INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng 2006; **66**:431–460
Published online 13 January 2006 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/nme.1557

Performance comparison of data-reordering algorithms for sparse matrix–vector multiplication in edge-based unstructured grid computations

Alvaro L. G. A. Coutinho*, †, Marcos A. D. Martins‡, Rubens M. Sydenstricker§ and Renato N. Elias¶

Department of Civil Engineering, Center for Parallel Computing, COPPE/Federal University of Rio de Janeiro, P.O. Box 68506, Rio de Janeiro RJ 21945-970, Brazil

SUMMARY

Several performance improvements for finite-element edge-based sparse matrix–vector multiplication algorithms on unstructured grids are presented and tested. Edge data structures for tetrahedral meshes and triangular interface elements are treated, focusing on nodal and edges renumbering strategies for improving processor and memory hierarchy use. Benchmark computations on Intel Itanium 2 and Pentium IV processors are performed. The results show performance improvements in CPU time ranging from 2 to 3. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: data reordering; unstructured grids; edge-based finite elements; cache-based machines

1. INTRODUCTION

Modern engineering applications demand accurate and realistic computational solutions. Either implicit finite-element computational solid or fluid mechanics simulations yield large linear equation systems which are often solved iteratively. Such solutions may require millions of sparse matrix–vector multiplications and frequently demand irregular data access driven by

*Correspondence to: Alvaro L. G. A. Coutinho, Department of Civil Engineering, Center for Parallel Computing, COPPE/Federal University of Rio de Janeiro, P.O. Box 68506, Rio de Janeiro RJ 21945-970, Brazil.
†E-mail: alvaro@nacad.ufrj.br
‡E-mail: marcos@nacad.ufrj.br
§E-mail: rubens@nacad.ufrj.br
¶E-mail: renato@nacad.ufrj.br

Contract/grant sponsor: CNPq; contract/grant number: 500196/2002-8
Contract/grant sponsor: Intel

Received 2 May 2005
Revised 23 September 2005
Accepted 23 September 2005

memory bandwidth and latency. Thus, an efficient simulation code must be optimized to improve performance regardless the computational environment.

Research on computational performance suggests that performance growth per year is about 60% for processor and 9% for memory [1]. Therefore, processors are about two times faster each 18 months, whereas the same progress would take about 10 years for memory. A recent investigation based on Top500 ranking [2] suggests that the doubling growth for processors performance is achieved each 13.5 months for the top 500 machines and each 16.6 months for the top 16 machines [3]. As the gap between processor and memory performance continues growing, an increasing fraction of computer workload will be dominated by memory access and transfer times rather than by processor time. Therefore, the efficient use of memory hierarchy resources (referred as cache and registers) turns out to be a critical issue in high-performance computing [4, 5].

Since processor communication is crucial for parallel performance, the traditional approach is to partition data followed by some data-reordering algorithm to improve data locality on each processor separately. Some results have been demonstrating though, that for real applications problems, the interrelated and simultaneous use of data partitioning and locality algorithms is a need for practicable results and in some cases, cache reuse may be more important than reducing communication frequency [5].

Techniques to reorder data along with parallel paradigms such as load-balancing, data partitioning for message passing, shared memory, hybrid and multithreading environment exploit the minimization of communications among processors, in conjunction with data locality algorithms to enhance the single-processor performance and then optimizing the overall performance alike [5–8]. Such techniques evidence that performance is substantially affected by the efficiency of the memory hierarchy use.

An efficient data-reordering procedure demands a careful analysis of data movement during the system solution, comprising: CPU time, memory access, memory contention and availability of CPU registers [4]. Tiling techniques, that is, the process of decomposing an overall computation into smaller blocks for doing the computation in each block one at a time, are frequently used for improving performance of matrix–vector products [1, 3, 5, 7–25]. Such techniques are based on some data-reordering procedures [26] and, additionally, unrolling the product evaluations [27]. The latter procedure gives the compiler the opportunity to schedule loops and also to reduce loop overheads which contributes to reduce memory latency effects.

For structured grid applications, algorithms can be designed to fit in memory hierarchy, taking into account data access and layout transformations, and cache associativity and layout [11]. However, the effects of memory hierarchy over final performance are not as obvious as a measurable parameter, related on how precisely data locality can contribute to cache use. Cache misses are difficult to model prior to computation, but may be measured after that [11]. In this context, data locality models [4, 14, 22] are employed in the evaluation of such parameter producing good results.

An efficient data locality management is mandatory, mainly for memory hierarchies with small caches, since the number of cache misses is more important for performance than the number of floating point operations. Thus, optimizations techniques based on memory-centric perspective, rather than flop-orientation, have been a trend [13] to achieve an effective use of the memory hierarchy.

For unstructured grids the memory accesses is irregular and it is not possible to predict its behaviour. In this case, cache optimizations depend less on the technical details of cache than

for structured grids, and indirect addressing harms the performance. The number of cycles to directly access data stored sequentially or not is the same, whereas the number of cycles for indirect access strongly depends on how data are stored (sequentially or not) [4]. This explains the great impact of data renumbering strategies on codes with a high proportion of indirect addressing compared to other operations. This concept suggests that in some cases it is more important to optimize indirect addressing accesses by data locality than simply reducing the number of these operations.

The data structure migration from element-based to edge-based observed since the last decade was motivated, among other factors, by the sparse matrix computations that arise from finite-element implementations [21, 28]. Edge-based data structures are an effective optimization technique since they provide reductions of floating point (flop) and indirect addressing (i/a) operations [20, 23], storage requirements and also improve memory access. As a result, the overall computational performance increases significantly, even when compared to the compressed row storage (CSR) scheme [23]. Edge optimization techniques have great potential, even for quadrilaterals and hexahedra, for which its implementation involves greater complexity [20, 23].

This paper presents performance results of several edge-based matrix–vector multiplication algorithms obtained in current processors. Computational optimizations focus the memory hierarchy and intend to minimize indirect addressing, floating point operations and registers use for unstructured grids composed by linear tetrahedra. Unstructured data are handled by reordering algorithms to improve data locality of nodes, edges and elements. All these concepts run quite suitably into the edge-based data structure paradigm.

Based on the nodal renumbering algorithms and concepts proposed in References [13, 16, 18] and edge renumbering algorithms proposed in Reference [20], algorithms for one and three degrees of freedom per node (hereafter, referred as dof) problems were developed, implemented and intensively tested. The techniques employed try to minimize indirect addressing operations. A sorting of edges, in increasing order by the edge first node number (namely hereafter, reduced i/a), halves the indirect addressing operations of the edge-based matrix–vector product algorithm [16].

Additionally, data locality algorithms were used in association with special edge groupings, which improve the edge-based matrix–vector product algorithm. This was implemented for tetrahedra, grouped into three and six edges, named, respectively, *superedge3* and *superedge6* for both monophasic flow and geomechanics problems [20]. For the latter, it was proposed and implemented an edge-based interface element with special groupings named *superedge4* and *superedge9*, comprising groups of four and nine edges, respectively [29].

All these groupings were compared to the reduced i/a edge concept, in cache-based machines and CPU-pipelined machines. Furthermore, an unrolled version of the matrix–vector multiplication algorithm for reduced i/a scheme was also proposed and tested, along with a chunkwise version for both schemes (*superedge* and reduced i/a). In addition, the alternative right-hand side evaluation in the matrix–vector product algorithm proposed by Löhner and Galle [16] was also implemented and tested.

Related to memory dependency, lists of edges are built where no pair of nodes in the same edge list shares the same node. This arrangement is referred here as nodal disjointing and is responsible for about 70% of the execution time spent in TLB (translation look-aside buffer) misses [13]. This lack of performance is reduced by an appropriate reverse Cuthill McKee (RCM) [26] algorithm in conjunction with edge and element sorting according to node numbering.

This work is organized as follows. In Section 2, the complexity of edge-based matrix–vector product algorithm is evaluated considering memory access, data locality and floating point operations. Section 3 presents the nodal reordering algorithm for reduced i/a list assembly proposed in Reference [16], and also briefly comments on the *superedge* grouping algorithm. Section 4 shows and comments performance studies in three problems. Finally, Section 5 presents our concluding remarks.

2. EDGE-BASED DATA STRUCTURES

Sparse matrix–vector multiplication algorithms may be considered the kernel of iterative solution methods. As a prototype of such procedures, the edge-based Laplacian loop algorithm as proposed in Reference [16], is presented in Algorithm 1 and is used as a pattern of comparison to other alternatives. This algorithm comprises 1 dof.

Algorithm 1
Laplacian loop for a single edge sparse matrix–vector product.

```
do edge = edge_begin, edge_end
    eq_1 = lm(1,edge)
    eq_2 = lm(2,edge)
    ap = a(edge) * (u(eq_2) - u(eq_1))
    p(eq_1) = p(eq_1) + ap
    p(eq_2) = p(eq_2) + ap
end do
```

This loop generates 4 i/a fetches, 2 i/a stores (resulting in six memory accesses) and four flops per edge. Array `lm` stores equation numbers. In order to achieve a good balance between memory accesses and flops, reordering techniques are suggested in the literature such as *superedges* [19, 20] and reduced i/a edges [16]. The *superedge* scheme reaches good balance of i/a and flops [30] without complex preprocessing codes [20]. In this case, computer costs are just related to the new order of the edge list, considering the edges agglomerated, in geometric sense, for example, in tetrahedral shape, swept by stride of six edges. The resulting

Table I. Computational parameters for matrix–vector multiplication algorithm for simple edge and *superedges* from tetrahedron for Laplacian loop.

Structure	Fetches	Stores	Flops	Reduction factor
Simple edge	4	2	4	$1.0=1.0$
*Superedge*6	8	4	24	$0.7/6=0.12$
*Superedge*3	6	3	15	$0.2/3=0.07$

code for Laplacian RHS evaluation comprising one dof is presented by Algorithm 2 [16, 20], for *superedge6*:

Algorithm 2
Laplacian loop for *superedge* sparse matrix–vector product.

```
do edge = edge_begin, edge_end, 6
    eq_1 = lm(1,edge)
    eq_2 = lm(2,edge)
    eq_3 = lm(1,edge+3)
    eq_4 = lm(2,edge+3)
    p1 = u(eq_1)
    p2 = u(eq_2)
    p3 = u(eq_3)
    p4 = u(eq_4)
    ap1 = a(edge) * (p2 - p1)
    ap2 = a(edge) * (p3 - p2)
    ap3 = a(edge) * (p3 - p1)
    ap4 = a(edge) * (p4 - p1)
    ap5 = a(edge) * (p4 - p2)
    ap6 = a(edge) * (p4 - p3)
    p(eq_1) = p(eq_1) + ap1 + ap3 + ap4
    p(eq_2) = p(eq_2) - ap1 + ap2 + ap5
    p(eq_3) = p(eq_3) - ap2 - ap3 + ap6
    p(eq_4) = p(eq_4) - ap4 - ap5 - ap6
end do
```

This loop requires 8 i/a fetches, 4 i/a stores (resulting in 12 memory accesses) and 24 flops for one *superedge* consisting of six edges. In previous works [20, 30], it was found that this kind of *superedge* (among several alternatives) achieves the best results for tetrahedral meshes in large-scale unstructured grid problems. For those grids, it was observed in Reference [20] that usually 70% of all edges can be grouped as *superedges* of six edges, 20% in *superedges* of three edges and 10% remains as simple edges. The equivalent code for one tetrahedral element comprises 8 i/a fetches, 4 i/a stores and 32 flops. A comparison among these parameters for these data structures is presented in Table I [20].

