Performance Optimization of GeoFEM on Various Computer Architecture

Kazuo Minami [(1)] and Hiroshi Okuda [(2)]

(1) Department of Computational Earth Sciences, Research Organization for Information Science and Technology (RIST), Tokyo, Japan (e-mail: minami@tokyo.rist.or.jp, phone: +81-3-3436-5271, fax: +81-3436-5274) (2) Department of Quantum Engineering and Systems Science, The University of Tokyo, Japan (e-mail: okuda@q.t.u-tokyo.ac.jp, phone: +81-3-5841-7426, fax: +81-3818-3455)

Abstract

In this research, we focus on improving the performance of GeoFEM on a single processor using a common data structure and coding approach in order to optimize GeoFEM for implementation on various computer architectures including parallel systems. A new data structure and direct access coding are developed for fluid analysis and implemented on scalar, vector, and pseudovector architecture. A 17% increase in peak performance is obtained on pseudovector and scalar architecture, and a 20% peak performance improvement is achieved on vector architecture. By applying a new direct access coding approach, the peak performance of the structure solver is increased by 23% on pseudovector architecture and 28% on vector architecture. Architecture-independent matrix assembly coding is developed and evaluated on vector and scalar machines. Performance of 736.8 Mflops was is obtained for the matrix assembly process and 900.7 Mflops for the entire code on an NEC SX-4 supercomputer. An average of 2.06 Gflops performance is obtained on a Fujitsu VPP5000 (peak: 9.6 Gflops), and 124 Mflops was is obtained for the matrix assembling process on 533-MHz 21164 Alpha system.

1. Introduction

The Science and Technology Agency of Japan inaugurated an Earth simulator project in 1997. The simulator is expected to be capable of predicting various Earth phenomena by analyzing an Earth model using a supercomputer. The specific research topics of the project are as follows:

1)Development of a high-performance massively parallel processing computer called the Earth Simulator (40 Tflops peak performance, 10 TBmemory)
2)Modeling of atmospheric and oceanic field phenomena and high-resolution simulations
3)Modeling and simulation of solid earth field phenomena
4)Development of large-scale parallel software for the Earth Simulator

The GeoFEM system, which is finite-element method (FEM) simulation code for unstructured grids, deals with topics 3 and 4. Development is planned in 2 phases:

Phase I: GeoFEM/Tiger (1997-1998): Multi-purpose parallel finite element software that may be applied to various fields in engineering and the sciences as well as becoming the basis for the solid earth simulator to be developed in Phase II.

Phase II: GeoFEM/Snake (1999-2001): A software system optimized for the Earth Simulator and developed specifically for the simulation of Earth phenomena such as mantle-core convection, buildup of tectonic stress, deformation of plates, and seismic wave propagation.

A wide range of computer architectures has been developed, and although GeoFEM is primarily targeted for implementation on the Earth simulator, it is expected to be used on various other platforms as multi-purpose parallel finite element software. As such computing-intensive applications require code that is optimized for a specific platform in order to achieve high performance, it is unlikely that optimized code will perform to the same level on other architectures, particularly in a parallel environment. Providing optimized versions of the software for a range of architectures is generally not feasible due to the increased burden in terms of development and maintenance, and inconvenience for users.

The purpose of the present authors' research is to develop a data structure and coding scheme for GeoFEM that is architecture independent. We consider that both parallel performance and single-processor performance are required to be good in order to realize optimal cross-platform performance of GeoFEM.

In 1998, good parallel performance was obtained for the computation of a very large-scale linear elastic problem by GeoFEM on a Hitachi SR2201 at the University of Tokyo using 1,000 processors.[1] The linear solver component of GeoFEM has also performed well on a symmetric multiprocessing (SMP) parallel computer in more recent research.[2] Based on these previous results, we consider that GeoFEM has the potential to achieve high parallel performance on various computer architectures. In the present research, we focus on optimizing the performance of GeoFEM on a single processor by developing a common data structure and coding scheme in order to achieve high platform-independent performance.

