1994

# Developing Modular Application Builders to Exploit MIMD Parallel Resources

C. Thornborrow
*University of Edinburg, Edinburgh Parallel Computer Center*

C. Faigle
*Syracuse University, Northeast Parallel Architectures Center*

# Developing Modular Application Builders to Exploit MIMD Parallel Resources

C. Thornborrow, A. J. S. Wilson

Edinburgh Parallel Computing Centre
University of Edinburgh
Edinburgh, Scotland EH9 3JZ

C. Faigle

Northeast Parallel Architectures Center
Syracuse University
Syracuse, NY.

## Abstract

*Modular application builders (MABs), such as AVS and Iris Explorer[6, 7] are increasingly being used in the visualisation community. Such systems can already place compute intensive modules on supercomputers in order to utilise their power. This paper details two major projects at EPCC which attempted to fully integrate the MAB concept with a distributed memory MIMD (DM-MIMD) environment.*

*The work presented was driven by two goals, efficient use of the resource and ease of use by programmer and end user.*

*We present a model of MABs and describe the major problems faced, giving solutions to them through two case studies.*

## 1 Introduction

Many modular application builders (MABs) have been built recently and their popularity is growing. It is often the case that the users of MABs wish to utilise parallel supercomputers within the environment. A few years ago, this was only possible by writing a special module that, typically, used socket connections to talk to a remote supercomputer and executed a single module on that platform. There is though, much demand for a closer integration of parallel supercomputers and networks of workstations running a MAB.

Recently, there have been developments in these areas. CM/AVS allows users of Thinking Machines CM5 Supercomputer to use array data types with AVS in a parallel manner. Modules may be written that describe data distribution in arrays and, if required, the layout of this data over the processors. Within an individual module, the support for parallel processing is purely SPMD. We define SPMD to mean many identical processes, potentially at different localities within their instruction streams, working on different areas of data.

There have been two projects with MABs at EPCC. The first aimed to produce a prototype library that could hook into an existing MAB. It was to allow use of parallel machines and facilitate distributed data models. The second was independent of the restraints of existing systems and was free to concentrate more on efficient use of resources in a more rigid model of computation. These two projects are described in detail in the following sections. Differences in the desired functionality of the systems, and the resulting implementations, are highlighted. Finally, conclusions and recommendations for further or repeat projects are presented.

## 2 Abstract model

In order to discuss current MABs we will first need to define our terminology. This is a new field with few accepted definitions. The following are for ease of discussion within this paper and are not, necessarily, generally accepted terms.

**MVE (Modular Visualisation Environment):** A package for data visualisation, consisting of a user interface allowing linking of modules in a pipeline. MVEs are a subset of MABs specifically geared to scientific visualisation.

**MODULE:** An entity which may be utilised in a pipeline. This can be thought of as a filter performing a function on input data to produce output data. It may also have a user interface. It may itself be a pipeline of modules.

**USER:** The person utilising the MAB. They may use the modules available or write new ones. A user who writes new modules is termed the writer.

**PIPELINE:** This is a set of modules and links between these modules. This implicitly defines the types of input, the functions performed on this input and the resulting types of output, e.g. an Explorer map definition.

**PARALLEL MODULE:** This is a module whose implementation consists of more than one concurrent process.

**TRIGGERING:** The firing of a module. This may be explicit with a given signal or implicit with certain inputs determining the firing.

**SYNCHRONISATION:** The method by which a MAB guarantees association of logical groups of data flowing through the pipeline.

### 2.1 Serial model of MABs

The MAB is considered to be a set of modules which can be connected to form pipelines. The function of the pipeline is implicitly defined by the function of each module and their connectivity. We refer to data flowing through the pipeline as meaning that it passes from one module to another in the order defined by the entry point and the links between modules. Typically there is a module in the pipeline, usually the last module, which renders the data.

There are two forms of parallelism in a serial MAB. The first, functional parallelism, is a facet of the fact that different modules may concurrently operate on different workstations. MABs are designed such that the user is encouraged to write modules that perform one function, in this way, different modules have differing functionality. Thus, we refer to a pipeline as having functional parallelism.