According to References [20, 22] and considering Table I, the *superedge* implementation reaches global values of $8 \times 0.12 + 6 \times 0.07 + 4 \times 0.1 = 1.78$ fetches, $4 \times 0.12 + 3 \times 0.07 + 2 \times 0.1 = 0.89$ stores and $24 \times 0.12 + 15 \times 0.07 + 4 \times 0.1 = 4.33$ flops per edge, thus saving 56% for memory accesses although overcharging 8.3% in flops when compared to simple edge implementation.

Geomechanical modelling often requires the simulation of faults or other contact discontinuities. An edge-based implementation of the interface triangular element proposed in Reference [29] was developed to render the code with a single data structure. For these elements, the *superedge* groupings are named *superedge4* and *superedge9*, corresponding to four and nine edges grouped, respectively. Table II outlines computational parameters for *superedges* arising from tetrahedra and interface elements, comprising three dof. Estimates in columns (b) and (d) are majored, respectively, by factors 1.5 and 6. These factors represent the observed ratio of edges by number of elements for each element type in a typical unstructured mesh.

Table II. Computational parameters for matrix–vector multiplication algorithm for *superedges* from tetrahedron and interface elements for geomechanical problems with three dof.

Parameter	Tetrahedron (a)	Edge tetrahedra (b)	Interface (c)	Edge interface (d)
flops	252	54	108	36
i/a	36	27	54	108

Table III. Comparison of computational parameters for matrix–vector multiplication algorithm for reduced i/a and *superedges* for three dofs.

Group/parameter	flops	i/a	flops/(i/a)
Simple edge	36	18	2.0
Superedge3	130	27	4.8
Superedge6	268	36	7.4
Superedge4	190	36	5.3
Superedge9	436	54	8.1
Reduced i/a	39	9	4.3

An alternative to reduce i/a is to convert an edge-based loop into a vertex-based loop [13] in which the edges are arranged in such a way that the first node always has the lower number and the first node number increases as the edge number increases with stride one. This loop reuses vertex-based data items in most or all of the accesses several times before discarding it. This approach increases flops but reduces i/a operations, whereas the edge has to be processed twice. This procedure, used along with nodal reordering for bandwidth reduction, results in a significant reduction of TLB misses [13]. Considering the second node [16] though (thus, the whole edge) in conjunction with node and edge number increasing with stride 1, the RHS evaluation can be changed from vertex-based to edge-based as shown in Algorithm 3, for 1 dof,

Algorithm 3
Laplacian loop for reduced i/a edge for sparse matrix–vector product.

```
do edge = edge_begin, edge_end
    eq_1 = shift + edge
    eq_2 = lm(2,edge)
    ap = a(edge) * (u(eq_2) - u(eq_1))
    p(eq_1) = p(eq_1) + ap
    p(eq_2) = p(eq_2) + ap
end do
```

where `shift` is the difference between the equation number of the current edge and the equation associated to the first node.

Algorithm 3 generates 2 i/a fetches, 2 i/a stores and five flops per edge. Table III shows a comparison of computation parameters for the matrix–vector multiplication algorithm for reduced i/a and *superedge* schemes, again, considering three dof.

In Table III, the computational intensity (here defined as the ratio between *flop* and i/a operations) of the reduced i/a scheme is lower than the *superedges*, indicating a minor potential

for optimization. Indeed, if the reduction factor shown in the last column of Table I is considered as typical for a mesh without interfaces, then a computational intensity of 3.77 will be found, resulting from 17.55 flops and 4.65 i/a per edge. However, the flops and i/a operations have to be weighted based on CPU bias for each scheme.

3. NODAL AND EDGE RENUMBERING

According to Reference [16], a nodal renumbering algorithm designed for an efficient memory handling, trying to yield uniform memory access and avoid contention, is shown in Algorithm 4.

Algorithm 4
Nodal renumbering algorithm.
Initialization:

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

Node Renumbering:

```
do while (npnew.ne.npoin)
    Obtain the node ipmax with the maximum value of lpsup(ip);
    npnew = npnew + 1
    lpsup(ipmax) = 0;
    do for all nodes jpoin surrounding ipmax
       lpsup(jpoin) = max(0,lpsup(jpoin) - 1)
    end do
end while
```

The critical point in Algorithm 4 is to obtain the node with maximum quantity of neighbours. This step presents linear time complexity according to Reference [18] and the normalized CPU time results for the code implemented here are depicted in Figure 1 for eight samples. It can be observed that indeed an almost linear behaviour is observed in practice. However, this reordering procedure can lead to inefficient use of memory hierarchy due to the jump in the nodal numbering per edge. It is suggested a two step renumbering procedure by Löhner and Galle [16]. In the first step, the nodes are ordered by a bandwidth minimization technique [26] and, in the second, the mesh is reordered according to Algorithm 4, not at once, but progressively in groups of nodes corresponding to the average bandwidth.

On the other hand, the main idea behind the *superedge* grouping algorithm is to check every element, marking all unused edges as a group of *superedge6* or *superedge3*, for tetrahedra and *superedge4* and *superedge9* for interface elements, assembling edge lists. In this work, the RCM algorithm [26] was used to improve data locality.

In all cases, the element list is reordered according to edge order and the nodal element connectivity is reordered locally to improve cache use [18]. The examples will graphically illustrate the effects of these reordering actions over nodes, edges and elements.

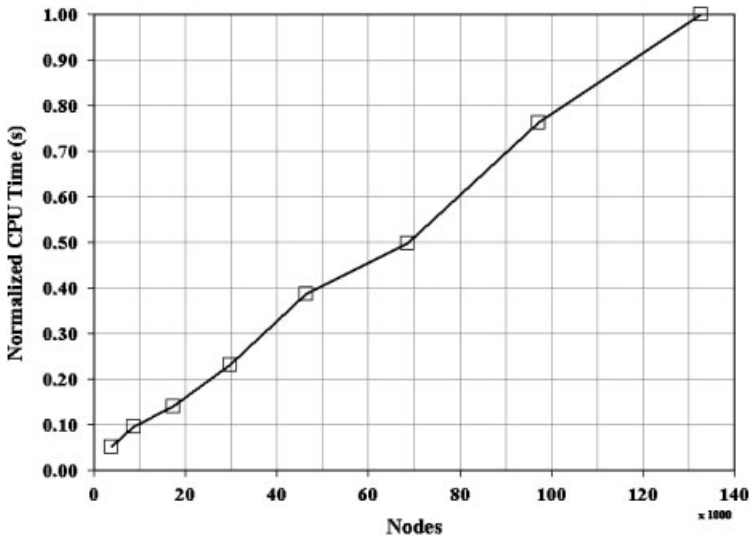

Figure 1. Time complexity for nodal renumbering algorithm.

Table IV. Basic hardware configurations.

Platform	CPU	RAM	Cache
Pentium IV	3.6 GHz	4 GB/533 MHz	L2 1 MB
Itanium 2	1.3 GHz	8 GB/266 MHz	L3 3 MB

4. EXAMPLES

The tests were carried out on Itanium 2 and Pentium IV Intel processors, present in most of the machines listed in the latest Top500 rank [2] along with Intel Fortran compiler release 7 for Linux systems. The basic hardware configurations are presented in Table IV.

To handle the memory dependency issue in the case of Itanium, the edges and elements were reordered by a nodal disjointing algorithm [12] and grouped into lists with constrained length, referred hereafter as vector length. The concept of vector length together with reduced i/a edge scheme was organized in levels as follows [16]:

Level 0: reduced i/a vector length set to *Nvec* at the maximum length, without usual i/a edges;
Level 1: reduced i/a vector length, as much as possible, set to *Nvec* and 64 at minimum, otherwise; the non-matched edges are arranged as usual i/a edges of *Nvec* at maximum;
Level 2: reduced i/a vector length, as much as possible, set to *Nvec* and arranged as usual i/a edges of *Nvec* at maximum, otherwise.

The layout of loop over edge lists constrained into specific lengths, hereafter referred as chunk length, was also implemented and tested in the absence of data dependency issue and therefore without nodal disjointing algorithm. Both chunk and vector length effects were investigated in

Table V. Keywords for data arrangement analysis.

Keyword #	Description	Option
1	Nodal disjointing	Yes/No
2	Chunks	Yes (List length)/No
3	Nodal ordering	Reduced/RCM
4	Edge ordering	Reduced (0/1/2)
5	Alternative RHS	Yes/No
6	Loop unrolling	Yes (2/3/6)/No

the numerical experiments. The unrolled loop layout was also evaluated in the case of reduced edge at level 0 to keep it up to *superedge* scheme and compare both. Furthermore, the algorithm for right-hand side evaluation proposed by Löhner and Galle [16] and referred as 'alternative RHS formation', for matrix–vector multiplication algorithm was tested for 1 and 3 dofs.

The tests were based on ordering data combinations driven by six keywords options as presented by Table V. These key combinations together with list lengths ranging from 64 to 16 384 and without limit also, with stride 64, lead to several possibilities. The schedule counts to 216 runs with nodal disjointing algorithm and 128 runs without it. In the latter, 112 runs were in chunkwise fashion.

The results comprise graphs representing the mesh topology according to its nodal and edge numbering to show how data is accessed in the loops. The analysis of these graphs helps to understand the behaviour of the matrix–vector product algorithm based on each data distribution and to forecast its performance for each data configuration. These graphs were assembled by connecting the first node of each edge by a grey line and the second node by a black one to emphasize the data distribution layout. The horizontal axis represents the edges and the vertical axis represents the nodal connectivity of edges. The most significant data distributions were presented.

Timing results of matrix–vector multiplication algorithm were presented for some specific data configuration and loop layout by bar graphs. In this graphs, the vertical axis corresponds to CPU time and the horizontal axis the specific data configuration and/or loop layout schemes. The most significant timing results were presented.

4.1. Potential flow around a submarine

This problem consists of a three-dimensional simulation of the potential flow around a Los Angeles class submarine [15]. The PCG solver converged after 350 iterations, for a relative residual norm decrease of six orders of magnitude. The mesh comprises 504 947 tetrahedral elements and 92 564 nodes, resulting in 623 003 edges, with 6.47% grouped as *superedge3* and 53.97% grouped as *superedge6*. Figure 2 shows a view of the surface mesh of this model.