2. strategy

GeoFEM consists of structure analysis and fluid analysis processes as discrete code. Each code can also be roughly divided into 2 parts; a linear equation solver, and coefficient matrix calculation (structure analysis) or vector calculation (fluid analysis) for the system equations. In this study, we approach each of these 4 components in turn. Test code is currently implemented in GeoFEM for the structure analysis components, and final GeoFEM code has been implemented for the fluid analysis components.

Based on the relative importance and performance-enhancing capacity of each of these 4 components, we prioritized the optimization of the solver for both structure and fluid analysis, followed by matrix assembly and the additional calculations for fluid analysis. For the solver component, we have developed a new data structure and coding scheme, and have evaluated the performance of the proposed approach on scalar, vector and pseudovector architectures. For matrix assembly, we have removed dependency between array elements, and have evaluated the performance on vector and scalar machines. We have yet to complete development of a new data structure and coding scheme for matrix assembly, and have not yet dealt with the additional calculations for fluid analysis; these will be reported in the future.

3. Linear Solver

3.1 Preliminary analysis of computational cost

In structure analysis, the linear solver represents approximately 85% of the total operations; the remaining 15% is devoted to matrix assembly. In fluid analysis, 91% of operations are devoted to the solver (see Table 1).

Table1. Computational Cost of Fluid and Structure Analysis Module

	Fluid Analysis (processing time)	Structure analysis (Operations)
Solver	91.2%	84.7%
Matrix Assembly/Additional calculations	8.8%	15.3%

Focusing on solver, the majority of the computational cost (96.8%) is attributed to the 4 operations listed in Table 2. This is true for both structure and fluid analysis. Therefore, these 4 processes are targeted for optimization.

Table 2. Computation cost of Solver

	Computation time/Operations	
Forward substitution	26.3%	
Backward substitution	26.1%	
Matrix-vector product (lower)	22.2%	
Matrix-vector product (upper)	22.2%	
Others		3.2%
Total	96.8%	3.2%

3.2 Coding

The 4 processes mentioned above have almost the same coding. An example of the original code for fluid analysis is given below.

```
 do iv= 1, NVECT
  iv0= IVECT(iv-1)
  do  j= 1, NLhyp(iv)
    iS= INL(NL*(iv-1)+j-1)
    iE= INL(NL*(iv-1)+j  )
    do i= iv0+1, iv0+iE-iS
       k= i+iS - iv0
      kk= IAL(k)
      W(i,Z)= W(i,Z) - AL(k) * W(kk,Z)
    enddo
  enddo
  ........
enddo                                                                    (1)
```

The original storage system for the coefficient matrix is the compressed row system (Fig. 1(a)). In the present implementation, we adopted a storage system suitable for vector processing. In this scheme, each line of the coefficient matrix is reordered in descending order of number of elements (Fig. 1(b)), and the direction of memory access is changed (Fig. 1(c)) in order to exploit long vector length and continuous memory access. This coding represents a matrix (AL) – vector (W) product (Fig. 2).

