GRAIL : *G*raphical *R*epresentation of *A*ctivity, *I*nterconnection and *L*oading

Susan Stepney

GEC-Marconi Research Centre, West Hanningfield Road,
Great Baddow, Chelmsford, Essex, CM2 8HN.

1 Introduction

It can be very difficult to know just what is going on inside your Transputers. Are they executing efficiently, or are they spending all their time waiting? And if so, just who is waiting where for whom?

All this assumes you have some program running. But what if your problem is more fundamental? How good are different algorithms for solving a problem? How does changing the topology affect the performance of an algorithm?

These questions need to be answered, and answering them needs tools. ParSiFal is an Alvey backed collaborative project[1] to build Transputer-based hardware and occam-based software tools. The aim is to help researchers, analysts and programmers gain the insight they need to understand, design and write parallel programs.

2 Graphical Occam

One of the tools being developed as part of ParSiFal is GRAIL. It represents an occam program graphically, in a way that highlights the parallel structure and communications. Activity information can be overlayed in colour.

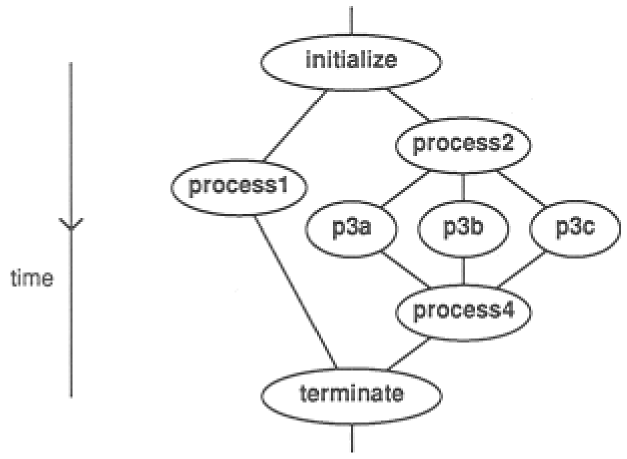
The current implementation uses a colour Sun workstation for the display. Interaction takes place via the mouse.

2.1 Processes

The display is based on occam's hierarchical process structure. Processes are drawn in nested rectangular boxes. Different style boxes are used for the different constructs `SEQ`, `WHILE`, `IF`, `PAR` and `ALT`. Sequential structures and deterministic choice (`IF`) are drawn vertically, as in the conventional way of writing the language. Parallel structures and non-deterministic choice (`ALT`) are drawn horizontally.

Consider a hypothetical parallel program with the following structure (here the lines represent flow of control, not communication channels):

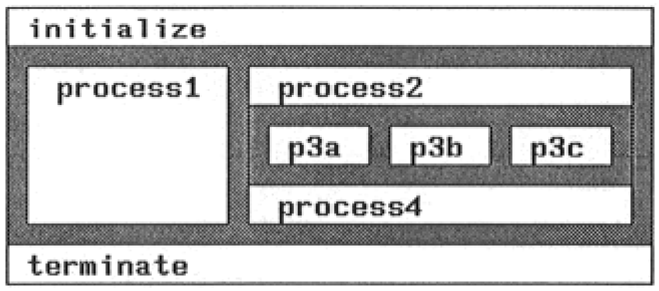
[1] The seven ParSiFal partners are Cambridge University Engineering Department, FEGS Ltd., GEC Research Ltd., Inmos Ltd., Logica p.l.c., Manchester University Computer Science Department and the Polytechnic of Central London.



The occam for such a structure is:

```
SEQ
  initialize
  PAR
    processl
    SEQ
      process2
      PAR
        p3a
        p3b
        p3c
      process4
  terminate
```

It is not very easy to relate this linear text to the parallel and sequential structure of the program. The GRAIL display of the above occam text is



Now you can see, at-a-glance, which processes are in parallel, and which are sequential.

2.2 Folds

In order to make the display manageable, the concept of folds has been carried across from the Inmos TDS folding editor. Folds may be opened, closed, entered and exited, using the mouse.

