COMMUNICATIONS IN NUMERICAL METHODS IN ENGINEERING
Commun. Numer. Meth. Engng 2002; **18**:335–343 (DOI: 10.1002/cnm.494)

Minimization of indirect addressing for edge-based field solvers

Rainald Löhner[1,*] and Martin Galle[2]

[1]*School of Computational Sciences, MS 4C7, George Mason University, Fairfax, VA 22030-4444, U.S.A.*
[2]*NEC European Supercomputer Systems, European HPC Technology Center, Stuttgart, Germany*

SUMMARY

A point renumbering scheme that halves the number of indirect addressing operations required for edge-based field solvers has been developed. At the same time, an alternative (and entirely serendipitous) assembly of right-hand side vectors was found to enhance performance considerably. The new loop structure is especially attractive for vector-parallel machines with direct memory access. Timings on the CRAY-SV1 and NEC-SX5 show that for Laplacian loops, which appear in many CFD and CEM codes, CPU can be reduced considerably. The new renumbering scheme does not require any new data structures or arrays, allowing for progressive porting of loops in large-scale production codes. Copyright © 2002 John Wiley & Sons, Ltd.

KEY WORDS: grid renumbering; unstructured grids; vector-parallel machines

1. INTRODUCTION

Even though a considerable portion of field solvers are run on workstations and servers that are based on RISC-chip technology (cache, commodity memory), for many time-critical applications vector-parallel machines are still the fastest computers available. These machines prefer an ordered access of data from memory. Accessing and storing data in an unordered way, as is typically the case for solvers based on unstructured grids, implies a CPU penalty that is considerable. Efforts to reduce the amount of indirect addressing (i/a) with respect to floating point operations (FLOPs) led to a switch from element-based codes to edge-based codes in the early 1990s [1–7]. A typical right-hand side (RHS) evaluation based on edges for the Laplacian may be written as follows:

Loop 1

```
1000 ipass=1,npass
nedg0=edpas(ipass)+1
```

*Correspondence to: R. Löhner, School of Computational Sciences, George Mason University, Fairfax, VA 22030-4444, U.S.A.

Contract/grant sponsor: Partially supported by AFOSR

Received 28 February 2001
Accepted 29 August 2001

```
      nedg1=edpas(ipass+1)
c$dir ivdep
       do 1100 iedge=nedg0,nedg1
       ipoi1=lnoed(1,iedge)
       ipoi2=lnoed(2,iedge)
       redge=geoed(  iedge)*(unkno(ipoi2)-unkno(ipoi1))
       rhspo(ipoi1)=rhspo(ipoi1)+redge
       rhspo(ipoi2)=rhspo(ipoi2)-redge
 1100 continue
 1000 continue
```

Here `edpas(npass)`, `lnoed(2,nedge)`, `geoed(nedge)`, `unkno(npoin)` and `rhspo(npoin)` denote the edge-pass array, edge-point connectivity, edge-Laplacian, point-unknown and point-rhsides, respectively. It is furthermore assumed that within every group of edges `nedg0:nedg1` the points are never accessed more than once, i.e. no memory contention is present. The array `lnoed(2,nedge)` could also have been stored as `lnoed(nedge,2)` for machines with direct memory access, but this form of storage would lead to potential cache-misses on cache-based machines. Inspection of Loop 1 indicates that it requires 4 i/a fetches, 2 i/a stores and (only) 4 FLOPS per edge. While many vendors quote impressive megaflops (10^6 FLOPs/s) or gigaflops (10^9 FLOPs/s) for their RISC chips, production codes seldomly achieve even 10 per cent of these peak numbers. The reason is that these chips only achieve peak performance for situations where a large number of operations is carried out for each item transferred from and to memory. As one can see, Loop 1, like most loops encountered in computational fluid dynamics (CFD) and computational electromagnetics (CEM) codes, does not fall into this category. Timings on the NEC-SX5 indicated that for a typical projection-type incompressible flow solver [8], Loop 1 accounts for 70 per cent of the total CPU time. Given that for Loop 1 i/a accounts for a large portion of its CPU cost, alternative data structures that reduce i/a even further will have a considerable payoff.