The second is the ability to 'hook' a parallel platform into a serial MAB. This is usually accomplished as described in the introduction to this paper. It is not a natural feature of the serial MAB and is considered serial as the data communication into and out of the modules is through one point and thus serial in nature.

### 2.2 Parallel environment

The parallel environment is considered to be a multiple instruction, multiple data surface with distributed memory (DM-MIMD). A parallel module is considered to be a collection of processes, each running identical code but, potentially, operating on a different area of the overall data. We refer to this as data parallelism.

There are two useful broad categories of communication within a parallel MAB utilising an SPMD model of programming. *Intramodule* is the communication type of two processes of the same module whilst *intermodule* is the communication type of two processes of different modules.

## 3 Serial to parallel migration

There are three main issues to be addressed when moving from the serial model of MABs to the parallel model.

**Data Distribution:** The method by which data is split up into smaller chunks for processing in parallel and then gathered again for further serial processing. Data distribution also covers the splitting and gathering of the data as it flows between parallel modules.

**Synchronisation:** The method by which the MVE implicitly triggers modules upon the arrival of data at the module. Keeping logically associated blocks of data (such as a 'frame' in apE [10]) associated as they flow through the pipeline is also part of synchronisation. These are intimately linked for it is usually the case that if frames of data become mixed or lost then it is due to incorrect triggering of modules within the pipeline. In the parallel environment, the sychronisation is complicated by the splitting of a module into processes. Data distribution must not lead to frames of data becoming seperated.

**Mapping:** The problem of how a description of module placement on hardware resources can be achieved. In other words, mapping covers the issues of how to describe and implement the parallelism of a module. The mapping of modules to resources will have a great influence on how efficiently the parallel pipeline will execute as a bad mapping may substantially increase the amount of message passing and decrease processor utilisation.

## 4 The MVE project

The MVE project at EPCC is a prototype library aimed at existing MVE systems to facilitate the use of DM-MIMD resources within such systems. The work was based on previous work done by Chris Thornborrow[1, 2]. The project firstly examined the design of existing systems in order to sublimate a model of the systems and thus attempt to address generic issues common to most systems. The library was named the NEVIS library[3]. There was, at development time, no existing MVE which was stable, freely available and small. It was thus necessary to write libraries, or wrappers, to imitate most of the common features of such systems. Into these prototype libraries, the NEVIS library would be slotted, thus demonstrating that, given a full system, DM-MIMD integration would be possible.

## 4.1 The design criteria

The two main criteria were ease of use and efficiency. Ease of use applies both for the writer and the user of the system. That the project was to fit within existing MVEs, or at least potentially do so, meant it had several other design criteria.

It was to be expected that seperate modules would be precompiled to executable *object code* and use libraries to communicate. The effect is that all the MVE project could aim to do was to extend the libraries. Existing systems encouraged the breakdown of functionally parallel units into seperate modules. It thus seemed natural to support an *SPMD* programming model within a module. The library must support the *triggering* rules of modules within existing MVEs and *multiple frames* of data. There should be *fan-in* and *fan-out* from as many input ports and output ports as required by the user.

In [2], Chris Thornborrow categorised the commonly occuring types of data in MVEs and demonstrated that these can be automatically distributed in a parallel environment. However, only arrays and single valued entities, known as parameters, were implemented in the prototype.

## 4.2 The efficiency criterion

The design should attempt to keep processors busy constantly. In order to keep processors busy, it is necessary to supply them with data as quickly as possible. A näive approach to data distribution within a parallel MAB is shown in diagram 1. It can be seen that the data must be gathered after each module and then split again for the next module. The task of the source is to divide data up amongst the workers and the task of the sink is to gather output data and make it into one coherent whole for the next modules source to distribute again. It is obvious there is a bottleneck and unnecessary inefficiency as parallel communication links are available.