2.3 Channels

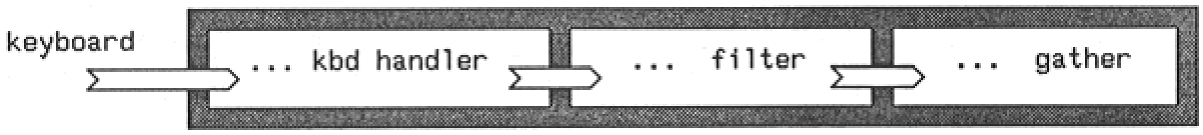
Channel information can be optionally overlaid on the process structure diagrams.

Channels are drawn as directed arrows between the processes they connect. The graphical display shows channels entering and leaving closed folds and procedure calls. It groups all the end points of a channel together, and indicates direction.

In occam, a simple pipeline might be written

```
PAR
  ... kbd handler
  ... filter
  ... gather
```

This gives no indication of the communication between the processes, except a hint in the fold names. In GRAIL, this would (if the hints are correct) be displayed as



3 Activity by Colour

The GRAIL display is also used to show the activity of a monitored run of the program. “Hot” areas, where a lot of activity is occurring, are shown in red. “Cold” areas of low activity are shown in blue. Either process or channel activity can be displayed.

Currently this activity is deduced by profiling a running program. As part of ParSiFal modifications have been made to the Inmos TDS compiler, implementing a profiling option. Statement execution counts are gathered as a program runs, and when it has finished (or deadlocked!) these results are wormed out of the Transputers. They are used to deduce the activity.

The next stage of our work in ParSiFal is to include timing information, to show where the communication bottlenecks are. Eventually we hope to have a fully dynamic version, which will show the activity as a program is running, so that hotspots that are localized in time, as well as space, can be observed.

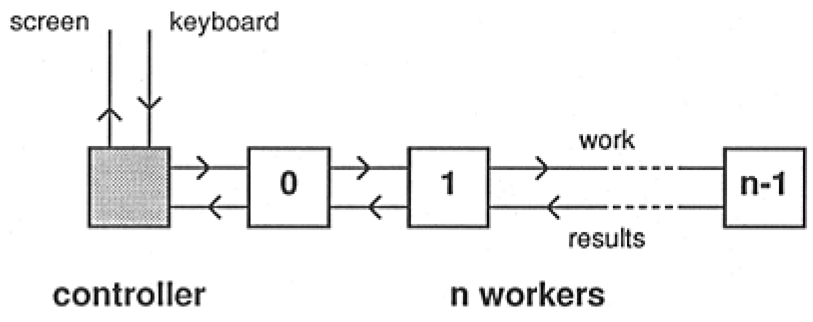
4 An Example - a Mandelbrot program

I have chosen a relatively well-known example to illustrate GRAIL, so that you do not have to cope with understanding a new program at the same time as trying to understand the new way of displaying it.

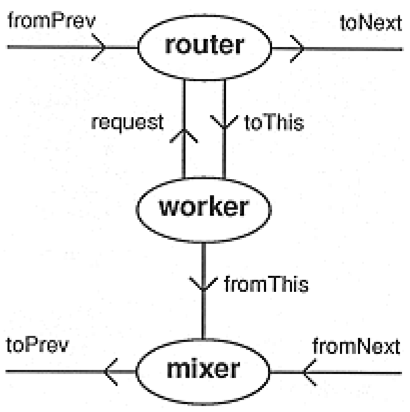
The example is the ubiquitous Mandelbrot pipeline. The ray-tracing example described in Inmos Technical Note 7 has the same structure, only the `worker` part is different.