One alternative to obtain more FLOPs per i/a is to agglomerate edges together into so-called stars, chains and superedges [9]. A superedge loop with count 6, which considers all the edges of a tetrahedron, looks as follows:

Loop 2

```
      do 1200 iedge=nedg0,nedg1,6
      ipoi1=lpoed(1,iedge )
      ipoi2=lpoed(2,iedge )
      ipoi3=lpoed(1,iedge+3)
      ipoi4=lpoed(2,iedge+3)
      upoi1=unkno(ipoi1)
      upoi2=unkno(ipoi2)
      upoi3=unkno(ipoi3)
      upoi4=unkno(ipoi4)
      redg1=edlap(iedge  )*(upoi2-upoi1)
      redg2=edlap(iedge+1)*(upoi3-upoi2)
      redg3=edlap(iedge+2)*(upoi3-upoi1)
      redg4=edlap(iedge+3)*(upoi4-upoi1)
```



```
      redg5=edlap(iedge+4)*(upoi4-upoi2)
      redg6=edlap(iedge+5)*(upoi4-upoi3)
      rhspo(ipoi1)=rhspo(ipoi1)+redg1+redg3+redg4
      rhspo(ipoi2)=rhspo(ipoi2)-redg1+redg2+redg5
      rhspo(ipoi3)=rhspo(ipoi3)-redg2-redg3+redg6
      rhspo(ipoi4)=rhspo(ipoi4)-redg4-redg5-redg6
1200 continue
```

This loop requires 8 i/a fetches and 4 i/a stores for every 6 edges. This represents a saving factor of 3:1 in i/a operations. As one can see from the superedge-loop above, the larger the number of edges in a group, the longer and more complicated the loops. At some stage register availability will become a constraining factor. Therefore, a balance has to be struck between efficiency and code simplicity, clarity and maintenance. For tetrahedral meshes, the most natural superedge is the tetrahedron itself. It was found that with a simple renumbering strategy, around 70 per cent of all edges could be covered with this type of superedge. Around 20 per cent of the edges were then grouped into triangles, the next most efficient superedge, with a saving factor of 2:1 in i/a operations. The remaining edges were treated as usual. This partitioning of edges seemed to be constant for a variety of different meshes. The saving factor in i/a for such a partitioning of edges is:

$$S = \frac{\text{i/a(usualedge)}}{\text{i/a(superedge)}} = \frac{1}{\frac{0.7}{3} + \frac{0.2}{2} + 0.1} = 2.3$$

Given that edges stored as stars, chains and superedges will lead to memory overwrite when accessed as in Loop 1, a complete code re-write is required when switching to them. For this reason, they have not had a great impact in production codes.

A second alternative to reduce the amount of i/a is to store the edges according to points, processing each edge twice. The central loop of such a code looks as follows:

Loop 3

```
      do 1300 iedge=nedg0,nedg1
      ipoi1=kpoi0+iedge
      ipoi2=lnoed(2,iedge)
      redge=geoed(  iedge)*(unkno(ipoi2)-unkno(ipoi1))
      rhspo(ipoi1)=rhspo(ipoi1)+redge
1300 continue
```

This loop requires 1 i/a fetch per edge. Given that each edge is processed twice as compared to the original Loop 1, a saving factor of 3:1 in i/a operations if achieved. However, the number of FLOPS has been increased by 50 per cent, and for more FLOP-intensive loops would have doubled. This does not represent a problem for the Laplacian loop. Indeed, timings on the NEC and CRAY indicate that even with twice the number of FLOPS, Loop 3 runs approximately twice as fast as Loop 1. This will not be the case for loops with more FLOPS per i/a, e.g. approximate Riemann solvers for CFD codes.

Given that stars, chains and superedges will lead to memory overwrite when edges are accessed as in Loop 1, and that Loop 3 incurs a heavy CPU penalty for FLOP-intensive edge-loops, an alternative was sought to reduce i/a while being compatible with the edge-loops of most production codes.

2. REDUCED I/A LOOP

Suppose the edges are defined in such a way that the first point always has a lower point-nr. than the second point. Furthermore, assume that the first point of each edge increases with stride one as one loops over the edges. In this case, Loop 1 may be rewritten as follows:

Loop 4

```
      do 1400 iedge=nedg0,nedg1
      ipoi1=kpoi0+iedge
      ipoi2=lnoed(2,iedge)
      redge=geoed(  iedge)*(unkno(ipoi2)-unkno(ipoi1))
      rhspo(ipoi1)=rhspo(ipoi1)+redge
      rhspo(ipoi2)=rhspo(ipoi2)-redge
1400  continue
```

Compared to Loop 1, the number of i/a fetch and store operations has been *halved* while the number of FLOPS remains unchanged. Moreover, the basic connectivity arrays remain unchanged, implying that a progressive rewrite of codes is possible. In the sequel, we describe a point and edge-renumbering algorithm that seeks to maximize the number of loops of this kind that can be obtained for tetrahedral grids.