Table VI shows the percentage of reduced i/a edges (Red %) and the vector length for the submarine mesh. According to Table VI, a considerable amount of reduced edges is reached, even when using large vector lengths for level 1, except for variable vector length at level 2.

Table VII shows the vector length for edges arranged as simple edges or *superedges*. The entries indicate a more regular group formation around the imposed vector length, even for larger vector length values, than reduced i/a scheme, implying in a better behaviour of the former during computations.

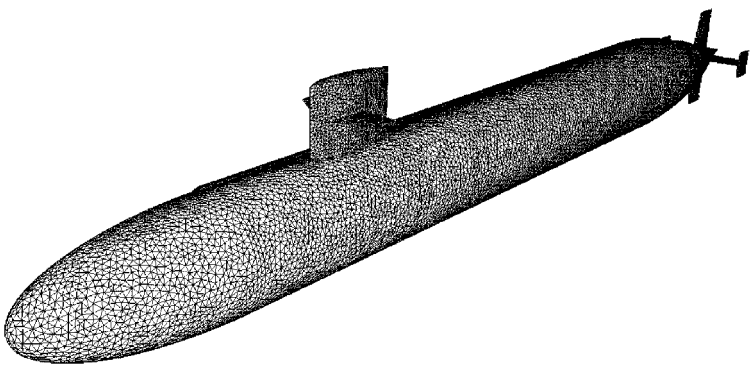

Figure 2. Surface mesh of Los Angeles class submarine.

Table VI. Vector length for reduced i/a scheme for submarine mesh.

		Nvec						
Level	Type	64	128	256	512	1024	2048	Free
0	Red %	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	Nvec avg	63	124	234	403	606	809	1091
1	Red %	98.2	98.9	99.1	99.1	99.1	99.1	99.1
	Nvec avg	64	126	243	431	674	935	1337
2	Red %	98.2	96.2	91.6	85.2	79.6	70.3	28.9
	Nvec avg	64	128	254	505	994	1929	10 559

Table VII. Vector length for *superedge* scheme for submarine mesh.

		Nvec						
Data structure		64	128	256	512	1024	2048	Free
Simple		64	128	255	506	1000	1953	12 714
Super	S6	60	126	252	509	1016	2025	14 618
	S3	63	126	255	510	1007	2014	10 072
	One edge	64	128	256	510	1019	2004	11 205

Table VIII shows the reordering times on Pentium IV. There is a trend of 75% overcharge in time considering the ratio between reduced i/a and *superedge* scheme. Note though that the reduced edge algorithm comprises both nodal and edge renumbering tasks while the *superedge* scheme just performs the edge renumbering task.

The resulting edge numbering for the RCM nodal ordered submarine mesh is illustrated in Figure 3. The edge list is sorted increasingly by first node after RCM algorithm. This picture corresponds to simple edge arrangement without nodal disjointing algorithm. It is clear the nodal agglomeration as edge number increases; the lesser the black line amplitude, the better

Table VIII. Timings for renumbering algorithms in Pentium IV (in s).

Scheme	Edges generation	Renumbering kernel	Total
Superedge	0.91	1.21	2.12
Reduced		2.16	3.07

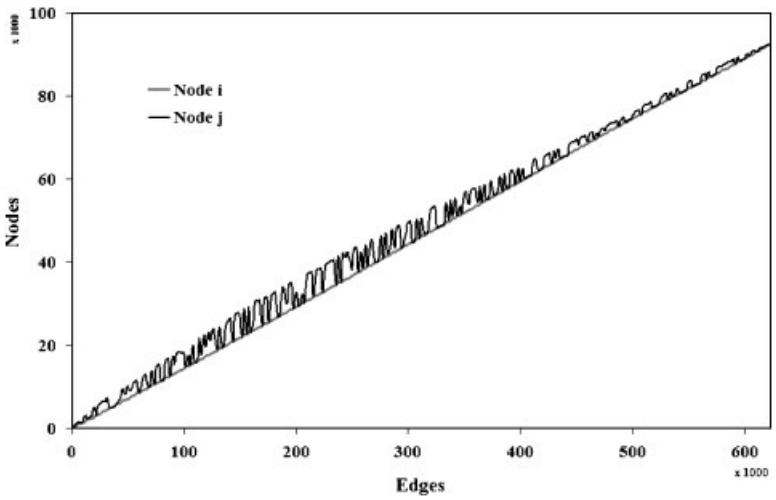

Figure 3. Nodal distribution according to the edges in submarine for original mesh: first edge node in grey and second one in black line.

the data locality. Consequently, we may expect a performance improvement due to a good data fitting into the cache. It is important to note though that the grey line is not straight and has little segments representing the several edges that share the same node.

Figure 4 shows the nodal distribution after use of Algorithm 4 on the mesh and edge ordering by level 0. The edges are ordered in such a way that the first node (grey line) is accessed with stride 1 as the edges are swept, forming groups with a saw aspect. The second node layout is formed by an irregular but well characterized strip, corresponding to a reasonable data locality, since the amplitude is somewhat large and jumps are not so smooth. This data distribution forecasts a reasonable use of cache. By comparing Figures 3 and 4, the second node distribution (black line) of the former is smoother then the latter, revealing its better adequacy for cache use.

Since the first node is not loaded from memory, the performance of the edge list represented in Figure 4 is related to the indirect addressing minimization along with the data locality of the second node. This combination has to be efficient to pay off the lesser data locality of this distribution in comparison to the one shown in Figure 3. Still related to second node, although it presents a regular spreading with a saw decay, its amplitude variation seems to be irregular enough to disturb data locality.

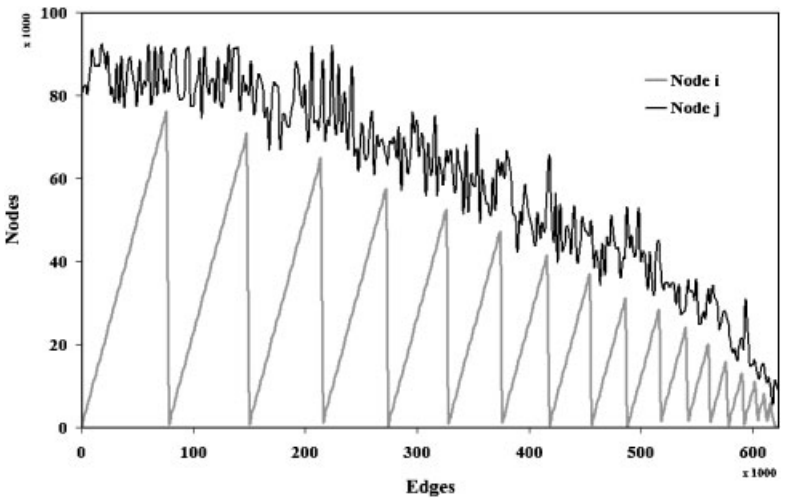

Figure 4. Nodal distribution according to the edges in submarine mesh after reduced nodal renumbering for level 0: first edge node in grey and second one in black line.

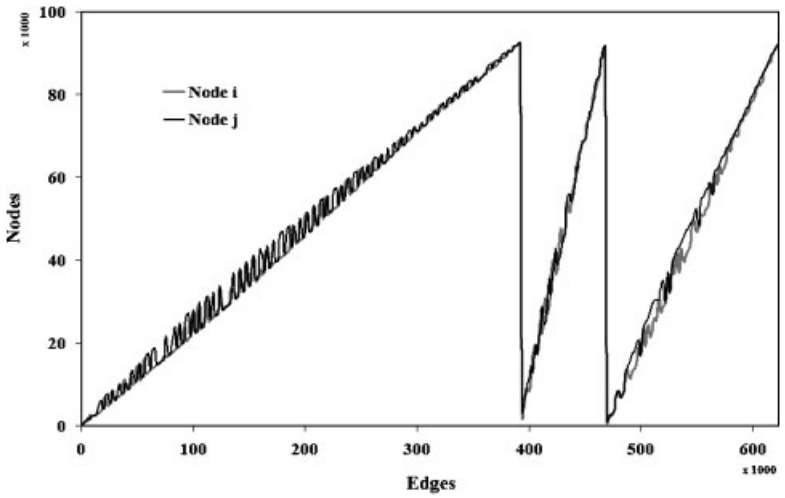

Figure 5. Nodal distribution according to the edges in submarine mesh after *superedge* groups assembling: first edge node in grey and second one in black line.

Figure 5 pictures the edge list arranged as *superedges*. The same behaviour of the nodal distribution presented in Figure 3 is kept here but in ranges of groups of six, three and simple edges, respectively. The number of edges grouped in six edges is larger than the other groups. The number of groups with three edges is the least.

The straight lines between two groups indicate the gap between the last and the first nodes of two contiguous edge groups. These jumps do not harm the performance of matrix–vector product algorithm since each group is handled by its own loop, that is, there are three loops

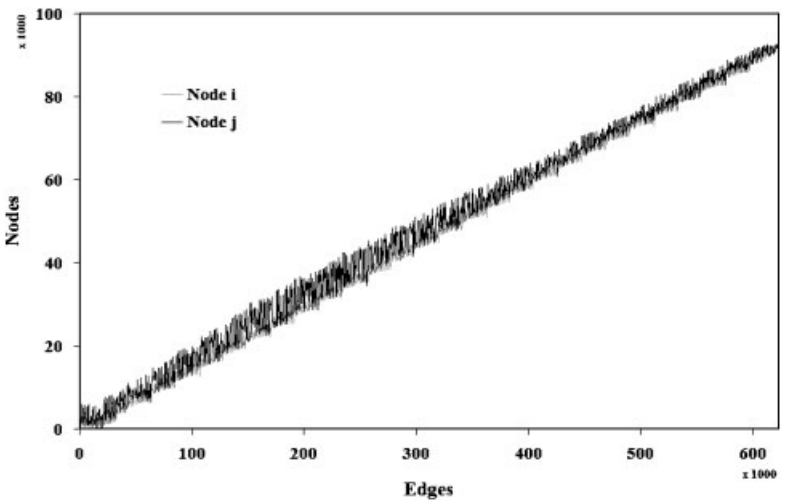

Figure 6. Nodal distribution according to the edges in submarine mesh after nodal disjointing renumbering for simple edge arrangement: first edge node (grey) and second one (black).

in the matrix–vector multiplication algorithm. Thus, in the first range of edges (*superedge6*), the compounding edges take the benefits of data locality besides the benefits of *superedge* loop (loop unrolling, indirect addressing minimization and optimal registers use). The same is valid for the second range composed by *superedge3*.