4.1 Program Structure

The work is handed out by a controller to a pipeline of n worker Transputers:



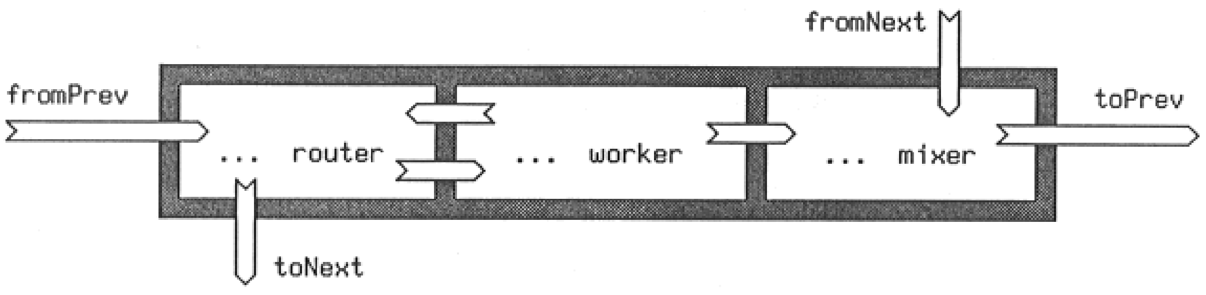
The program on each Transputer has a structure like



In occam this would be something like (ignoring complications of `PRI PAR`s):

```
PAR
  ... router
  ... worker
  ... mixer
```

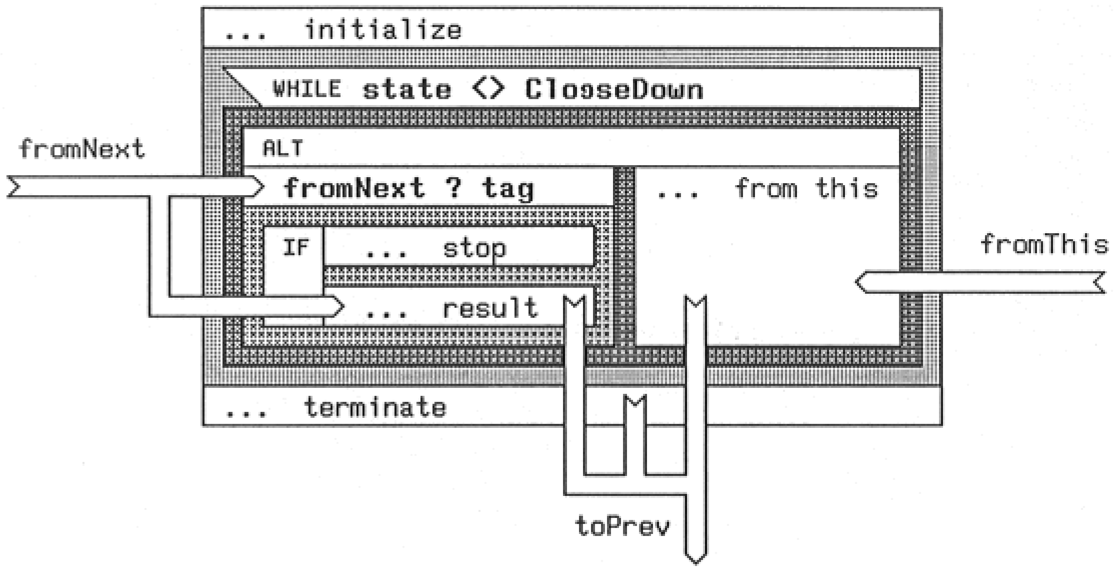
The corresponding GRAIL display is



Entering the mixer fold in occam results in something like:

```
SEQ
  ... initialize
  WHILE state <> CloseDown
    ALT
      fromNext ? tag
        IF
          ... stop
          ... result
      ... from this
  ... terminate
```

