Structured Parallel Programming: Parallel Abstract Data Types

Hing Wing To

Imperial College, London

hwt@doc.ic.ac.uk

February 1997

Motivation

To write parallel programs over irregular data structures, e.g. graphs and trees, which:

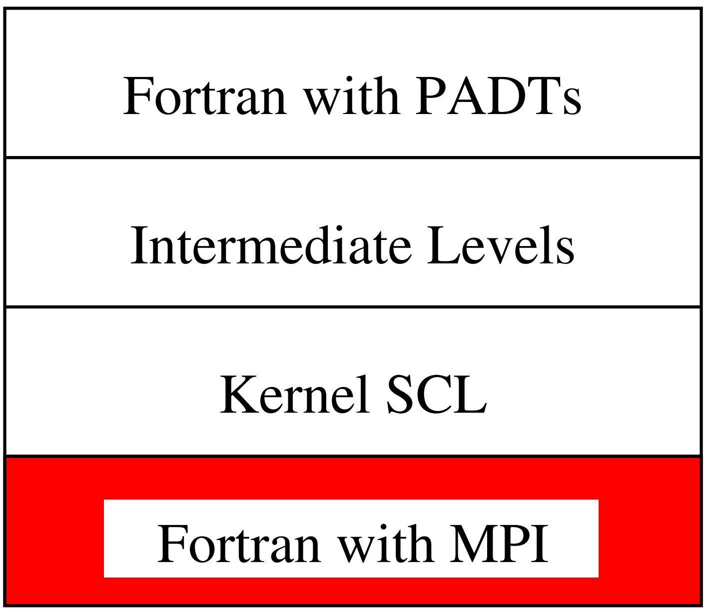
- are expressive and natural
- have an optimise implementation

Solution

Exploit the extensibility of our Structured Co-ordination Language (SCL).

Build parallel abstract data types (PADTs) on top of the existing language.

Avoid building new primitives into the language.



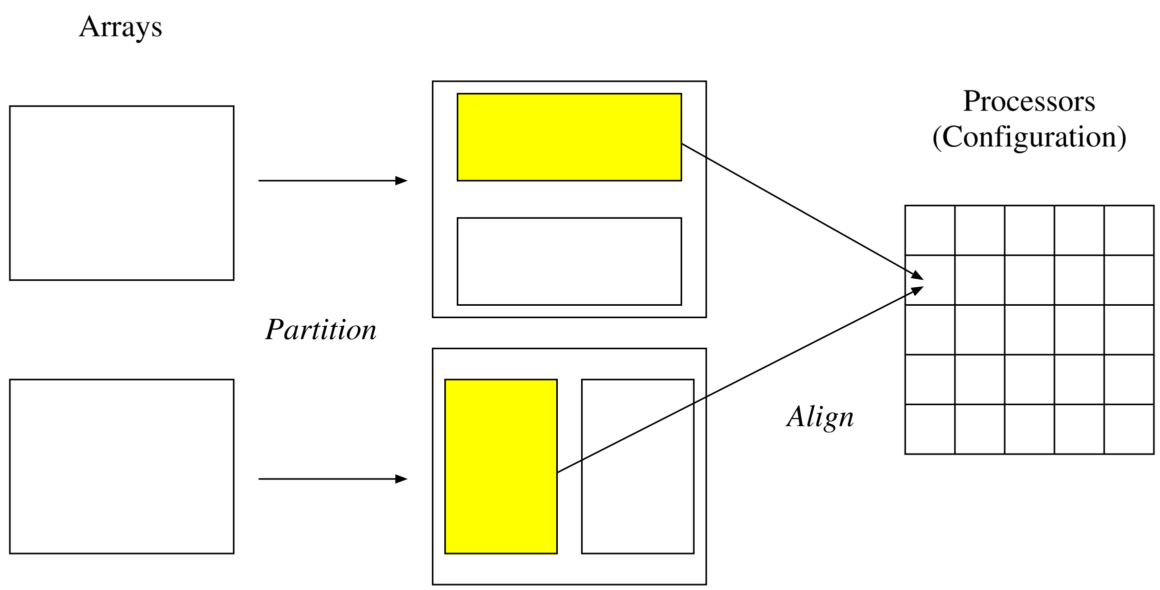
Structured Co-ordination Language—SCL

A structured language for co-ordinating the concurrent activities of Fortran code.