Important to note in Figure 5 is the capability of assembling *superedges* groupings without destroying the good data locality inherited by the simple edge arrangement. Thus, this arrangement represents an increasing of capabilities in efficient use of computer resources, beyond good use of memory hierarchy. The reordering disturbance is caused mainly in the first edge as can be seen by its distribution somewhat irregular in some places but with good amplitude similar to the second node. However, one has to be aware of the computational complexity of matrix–vector multiplication algorithm for *superedge* groups. Depending on the architecture, one has to choose the best combination, which sometimes can comprise only *superedges3* and simple edges.

The effect of nodal disjointing is now shown by Figure 6 which presents the edge nodal connectivity for simple edge arrangement and glimpses the other ones. The list length is set to 2048 edges for better viewing purposes, although lists ranging from 64 up to 16 394 and without limit were assembled and tested. The effect of nodal disjointing is represented by a saw aspect. This leads to worse cache use, compared to the previous orderings, since this distribution slightly disturbs data locality. The nodal disjointing algorithm tries to keep the nodal locality, but assembling edge lists without common nodes. Despite this disturbance, the nodal locality is better than that observed in the case of the second node in the reduced edge ordering. This means that in cache-based environments the present arrangement can be even more efficient than the reduced edge scheme.

Figure 7 shows the CPU times in Pentium IV for the mesh ordered by RCM and Algorithm 4, hereafter indicated in pictures as RCM and R'd, respectively, for both conventional and alternative RHS evaluation loop layouts, hereafter indicated in pictures as C'RHS and A'RHS,

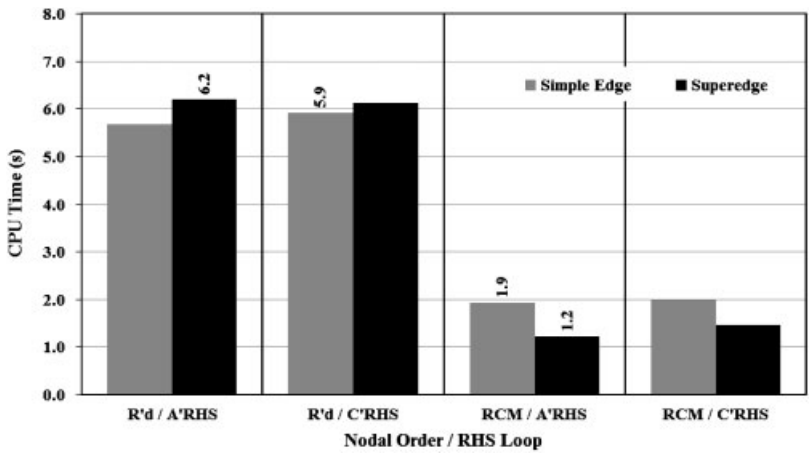

Figure 7. Timings of reduced nodal and simple/*superedge* edge ordered mesh and RHS evaluation loop versus CPU time for Submarine analysis in Pentium IV.

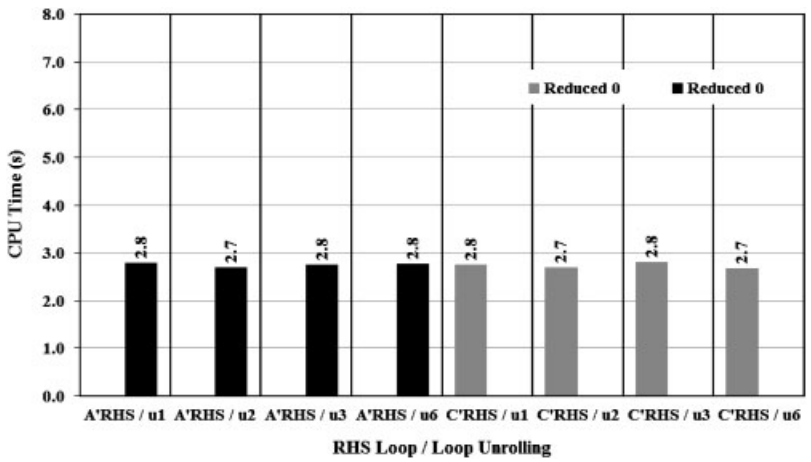

Figure 8. Timings of reduced nodal and edge level 0 ordered mesh and RHS evaluation loop versus CPU time for Submarine analysis on Pentium IV.

respectively, for continuous loop mode. That is, without chunks or nodal disjointing algorithm, for simple and *superedge* schemes. The best results point to mesh ordered by RCM algorithm since it provides a better data locality distribution, reaching a ratio of 3 between reduced and RCM orderings. There is no difference between RHS loop layouts and the *superedge* scheme produces gains about 35% over the simple edge one.

Figure 8 shows CPU time results for the mesh ordered by Algorithm 4 and edge ordered by level 0 along with both conventional and alternative RHS evaluation loop layouts, for continuous loop mode, besides not unrolled, unrolled into two, three and six edges loop layout, in Pentium IV, hereafter indicated in pictures as u1, u2, u3 and u6, respectively. The results do not show significant differences between these loop configurations.

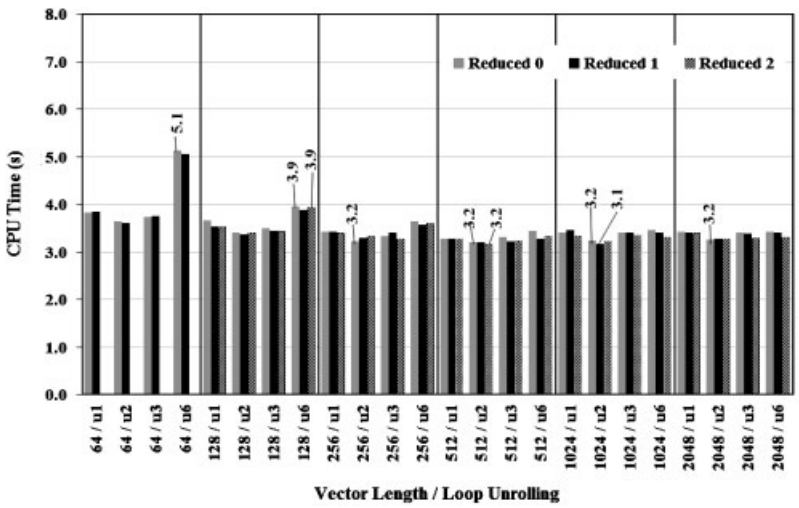

Figure 9. Timings of reduced nodal and edge ordered mesh and conventional RHS evaluation loop versus CPU time for Submarine analysis on Itanium 2 for reduced edge scheme.

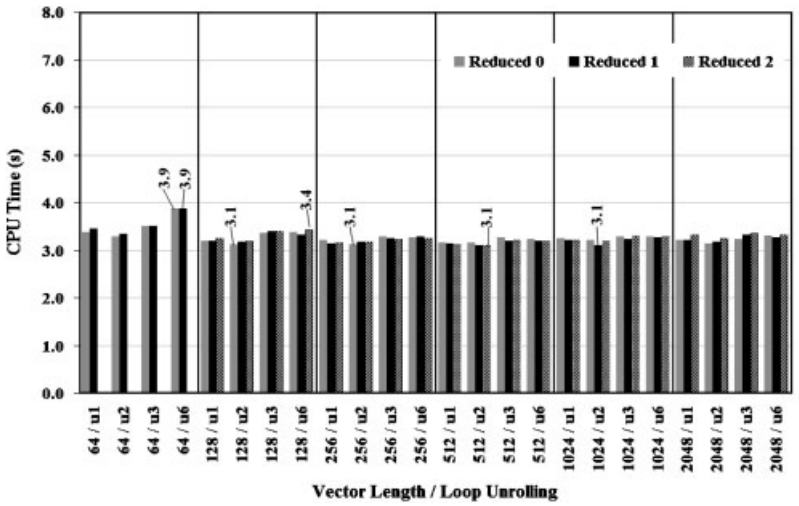

Figure 10. Timings of reduced nodal and edge ordered mesh and alternative RHS evaluation loop versus CPU time for Submarine analysis on Itanium 2 for reduced edge scheme.

Figure 9 pictures timing results for the mesh ordered by Algorithm 4 and edges ordered by reduced levels 0, 1 and 2 comprising loop layouts with vector lengths ranging from 64 up to 2048 and not unrolled, unrolled into two, three and six edges. For Itanium 2 processor it was also considered the conventional RHS evaluation. The best results come up after vector length 256 and for loop unrolled into two edges. Results are similar for the three levels.

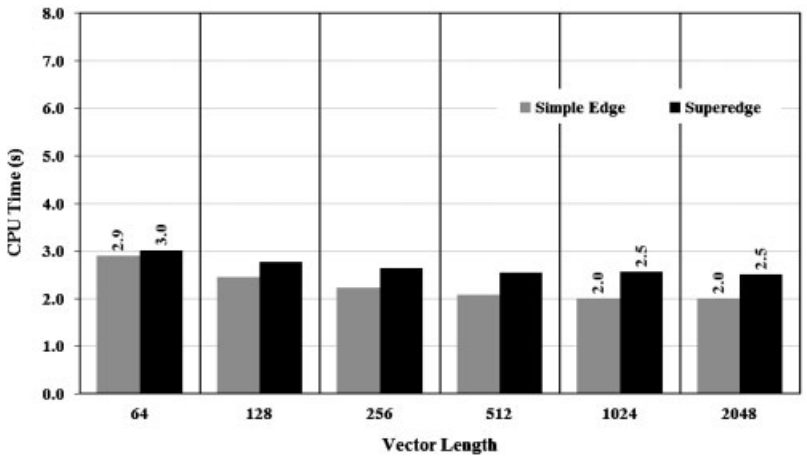

Figure 11. Timings of RCM nodal and simple and *superedge* edge ordered mesh and conventional RHS evaluation loop versus CPU time for Submarine analysis on Itanium 2.

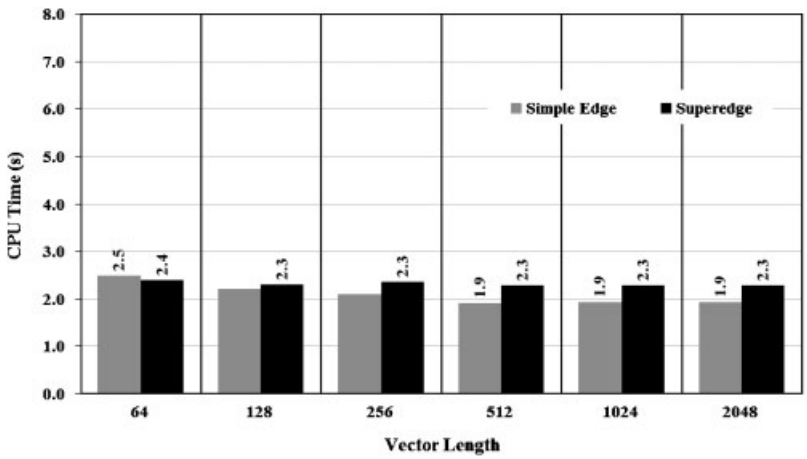

Figure 12. Timings of RCM nodal and simple and *superedge* edge ordered mesh and alternative RHS evaluation loop versus CPU time for Submarine analysis on Itanium 2.