and the corresponding GRAIL display is

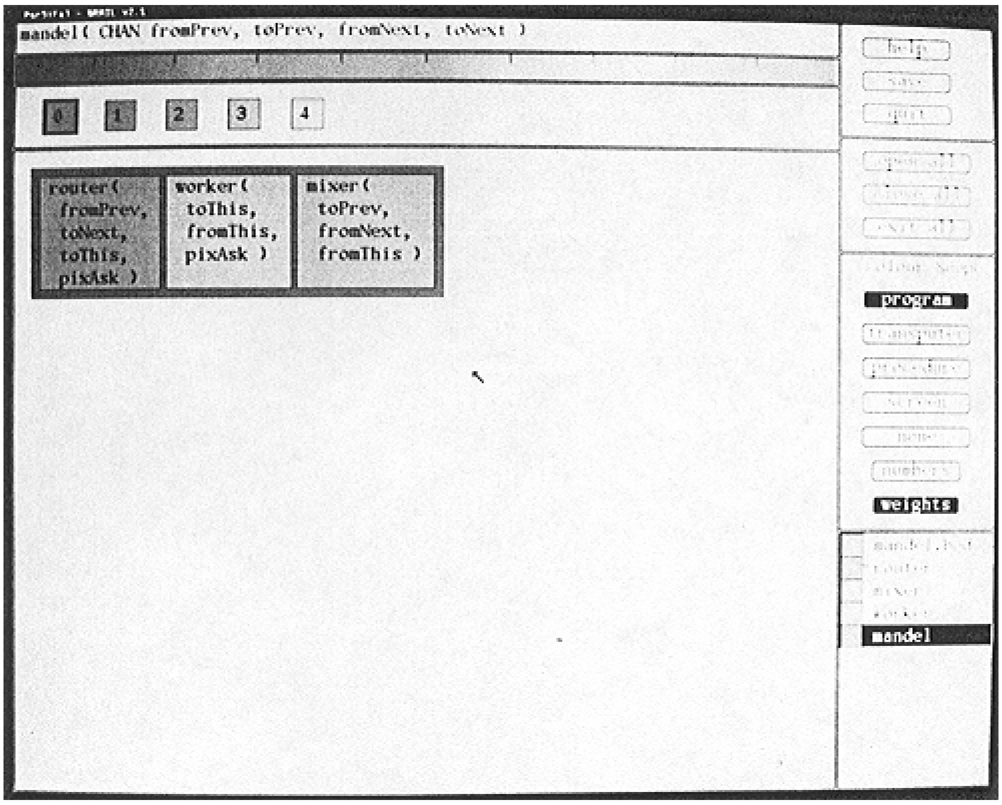


Here you can see the way GRAIL draws `IF`s and `ALT`s. Channels entering and leaving folds show up clearly.

4.2 Program Activity

This program has been run on a pipeline of Transputers, and has been profiled. The colour photographs are actual screen shots of GRAIL's display.

The first photograph shows the results for the program described above. There are various windows which show different information, and allow the user to interact with the display.



```
mandel( CHAN fromPrev, toPrev, fromNext, toNext )
```

```
router(          worker(          mixer(
  fromPrev,        toThis,          toPrev,
  toNext,          fromThis,        fromNext,
  toThis,          pixAsk )         fromThis )
  pixAsk )
```

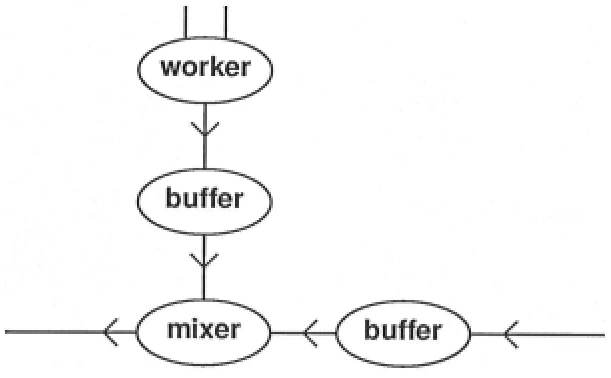
The top left window shows the name of the procedure (or entered fold) currently displayed. The one below it shows the colours used to represent activity, from blue (inactive) to red (most active). Below that is a window showing the total activity of each Transputer. The currently displayed Transputer, 0 in this case, is shown with a black border. Another Transputer can be made current by clicking the mouse over the relevant square. The large window shows the currently selected procedure (or entered fold) on the current Transputer. Other procedures can be selected, and folds entered or opened, by clicking the mouse over the relevant rectangle.

The smaller windows down the right hand side also allow interaction with the display. For example, the second one allows the user to open, close or exit all fold in the current procedure at one go.