A parallel program has two levels:

- SCL = parallel behaviour
- Fortran = lower-level sequential computation

Basic parallel data structure is the array:




```
distribution f g A B =
   align (partition f A) (partition g B)
```

An example

Parallel matrix multiply:

```
const l,m,n = 6
const p = 2
real, dimension(l,m) :: A
real, dimension(m,n) :: B
progroup, dimension(p,p) :: shape

mm_block_ver A B
   =  iocall check_res C
      where
      dl = iocall init (A,B)
      fl = ((blk,blk), (blk,blk), (blk,blk))
      dist = rotation df1 (distribution fl dl)
      <da, db, dc>  = iterfor p step dist
      C = gather (blk,blk) dc

step i <da, db, dc>
   = spmd gf seq_mm <da, db, dc>
     where
     gf x = rotation df2 <da, db, x>

rotation df <da, db, dc>
   = <da1, db1, dc>
```

```
where
da1 = rotate_2 1 df da
db1 = rotate_2 2 df db
```

```
|| Fortran code :

     spf_subroutine init(X,Y,Z)
     spf_subroutine seq_mm(X, Y, Z)
     spf_subroutine check_res(C)
     spf_function df1(rcnum)
     spf_function df2(rcnum)
```

Irregular Data Structures

Kernel SCL is very appropriate for structuring regular computations over arrays.

It is less suitable for coding programs with irregular data structures.

Encoding irregular data structures as arrays is error prone and complex, but gives good performance.

Solution: Exploit ability to extend SCL by building irregular data structures on top of kernel SCL.

global name space operators

↓

distributed name space operators

↓

distributed name space operators with sharing made explicit

↓

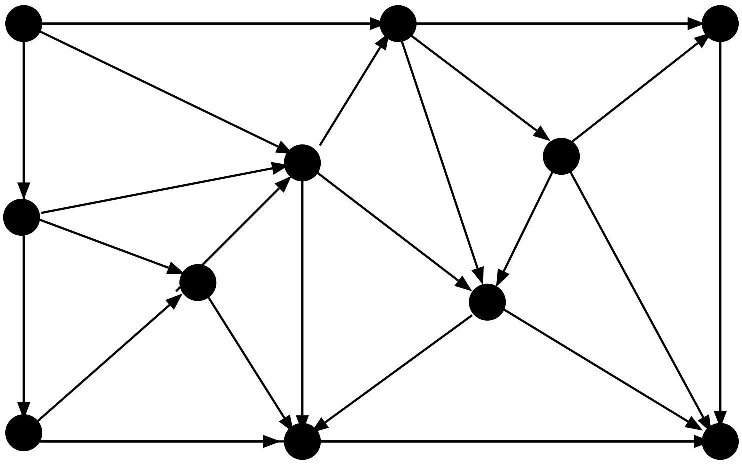
encoded into distributed arrays in kernel SCL

Euler Example

Take fragment of code from an unstructured Euler solver used to study the flow of air over an airfoil.

Physical values are associated with each mesh vertex.

Euler Solver: Global Address Space



Sequential Fortran code:

```
do i = 1, numEdges
  node1 = edge(i, 1)
  node2 = edge(i, 2)
  y(node1) = y(node1) + f(x(node1), x(node2))
  y(node2) = y(node2) + g(x(node1), x(node2))
end do
```

Parallel Abstract Data Type for Mesh

Parallel abstract data type for mesh:

```
Mesh  :: ( Nodes, Edges )
Nodes :: { ( Key, (x_value, y_value ) )}
Edges :: { ( Key, Key ) }
```

Define operators over mesh PADT:

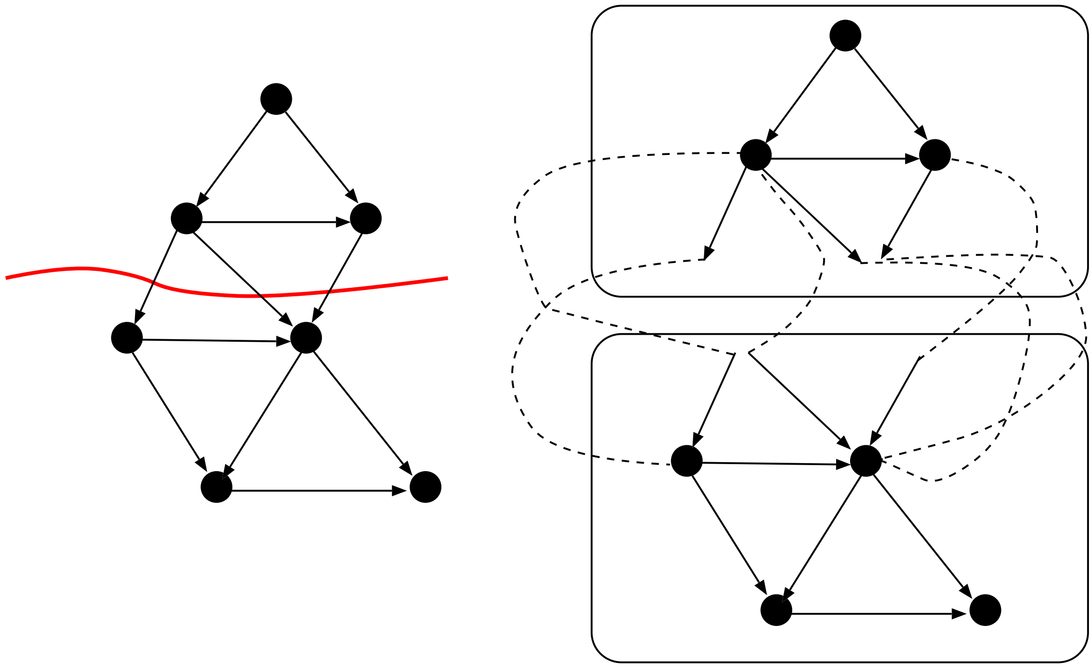
```
M_map f ( N, E) =
  ( { (k, f x) | (k, x) ∈ N }, E )


M_LNO inSum outSum inWeight outWeight (N, E) =
  ( { (k, update k x) | ( k, x) ∈ N }, E )
  where
    update k x =
      l_fold (l_fold x inSum (inCont k x))
           outSum (outCont k x)
    inCont k x = { inWeight(N(k'), x)
                 | (k', k) ∈ E }
    outCont k x = { outWeight(x, N(k'))
                  | (k, k') ∈ E }
```

The fragment of the Euler solver is then:

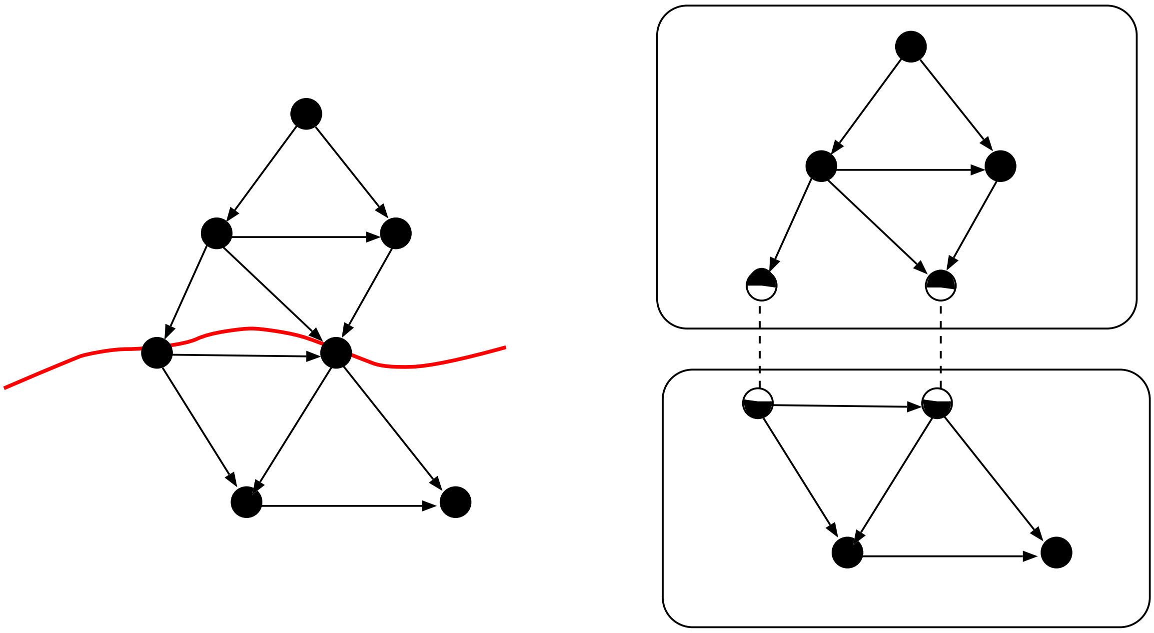
```
Euler a_mesh = M_LNO ADD ADD G F a_mesh

subroutine ADD (x, y, z)
real, intent (input) :: x
real, intent (input) :: z
real, intent (inoutput) :: y
y = z + y

subroutine G (x0, y0, x1, y1, z)
real, intent (input) :: x0
real, intent (input) :: y0
real, intent (input) :: x1
real, intent (input) :: y1
real, intent (output):: z
z =  g (x0, x1)

subroutine F (x0, y0, x1, y1, z)
real, intent (input) :: x0
real, intent (input) :: y0
real, intent (input) :: x1
real, intent (input) :: y1
real, intent (output):: z
z =  f (x0, x1)
```

Distributing the Mesh

1. Distribute by nodes with shared edges.



2. Distribute by edges with shared nodes.



PADT for Distributed Mesh

New data type for partitioned mesh:

```
P_Mesh :: { (Partition_Id, Mesh) }
```

Operator from mesh to partitioned mesh:

```
Partition_Mesh :: Strategy -> Mesh -> P_Mesh
```

Edges must be transformed to refer to other partitions where appropriate.

Operators over new PADT:

```
PM_map f M = { (i, f m) | (i, m) ∈ M }
```

By data type transformation:

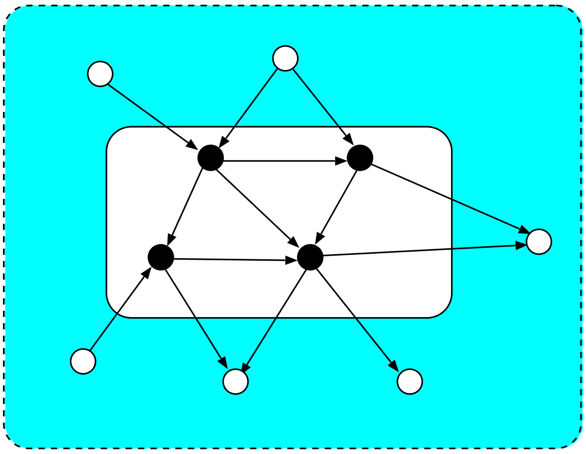
```
Euler' a_p_mesh =
  PM_map (M_LNO ADD ADD G F) a_p_mesh
```

Implies that sequential abstract data type must be provided.

References to non-local node values can then be handled in ADT.

Distributed Version with Halo

Aim: Remove non-local references in local computation.



New data type for partition mesh with halo:

```
HP_Mesh :: { (Partition_Id, Mesh) }
```

Operator from partitioned mesh to partitioned mesh with halo:

```
Add_Halo :: P_Mesh -> HP_Mesh
```

Edges must be transformed to copies in halo where appropriate.