The scenario presented by Figure 9 is slightly modified when alternative RHS evaluation is used as shown by Figure 10. This technique reduces timings in almost 25% for worst case, but keeps on the same behaviour for the further vector lengths. Besides, it can be observed a subtle trend in smoothing the differences among the three levels for greater vector lengths.

Figures 11 and 12 present results for the mesh ordered by RCM algorithm and simple and *superedge* ordered together with vector length loop layouts ranging from 64 up to 2048 edge lengths, in Itanium 2. Again, the effects are significant for the lesser lengths of 64 and 128,

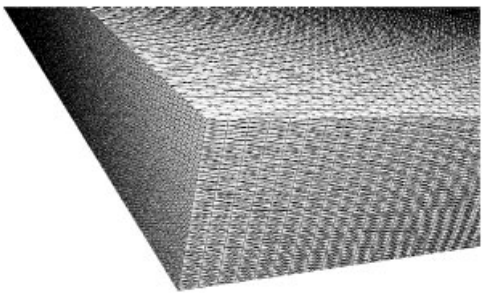

Figure 13. SPE10 surface mesh detail.

Table IX. Vector length values for reduced i/a scheme for SPE10 mesh.

		Nvec						
Level	Type	64	128	256	512	1024	2048	Free
0	Red %	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	Nvec avg	60	113	196	310	436	49	459
1	Red %	93.9	96.0	96.0	96.0	96.0	96.0	94.1
	Nvec avg	64	128	255	507	1000	1959.0	5582
2	Red %	93.9	95.8	95.7	95.5	95.4	94.4	74.5
	Nvec avg	64	128	256	512	1023	2044	16 064

reaching 20% reduction for simple edge scheme at 64 vector length. These results indicate that the best scenario of data arrangement is the simple edge with vector length no lesser than 512.

4.2. SPE10

This problem consists in the pressure determination in a heterogeneous three-dimensional five-spot problem based on the 10th SPE Comparative Solution Model [29]. The model comprises 5 610 000 tetrahedral elements and 1 159 366 nodes, resulting in 6 843 365 edges, with 5.7% grouped as *superedge3* and 61.3% grouped as *superedge6*. The elements were generated over a structured hexahedra mesh, similar to a finite difference mesh, dividing each hexahedron in 5 tetrahedra. A surface mesh detail is presented in Figure 13. Table IX shows the percentage of reduced i/a edges and vector length values according to the edge scheme preprocessing.

By comparing Table IX to Table VI, one can note the little effect of mesh topology over the nodal renumbering strategy, since the results remain almost unchanged from Table VI. The differences come from the vector length values over the amount of edges available for lists assembly. Table X shows the vector length values for usual i/a edges arranged as simple or *superedges*.

The following figures show the timing results for matrix–vector multiplication algorithm for all data and loop layout combinations after 1620 iterations for a relative residual norm decrease of six orders of magnitude.

Table X. Vector length distribution for *superedge* scheme preprocessing for SPE10 mesh.

Data structure		Nvec						
		64	128	256	512	1024	2048	Free
Simple		64	128	255	506	1000	1953	11 754
Super	S6	60	126	252	509	1016	2025	14 618
	S3	63	126	255	510	1007	2014	10 072
	One edge	64	128	256	510	1019	2004	11 204

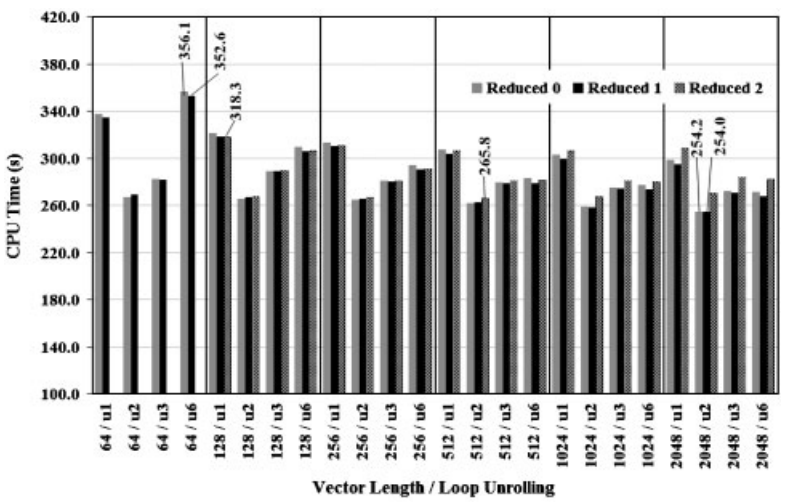

Figure 14. Timings of reduced nodal and edge ordered mesh and conventional RHS evaluation loop versus CPU time for SPE analysis on Itanium 2.

Figure 14 indicates the best results for reduced level 0 and 1 schemes, organized in lists of length 2048 and loop unrolled into two edges, considering the mesh ordered by Algorithm 4. Comparison between this figure and Figure 15 shows the different effects of alternative RHS evaluation over performance, since it produces gains of about 10% over previous worst results for lists of length 64 and about 5% for remaining vector lengths but, otherwise, worsen the best result in about 7%. In this case, these figures show that alternative and conventional RHS evaluation option is critical and has to be determined experimentally since there is no way to predict the algorithm behaviour under such data configurations.

Comparing Figures 16 and 17 we may see the great potential for performance optimization by just changing to the alternative RHS loop layout for this mesh. There is a time reduction of about 25% for simple and *superedges* simultaneously. The advantage of the *superedge* scheme over simple edge, comprising the registers reuse and flops and i/a reduction, is not enough to pay off data locality disturbance as can be seen by the gap of about 20%, for the best results. Regarding vector length, the best results correspond to list lengths greater then 512.

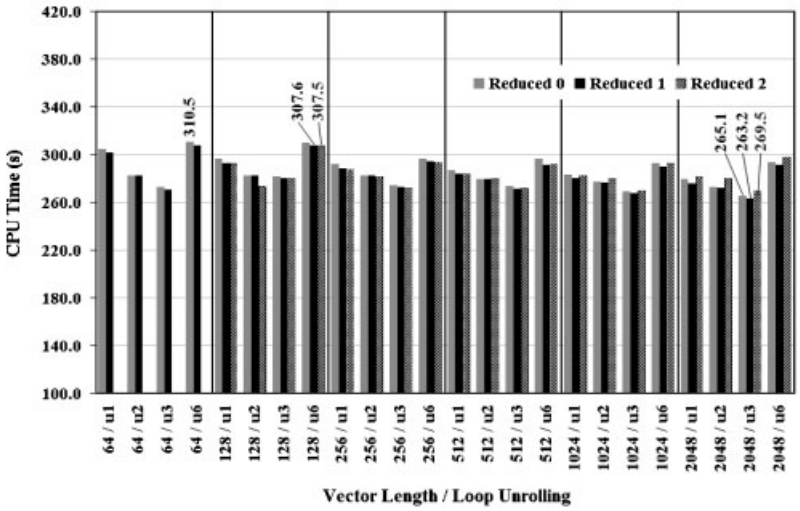

Figure 15. Timings of reduced nodal and edge ordered mesh and alternative RHS evaluation loop versus CPU time for SPE analysis on Itanium 2.

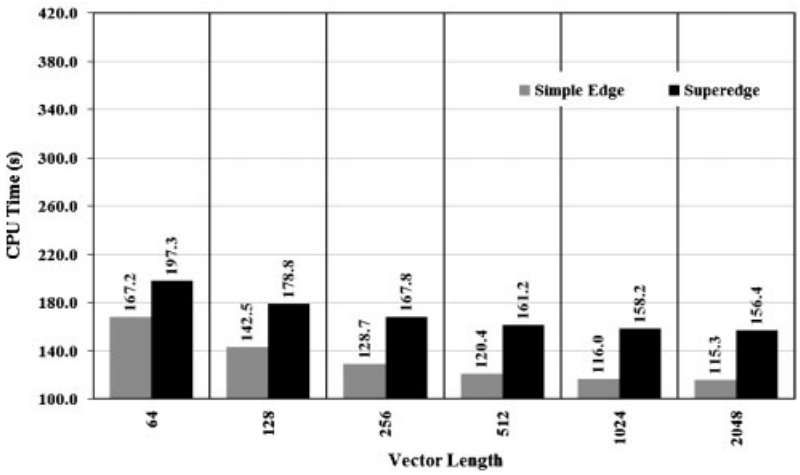

Figure 16. Timings of RCM nodal and simple and *superedge* edge ordered mesh and conventional RHS evaluation loop versus CPU time for SPE analysis on Itanium 2.

4.3. Sedimentary basin

This example simulates the interaction of four blocks of a sedimentary basin separated by three geological faults. Geometry and material properties are similar to a portion of a sedimentary basin in the northeast of Brazil. The blocks were submitted to a longitudinal compression. All displacements on top are free and, at the basis and lateral surfaces, normal displacements are prescribed. Two meshes were used and the first one comprises 26 860 nodes, 94 058 linear

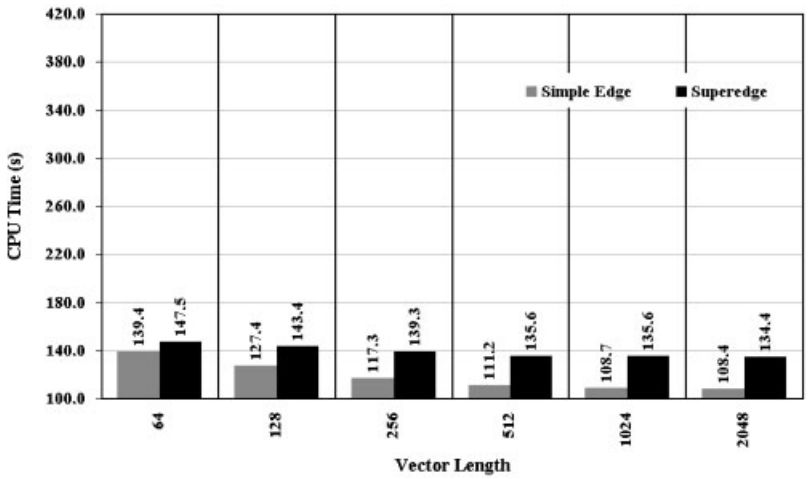

Figure 17. Timings of RCM nodal and simple and *superedge* edge ordered mesh and alternative RHS evaluation loop versus CPU time for SPE analysis on Itanium 2.