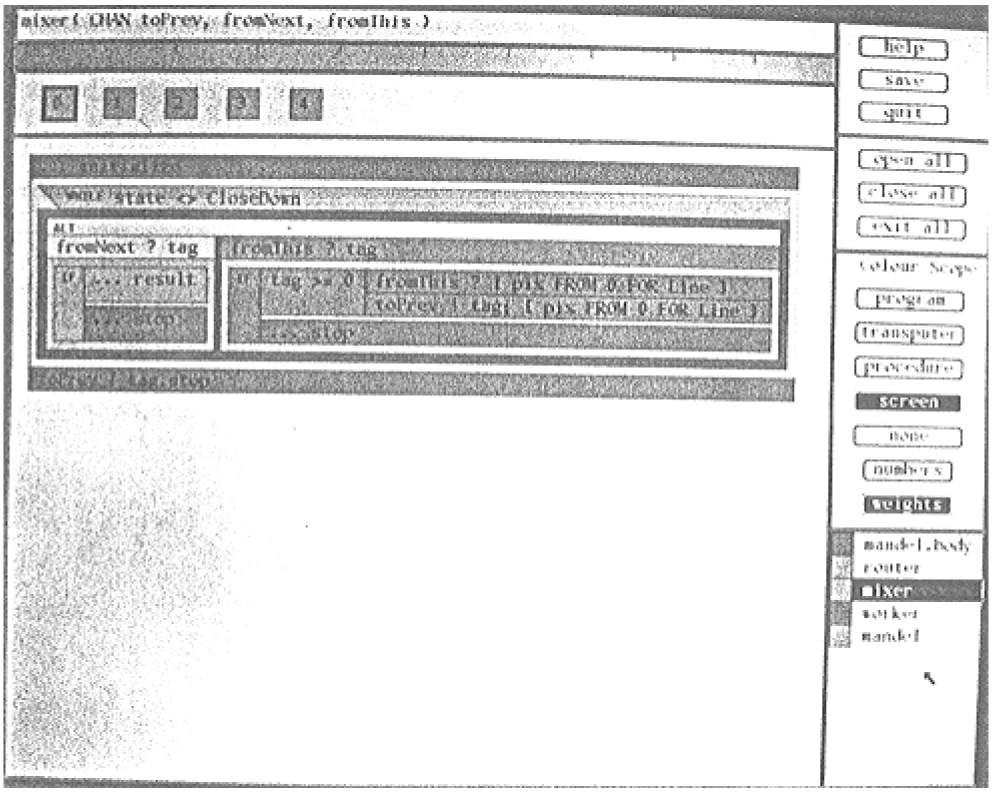
The third window changes the colour scope. Red corresponds to the "most active" process; this window allows that process to be the whole program, the process on the current Transputer, the current process, the entered fold being displayed on the screen, or none at all (to get rid of the colour).

The bottom right hand window is a procedure menu. The colours indicate the activity of each procedure on the current Transputer. The current procedure is indicated in reverse video.

Note how the Transputers at the end of the pipeline are "cooler", less red, than those near the beginning. They are less active, because they are being starved of work. The well known solution to this is to make the communications (`router` and `mixer`) take place at high-priority, the calculations (`worker`) take place at low priority, and add a `buffer` process to each input of `mixer`:



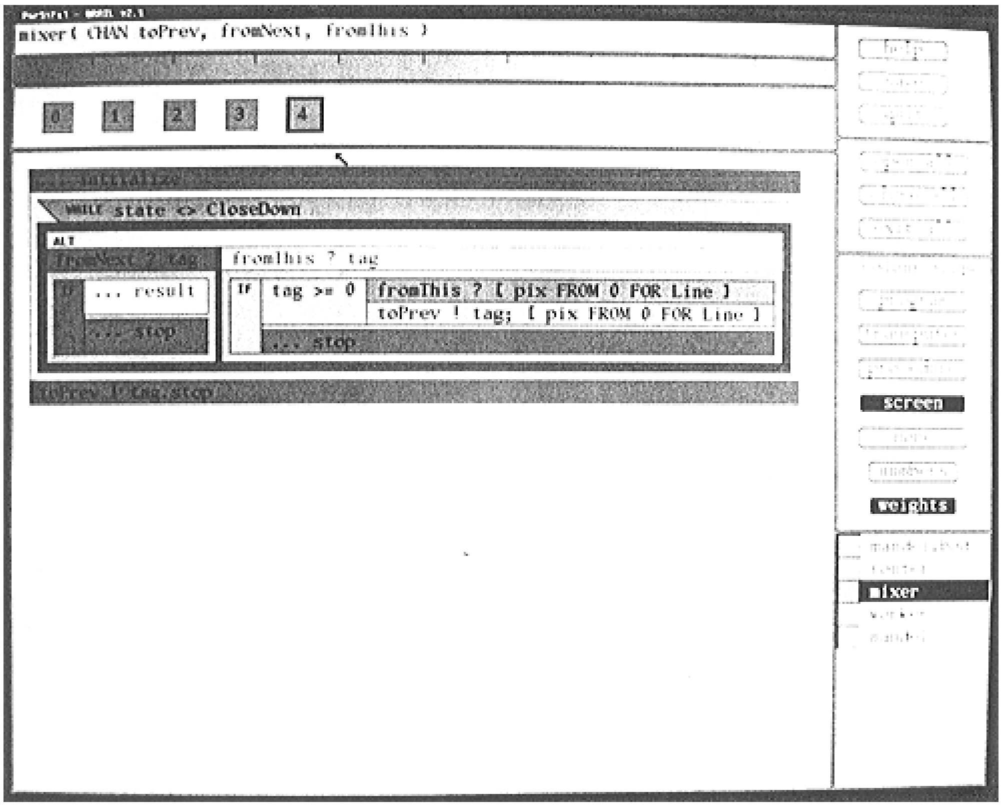
The next photograph shows the result of this.



```
mixer( CHAN toPrev, fromNext, fromThis )
... initialise
WHILE state <> CloseDown
  ALT
    fromNext ? tag
      IF
        ... result
        ... stop
    fromThis ? tag
      IF
        tag >= 0
          fromThis ? [ pix FROM 0 FOR Line ]
          toPrev ! tag; [ pix FROM 0 FOR Line ]
        ... stop
toPrev ! tag.stop
```

Now all the Transputers are glowing “red hot”. In this picture the current procedure is `mixer`, on Transputer 0. Notice how the branch of the `ALT` taking input from the next Transputer is much more active than that taking input from its own worker. Also, most of the work in this branch is being done in the “result” branch of the `IF`, rather than the “stop” branch (as would be expected).

The third photograph shows a different behaviour on Transputer 4, at the end of the pipe.



```
mixer( CHAN toPrev, fromNext, fromThis )
... initialize
WHILE state <> CloseDown
ALT
fromNext ? tag
  IF
    ... result
    ... stop
fromThis ? tag
  IF
    tag >= 0
      fromThis ? [ pix FROM 0 FOR Line ]
      toPrev ! tag; [ pix FROM 0 FOR Line ]
    ... stop
toPrev ! tag.stop
```

Now most of the work in the ALT is in the branch taking input from its own worker process. The only work done in the other branch is the receipt of the "stop" signal. There are no results (indicated by a white box), since there are no Transputers further down the pipe sending them on.

The procedure menu window (bottom right) shows the activity in the different procedures. Notice how on Transputer 4 the worker is green (more active) than the communication processes router and mixer; the opposite is true on Transputer 0.

Eventually, as the pipe gets longer, the first Transputer will be doing all routing and no work. The pipe is then saturated, and making it longer will not result in any speedup. The picture can give a crude estimate of when this will happen. It is not at five Transputers, however! For purposes of this example an artificially small packet size was chosen. GRAIL lets us experiment with different size work packets, and see how the comms/compute ratio changes.

5 Conclusions

GRAIL provides a way of looking at occam programs that highlights their parallel structure and communications. With a little practice these diagrams can be easier to understand than linear code.

Using colour to indicate activity allows potential troublespots and bottlenecks to be spotted early. The information gained can be used to make reasoned choices of allocation of processes to Transputers. GRAIL is integrated with ParSiFal's Topology Tool to help automate the generation of PLACE statements.

This method of displaying occam can have uses other than displaying activity. Ones that have been suggested are as an occam editor, and as an interface to a debugger.