Multigrid aircraft computations using the OPlus parallel library

Paul I. Crumpton and Michael B. Giles [a] *

[a]Oxford University Computing Laboratory, Parks Road, Oxford, OX1 3QD, UK

This paper presents the OPlus library which is a flexible library for distributed memory parallel computations on unstructured grids through the straightforward insertion of simple subroutine calls. It is based on an underlying abstraction involving sets, pointers (or mappings) between sets, and operations performed on sets. The key restriction enabling parallelisation is that operations on a particular set can be performed in any order.

The set partitioning, computation of halo regions, and the exchange of halo data as required is performed automatically by the OPlus library after the user specifies the sets and pointers. A single source OPlus application code can be compiled for execution on either a parallel or a sequential machine, greatly easing maintainability of the code.

The capabilities of the library are demonstrated by its use within a program for the calculation of inviscid flow over a complete aircraft using multigrid on a sequence of independent tetrahedral grids. Good computational efficiency is achieved on an 8-processor IBM SP1 and a 4-processor Silicon Graphics Power Challenge.

1. INTRODUCTION

Algorithms for unstructured grids are becoming increasingly popular, especially within the CFD community where the geometrical flexibility of unstructured grids enables whole aircraft to be modelled. The resulting calculations are often huge and so there is a need to fully exploit modern distributed memory parallel computers. Writing an individual, machine-specific parallel program can be time consuming, expensive and difficult to maintain. Therefore there is a need for tools to simplify the task and generate very efficient parallel implementations. This paper describes the development of OPlus (Oxford Parallel library for unstructured solvers), a FORTRAN subroutine library which enables the parallelisation of a large class of applications using unstructured grids, removing the parallelisation burden from the application programmer as far as possible [1].

In the design of the library emphasis was placed on the following aspects:

generality: OPlus uses general data structures. In a CFD application, for example, it allows cell, edge, face and other data structures, The cells could also be of any type, such as tetrahedra, prisms or hexahedra.

performance: Messages are sent only when data has been modified and are concatenated to reduce latency. Also, local renumbering is used to improve the cache performance on RISC processors.

*Research supported by Rolls-Royce plc, EPSRC and DTI

single source: A single source code can be executed either sequentially (without any message-passing of other parallel library) or in parallel, with identical treatment of disk and terminal i/o. This greatly simplifies development and maintenance of parallel codes.

This paper will first describe the concepts behind the OPlus framework, and then various aspects of the implementation. Finally some results are presented for an application code modelling the inviscid flow over an aircraft using multiple tetrahedral meshes. A companion paper discusses the use of the distributed visualisation software pV3 which was a vital tool in this work [5].

The PARTI library developed by Das *et al* [7,6] has similar objectives in dealing with parallel computations on generic sets. There are a number of detailed differences between PARTI and OPlus but the principal difference is that with OPlus the programmer is not aware of the message-passing required for the parallel execution. This greatly simplifies the programmer's task. PARTI has the same long-term objective but the aim is to achieve it through the incorporation of PARTI within an automatic parallelising compiler. At present, the programmer must still explicitly specify the message-passing to be performed.

2. OPlus LIBRARY

2.1. Top level concept

The concept behind the OPlus framework is that unstructured grid applications have three key components.

sets Examples of sets are nodes, edges, triangular faces, quadrilateral faces, cells of a variety of types, far-field boundary nodes, wall boundary faces, etc. Data is associated with these sets, for example the grid coordinates at nodes, the volumes of cells and the normals on faces.

pointers The connectivity of the computational problem is expressed by pointers from elements of one set to another. For example, cell to node connectivity could define tetrahedra, and face to node connectivity would define the corresponding faces.

operations over sets All of the numerically-intensive parts of unstructured applications can be described as operations over sets. For example, looping over the set of cells using cell–node pointers and nodal data to calculate a residual and scatter an update to the nodes, or looping over the nonzeros of a sparse matrix accumulating a matrix-vector product.

The OPlus framework makes the important restriction that an operation over a set can be applied in *any* order without affecting the final result. Consequently, the OPlus routines can choose an ordering to achieve maximum parallel efficiency. This restriction prevents the use of OPlus for certain numerical algorithms such as Gauss–Seidel iteration or globally implicit approximate factorisation time-marching methods. However, most numerical algorithms on unstructured grids in current use in CFD, and many other application areas, satisfy this restriction. Specific examples include explicit time-marching methods, multigrid calculations using explicit smoothing operators and conjugate gradient methods using local preconditioning.

Another current restriction is that the sets and pointers are declared at the start of the program execution and must then remain unaltered throughout the computation. Therefore, dynamic grid refinement cannot be treated at present. This is an area for future development.

2.2. Parallelisation approach

The implementation uses a standard data-parallel approach in which the computational domain is partitioned into a number of regions and each partition is treated by a separate process, usually on a separate processor.