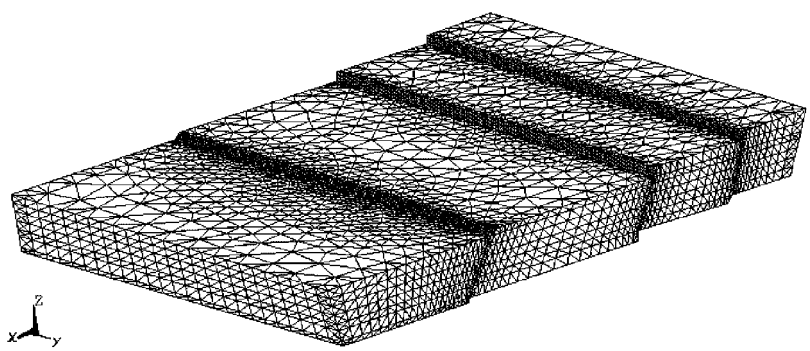

Figure 18. Surface mesh 1 of sedimentary basin model.

tetrahedral elements, 12 960 interface elements and 76 380 equations. The resulting number of edges is 187 266 and 40.08% were grouped in *superedge6*, 16.99% in *superedge3*, 8.94% in *superedge9* and 0.88% in *superedge4*; the percentage of remaining simple edges is 33.12%. Figure 18 shows a view of the surface mesh of this model.

Table XI shows the percentage of reduced i/a edges and vector length according to this edge scheme assembly for mesh 1. As one can see in Table XI, the average vector length obtained does not keep up with the length proposed for level 0 and 1, due to mesh size. The best results go through up to length of 256. At level 2 though, the vector length obtained is reasonable up to 512. Table XII shows the vector length for usual i/a edges arranged as simple edges or *superedges* in mesh 1. The vector lengths obtained match with the values imposed, with subtle difference for high values, unless for S4 groups, where the differences are higher, because of the low quantity of interface elements.

Table XI. Vector length for reduced i/a scheme for sedimentary basin mesh 1.

		Nvec						
Level	Type	64	128	256	512	1024	2048	Free
0	Red %	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	Nvec avg	64	128	233	354	482	569	670
1	Red %	84.1	85.2	85.3	85.2	85.4	85.2	85.4
	Nvec avg	64	126	236	402	601	798	982
2	Red %		81.0	75.2	68.3	58.3	46.8	
	Nvec avg		128	254	501	962	1804	

Table XII. Vector length for *superedge* scheme for sedimentary basin mesh 1.

		Nvec						
Data structure		64	128	256	512	1024	2048	Free
Simple		64	128	255	507	1007	1985	6731
Super	S9	63	126	251	501	1011	2000	11 907
	S6	60	126	251	503	988	1899	9820
	S4	63	125	235	428	612	920	940
	S3	63	126	252	501	968	1737	6082
	One edge	64	127	253	500	983	1923	6754

Table XIII. Timings for renumbering algorithms in Pentium IV (in s) for mesh 1.

Scheme	Edges generation	Renumbering kernel	Total
Superedge	0.34	0.24	0.58
Reduced		0.78	1.12

Table XIII shows the time spent for preprocessing phase in the Pentium IV for mesh 1. The values presented comprise the edges generation and the renumbering procedures for edges in the case of *superedges* and edges and nodes in the case of reduced edges.

The nodal connectivity of edges for both meshes ordered by RCM is similar to Figure 3 (submarine mesh), that is, the nodal distribution along edges is regular and rather favourable for efficient cache use, because the second node amplitude is small along all edges. Equivalent similarity occurs for the nodal connectivity of reduced edges, which presents the second node distribution with irregular amplitude.

The nodal connectivity of edges is now submitted to *superedge* technique to assembly S6, S3, S9 and S4, that is, *superedges* groups set by six, three, nine and four edges, respectively. The resulting nodal connectivity of edges is shown in Figure 19 for mesh 1. The S6 and

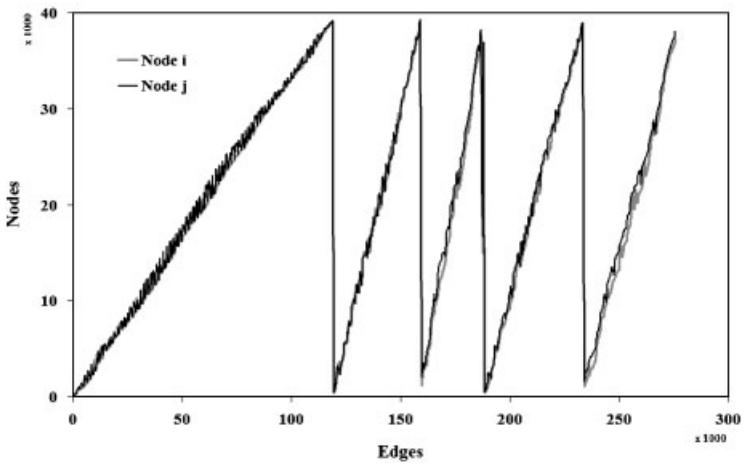

Figure 19. Nodal distribution according to the edges in mesh 1 after *superedge* groups assembling: first edge node in grey and second one in black line for mesh 1.

S3 groups came out from tetrahedra and S9 and S4 from interface elements. The groups are characterized by four ramps and the remaining, simple edges are in the last ramp. Each ramp corresponds to S6, S3, S9, S4 and simple edge connectivity respectively. All sets present good data locality and forecast good use of cache facilities. Again (as before in the submarine mesh—*viz*. Figure 5), the edge renumbering in *superedge* groups does not disturb data locality of original edge list.

Edge distributions are sensitive to the nodal disjointing technique, and similarly to the submarine mesh, present a saw aspect. For reduced edge level 0 though, the effect is more impacting than for the submarine mesh. In general, for all distributions with nodal disjointing, it is observed a worsening in data locality.

Due to material non-linearities, the trade-off between matrix–vector products and Jacobian matrix coefficient and residuals evaluations has to be weighted regarding the element connectivity by edges. Besides, the nodal connectivity of elements plays an important role during stress integration computations.

For mesh ordered by RCM, these connectivities present good data locality, similar to edge connectivity by nodes, even when organized as *superedges*. However, for mesh ordered by Algorithm 4, level 0, the resulting distribution reveals very bad data locality and forecast bad performance of this distribution during non-linear computations.

Timing results on Pentium IV for mesh 1 were collected after 720 PCG iterations, that is, for a relative residual tolerance of 10^{-6}. Figure 20 shows timing results for mesh 1 ordered by RCM and Algorithm 4 for both conventional and alternative RHS evaluation loop layouts, for continuous loop mode. That is, without chunks or the nodal disjointing algorithm, for simple and *superedge* schemes. The best results are for the mesh ordered by RCM algorithm, due to its better data locality, reaching a ratio of 2 between reduced and RCM orderings. The alternative RHS loop layouts produce reductions of about 15% in time compared to the conventional one.

Figure 21 shows the timing results in Pentium IV for mesh 1. Nodes are ordered by Algorithm 4 and edges ordered by level 0 along with both conventional and alternative RHS

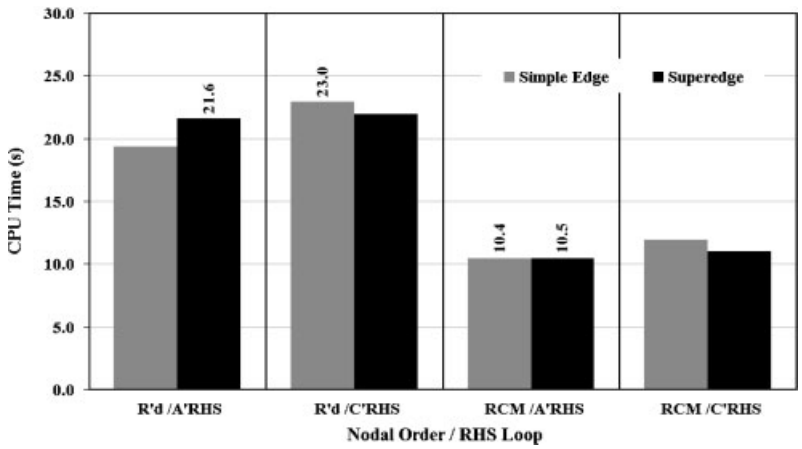

Figure 20. Timings of RCM nodal and simple and *superedge* edge ordered mesh and conventional RHS evaluation loop versus CPU time for mesh 1 analysis on Pentium IV.

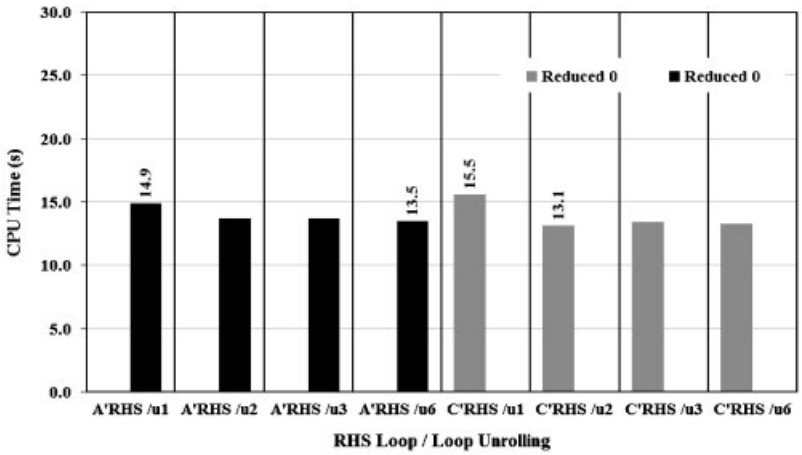

Figure 21. Timings of RCM nodal and simple and *superedge* edge ordered mesh and alternative RHS evaluation loop versus CPU time for mesh 1 analysis on Pentium IV.

evaluation loop layouts, for continuous loop mode, besides not unrolled, unrolled into two, three and six edges loop. The results do not show significant differences between configurations with a slight advantage of alternative RHS over the conventional one. Unrolled loops are subtly more efficient then not-unrolled, with reductions of about 10%.

The second mesh comprises 371 244 nodes, 2 064 940 linear tetrahedral elements, 17 317 interface elements and 1 098 257 equations. The resulting number of edges is 2 553 563 of which 63.47% were grouped in *superedge6*, 12.75% in *superedge3*, 0.99% in *superedge9* and 0.03% in *superedge4* and the percentage of remaining simple edges is 22.76%. Figure 22 shows a detail of the surface mesh around a fault.

