Multi-threading for ESO Pipelines

Lander de Bilbao[1, 2, 3, *], Lars Kr. Lundin[1], Pascal Ballester[1], Klaus Banse[1], Carlo Izzo[1], César E. García-Dabó[4], Ralf Palsa[1]

1: European Southern Observatory – Software Development Division, Karl-Schwarzschild-Str. 2 D-85748 Garching bei München, Germany
2: Instituto de Física de Cantabria – CSIC, Avda. de los Castros s/n E-39005 Santander, Spain
3: Fundación Española para la Ciencia y la Tecnología, Rosario Pino 14-16 E-28020 Madrid, Spain
4: FRACTAL SLNE, Tulipán 2, portal 13, 1. A E-28231 Las Rozas de Madrid, Spain

Abstract. The second generation of instruments for ESO's VLT at Paranal Observatory will increase dramatically the volume of raw data per night. This in turn leads to very high computational needs that can be, up to some level, addressed by existing multi-core, shared-memory computers. To fully utilize these multi-processor systems we need to implement a programming environment supporting multi-threaded execution of applications. Such parallel execution needs to be introduced at the level of pipeline recipes as well as within the Common Pipeline Library (CPL)[1] on which all operational VLT pipelines of ESO are based. We describe our approach to providing such a new, multi-threading pipeline setup and evaluate possible implementation solutions with some performance measurements.

1. Introduction

ESO currently maintains a number of instrument pipelines for the VLT. They consist of a number of sequential recipes, which are built on top of a common, thread-unsafe library (CPL). Also the second generation instrument pipelines, currently in development, are based on the CPL. They vary significantly with respect to their algorithms.

Initial attempts to parallelize recipe execution have been done by enabling, via batch scheduling, the concurrent execution of independent data reduction processes, showing limited potential. Therefore the option of multi-threading recipes is considered as well.

Our aim is to enable the implementation of multi-threading recipes on top of a thread-safe CPL, while still coping with the maintenance of the legacy sequential code of the existing pipelines. We do not intend to develop a new thread-safe library but rather to modify the CPL where necessary. For this purpose we have chosen OpenMP [2].

2. CPL

The CPL includes a number of global variables which make the library inherently thread-unsafe. In some cases, global variables cannot simply be made local. The necessary API modifications would impose an unfeasible amount of code changes in the CPL. This is the case with the error handling modules.

A solution to this problem is to provide private copies of global variables for each thread. This is easily enabled using simple OpenMP pragmas:

```
static cpl_boolean cpl_error_status = CPL_FALSE;
static cpl_boolean cpl_error_read_only = CPL_FALSE;

#pragma omp threadprivate(cpl_error_status, cpl_error_read_only)
```

This approach implies new semantics for error handling in a multi-threading environment.