3. POINT RENUMBERING

In order to obtain as many uniformly accessed edge-groups as possible, the number of edges attached to a point should decrease uniformly with point-number. In this way, the probability of obtaining an available second point to avoid memory contention in each vector-group is maximized. The following algorithm achieves such a point renumbering:

Initialization:

- From the edge-connectivity array `lnoed`:
  Obtain the points that surround each point;
- Store the number of points surrounding each point: lpsup(1:npoin);
- Set `npnew=0`;

Point renumbering:

- `while(npnew.ne.npoin)`:
  - Obtain the point `ipmax` with the maximum value of `lpsup(ip)`;
  - `npnew=npnew+1` ! update new point counter
  - `lpnew(npnew)=ipmax` ! store the new point
  - `lpsup(ipmax)=  0` ! update lpsup
  - `do`: for all points `jpoin` surrounding `ipmax`:
    - `lpsup(jpoin)=max(0,lpsup(jpoin)-1)`
  - `enddo`
- `endwhile`

Once the points have been renumbered, the edges can be grouped into so-called vector-groups to avoid memory contention [10, 11]. It is clear that not all of these groups will lead to a uniform stride access of the first point. These loops are still processed as before in Loop 1. The final form for the edge-loop then takes the form

Loop 5

```
      1000 ipass=1,npass
      nedg0=edpas(ipass)+1
      nedg1=edpas(ipass+1)
      if(lnoed(1,nedg1).ne.lnoed(1,nedg0)+nedg1-nedg0) then
c$dir ivdep
         do 1100 iedge=nedg0,nedg1
         ipoi1=lnoed(1,iedge)
         ipoi2=lnoed(2,iedge)
         redge=geoed(  iedge)*(unkno(ipoi2)-unkno(ipoi1))
         rhspo(ipoi1)=rhspo(ipoi1)+redge
         rhspo(ipoi2)=rhspo(ipoi2)-redge
 1100   continue
      else
         kpoi0=lnoed(1,nedg0)-nedg0
c$dir ivdep
         do 1400 iedge=nedg0,nedg1
         ipoi1=kpoi0+iedge
         ipoi2=lnoed(2,iedge)
         redge=geoed(  iedge)*(unkno(ipoi2)-unkno(ipoi1))
         rhspo(ipoi1)=rhspo(ipoi1)+redge
         rhspo(ipoi2)=rhspo(ipoi2)-redge
 1400   continue
      endif
 1000 continue
```


4. ALTERNATIVE RHS FORMATION

A number of test cases (see below) were run on both the CRAY-SV1 and NEC-SX5 using both the conventional Loop 1 and Loop 5. The results were rather disappointing: Loop 5 was slightly more expensive than Loop 1, even for moderate (>256) vector lengths and more than 90 per cent of edges processed in reduced i/a mode. Careful analysis on the NEC-SX5 revealed that the problem was not in the fetches, but rather in the stores. Removing one of the stores almost doubled CPU performance. This observation led to the unconventional formation of the RHS with two vectors

Loop 6

```
      1000 ipass=1,npass
      nedg0=edpas(ipass)+1
      nedg1=edpas(ipass+1)
```