Operators over new PADT:

```
HPM_map f M =
  { (i, (update (nodes, edges) ∪ halo, edges)
   | (i, (nodes, edges) ∈ M }
  where
   update (nodes, edges) =
     fst (f ( part i nodes, edges))
   part i nodes =
     { n | n ∈ nodes, n ∈ domain(i) }
   halo i nodes =
     { n | n ∈ nodes, n ∉ domain(i) }
```

`domain` returns the domain of the nodes in the partition without halo.

By data type transformation on `Euler'`:

```
Euler'' a_hp_mesh =
  (update_halo o
   HPM_map (M_LNO ADD ADD G F)) a_hp_mesh
```

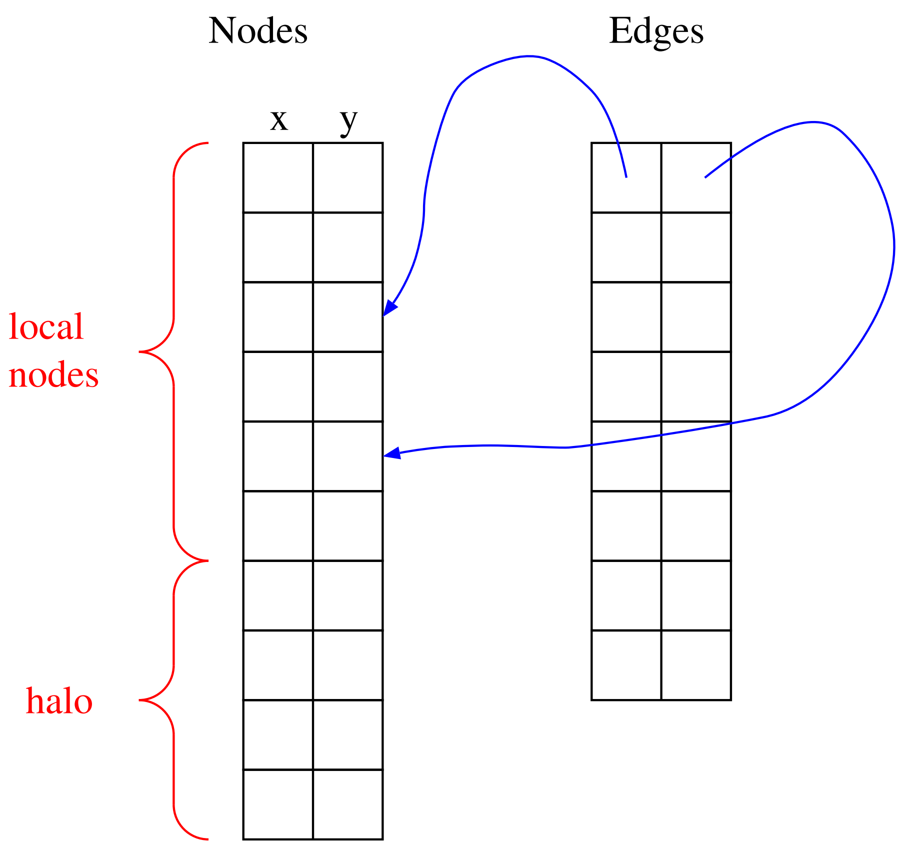
Note that `M_LNO` no longer will refer to non-local node values.

Could use Fortran code instead of `M_LNO`.

Concrete Representation

Set of distributed meshes can be represented by distributed array in SCL.

```
HP_Mesh :: ParArray Index Mesh

Mesh :: (SeqArray Index (x_value, y_value),
         SeqArray Index (Int, Int))
```



`Partition_Mesh` and `Add_Halo` can be implemented using `partition`.

Conclusions

Presented an approach which enables irregular applications to be written:

- which is natural and expressive
- without extending primitives of the basic language
- with high performance

Could apply the approach to HPF-1.

There are still problems with the interface with Fortran.

Is the current kernel a sufficiently rich base to build all useful PADTs?