The communication requirements between partitions arise because of the pointer connectivity at the boundaries between partitions. An illustrative example is matrix-vector multiplication for a symmetric sparse matrix:

$$y_i = \sum_j A_{ij} x_j$$

If the matrix A is symmetric, then defining an edge k to correspond to nodes i_k, j_k for which $A_{ij} \neq 0$, the product can be evaluated by the following algorithm:

```
For all nodes $i$,   $y_i := 0$

For all edges $k$,   $y_{i_k} := y_{i_k} + A_k x_{j_k}$
                     $y_{j_k} := y_{j_k} + A_k x_{i_k}$
```

Expressed in FORTRAN this algorithm becomes

```
      DO I = 1, NNODES
        Y(I) = 0.0
      ENDDO
C
      DO K = 1, NEDGES
        I    = P(1,K)
        J    = P(2,K)
        Y(I) = Y(I) + A(K)*X(J)
        Y(J) = Y(J) + A(K)*X(I)
      ENDDO
```

The integer array P is the pointer table defining the edge to node connectivity. Note that operations on edges can be performed in any order and the final result will remain the same, so this example satisfies the restriction required by the OPlus framework.

In the data parallel approach the edges and nodes are partitioned so that each individual edge or node *belongs* to only one partition. There is no difficulty in performing the edge operations when the edge and its two nodes belong to the same partition. When more than one partition is involved the approach used is to perform the edge operation on

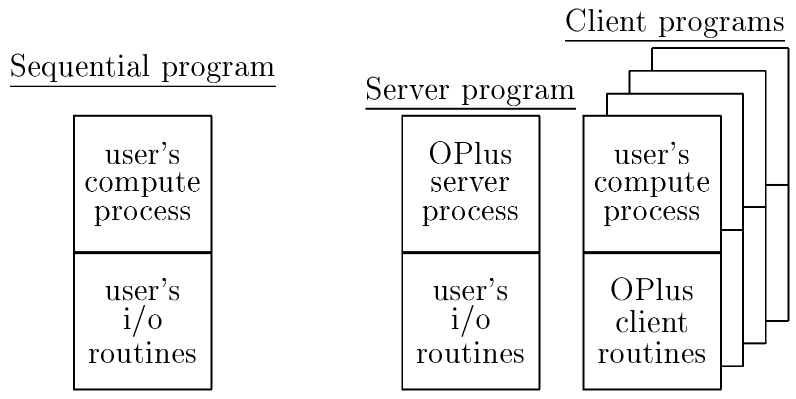

Figure 1. Sequential and parallel versions of user's program

each partition whose owned data is affected by the operation. In this case that means performing the calculation for each partition owning one of the two nodes. To carry out this operation, temporary copies must be obtained of the unowned data from the other node and/or the edge belonging to a different partition; this is commonly referred to as *halo data*.

2.3. Software architecture and communications

A key design goal for the OPlus framework was to allow users to write a *single source code* which will execute either sequentially or in parallel depending only on the library to which it is linked. Moreover, the sequential and parallel execution should result in identical disk and terminal i/o. To achieve this it was necessary to adopt the program structure shown in Figure 1 in which all disk and terminal i/o is handled via subroutines with a specified interface.

For sequential execution the user's main program is linked to user–written subroutines which handle all i/o. This enables the user to develop, debug and maintain their sequential code without any parallel message passing libraries.

For parallel execution the OPlus framework creates server and client programs from the user's single source. The server program is formed by linking the OPlus server process to the user's i/o routines, while the client program is created by linking the user's compute process to the OPlus client routines. When the client routines collectively need to input an array of data from disk, a request is passed to the server process; it reads the data from disk and passes to each client its piece of the array corresponding to its partition of the overall problem.

The communication between the client and the server is performed using PVM 3.3 to allow the server to be on a different type of machine from the clients. For the critical client-client communication in the main parallel computation phase, BSP FORTRAN is employed. This is a FORTRAN library with strong similarities to the `shmem_put` and `shmem_get` directives for communication on the CRAY–T3D. It has been implemented on a wide range of machines using the most efficient communication method in each case, e.g. MPL on the IBM SP1/SP2 and shared-memory pages on the Silicon Graphics Power Challenge [8].

2.4. Initialisation phase

At the beginning of the application program the user declares the sets and pointers to be used in the application. At the beginning of the parallel execution these are then used in the following key steps:

partitioning All sets are partitioned. Simple recursive inertial bisection is used to initially partition one or more sets. The other sets are partitioned consistently using the connectivity information contained in the pointers; see [5] for examples of inherited partitioning.

construction of import/export lists The initialisation phase constructs, for each partition, lists of the set members which may need to be imported during the main execution phase. Correspondingly, each partition also has export lists of the owned data which may need to be imported by other partitions.

local renumbering Each partition should only need to allocate sufficient memory to store the small fraction of each set which it either owns or imports. To enable this, it is necessary to locally renumber the set members. This local renumbering of each set forces a consistent renumbering of all of the pointer information. The local-global mapping is also maintained for i/o purposes.