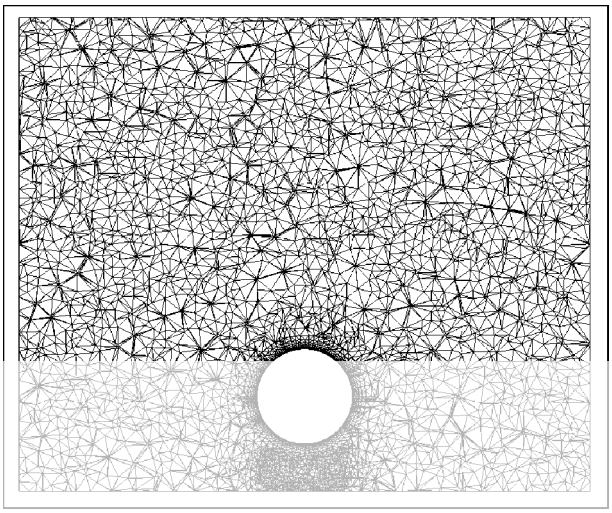

Figure 1. Sphere in wall proximity: mesh in cut plane.


```
c$dir ivdep
      do 1100 iedge=nedg0,nedg1
      ipoi1=lnoed(1,iedge)
      ipoi2=lnoed(2,iedge)
      redge=geoed(  iedge)*(unkno(ipoi2)-unkno(ipoi1))
      rhsp0(ipoi1)=rhsp0(ipoi1)+redge
      rhsp1(ipoi2)=rhsp1(ipoi2)-redge
1100  continue
      do 1000 iedge=nedg1,nedge
      ipoi1=lnoed(1,iedge)
      ipoi2=lnoed(2,iedge)
      redge=geoed(  iedge)*(unkno(ipoi2)-unkno(ipoi1))
      rhspo(ipoi1)=rhspo(ipoi1)+redge
      rhspo(ipoi2)=rhspo(ipoi2)-redge
1000  continue
```

Since the compiler cannot exclude that `ipoi1` and `ipoi2` are identical, the fetch of `rhspo(ipoi2)` in Loop 5 has to wait until the store of `rhspo(ipoi1)` has finished. The introduction of the dual RHS enables the compiler to schedule the load of `rhsp1(ipoi2)` earlier and to hide the latency behind other operations. We remark that if `rhsp0` and `rhsp1` are physically identical, no additional initialization or summation of the two arrays is required. The same use of dual RHS vectors was also implemented in Loop 5, and is denoted as Loop 7 in the sequel. Finally, the `if`-test in Loop 5 may be removed by reordering the edge-groups in such a way that all usual i/a groups are treated first and all reduced i/a thereafter. This loop is denoted as Loop 8.

5. EXAMPLES AND TIMINGS

In order to see what percentage of edges can typically be processed in reduced i/a mode (Loop 4), several grids were tested. The first is a sphere close to a wall, typical of low Reynolds-number incompressible flows. Figure 1 shows the mesh in a plane cut through the sphere. This case had 49 574 points, 272 434 elements and 328 634 edges. Table I shows the

Table I. Sphere close to wall: `nedge=328 634`.

`mvecl`	% redi/a	`nvecl`	% redi/a	`nvecl`
128	89.53	126	94.88	119
256	87.23	251	94.94	218
512	84.50	490	94.69	371
1024	78.58	947	93.10	592
2048	69.85	1748	90.85	797

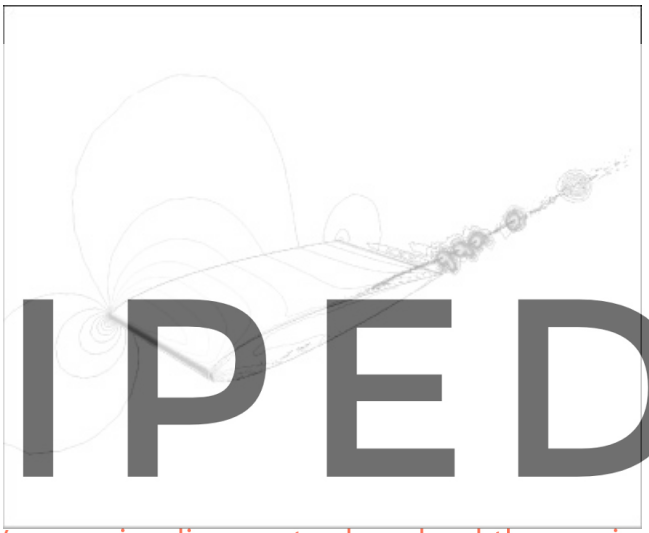

Figure 2. NACA0012: surface pressure and vortex core visualization.

percentage of edges processed in reduced i/a mode as a function of the desired vector length (`mvecl`) chosen, as well as the resulting average vector length (`nvecl`). The table contains two values: the first is obtained if one insists on the vector length chosen; the second is obtained if the usual i/a vector groups are examined further, and snippets of sufficiently long (>64) reduced i/a edge-groups are extracted from them. Observe that even with considerable vector lengths, more than 90 per cent of the edges can be processed in reduced i/a mode.