To avoid this, it was decided to split the source and sink processes up, associating one process with each of the worker processes. In practice, it was possible to make these processes linked libraries and thus avoid context switching. The scheme is depicted in figure 2.

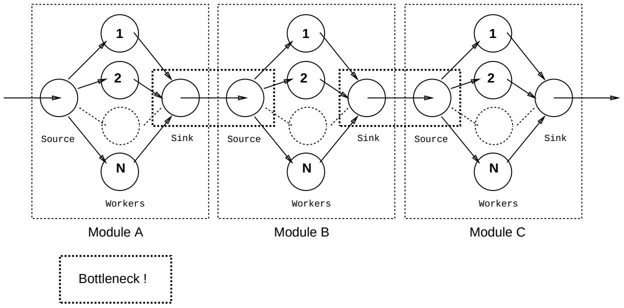

Figure 1: Three Parallel Modules Demonstrating Possible Bottlenecks

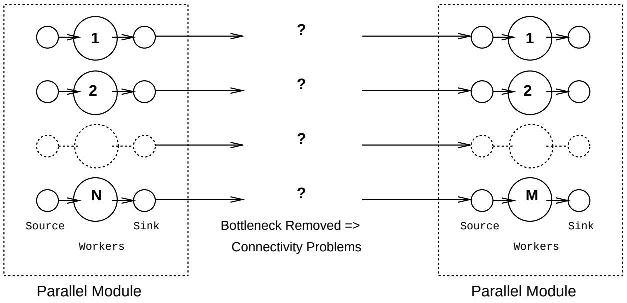

Figure 2: Splitting the Source and Sink

Thus each sink is responsible for splitting data from its associated process to forward to the correct sources of the next module. These sources then gather all such inputs from the previous module into one chunk of data. This gathering enables the data to be forwarded to the normal triggering routines of the MVE.

There are three interesting consequences of this scheme :

**Role Swapping:** The source and sink have now swapped roles. The sink library now splits data for the next module and the source library now gathers data into one chunk ready for forwarding to the worker process.

**Loose Synchrony:** As each process is essentially distinct from others in the same module, they trigger at slightly different times. This enables us to keep processors busier than in the näive approach.

**Synchronisation:** As long as the splitting and gathering of data work correctly, correct synchronisation of the module is guaranteed. This is tricky to see, but relies on the fact that the data that is forwarded to the existing MVE libraries in a serial form (i.e. each process only sees what it would, if it were running singly on a smaller data set).

In practice, the pipeline was found to execute correctly and to run faster with the distributed source and sink.

## 4.3 Ease of use criterion

It was decided the code should look as similar to serial code as possible. The number of processors was to be input by the user, not the writer who could only supply a default. Assigning numbers of processes to each module automatically is impossible without some knowledge of the data and the execution rate of modules as Thornborrow shows in [1], thus no automation was attempted. It was decided to attempt to hide as much message passing as possible.

**Intermodule:** The NEVIS library was designed to hide all such communication, as existing MAB wrappers hide socket connections between workstations. Thus, the library becomes part of the communications wrapper of an existing MVE.

**Intramodule:** The NEVIS library was not originally intended to deal with this, another project at EPCC, called the Parallel Utilities Library Key Technology Project (PUL)[5] developed libraries to support the parallel programming paradigms of regular domain decomposition and scattered spacial decomposition. These hide the message passing totally from the user.

Thus, using the PUL and NEVIS libraries in conjunction within the MVE project, *all* message passing could be hidden from the writer of parallel modules.

