

correct all batches of the raw data. This keeps the inter-processor communication for the auto-focus to a minimum.

3.3 Problems And Solutions

In order to process the raw radar data, a series of imaging parameters must be distributed to the various processing stages. These parameters can not be determined a priori, therefore they must be synchronised with the data to which they refer. This synchronisation of the data flow is essential to avoid data flow deadlocks particularly at schedule interfaces. The simplest way we found to ensure synchronisation is to duplicate all of the interpolation, decimation, overlap/hold etc. on both the data and the parameters.

Non-deterministic data flow is a significant problem especially when running partitioned on an embedded system. Flowgraphs, which work perfectly when run on the host, can easily become deadlocked when distributed on an embedded system. In many situations it is possible to simplify the data flow by using dynamic input/output. Non-deterministic data flow is only necessary where the system must continue processing even when one or more of the inputs are unavailable, or where the number of tokens required varies. Combine this with the additional complexity of debugging a user written “nondet” primitive that may work fine on the host and it becomes important to limit the use of non-deterministic primitives and keep their functionality as simple as possible.

Inter-processor communication is essential for a multiprocessor implementation, but since these transfers take time, it is important to keep the amount of data being transferred to a minimum and to select the transfer mechanisms carefully. Different transfer mechanisms allow a trade-off to be made between memory requirement and transfer speeds. Non-blocking transfers ensure that processing continues without the need to wait for the transfer to complete, although normally at the cost of more memory. The target hardware will also impose restriction on the number and rate of the transfers. By considering the necessary transfers early a design can be made more flexible and therefore be ported to a wider variety of target platforms.

In porting our design to an embedded system we have also had to cope with interfacing between Gedae and external hardware (such as a disk-array to read the data and a graphics card to display the results). This has required the use of external source code and libraries, both of which are easily added as entries in the Personal_Obj_List and Personal_Emb_Obj_List files.

Our design initially used the standard interleaved complex data type, as the split complex data type was only introduced in version 4.0. Certain functions, such as FFTs can be implemented more efficiently for the split complex data type (due the order in which individual values are used) as shown in Figure 2.

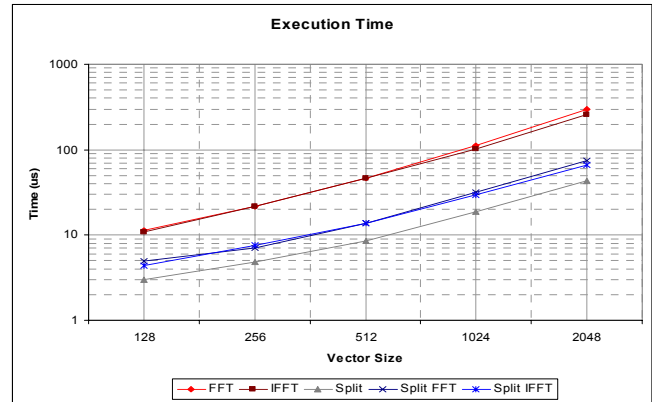


Figure 2 Complex FFT Execution Times

For reference, the time to perform the de-interleaving from the standard complex data type into split complex is shown as “split”. Depending on the size of FFT there is a speed improvement between 2 and 4 times. There is still a trade-off to be made as transferring data between processors requires two ports to be established for split complex, each transferring half the data, whereas the interleaved complex data type uses only one port.

Inevitably with a product that is evolving and advancing as quickly as Gedae there will be bugs to contend with. Without exception these have been solved by way of an iterative technical dialogue, between Thales Sensors and Blue Horizon, which has helped in debugging both our data flow as well as solving issues with the board support packages and Gedae itself.

3.4 Novel Language Usage

As with all design projects it is important to include additional information for debugging. We have made extensive use of “families” to achieve this throughout our development. Using a null family (e.g. a range 0..-1) allows the additional debug information to be easily switched on or off and has the advantage of not requiring memory to be allocated for these tasks if they are not needed.

