_program

1. Matlab supports MPI jobs submitting parallel task to a ***cluster*** :

*cl = parcluster();*

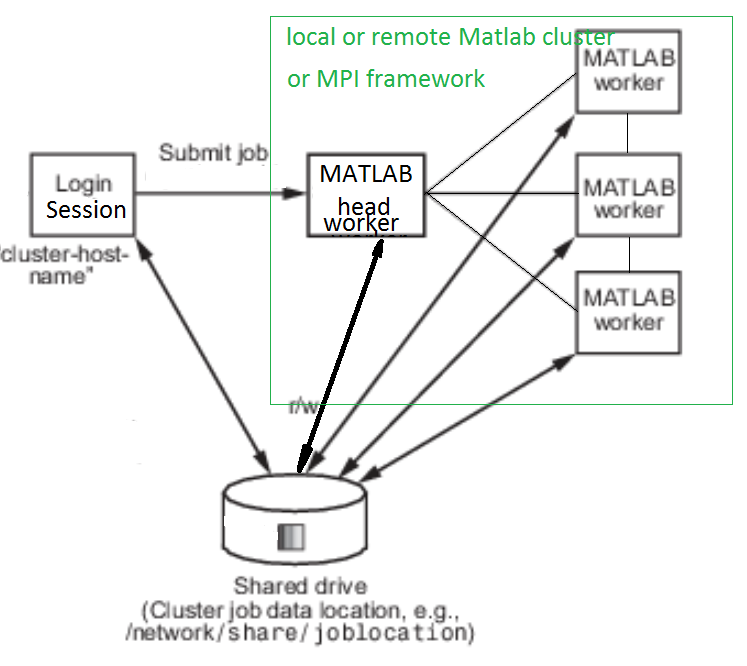
cjob = createCommunicatingJob(cl,'Type','SPMD');

task = createTask(cjob,UserFunctionHandle ,0,{user\_function\_inpts});

submit(cjob);

where the **submit** function is equivalent to the start of mpiexec function and number of workers (***n\_workers***) is defined previously within a ***cluster*** configuration (***n\_workers*** = cl.NumWorkers)

1. The Matlab ***cluster*** configuration necessary and sufficient to run Horace MPI jobs 2.1 above has the form, where the presence of common file system (parallel for performance reason) assumed to be mandatory.



1. We expect to design and create software, working in 3 main hardware/software configurations:
   1. A powerful user machine with or without parallel computing toolbox powerful enough to run number of headless Matlab sessions and accessing fast file system.
   2. A user machine with parallel computing toolbox installed and connected to a parallel file system and accessing Matlab distributed computing server installed on a parallel cluster (like SCARF or number of DaaaS virtual machines)
   3. A user function compiled on Matlab with Horace using Matlab compiler and deployed as part of an MPI job on a parallel cluster having compiled Horace installed on each node.

All these configurations can be united under the logical configuration 4)

1. Matlab parallel computing toolbox and distributed Matlab server provide range of classes and functions (blocks) to deliver system independent operations. The blocks we are decided to use are :
   1. parcluster (parpool?) -- provides interface to physical resources and job control operations
   2. MPI commands (e.g. labnum, labsend, labreceive etc), working in a spmd block only and providing low level communication and synchronization routines between different Matlab workers.
   3. SPMD blocks itself, hiding the procedure of task splitting and MPI jobs dispatching on the selected parcluster.

We are going to wrap these blocks into custom classes with common interface, to allow Horace user experience to be independent on presence of parallel computing toolbox and distributed server. The only difference should be in the performance of different configurations.

## Design choices.

Set of polymorphic classes will be developed to satisfy user requests 1) with the conditions 2) to support three user cases, namely:

a) User has parallel computing toolbox.

b) User have powerful machine without parallel computer toolbox

c) User has access to cluster where compiled Horace is installed

These classes are:

1. cluster : the interface to parpool class in case a), or the class which supports starting and managing the Matlab workers (headless Matlab sessions) on local machine case b) or MPI framework configurator and interface to mpiexec and MPI framework configuration in the case c)
2. MPI\_communicator: the wrapper for Matlab MPI commands in case a) , or class writing files to be interpreted as MPI messages in case b) or Matlab wrapper around mex file providing access C++ MPI framework in case c).
3. Job\_executor – the abstract class providing access to *Do\_processing* and *reduce\_results\_process\_condition* methods from chapter 2.1. Each supported algorithm from 1.5 would implement this class and provide its own algorithm specific implementations for these methods.
4. Job\_dispatcher – the class which execute Job\_executor’s *Do\_processing*  and *reduce\_results\_process\_condition* methods depending on parallel framework settings and resources, available to user
5. An sqw object and set of sqw objects will get *get\_pixels* method, which would provide access to specified number of pixels out of range of all pixels present in sqw object or sqw objects set regardless of these objects physical location (in memory or in a file)

Finally, a parallel\_config class would allow switching between all these three configurations fine-tuning of the configurations and access to the parameters of these frameworks.

## Note

Only options 3.a and 3.b would be implemented from the beginning of the project. Option 3.c will be implemented if sufficient resources are available.

## Current (01/01/2018) Implementation:

On the date specified in the header, three substantially different blocks of code are written to support the Horace parallel capabilities namely:

1. Herbert Cluster, running multiple Matlab sessions to perform parallel job and using message files to exchange data between workers (Poor-man MPI).

*Its advantages:*

a) No need in parallel computing toolbox and additional Matlab licences

b) The code is fully controlled by Horace development group and can be if necessary extended to provide MPI and cluster capability with no licensing cost (but substantial development cost)

*Its disadvantages:*

1. Runs only on a single node.
2. Becomes extremely slow if an algorithm needs active inter-process communications.
3. Matlab (parpool) cluster running multiple Matlab sessions using Matlab parallel computing toolbox and embedded Matlab MPI intercommunications. Fails back to filebased messages exchange mechanism in case of Matlab MPI being not enabled.

*Its advantages:*

1. Fast (MPI-defined) interprocess communication allowing to run the whole range of Horace algorithms (see below)
2. The possibility to run on a multi-node machine with nodes connected by MPI link or on a parallel cloud if appropriate licenses are available

*Its disadvantages:*

1. Needs parallel computing toolbox license to run on a single node and additional licenses to run on a cluster or cloud
2. The root of developing code without Matlab and additional licenses is completely closed
3. C++ code used to combine binary sqw files into single sqw file

*Its advantages:*

1. Extremely fast and efficient in its area of applicability.
2. With substantial development and inclusion of 3rd party code (Mantid fake MPI (Simon Heybrock private communication http://docs.mantidproject.org/v3.11.0/development/AlgorithmMPISupport.html#units-tests.) or https://www.codeproject.com/articles/1092727/asynchronous-multicast-callbacks-with-inter-thread) may become a communication hub for the Herbert cluster, overcoming its current disadvantages.

*Its disadvantages:*

1. Currently have very narrow applicability area
2. Has bugs so fails sometimes and currently does not work on SCARF (bugs)
3. C++ code – high cost of development.

As the majority of commonly used Horace algorithms are limited by file-IO speed, one needs fast parallel file system to gain advantage of Horace parallel capabilities.

In additional to that, Herbert Cluster needs powerful multiprocessor machine with multiple processors to run multiple Matlab sessions (At least 64 computational threads and 0.25Tb memory to fully accelerate gen\_sqw on current parallel file system or more if a symmetrisation during file generation is neded.

A Matlab Cluster (parpool) needs either similar machine + parallel computing toolbox licence for each user or multiple nodes connected by parallel file system and fast MPI link + the same toolbox and + Matlab distributed server licenses for each node.

The Horace algorithms and their benefits form each approach are summarized in the table below:

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Herbert Cluster | Parpool cluster | C++ code |
| Gen tmp files (part of gen\_sqw) | 10-fold acceleration | 10-fold acceleration | N/A |
| write\_nsqw\_to\_sqw  (as part of gen\_sqw or separate) | 4-fold deceleration wrt. to serial Matlab code | Substantial acceleration (needs further testing for numbers) | 5-10 fold acceleration |
| Accumulate headers  (as part of gen\_sqw) | Substantial acceleration, a bit slower than Parpool | Substantial acceleration | Currently N/A |
| Accumulate cut (developing) | 10-fold acceleration expected | 10-fold acceleration expected | N/A |
| sqw\_eval (planned development) | n-workers acceleration expected | n-workers acceleration expected | N/A |

## Appendix 1. Dataflows in various parallel algorithms.

The data-flow diagrams below use the following markers:



The direction of the row In addition to this, the thickness of the lines identifies the amount of the data to be transferred through the correspondent media, namely:



Using this marks, the parallel Horace algorithms can be written as follows:

### gen\_sqw algorithm dataflow:

Gen\_sqw – this algorithm actually consist of three parts. The first part, gen\_tmp, takes all contributed spe or nxspe files and convert each into single sqw file:



Figure Gen\_tmp. Convert multiple nxspe files into multiple sqw (tmp) files.