## 4.4 Module writers interface

It was possible to use few parallel calls. In the pseudo-code below, extra lines of code are highlighted with a plus sign at the start. Note that the line that indicates processing of the data has a code dependent number of message passing calls as processing the data requires. If the PUL libraries were utilised then there would be no message passing calls here.

```
main {
+  Initialise the Message Passing System
+  Initialise the NEVIS library
   Declare all the ports for this module

   loop until finished
           Receive from input ports
           nv_receive_ports(stderr);

           Read input port data
           Create header and data space for output data
           Fill in header fields
+          Process data
           Put data on output port

           Send output ports
           nv_send_ports(stderr);
   repeat

+  Exit NEVIS
+  Exit message passing system
}
```

## 4.5 Analysis of the MVE project

### Data distribution

Only arrays have to be mapped across processes in the prototype. There are two problems to be solved:

**Specification:** Suppose a module performs a reduction operation on a 2D data set, down to a 1D data set. If all the values in the Y axis are to be reduced to one value for each entry in the X axis, it makes little sense to split the data into square chunks across processes and introduce the need for communication of partial sums. Instead we would wish to assign strips of data to processes and avoid communication totally. To this end, the writer needs to specify the data distribution over processes. In the NEVIS library this was achieved by allowing the writer to specify which dimensions data should *not* be split across. This technique was used because the default is to split across all dimensions as equally as possible. The data split is decided dynamically, given the prefered type of data split by the writer and the number of processes by the user. A piece of code can be written that deterministically returns a mapping of data space to processes given the number of processes, the dimensions and size of the data in each dimension and the type of data split required, specified from a fixed set of possibilities.

**Re-mapping:** As data flows from one module to another, it is re-mapped to the correct processes in the next module. This is achieved by each module having knowledge about its immediate downstream modules. For each of these modules, the code described above is called with the size and dimensions of the data to be output, the number of processes of the downstream module and the prefered data split of the downstream module. This is fine if the size of the output data is known apriori. If it is not, then we effectively have 1D data (it is impossible to assign dimensions of data if we do not know their size). This does not mean the data has no spacial co-ordinates, simply that the data shape itself is 1D. Data may thus be split into chunks by each process individually and forwarded on a round-robin basis to the next module. If the number of processes of the modules is different, then each process would begin it's round robin distribution at a different process. This scheme allows loose synchrony of processes even when the data size is unknown, and, given small enough chunks of data, ensures a fairly even distribution of data in the downstream module.

### Synchronisation

It has already been shown that synchronisation is guaranteed by the correct operation of the splitting and gathering libraries. The standard firing libraries of the existing MVE may be utilised. Loose synchrony of the processes is achieved, which is useful for efficiency.

### Mapping

Some work has gone into automatic efficient mapping of processes and is detailed in [2]. This is still in the theoretical stage. In the prototype, mapping was one process per processor (excepting processes for the message passing and PUL libraries).

## 5 The Euphrates project

Euphrates is one of a series of collaborative projects between EPCC and the petroleum industry. A previous project developed a number of new 3D seismic processing techniques and implemented these on DM-MIMD platform [8]. The goal of the Euphrates project was to take these developments and present them in a form suitable for use in operating companies.

The approach taken was to build a system in which a scientist could interactively prototype a processing sequence, running small jobs to test that the desired effect was achieved, and then submit a batch job, which would then execute on a much larger, production, data set.

In order to achieve this within the time available it was decided to take advantage of existing MABs. A number of modules were prepared for such systems to allow users to interactively prototype their processing sequence which may then be saved in the map file format specific to that particular MAB. This is then translated into Euphrates Map Language (EML) a MAB-independent format. Finally, a batch job is created from this map description. This is achieved by generating source code which links against separate libraries which provide the required application functionality and a framework in which to parallelise this.

### 5.1 The design criteria

Once again the two main criteria for design were ease of use, for writer and user, and efficiency. Ease of use was particularly important as the system was for use by geoscientists who had no wish to become programmers, let alone parallel programmers. It must be possible to process very large seismic images (a typical 3D seismic survey of around 10,000 Km lines will occupy around 920 Gbytes). Infact, *arbitrarily large data* sets must be coped with. Modules do not communicate through libraries, but rather are linked, in sequence, from a single piece of source code, generated at the time the pipeline is instanced. Also, it was only necessary to support a *single frame* of data flowing through the system. It must be possible to have *fan-in* and *fan-out* between modules. The target architectures are networks of workstations such as Suns and RS6000s, and more closely coupled machines such as Meiko Computing surfaces, so the system had to be *portable*.