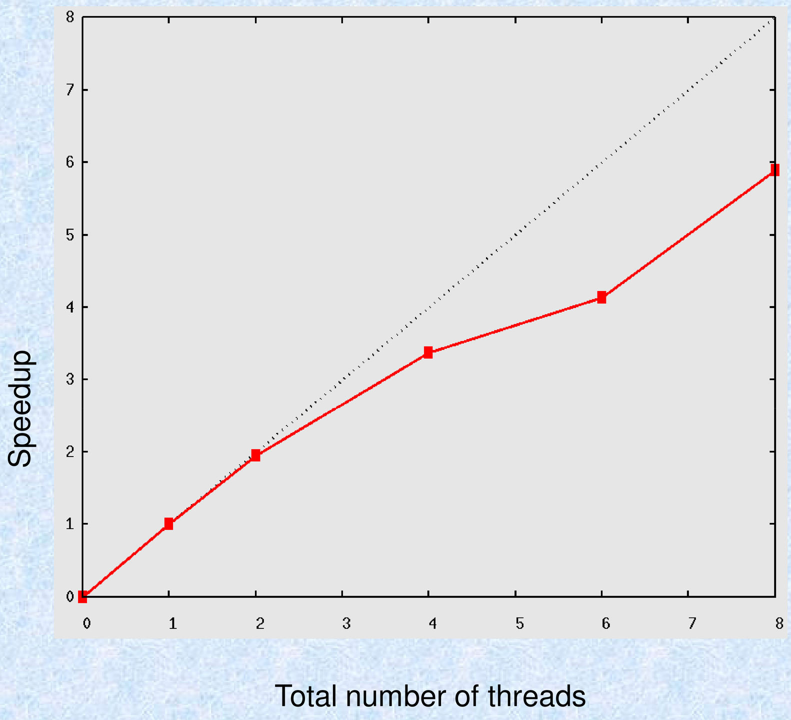
3. VISIR

A first candidate for parallelization are the VISIR[3] science recipes, for which an even number of 256 x 256 (nodding + chopping) images are loaded into memory (512kB RAM each), normalized with respect to DIT (Detector Integration Time) and pairwise added. The final step consists of destriping the added images, using an iterative filtering.

Parallelization can be achieved in this case by adding, on top of the loop over the images, a single pragma of the type:

```
#pragma omp parallel for private([private variable list]) schedule(static, 2)
for ( ; ; ) {
     /* Code for each iteration */
}
```

The `schedule` construct is added due to the need to process the frames pairwise.

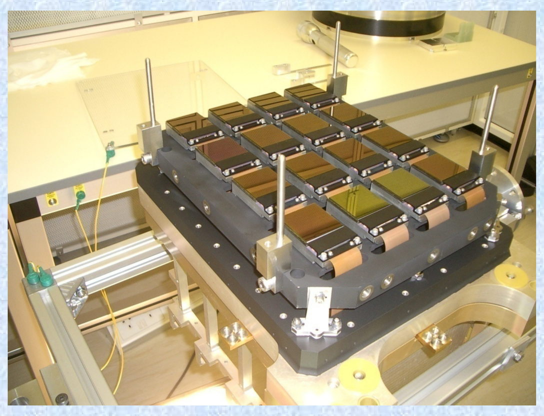


Results with VISIR

Figure 1:
Speedup in VISIR execution (see 6 for hardware specification).
It shows a good scalability up to 4 threads, with a maximum of 5.89 on 8 threads (74% efficient). On 6 threads, the speedup is lower than expected, because this amount of threads is not optimal for the machine in use.

4. VIRCAM / VISTA

VIRCAM is the most demanding pipeline [4] in terms of computational resources at the time of writing. It is a 16-chip instrument and therefore it offers a great potential for parallelization. This makes it an excellent testbed to assess the potential of OpenMP.



VISTA Focal Plane, 16 detectors on their mount. Credit: VISTA

From a set of 2048 x 2048 flat images, the recipe vircam_twilight_flat_combine removes any image which is saturated or underexposed, linearizes the remaining ones, removes the dark current from them, combines them with rejection and normalizes the result by its median.

Parallelism at the FITS extension level is the obvious strategy. However, it is not the most efficient one due to its lack of data locality (repeated reading of data from the cache reducing the high-latency access to main memory).

A combined strategy, introducing parallelism a) at the FITS extension level and b) at the image level, is applied.