The second case is a NACA0012 wing, typical of inviscid incompressible flows. This case had 68 585 points, 370 541 elements and 451 108 edges. Figure 2 shows some results. Table II shows the percentage of edges processed in reduced i/a mode as a function of the desired vector length chosen, as well as the resulting average vector length. Note that, as before, even for large vector lengths, more than 90 per cent of the edges can be processed in reduced i/a mode.

We now proceed to timings for the different loops. The case taken is the sphere in close proximity to a wall shown in Figure 1. Table III shows the relative timings recorded for a desired edge-group length of 2048 on the SGI Origin2000, CRAY-SV1 and NEC-SX5. One can see that gains are achieved in all cases, even though these machines vary in speed by

Table II. NACA0012 wing: `nedge=451 108`.

`mvecl`	% redi/a	`nvecl`	% redi/a	`nvecl`
128	90.24	127	95.67	119
256	87.77	253	95.73	220
512	86.07	502	95.80	380
1024	82.55	993	94.90	612
2048	73.47	1919	92.90	835

Table III. Laplacian RHS evaluation (relative timings).

Loop type	O2K	SV1	SX5
Loop 1	1.0000	1.0000	1.0000
Loop 5	0.9563	1.0077	0.8362
Loop 6	0.9943	0.8901	0.7554
Loop 7	0.9484	0.8416	0.7331
Loop 8	—	—	0.7073

approximately an order of magnitude, and the SGI has an L1 and L2 cache, i.e. no direct memory access. The biggest gains are achieved on the NEC-SX5, which performed Loop 8 at approximately 650 MFLOPS.

6. CONCLUSIONS

A point renumbering scheme that allows to halve the number of indirect addressing operations required for edge-based field solvers has been developed. At the same time, an alternative (and entirely serendipitous) assembly of right-hand side vectors was found to enhance performance considerably. The new loop structure is especially attractive for vector-parallel machines with direct memory access, where the cost of indirect addressing can be high as compared to floating point operations. Timings on the CRAY-SV1 and NEC-SX5 show that for Laplacian loops, which appear in many CFD and CEM codes, CPU can be reduced considerably. Slight gains are also achieved on the SGI Origin2000, implying that no penalty is incurred for cache-based machines. The new renumbering scheme does not require any new data structures or arrays, allowing for progressive porting of loops in large-scale production codes.

Future efforts will center on optimizing the present renumbering techniques to cache-based RISC machines, where cache-misses have to be considered for very large problems.

ACKNOWLEDGEMENTS

The first author was partially supported by AFOSR, with Dr Leonidas Sakell as the technical monitor.

REFERENCES

1. Barth T. A 3-D upwind Euler solver for unstructured meshes. *AIAA*-91-1548-*CP*, 1991.
2. Mavriplis D. Three-dimensional unstructured multigrid for the Euler equations. *AIAA*-91-1549-*CP*, 1991.
3. Jameson A. The AIRPLANE code. *Private Communication*, January, 1992.
4. Peraire J, Peiro J, Morgan K. A three-dimensional finite element multi-grid solver for the Euler equations. *AIAA*-92-0449, 1992.
5. Luo H, Baum JD, Löhner R, Cabello J. Adaptive edge-based finite element schemes for the Euler and Navier–Stokes equations. *AIAA*-93-0336, 1993.
6. Weatherill NP, Hassan O, Marcum DL. Calculation of steady compressible flowfields with the finite element method. *AIAA*-93-0341, 1993.
7. Galle M. Solution of the Euler and Navier–Stokes equations on hybrid grids. *AGARD F.D.P. Symposium on Progress and Challenges in CFD*, *Methods and Algorithms*, Seville, October, 1995.
8. Löhner R, Chi Yang, Oñate E, Idelssohn S. An unstructured grid-based, parallel free surface solver. *Applied Numerical Mathematics* 1999; **31**:271–293.
9. Löhner R. Edges, stars, superedges and chains. *Computer Methods in Applied Mechanics and Engineering* 1994; **111**:255–263.
10. Löhner R. Some useful renumbering strategies for unstructured grids. *International Journal for Numerical Methods in Engineering* 1993; **36**:3259–3270.
11. Löhner R. Renumbering strategies for unstructured-grid solvers operating on shared-memory, cache-based parallel Machines. *Computer Methods in Applied Mechanics and Engineering* 1998; **163**:95–109.