It is important to note again that all of the above phases are performed automatically by the OPlus library, not the application code. In all applications performed to date, the CPU time taken for these initialisation phases has been significantly less than the time required for the disk i/o, and so is considered to be negligible.

2.5. Computation phase

The heart of a parallel OPlus application is a DO-loop carrying out in parallel operations on a distributed partitioned set. Continuing with the example of the sparse matrix-vector product, using the OPlus library the FORTRAN code for the main edge loop is:

```
  DO WHILE(OP_PAR_LOOP(EDGES,K1,K2))
    CALL OP_ACCESS('read'  ,X,1,NODES,P,2,1,1,1,2)
    CALL OP_ACCESS('update',Y,1,NODES,P,2,1,1,1,2)
    CALL OP_ACCESS('read'  ,A,1,EDGES,0,0,1,1,0,0)
C
    DO K = K1, K2
      I    = P(1,K)
      J    = P(2,K)
      Y(I) = Y(I) + A(K)*X(J)
      Y(J) = Y(J) + A(K)*X(I)
    ENDDO
  END WHILE
```

The purpose of the OP_ACCESS calls is to inform the library which distributed arrays are being used within the DO-loop, which sets they are associated with, which pointers are being used to address those sets and the type of operation (read, write or update) being undertaken with the data. This is the information needed by the library to decide which data must be imported from neighbouring partitions. Full details of the arguments of OP_ACCESS and the other OPlus routines are available [2].

The logical function OP_PAR_LOOP controls the number of times execution passes though the interior of the DO WHILE loop. The first argument declares that the operations are to be performed over the set of edges, and the second and third arguments set by the function are the start and finish indices of the inner loop. For sequential execution, there is just one pass through the DO WHILE loop with K1 and K2 set to 1 and NEDGES respectively. For parallel execution there are a number of preliminary passes through the DO WHILE loop during which K1 is set to a higher value than K2 so the inner DO loop is skipped; these passes process the information in the OP_ACCESS calls and export the necessary halo data to neighbouring partitions. Next comes a single pass through the DO loop performing those calculations which do not depend on halo data. There are then a number of passes which receive the incoming imported data from neighbouring partitions but perform no calculations, and finally there is an execution pass which performs the computations that do depend on the halo data. In this way it is possible to overlap interior computations with the exchange of halo data, but so far this overlapping has not yielded significant benefits on any of the machines tested.

A point to emphasise is that the lines of FORTRAN which form the contents of the inner DO loop have not changed from the original sequential code. In this trivial example the number of OPlus subroutine calls which have been added is comparable to the number of lines of application code, but in a real application there could be a hundred lines or more of FORTRAN inside the DO loop and it is crucial that this does not have to be changed.

3. MULTIGRID AIRCRAFT COMPUTATION

The utility and efficiency of the OPlus library is illustrated here by its use for the computation of the steady inviscid flow around a complex aircraft geometry. The CFD algorithm uses a multigrid procedure based on a Lax-Wendroff solution algorithm [3,4]. In this application five tetrahedral grids are used. The number of cells varies from 750k on the finest grid to 28k on the coarsest. The surface triangulation of one of the grids is shown in Figure 2 together with the final surface pressure contours. Previous research [3] showed that a W-cycle multigrid iteration is twice as fast as a V-cycle iteration, and so a W-cycle iteration is used in this work. However, this presents a great challenge for parallel efficiency because of the very large number of iterations performed on the coarsest grids which have relatively more communication and redundant computation.

On each of the five grids there are four sets, tetrahedra, nodes, boundary faces and boundary nodes. For each grid there are pointers from the tetrahedra, boundary faces and boundary nodes to the regular nodes. There are also pointers between grid levels, giving for each node the four nodes of the enclosing tetrahedra on the finer and coarser grids. These are required for the transfer and interpolation operations within the multigrid

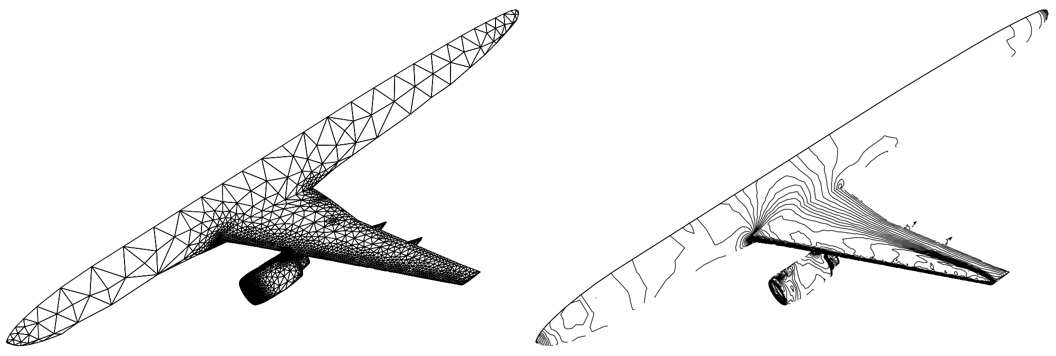

Figure 2. One of the grids used for the aircraft computation and computed contours of surface pressure

procedure.