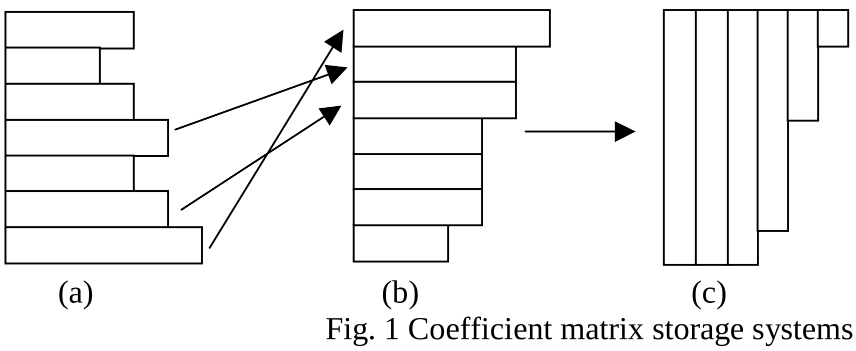


Fig. 1 Coefficient matrix storage systems

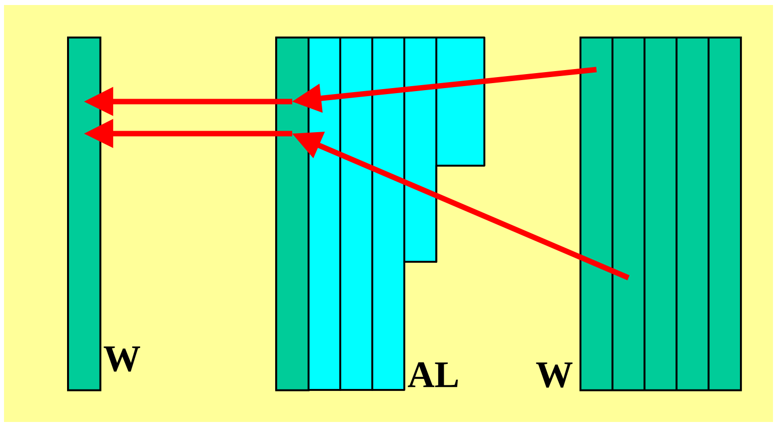


Fig. 2 Graphic illustration of array compression

An example of code for structure analysis is shown below. The coding is similar to that for fluid analysis, except that this code takes the product of a 3×3 matrix and 3-element vector in contrast to the 1×1 matrix and 1-element vector product taken in fluid analysis.

```
        do iv= 1, NVECT
        iv0= IVECT(iv-1)
        do  j= 1, NLhyp(iv)
          iS= INL(NL*(iv-1)+j-1)
          iE= INL(NL*(iv-1)+j  )
          do i= iv0+1, iv0+iE-iS
             k= i+iS - iv0
            kk= IAL(k)
            Zm2= W(3*kk-2,Z)
            Zm1= W(3*kk-1,Z)
            Zm0= W(3*kk  ,Z)
            W(3*i-2,Z)= W(3*i-2,Z) - AL(9*k-8)*Zm2 - AL(9*k-7)*Zm1      &
     &                                             - AL(9*k-6)*Zm0
            W(3*i-1,Z)= W(3*i-1,Z) - AL(9*k-5)*Zm2 - AL(9*k-4)*Zm1      &
     &                                             - AL(9*k-3)*Zm0
            W(3*i  ,Z)= W(3*i  ,Z) - AL(9*k-2)*Zm2 - AL(9*k-1)*Zm1      &
     &                                             - AL(9*k  )*Zm0
          enddo
        enddo
        ...........
      enddo
```

In the two codes, array W with index kk in the innermost loop is accessed indirectly, whereas all other arrays without index kk are accessed directly. In structure analysis, this direct access is in constant intervals.

The proposed fluid analysis code is shown below, modeled from the original code (1). Direct access is adopted for array AL in code (1) and array b AL in code (3), and indirect access is adopted for array W AL in code (1) and array a AL in code (3).

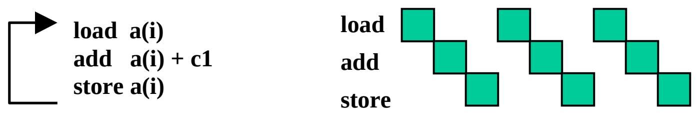
```
      do j=1,1000
      do i=1,1000
        a(i) = a(i) + b(1000*(j-1)+i)*a(1000+N(i))
      enddo
      enddo
```
(3)

Similarly, the proposed code for structure analysis is given below, as modified from the original code (2). Array access is identical to that for fluid analysis.