### 5.2 Implementation

Euphrates translates a description of an application in EML into an equivalent program for a DM-MIMD computer. In common with many geophysical applications, the functionality initially targeted featured processing of a regular mesh of values by finite difference operations. Previous work at EPCC indicated that the most effective method for parallelising such operations was to introduce data parallelism.

Clearly this task is too difficult to be tackled in the general case since this would amount to the construction of an all-purpose parallelising compiler for DM-MIMD computers. Instead the approach taken was to define a class of operations which would be supported and a module developers interface. Any operation which falls into this class, and which is programmed in accordance with the module developers interface, will be automatically, and correctly, parallelised by Euphrates.

### Characterising operations

Operations on meshes can be discussed in terms of the *task* which must be performed at each site on the mesh and the *perspective* of the operation as a whole. Three properties of tasks can be identified which are key indicators of the efficiency which can be expected from a parallel implementation. These properties are the *spatial dependence* of each task, which describes the dependence of a task at one site upon information from other sites; the *activity* of the operation which describes the distribution of tasks across the mesh; and *precedence* which describes the order in which tasks must be executed. Operations which have local spatial dependence, global activity and no precedence re-

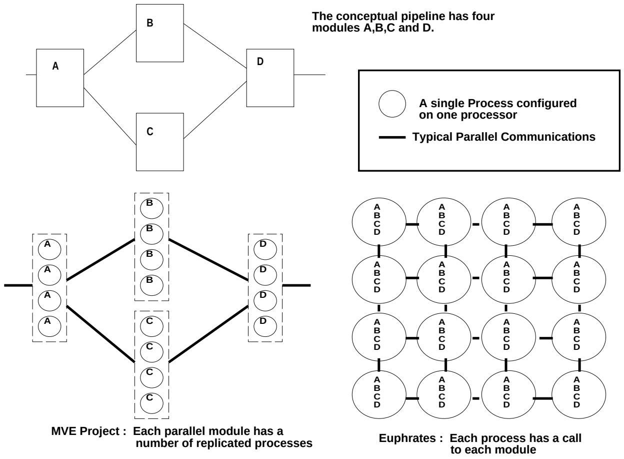
The conceptual pipeline has four modules A,B,C and D.
A
B
C
D
A single Process configured on one processor
Typical Parallel Communications
MVE Project : Each parallel module has a number of replicated processes
Euphrates : Each process has a call to each module

Figure 3: A Comparison of a 16 Processor Configuration of a 4 module Pipeline by the MVE project and Euphrates

lations are the easiest to parallelise and are termed *regular* since a regular geometric decomposition may be expected to provide an efficient parallelisation.

The perspective of an operation may be *global* or *local*. Global operations, such as a global sum or maximum, require all tasks to be completed before another operation can begin. This is not the case for operations with a local perspective and so a sequence of these operations may be applied to a large data set on a tile by tile basis allowing overlap of file I/O and computation and greatly reducing memory requirements.

## 5.3 The Efficiency criterion

Euphrates chose to tackle regular operations only, since these were the easiest to parallelise, and use could be made of the PUL library [5] which supports regular decomposition. Only fully occupied 3D rectangular meshes are currently supported, although voxels may have any aspect ratio.

Regular operations may be effectively parallelised by the SPMD approach. Euphrates follows this approach and, in order to minimise context switching, *combines the entire functionality of the pipeline into one application process per processor.* This approach is best illustrated in comparison to the MVE project approach and is depicted in figure 3.

### Providing arbitrary connectivity

The order in which calls to application modules are made is determined during the translation by analyzing the topology of the input map. Arbitrary connectivity can be achieved by ensuring that the modules are sorted into an order where each module only uses data output from a module which occurs earlier in the list. This process also checks for cyclicity; maps which contain feedback loops cannot be supported since there is no user interaction and such an application, once set running, would run forever.