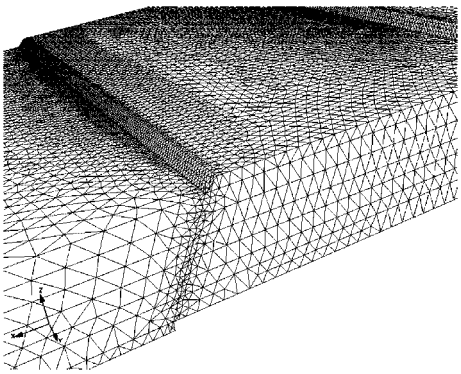

Figure 22. Detail of surface mesh 2 of sedimentary basin model.

Table XIV. Vector length for reduced i/a scheme for sedimentary basin mesh 2.

		Nvec						
Level	Type	64	128	256	512	1024	2048	Free
0	Red %	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	Nvec avg	64	126	234	398	598	786	1069
1	Red %	93.7	94.4	94.7	94.8	94.8	94.8	94.8
	Nvec avg	64	127	243	439	717	1032	1626
2	Red %		91.6	88.5	85.1	81.5	76.8	
	Nvec avg		128	256	511	1020	2025	

Table XV. Vector length for *superedge* scheme for sedimentary basin mesh 2.

		Nvec						
Data structure		64	128	256	512	1024	2048	Free
Simple		64	128	256	511	1022	2040	15 110
Super	S9	63	125	251	512	1013	1947	12 659
	S6	60	126	252	523	1018	2036	15 890
	S4	64	127	212	318	780	780	780
	S3	63	126	254	423	1014	2010	13 565
	One edge	64	128	255	521	1019	2018	13 206

Table XIV shows the percentage of reduced i/a edges and vector length according to this edge scheme assembly for mesh 2. The benefits over reduced edge assembly from increasing mesh size can be observed by comparing Tables XIV and XI. However, the average vector length has the same behaviour, without further improvements.

Table XV shows the vector length for usual i/a edges arranged as simple edges or *superedges* in mesh 2. The benefits of mesh size increasing are observed granting steady average vector

Table XVI. Timings for renumbering algorithms in Itanium 2 (in s) for mesh 2.

Scheme	Edges generation	Renumbering kernel	Total
Superedge	6.13	12.64	18.77
Reduced level 0		42.33	48.46

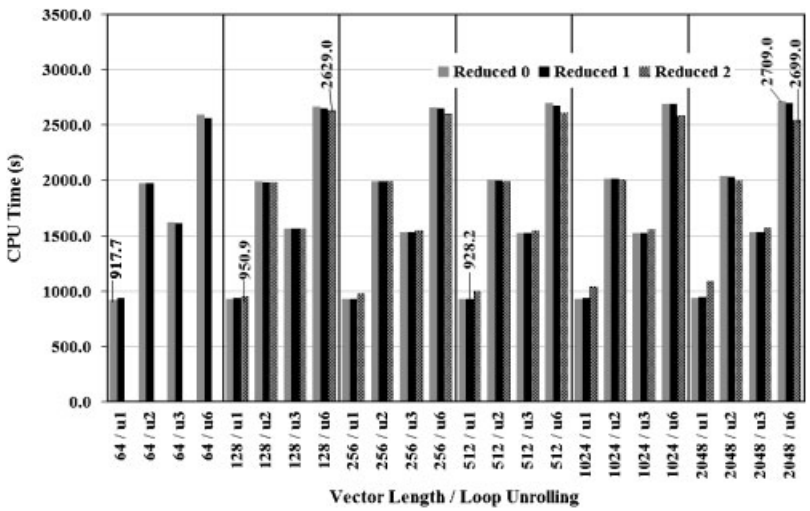

Figure 23. Timings of reduced nodal and edge ordered mesh and conventional RHS evaluation loop versus CPU time for mesh 2 analysis on Itanium 2 for reduced edge scheme.

lengths up to 2048, mainly for simple edge scheme. For S4 *superedge* type however, the vector length sticks at 780 since this is the total number of this group type. By comparing Tables XII and XV, it can be concluded that refining the fault mesh leads to an increasing amount of S9 due to S4 assembly decreasing. An overall comparison between *superedges* assembly of meshes 1 and 2 indicates ratios of S6, S3, S9, S4 and simple edges of about 1.6, 0.75, 0.11, 0.034 and 0.69, respectively.

Table XVI shows the time spent for preprocessing phase in the Itanium 2 for mesh 2. The values presented comprise the edges generation and the renumbering procedures for edges in the case of *superedges* and edges and nodes in the case of reduced edges for level 0 only.

Figures 23–26 present timing results for the mesh 2, which were collected after 3180 PCG iterations, that is, convergence was achieved for a relative residual tolerance of 10^{-6}.

The effect of both modes for RHS evaluation loops is shown in Figures 23 and 24. Disregarding the vector lengths, the unrolling level suffers from RHS alternatives as a pattern. Considering the use of RHS loop mode from conventional to alternative, for the non-unrolled loops there is a gain of about 10%; unrolling loops in two edges produces significant gains, of about 30%; unrolling loops in three and six edges worsen results in about 5%. These figures

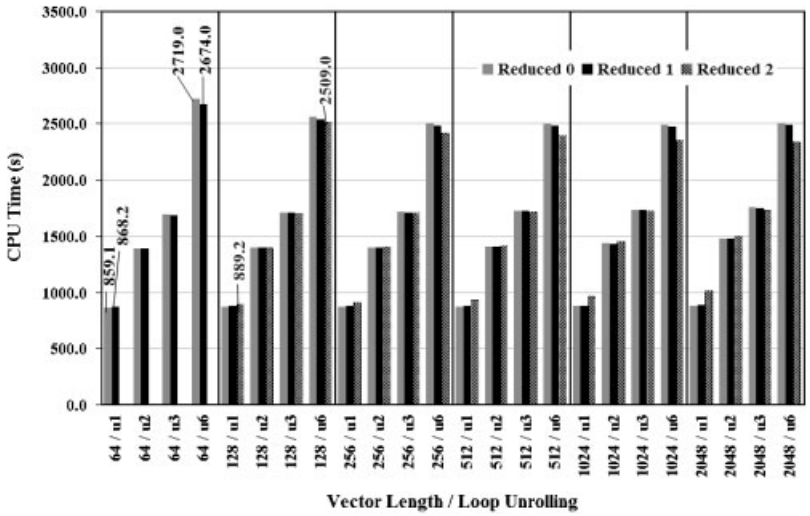

Figure 24. Timings of reduced nodal and edge ordered mesh and alternative RHS evaluation loop versus CPU time for mesh 2 analysis on Itanium 2 for reduced edge scheme.

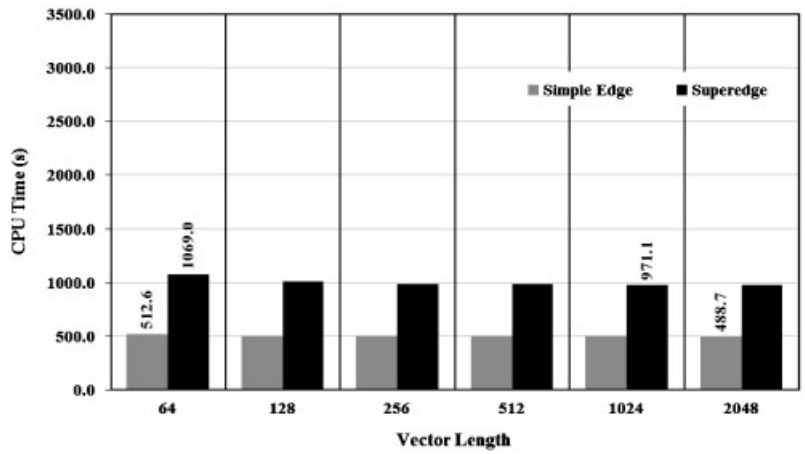

Figure 25. Timings of RCM nodal and simple and *superedge* edge ordered mesh and conventional RHS evaluation loop versus CPU time for mesh 2 analysis on Itanium 2 for simple and *superedge* schemes.

illustrate the complete unforeseeable behaviour of this scenario regarding to the choice of RHS evaluation loop modes.

Figures 25 and 26 shows a gain of about 25% in performance for the *superedge* scheme when the mesh is ordered by RCM algorithm. The performance of simple edge keeps stable without relevant changes and sticks as best performance of data and loop layout combination for this scenario.

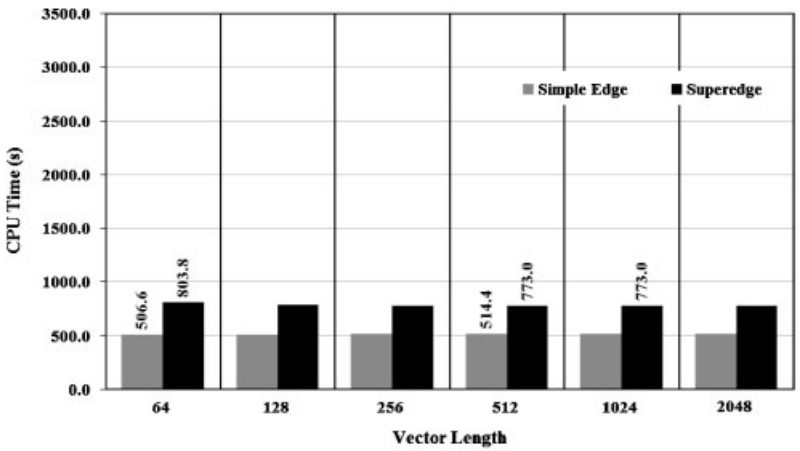

Figure 26. Timings of RCM nodal and simple and *superedge* edge ordered mesh and alternative RHS evaluation loop versus CPU time for mesh 2 analysis on Itanium 2 for simple and *superedge* schemes.

5. CONCLUSIONS

This work presented a comparison of performance results among several edge-based data structures for finite-element analyses driven by iterative solvers. Several optimizations techniques were employed into the symmetric matrix–vector product algorithm, focusing on serial performance. The analyses comprised three-dimensional domains for flows and geomechanical problems modelled by unstructured grids composed of tetrahedral elements.

As a critical issue in iterative solvers behaviour, the underlying computational parameters of the edge-based matrix–vector product algorithm were investigated, considering CPU and memory hierarchy use, that is, floating point operations (flops), indirect addressing operations (i/a), registers reuse and data locality concepts. Serial processing optimization was focused, since it is expected that the local processor improvements are spread to multiprocessor systems [14]. Some additional devices were investigated, namely: loop unrolling, loop splitting in chunks and memory sharing (by residual vector) during right-hand side evaluations [16].

Regarding all these concepts, Table XVII briefly presents the main characteristics for 1 dof/node edge-based matrix–vector multiplication loop, emphasizing the computational complexity of each configuration. In this table, 'Var' is the number of variables within the loop, 'i/a/edge' means indirect addresses per edge and 'flops/edge' the number of floating point operations per edge. The final column remarks the main characteristics of each configuration, based on a complete compilation over hundreds of results collected.