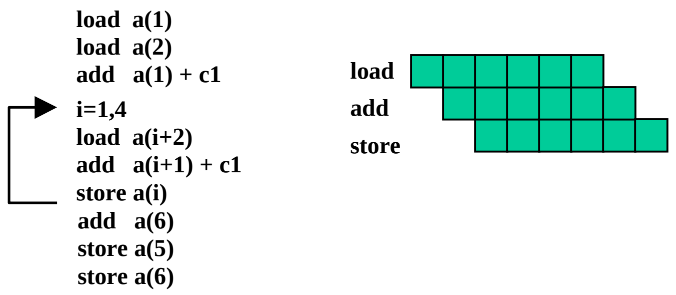
```
  do j=1,1000
  do i=1,1000
        k = i
       kk = N0(k)
      Zm2 = a(3*kk-2)
      Zm1 = a(3*kk-1)
      Zm0 = a(3*kk  )
      a(3*i-2)= a(3*i-2) - b(9*k-8)*Zm2 - b(9*k-7)*Zm1      &
 &                                    - b(9*k-6)*Zm0
      a(3*i-1)= a(3*i-1) - b(9*k-5)*Zm2 - b(9*k-4)*Zm1      &
 &                                    - b(9*k-3)*Zm0
      a(3*i  )= a(3*i  ) - b(9*k-2)*Zm2 - b(9*k-1)*Zm1      &
 &                                    - b(9*k  )*Zm0
  enddo
  enddo
```
(4)

3.3 Pseudo Vector Architecture

In this study, we used a Hitachi SR8000 128-node system at the Computing Center, University of Tokyo. The SR8000 system has a pseudovector architecture, and can function as scalar machine and makes use of software-pipelining (Figs. 3(a) and (b)). Pipelining for elements of an array is possible if dependency between array elements is removed. In pseudovector processing, the SR8000 has preload and prefetch functions (Fig. 4). Each processor has 128 floating registers that can be used as vector resisters. Preload loads data directly from memory into the floating registers without caching in advance, thus avoiding load latency. Preload is usually applied for indirect access. Prefetch loads data from memory into a cache before use, again removing load latency. Prefetch is usually applied for direct access.



(a) No software pipelining



(b) Software pipelining

Fig. 3 Schematic of effect of software pipelining

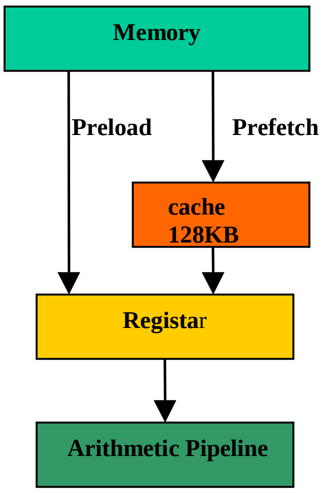


Fig.4 Preload and Prefetch function

3.4 Performance Factor

The proposed fluid analysis code (3) performs at 68 Mflops on the SR8000, which is comparable to that of the existing GeoFEM code. This code was then further refined by changing the access rule for array a from indirect access to direct access, as shown below. The performance was improved to 520 Mflops using this modified code.

```
do j=1,1000
do i=1,1000
  a(i) = a(i) + b(1000*(j-1)+i)*a(1000+i)
enddo
enddo
```
(5)

According to compiler messages, this 8-fold increase in performance was achieved exclusively in the outer loop expansion and the direct access rule. Therefore, performance can be improved even further by minimizing load/store latency related to the outer loop expansion. An example of the original relevant code is shown below.

```
do i=1,1000
do j=1,1000
  y(i) = y(i) + a(i,j)*x(j)
enddo
enddo
```
(6)

The improved code is as follows:

```
do is=1,1000,10
do j=1,1000
  y(is  ) = y(is  ) + a(is  ,j)*x(j)
  y(is+1) = y(is+1) + a(is+1,j)*x(j)
  .......
  y(is+9) = y(is+9) + a(is+9,j)*x(j)
enddo
enddo
```
(7)

The original coding (6) has 2 load operations and 2 floating point operations, hence, the ratio of load/store operations to floating point operations is 1:1. The improved code (7) has 11 load operations and 20 floating point operation, reducing the ratio of load/store operations to floating point operations to 11:20. This ratio is effective for increasing performance.

However, applying the same rationale for array b by replacing array a with array b in code (7), as given below, performance was degraded from 520 Mflops to 140 Mflops from 520MFlops in code (8).

```
do j=1,1000
do i=1,1000
  a(i) = a(i) + a(1000*(j-1)+i)*b(1000+i)
enddo
enddo
                                                                    (8)
```

Based on compiler messages, the poor performance of this code was due to the failure of software pipelining, although the pseudovector process succeeded.