This is massively parallel algorithm, where no interprocess communication is necessary.

The second part, accumulate\_headeders runs over all generated sqw (tmp) files and collects information on final layout of the pixels data within the combined sqw file:



Figure . Accumulate\_headers. Collect and process information on the pixels location within the final sqw file.

This algorithm exchanges information about partial location of pixels within the file. The amount of this information is the order of the dnd image, so the algorithm would benefit from MPI communications but could work with filebased exchange of the information between the nodes.

The final part of the algorithm write\_nsqw\_to\_sqw combines all pixel information within the final sqw file:



Figure . Combine the pixels information and write final sqw file.

This algorithm requests the exchange of the whole pixels information between the nodes, so needs efficient MPI communications between the nodes. It will work much slower if implemented using file-based communication mechanisms.

### cut\_sqw algorithm dataflow:

Cut\_sqw benefit from parallel implementation if the initial sqw object is big enough so it cannot fit the memory. In this case, cut\_sqw works from the sqw object, located on a filesystem (file). Depending on the operation mode, it request three different dataflows:



Figure Cut\_sqw -- nopix mode.

If invoked with ***-nopix*** option, only signal/error/npix information is exchanged between nodes, so no substantial interprocess communications are necessary. The acceleration is achieved due to multiple workers performing parallel read operations from sqw file, locate on parallel file system.

If pixel information is needed but the target cut can be placed in the memory of a single node, interprocess communication demands are higher but still relatively moderate:



Figure . cut\_sqw in pix mode when target cut fits the memory.

Finally, if the cut object does not fit memory, the data flow becomes very similar to dataflow used by first stage of gen\_sqw algorithm except the source not is not a multiple spe files but single sqw file:



Figure Cut\_sqw in filebased mode.Stage 1.

The stages of collecting the intermedium tmp files into sqw files are the same as the second (**Figure 2**) and third (**Figure** **3**) stages of gen\_sqw algorithm.

### Unit operations:

Horace does not implement operations affecting the pixels. The operations on the image would not benefit from MPI parallelization so the main unit operation, benefiting from parallelization would be operation, similar to cut\_sqw in nopix mode (Figure 4):



Figure Unit operation without change to pixel information.

The only possible change to sqw file would be change in the image (DND) information. This change can be efficiently applied from the interactive note. In the case when change in pixels is still necessary, the dataflow could be similar to the one, applied for binary operation below, except in the case of a unit operation only one source file will be used.

### Binary operations:

Binary operations are very similar to unary operations except it may be reasonable to implement them on pixels. Current binary operations implementation works only when pixels number and their coordinates are equivalent. It may be reasonable to implement operations, which would combine pixels. The dataflow for this kind of operation would look like the one below:



Figure . Filebased binary operation.

### sqw\_eval

sqw\_eval can be considered as a particular case of the unit operation, except there is no need to store changed dnd information in the file and the initial object is located in memory, so if the evaluated function is expensive, it makes sense to separate its calculations among multiple workers.



Figure sqw\_eval in case of object can be fit in a single node memory

If the sqw object does not fit the memory, the sqw\_eval dataflow becomes similar to a binary operation:



Figure sqw\_eval. Filebased operations

### Tobyfit, fit:

Tobyfit and fit algorithms are the only algorithms, requested repetitive exchange of data between central node (node0) and other workers. It is reasonable to implement linear fit algorithm on a single node and use all other nodes to calculate error function. In this case, the data exchange between the nodes becomes minimal, but the changes in the parallel worker may be necessary, as this type of the algorithm does not fit the parallel pattern, described in the Chapter 2:



Figure Fit algorithm in the case when cut info is provided as input.

The changes to the parallel worker do not look extensive or overcomplicated.

### Symmetrise:

The dataflow of the symmetrise algorithm is the same as for cut algorithm except the contributed pixels and pixels transformation are calculated differently.