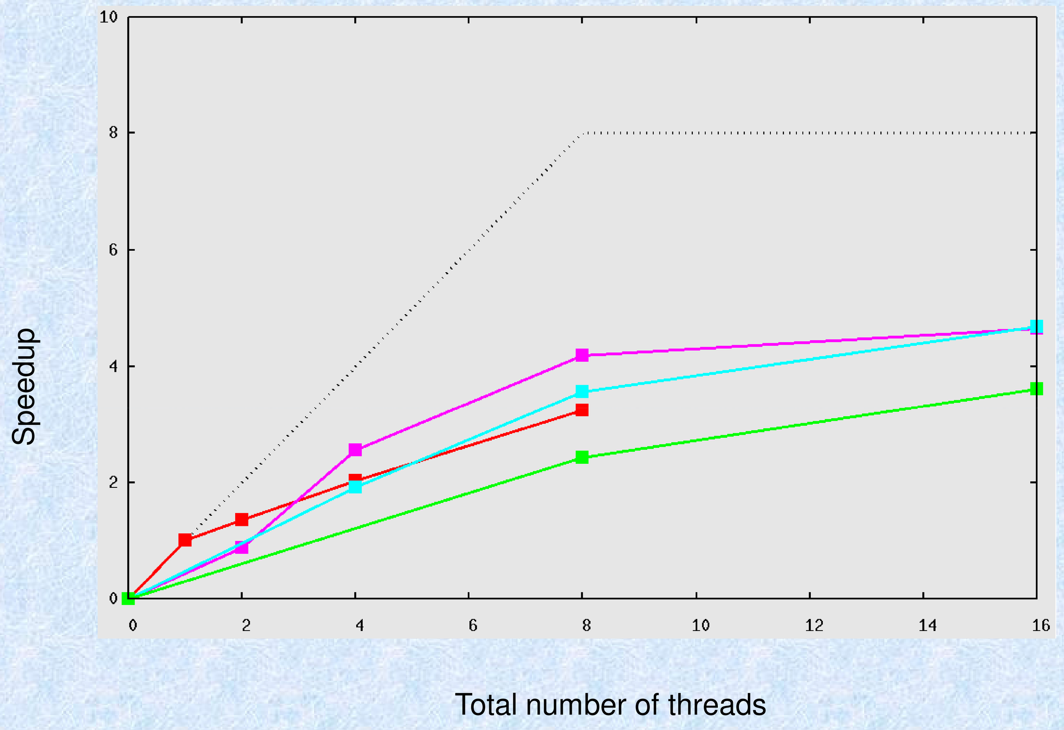
At the first level (a) the introduction of the following pragmas is needed:

```
#pragma omp threadprivate ([global variable list])

#pragma omp parallel for ordered                          \
                         private([private variable list]) \
                         shared([shared variable list])   \
                         schedule(dynamic)                \
                         num_threads([Nth])
 for ( ; ; ) {
     /* Code to process each extension */

     #pragma omp ordered
     save_ext();
}
```

The `ordered` construct ensures that the products are appended to the result FITS file in the proper order.

At the second level (b) several pragmas of the type already used in VISIR are added on top of every loop traversing the pixel buffer. In order to avoid false sharing (undesired writing to a single cache line by more than one thread) at this level, an alternative memory allocator, assuring cache line alignment, HOARD[5], has been used.

Fig. 3 shows that scalability is rather poor. Images are large; the recipe execution is rather memory-intensive. The 8 cores available share 2 L2 caches only, while during the sequential execution, a single process has a whole L2 cache at its disposal. Applying nested levels of parallelism improves the data locality and shows better results. Nevertheless, this multi-threaded solution pays off in comparison with the parallelization via batch scheduling (in this case, launching a separate data reduction process for each chip), which offered a speedup, in regard to the sequential execution, only close to 2. Here the maximum speedup is 4.69 (59% efficient) on 16 threads, with 4 threads at the extension level, then each one launching other 4 at the image level.



Results with VISTA

Figure 3:
Speedup in VISTA execution, using different parallelization strategies (see 6 for hardware specification). There are 4 variants: single thread at the image level (red), 2 threads at the image level (purple), 4 threads at the image level (blue) and 8 threads at the image level (green).

5. Conclusions and future work

Given a suitable parallelization strategy, multi-threaded recipes bring substantial advantage in performance. Nevertheless, their scalability is limited by the nature of the reduction task itself. The optimal number of threads might not necessarily coincide with the number of available cores. For future machines, a combined solution where several independent data reduction processes are launched, each process in turn launching in a number of threads, is expected to be the most efficient one.

The parallelization strategy presented with the VISIR pipeline will be very likely adopted by the pipeline for the second generation instrument MUSE in order to reduce data cubes.

With regard to future work, the consequences of the new semantics for error handling in CPL must be analyzed, especially the case where one might need to propagate an error from within a parallel region. Implementing multi-threaded versions of highly computation-demanding CPL functions is also a possibility.

6. Data sheet

The machine used for these tests is a workstation with 8 cores, the Intel Xeon E5420 (Dual die @ 2.50Ghz, 4-cores + 6MB cache each). The multi-threaded recipes have been built on top of an internal development version of CPL 5.1 with CFITSIO 3.181 (reentrant).

7. References

[1] http://eso.org/cpl
[2] http://openmp.org
[3] http://eso.org/sci/facilities/paranal/instruments/visir
[4] http://eso.org/sci/facilities/paranal/instruments/vista
[5] http://www.hoard.org/

(*) Lander de Bilbao's position at ESO is funded by Spain as part of the in-kind contribution to ESO. ***Contact:*** *lbilbao@eso.org*