Families can also be used to switch between alternate processing chains. When implementing our design we wanted the ability to be able to switch between processing simulated data and real radar data without having to constantly modify connections. We achieved this by connecting the outputs of both data sources to the input of the SAR processor (using a text editor to modify the connections) and then ensuring that only one of the data sources is instantiated at any time by using complementary families. The families can be set up using two ranges ‘simulated’ and ‘real’ defined as “simulated = 0..Test?0:-1” and “real = 0..Test?-1:0”.

4. RESULTS

Our original (modest) aim was to implement the SAR processing algorithms and process a range swath of 1024 samples of data in real-time. The current performance has exceeded this with 1024 range cells processed in ~50% of real-time (i.e. taking half the available time to complete the processing). This is with all the processors active for in excess of 75% of the time and the inter-processor communications taking up just 0.2% of the time. We have also processed a larger swath, with 2048 range cells, in

~90% of real-time. This represents the complete SAR processing chain.

An example of the imagery produced in this project is shown in Figure 3, with a rough categorisation of the land usage shown in Figure 4.

The auto-focus provides approximately 40% improvement in the contrast on this data set apart from the section across the wetlands (see Figure 5 which has been approximately aligned with the Figures 3 and 4). The data in this area does not contain enough information to allow the auto-focus to produce a reliable estimate of the phase errors. Therefore, the auto-focus is automatically switched off in regions where the contrast in the original image is too low and the data-flow adjusted accordingly.

5. CONCLUSIONS

Within a five month period we have demonstrated a fully featured SAR processor running in real-time on embedded hardware. During this development we have learnt a number of lessons about both general multi-processor designs and more specifically about their implementation in Gedae. The main points that have been learnt are:

- Ensure that the dataflow is synchronised.
- Use non-deterministic data flow only where it is essential.
- Keep inter-processor communication to a minimum.

6. FUTURE PLANS

The next task will be to integrate the SAR processor into a complete multi-mode radar demonstration. Currently the SAR processor uses the majority of the available memory. Therefore, we will need to use segmentation and exclusive memory allocation to allow re-use of this memory for other tasks. Segmentation will also allow proper initialisation at mode changes. The current version of Gedae includes the language syntax necessary to describe segmentation but partitioning of segmented graphs will not be supported until the next version.

The current graphics card is only capable of displaying the SAR data with a reduced update-rate (refreshing every four lines of data). This will be replaced with a more powerful PCI version to allow implementation of a smooth scrolling display with the addition of image exploitation tools.

7. REFERENCES

[1] Carrara, W.G. Spotlight Synthetic Aperture Radar Signal Processing Algorithms. Artech House, 1995, 252-260.



Figure 3 SAR Image generated using Gedae



Key: Urban Agricultural Wetland River Airfield

Figure 4 Land Usage

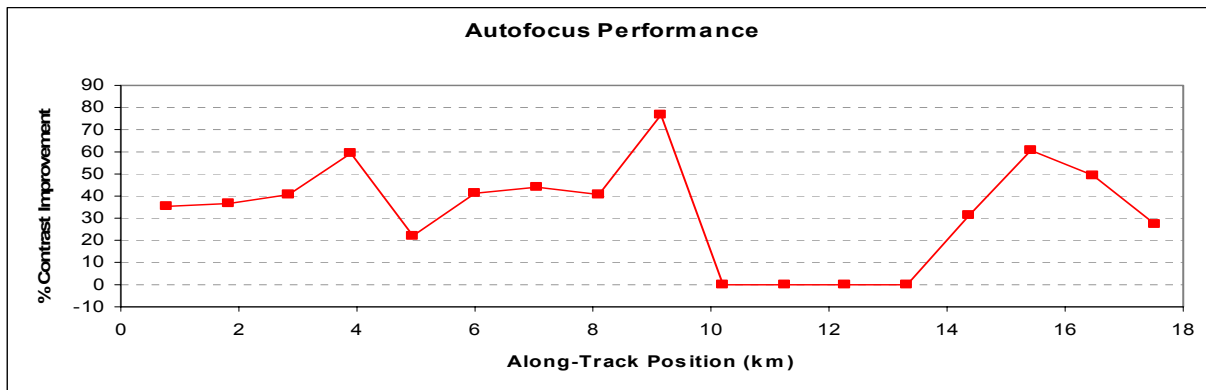


Figure 5 Auto-focus Performance