## 5.4 Ease of use criterion

The module developers interface is remarkably simple. Developers are required to provide a C function call, and a plain text file which describes the arguments to this call. In this plain text file arguments are associated with ports and are described as either input or output, parameter or data. Parameters may be of any type, but arguments corresponding to data ports may only be of one type; a C data structure which describes an array, and an arbitrary region of this array. Modules must not alter input data arrays, and must write data to fill the indicated region of output data arrays.

In terms of our two categories of message passing found in SPMD code, we find the following:

**Intramodule:** The writer is forced to use the PUL libraries and thus no explicit message passing code need be written, simply calls to manipulate areas of data.
**Intermodule:** These calls cease to exist as they are implicit in the source code generated from the EML description of the pipeline. Data is simply passed as pointers from one C function call (module) to another.

## 5.5 Processing arbitrarily large images

The need to process large images with limited amounts of memory available at each node, and no virtual memory system, led to a decision not to support operations with a global perspective. Such operations would have required Euphrates to implement an equivalent to a global memory paging system for DM-MIMD systems. This was considered inappropriate since current and future DM-MIMD systems seem likely to based around commodity processors and will either have virtual memory systems on each node or have a global address space.

Instead Euphrates pre-calculates the memory requirements of an application in terms of the maximum number of images which will exist at any one time during one run of the application. Given this information, and an upper limit on the amount of memory available at each node, it is possible to process arbitrarily large images using regular operations with local perspective. This is achieved by dividing the input images into regions and then processing each region in turn. This decomposition is termed the *primary decomposition* in order to distinguish it from the *secondary decomposition* in which a primary region is distributed over a number of processors. Both the primary and secondary decomposition are performed at run-time, allowing the same DM-MIMD executable to run on a systems with varying numbers of processors and with varying amounts of memory per processor, without requiring retranslation or recompilation.

### 5.6 Analyzing Euphrates

By moving out of the MAB environment before parallelising, Euphrates sidesteps many of the complications that the MVE project has to deal with.

#### Data Mapping

Euphrates assumes that each module employs the same data distribution. This allows Euphrates to perform the primary decomposition and to manage all associated file I/O. Because the functionality of an entire application is combined into one process there is only one splitting and one gathering event per primary region.

#### Synchronisation

Issues of module triggering do not arise, since each module is executed only once and in a predetermined order. There is no triggering library level, as triggering is implicitly defined by the source code generated from the EML which calls each of the modules in turn. Similarly it is not possible for multiple frames of data to become disassociated, since only one frame of data is processed at a time.

#### Mapping

Mapping issues disappear completely since there is only one kind of application process and Euphrates assumes one application process will be placed on each processor. However, the user may easily modify this, so that a more powerful processor receives more than one process. Intramodule message passing is handled by the PUL libraries. These support non-blocking boundary exchange which enforces loose synchrony among the processes.

## 6 Results and conclusions

A model of MABs has been presented. Using this the issues relevant to parallelising an MAB have been discussed. Two projects at EPCC with differing design considerations, but both addressing these same issues have been presented.

### 6.1 Synchronisation

It was surprising to find that there appear to be few synchronisation problems when parallelising MABs. In the Euphrates project, this is a natural function of the fact that source code calls are used to trigger modules, which are in effect, simply libraries. In the MVE project, an extra layer of library was used to forward data to existing libraries that trigger modules upon data arriving or being present at a port. As long as the new library guarantees that it has gathered all data necessary to begin execution on a per process basis, then the processes may execute with loose synchrony.

### 6.2 Data distribution

It would seem that enforcing an SPMD model of parallel programming is both sufficient for the needs of writers and convenient for system builders. Assuming this, there are two factors, splitting of data and gathering. In the MVE project, it was discovered that as long as data sizes and shapes are known, together with the number of processes of each module then data distribution can be automated. The writer can give guidelines for splitting of data over processes within a module.