Calculations were performed on an 8 node distributed memory IBM SP1 and a 4 processor shared memory SGI Power Challenge. The elapsed times, in seconds per multigrid cycle, and the corresponding speedup over the single processor performance, S, is given in the following table.

		IBM SP1		SGI PC	
proc	$S_{\max}$	time	S	time	S
1	1.0	1006	1.0	419	1.0
2	1.9	556	1.8	216	1.9
3	2.7	384	2.6	149	2.8
4	3.5	310	3.2	116	3.6
5	4.1	270	3.7	—	—
6	4.8	234	4.3	—	—
7	5.4	211	4.8	—	—
8	6.1	190	5.3	—	—

The maximum achieveable speed-up, $S_{\max}$, is defined as the ratio of the total sequential work to the maximum work performed on any of the OPlus clients,

$$S_{\max} = \frac{\text{sequential work}}{\text{max slave work}}.$$

This is the speedup which would be achieved in the complete absence of communication costs. There are two reasons why it does not increase linearly with the number of processors. One is that it is difficult to achieve load-balancing on the coarser grid levels; the grid nodes may be load-balanced but the boundary faces may not be. The other is that redundant computations are performed at the interfaces between partitions; the proportion of these becomes larger on the coarser grids. Nevertheless, very good parallel execution speed up has been achieved for a highly complex application which is in many ways a worst case in that it employs the W-cycle multigrid iteration which is known to be a challenge to efficient parallel execution. Note that the factor of two saved by the use of W-cycle multigrid compared to V-cycle multigrid is still much greater than any loss of parallel efficiency associated with the increased number of iterations on the coarse grid levels.

4. CONCLUSIONS

A flexible and general library has been developed to parallelise a large class of unstructured grid applications. The programmer specifies the sets and pointers to be used in the application and the library determines an appropriate partitioning for data-parallel execution. The transfer of halo data is performed automatically by the library given the programmer's specification of the data being used in operations performed on the members of the distributed sets. The resulting single source code will execute on a sequential machine without the need for any parallel libraries, or in parallel on a MIMD architecture.

The use of the OPlus library has been demonstrated for a multigrid computation of the inviscid compressible flow over a complete aircraft configuration. For this realistic industrial application good parallel efficiency has been achieved with very little effort from the application programmer.

REFERENCES

1. D.A. Burgess, P.I. Crumpton, and M.B. Giles. A parallel framework for unstructured grid solvers. In S. Wagner, E.H. Hirschel, J. Périaux, and R. Piva, editors, *Computational Fluid Dynamics '94. Proceedings of the Second European Computational Fluid Dynamics Conference 5-8 September 1994 Stuttgart, Germany*, pages 391–396. John Wiley & Sons, 1994.
2. P. Crumpton and M. Giles. OPlus programmer's guide, rev. 1.0. 1993.
3. P. Crumpton and M.B. Giles. Aircraft computations using multigrid and an unstructured parallel library. AIAA Paper 95-0210, 1995.
4. P.I. Crumpton and M.B. Giles. Implicit time accurate solutions on unstructured dynamic grids. AIAA Paper 95-1671, 1995.
5. P.I. Crumpton and R. Haimes. Parallel visualisation of unstructured grids. In S. Taylor, A. Ecer, J. Periaux, and N. Satofuka, editors, *Proceedings of Parallel CFD'95, Pasadena, CA, USA 26–29 June*, 1995.
6. R. Das, D.J. Mavriplis, J. Saltz, S. Gupta, and R. Ponnusamy. Design and implementation of a parallel unstructured Euler solver using software primitives. *AIAA Journal*, 32(3):489–496, 1994.
7. R. Das, J. Saltz, and H. Berryman. A manual for PARTI runtime primitives, revision 1. Technical report, ICASE, NASA Langley Research Centre, Hampton, USA, 1993.
8. R. Miller. A library for bulk synchronous parallel programming. In *Proceedings of the BCS Parallel Processing Special Interest Group Workshop on General Purpose Parallel Computing*, Dec 1993. http://www.comlab.ox.ac.uk/oucl/oxpara/ppsg.html.