Further, the final performance is significantly affected by loop layout alternatives. Based on results, it was observed that for the Pentium IV, regarding 1 dof/node and all edge schemes, the loop unrolling technique and alternative RHS evaluation were not worth use unless when using the latter one along with loop splitted into chunks for *superedge* scheme, producing gains of about 30% compared to conventional RHS evaluation layout as can be concluded by observing Figure 7.

Table XVII. Edge schemes and main characteristics for 1 dof/node.

Scheme	Var	i/a/edge	flops/edge	Remarks
Simple edge	13	6	4	• Biased to data locality exploitation; • Simplest and most flexible structure
Superedge6	38	12/6	31/6	• Biased to data reuse by registers;
Superedge3	24	9/3	15/3	• Data locality disturbed by *superedge* assembly; • Most complex structure
Reduced 0	12	3	5	• Data locality disturbed by reduced scheme assembly

For Itanium 2 under the same conditions, loop unrolling for reduced scheme is recommended into two edges as can be seen in Figure 14 and splitting into chunks greater then 128 is a better choice. The best results are observed for loop unrolled into two edges and chunks of 2048 edges. Although the alternative RHS evaluation loop has produced a favourable impact of about 10% in time reduction for the unrolled loop, it is not recommended since the final trade off is not worth as can be concluded by comparing Figures 14 and 15. These effects are not perceivable for the submarine mesh as shown in Figures 9 and 10.

Regarding simple and superedge schemes, the combination of chunks greater then 512 edges with alternative RHS evaluation loop produced the best results of all analyses and are strongly recommended mainly for simple edges schemes which corresponds to 20% in gain over *superedge* one.

For 3 dof/node, results shown that the more complex the algorithm, the simpler the structure together with good data locality distribution is needed to reach good results. The best results point to simple edge scheme with no sensitivity to chunk length and even RHS layout loop type as shown by Figures 25 and 26. By considering *superedges*, in other words, a more complex scheme, the alternative RHS feature can be used in order to minimize algorithm complexity but not enough to reach simple edge results.

The lack of data locality produced by the reduced (i/a) scheme does not allow better results when compared to simple edges, although there are more i/a operations in the latter compared to the former (*viz.* Table XVII). Further, it is important to note that even this drawback can be minimized by alternative RHS evaluation loop, this is not enough to pay off, as can be observed by comparing Figures 23 and 24.

Table XVIII presents the summary of all results related to machine, dof/node, scheme and gives some remarks about each configuration. In all results, data locality is the backbone for best results.

The overall results reveal a great variety of behaviour for all loop layouts, underlying physics that is the number of degrees of freedom per node, and processor types, besides mesh size. Considering specific cases of configurations, that is, the synergy between data and architecture, the gains in processing time are significant. However these configurations are almost impossible to be set previously. Regarding all these characteristics and consequently the awful task of searching the best combination of data structure and compiler options for a specific platform, as a future work, it is proposed the development of an auto-configurable set up software based on partial timing results in order to achieve the best configuration comprising all variables, that is, the physics involved, hardware platform and compiler characteristics, by just choosing the best timing results.

Table XVIII. Results summary (DOF = degrees of freedom per node; ND = nodal disjointing).

Platform	DOF/ND	Scheme	Remarks
Pentium IV	1/No	• RCM • *Superedge* • Chunks of 128 edges	• Both edge and nodal reduced ordering produce bad results; • Sensitivity to chunk length; • Little sensitivity to loop unrolling; • Great sensitivity to data locality; • No sensitivity to RHS evaluation mode.
	3/No	• RCM • *Simple edge* • Alternative RHS evaluation loop	• Both edge and nodal reduced ordering produce bad results; • No sensitivity to chunk length; • Worst results for not unrolled loop; results equally distributed for loop unrolling in two, three or six edges; • Great sensitivity to data locality; • Little sensitivity to RHS evaluation mode.
Itanium	1/Yes	• RCM • *Simple edge* • Chunks greater than 512 edges	• Both edge and nodal reduced ordering produce reasonable results; • Sensitivity to chunk length; • Sensitivity to loop unrolling; • Great sensitivity to data locality; • Sensitivity to RHS evaluation mode.
	3/Yes	*Simple edge*	• Good results for RCM simple edge scheme; • Bad results for *superedge* under conventional RHS evaluation loop layout; • Little sensitivity to chunk length; • Great sensitivity to data locality;

ACKNOWLEDGEMENTS

This work was partially supported by CNPq grant 500196/2002-8 and Intel under the research contract 'New Algorithms for Improving Processor Efficiency in Unstructured Grid Parallel Computations'. Computer time was provided by the Center of Parallel Computation of COPPE/UFRJ. The authors gratefully acknowledge Dr A. Aveleda from the Center of Parallel Computation of COPPE/UFRJ for technical support. We are also indebted to Prof M. Behr from RWTH Aachen University, Germany, by the submarine mesh.

REFERENCES

1. Hennessy JL, Patterson DA, Goldberg D. *Computer Architecture*: *A Quantitative Approach* (3rd edn). Morgan Kaufmann: Los Altos, CA, 2002.
2. www.top500.org, visited in November, 2004.
3. Feitelson DG. The supercomputer industry in light of the Top500 data. *Computing in Science and Engineering*, January/February, 2005; 42–47.
4. Burgess DA, Giles MB. Renumbering unstructured grids to improve the performance of codes on hierarchical memory machines. *Advances in Engineering Software* 1997; **28**:189–201.
5. Oliker L, Li X, Heber G, Biswas R. Parallel conjugate gradient: effects of ordering strategies, programming paradigms, and architectural platforms. *IEEE Transactions on Parallel and Distributed Systems* 2000; **11**(9):931–940.
6. Biswas R, Oliker L, Shan H. Parallel computing strategies for irregular algorithms. In *Annual Review of Scalable Computing*, Kwong YC (ed.), vol. 5, Chapter 1. World Scientific Publishing Company Inc.: Singapore, 2003.
7. Oliker L, Li X, Heber G, Biswas R. *Ordering Unstructured Meshes for Sparse Matrix Computations on Leading Parallel Systems*. Lecture Notes in Computer Science, vol. 1800, 2000; 497–503.

8. Oliker L, Li X, Husbands P, Biswas R. Effects of ordering strategies and programming paradigms on sparse matrix computations. *SIAM Review* **44**(3):373–393.
9. Pinar A, Heath MT. Improving performance of sparse matrix–vector multiplication. *Conference on High Performance Networking and Computing*, *Proceedings of the 1999 ACM/IEEE Conference on Supercomputing* (*CDROM*), Portland, Oregon, U.S.A., 1999, Article No. 30, ISBN:1-58113-091-0.
10. Vuduc R, Demmel JW, Yelick KA, Kamil S, Nishtala R, Lee B. Performance optimizations and bounds for sparse matrix–vector multiply. *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, Baltimore, MA, U.S.A., 16–22 November 2002; 1–35. CD-ROM ACM, 2002.
11. Douglas CC, Hu J, Kowarschik M, Rude U, Weiss C. Cache optimization for structured and unstructured grid multigrid. *Electronic Transactions on Numerical Analysis* 2000; **10**:21–40.
12. Ferencz RM, Hughes TJR. Iterative finite element solutions in nonlinear solid mechanics. *Handbook of Numerical Analysis*, vol. 6. Elsevier Science: Amsterdam, 1998.
13. Gropp WD, Kaushik DK, Keyes DE, Smith BF. Performance modeling and tuning of an unstructured mesh CFD application. *Proceedings of SC 2000*, *IEEE Computer Society*, Dallas, TX, U.S.A., 2000, Article No. 34, ISBN:0-7803-9802-5.
14. Heras DB, Blanco V, Cabaleiro JC, Rivera FF. Modeling and improving locality for the sparse-matrix–vector product on cache machines. *Future Generation Computer Systems* 2001; **18**:55–67.
15. Kennedy JG, Behr M, Kalro V, Tezduyar TE. Implementation of implicit finite element methods for incompressible flows on the CM-5. *Computer Methods in Applied Mechanics and Engineering* 1994; **119**:95–111.
16. Löhner R, Galle M. Minimization of indirect addressing for edge-based field solvers. *Communications in Numerical Methods in Engineering* 2002; **18**:335–343.
17. 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.
18. Löhner R. Some useful renumbering strategies for unstructured grids. *International Journal for Numerical Methods in Engineering* 1993; **36**:3259–3270.
19. Löhner R. Edges, stars, superedges and chains. *Computer Methods in Applied Mechanics and Engineering* 1994; **111**:255–263.
20. Martins MAD, Alves JLD, Coutinho ALGA. *Parallel Edge-based Finite Techniques for Nonlinear Solid Mechanics*. Lecture Notes on Computer Science, vol. 1981. Springer: Berlin, Heidelberg, 2001; 506–518.
21. Mavriplis D. Three-dimensional unstructured multigrid for the Euler equations. *Paper AIAA-91-1549-CP*, 1991.
22. Oliker L, Canning A, Carter J, Shalf J, Skinner D. Evaluation of cache-based superscalar and cacheless vector architectures for scientific computations. *Proceedings of the 18th Annual International Conference on Supercomputing*, Malo, France, 2004, ISBN:1-58113-839-3.
23. Ribeiro FLB, Coutinho ALGA. Comparison between element, edge and compressed storage schemes for iterative solutions in finite element analyses. *International Journal for Numerical Methods in Engineering* 2005; **63**(4):569–588.
24. Strout MM, Carter L, Ferrante J, Kreaseck B. Sparse tiling for stationary iterative methods. *International Journal of High Performance Computing Applications* 2004; **18**(1):95–113.
25. Carey GF, Swift S, McLey RT. Maximizing sparse matrix–vector product performance on RISC based MIMD computers. *Journal of Parallel and Distributed Computing* 1996; **37**:146–158.
26. Cuthill E, McKee J. Reducing the bandwidth of sparse symmetric matrices. *Proceedings of the ACM Nature Conference*, 1969; 157–172.
27. Dongarra J, Foster I, Fox G, Kennedy K, White A, Torczon L. *Sourcebook of Parallel Computing*. Morgan Kaufmann: Los Altos, CA, 2002.
28. Barth T. A 3-D upwind Euler solver for unstructured meshes. *AIAA-91-1548-CP*, 1991.
29. Sydenstricker RM, Martins MAD, Coutinho ALGA, Alves JLD. *Edge-based Interface Elements for Solution of Three-dimensional Geomechanical Problems*. Lecture Notes in Computer Science, vol. 2565, 2003; 53–64.
30. Coutinho ALGA, Martins MAD, Alves JLD, Landau L, Moraes A. Edge-based finite element techniques for nonlinear solid mechanics problems. *International Journal for Numerical Methods in Engineering* 2001; **50**(9):2053–2068.
31. Web Site for 10th SPE Comparative Solution Project, http://www.spe.org/csp, 10th February, 2004.