During the MVE project it became apparent that if the size of data to be output by a module is not known, then data distribution becomes complicated if we wish to maintain loose synchrony of the next module's processes. We suggested that an intelligent round-robin scheme of distribution be used to forward data to the next module or modules. When data ordering is important, such as the data for a histogram where each entry has a meaning depending on its position, we had to collecte a complete data set before forwarding to the next module when the PUL libraries were used.

### 6.3 Mapping

When dealing with SPMD programming models, there are two broad categories of message passing. Intramodule message passing cannot easily be controlled by the system. If intramodule data access patterns are of a fixed kind (as in the Euphrates project) then assumptions can be made and libraries developed to support this. The second category of message passing is intermodule. Again, if the patterns are known optimisations can be made, however, once again, if we are dealing with a general parallel MAB then the situation is less clear. Tests using a package called D3[9], written by Mike Norman of EPCC, strongly suggest that co-location of processes from different modules on the same processor would improve efficiency.

If the model of possible SPMD processing is limited, then great efficiency can be achieved by ordering the pipeline, removing cycles and then compiling down the pipeline into a single process which is then replicated once per processing element. This scheme presupposes that the data distribution is the same in each module.

### 6.4 Dynamic module addition

Neither the Euphrates project, nor the MVE project were dynamic in nature. In other words, once a pipeline was configured, it was impossible to add a module to the pipeline, or remove one. This is reminiscent of apE but neither AVS, nor Iris Explorer work in this way. In order to facilitate this within the MVE project, it would be necessary to use a dynamic message passing system, one that supported the creation and deletion of processes. Interestingly, MPI appears to have no support for this.

## 7 Future work

There are a number of issues left to resolve. The first problem for the MVE project is that it does not deal with arbitrarily large data sets. This was not a big priority because it was felt that future machines would have virtual memory at each node, or that there would be a global address space which would mean that the system as a whole might run out of memory but that an individual process would not.

It is not yet clear how co-location of processes of the MVE project would improve throughput. The prototype used a small number of processes for each module, but each had a dedicated processor. This meant that whenever data was split or gathered, it was sent in a message, rather than being a pointer passed from one module to another, as would be the case should two processes be co-located. This would reduce intermodule communication costs. However, the more co-location that occurs, the more competition there is for memory between the modules. Thus the modules themselves must be split into more processes, spread over more processing elements. This the intramodule and intermodule message passing occuring. A proper investigation of this would be useful.

## 8 Acknowledgements

The Euphrates team were Andrew Wilson, Gordon Cameron, Ian Flockhart and Chris Thornborrow. The project was managed by Nick Radcliffe. The MVE project was developed by Chris Thornborrow and Chris Faigle, who also wrote the NEVIS library code during a Summer Scholarship Programme placement at EPCC. The project was managed by Matthew White.

## References

[1] Chris Thornborrow *"Utilising MIMD Parallelism in Modular Visualisation Environments"*, Proceedings of 10th Eurographics U.K. Conference (1992)

[2] Chris Thornborrow *"EPCC-KTP-NEVIS-MVE-CONCEPTS"*, EPCC Technical Report (1992)

[3] Chris Faigle *"The NEVIS Prototype User's and Developer's Guide"*, EPCC Summer Scholarship Programme Report (1992)

[4] Andrew J. S. Wilson *"Euphrates-P Concepts Document"*, EPCC Internal Report (1992)

[5] Simon Chapple *"PUL-RD User's Guide"* EPCC Technical Report (1992)

[6] *"IRIS Explorer User's Guide (Beta Draft)"*, SGI Confidential Document (1991)

[7] *"IRIS Explorer Module Writer's Guide (Beta Draft)"*, SGI Confidential Document (1991)

[8] A.J.S. Wilson, M.G. Norman, and J.G. Mills *"Bodyscan: A Transputer Based 3D Image Analysis Package"*, EPCC Technical Report (1990)

[9] M. G. Norman *"A Parallel 3D Graphics Utility for Parallel Programs."* Applications of Transputers, Vol 1 (1990)

[10] The Ohio Supercomputer Graphics Project *"apE Version 2.0 Users Manual"*, Ohio State University (1990)