Therefore, 5 factors are identified as influencing the performance of the alrgorithm, as follows:

(1) Pseudo vectorization for the loop
(2) Software pipelining for the loop
(3) Outer loop expansion
(4) Low load/store latency for the loop
(5) Direct access to arrays in the loop

Item (1) is achieved via the compiler by exploiting the pseudovector architecture. Item (2) is also realized via the compiler, using a function of the pseudovector and scalar architecture. Thus, items (1) and (2) are compiler-dependent. On the other hand, items (3), (4) and (5) are compiler-independent, and as such the modifications are applicable to all architectures.

3.7 Evaluation of Performance of proposed fluid analysis code

Factors (1) and (2) above were implemented in code (3). The remaining factors, related to outer loop expansion, were then implemented, and the data structure of array b was modified so as to fully exploit the prefetch function. The simplified code for the 4 operations given in Table 2 is given below.

```
do j=1,2
do i=1,1000
  a(i) = a(i) + b0(i,j)*a(L(i),j)
enddo
enddo
                                                                    (9)
```

Array b0, declared as b0(1000,2) in the code above, is modified to the new data structure b1(2,1000). The resultant code is as follows:

```
do i=1,1000
  a(i) = a(i) + b1(1,i)*a(L(i),1) + b1(2,i)*a(L(i),2)
enddo
                                                                   (10)
```

In this code, the low load/store latency for the loop is satisfied because b1(1,i) and b1(2,i) are accessed continuously. Both outer loop expansion and low load/store latency for the loop are satisfied by this new data structure.

An example of the solver code for fluid analysis including the modification in (10) is shown below.

```
    do j=1,1000,10
    do i=1,1000
      a(i) = a(i) + b(10*(i-1)+1)*a(N0(10*(i-1)+1)) &
   &              + b(10*(i-1)+2)*a(N0(10*(i-1)+2)) &
   &              + b(10*(i-1)+3)*a(N0(10*(i-1)+3)) &
   &              + b(10*(i-1)+4)*a(N0(10*(i-1)+4)) &
   &              + b(10*(i-1)+5)*a(N0(10*(i-1)+5)) &
   &              + b(10*(i-1)+6)*a(N0(10*(i-1)+6)) &
   &              + b(10*(i-1)+7)*a(N0(10*(i-1)+7)) &
   &              + b(10*(i-1)+8)*a(N0(10*(i-1)+8)) &
   &              + b(10*(i-1)+9)*a(N0(10*(i-1)+9)) &
   &              + b(10*(i-1)+10)*a(N0(10*(i-1)+10))
    enddo
    enddo
```
(11)

Code (3) involved 4 load/store operations and 2 floating point operations (ratio 2:1). The code above includes 22 load operations and 20 floating point operations (ratio 11:10). The number of load/store operations can be reduced by fetching many data in one fetch operation. This is possible because the elements of array b are accessed continuously in code (11). The performance of code (11) is 174 Mflops on the SR8000, improved from 68 Mflops in code(11). A summary of the factors and performance is given in Table 3.

A new code addressing factor (5) was then evaluated. In this code, direct access for array a is implemented in code (11). It is expected that good performance will be obtained if all arrays in the loop are accessed directly.

The reording process for array a is as follows:

```
    do j=1,2
    do i=1,1000
      x(j,i) = a(L(i),j)
    enddo
    enddo
```
(12)

The direct access code is then

```
    do j=1,1000
      a(i) = a(i) + b1(1,i)*x(1,i) + b1(2,i)*x(2,i)
    do i=1,1000
    enddo
    enddo
```
(13)

The reording process for the fluid analysis solver using then becomes

```
    do j=1,1000,5
    do m=1,10
    do i=1,100
      kk = 500*(m-1)+5*i-5
      x1(kk+1) = a(N0(kk+1))
      x1(kk+2) = a(N0(kk+2))
      x1(kk+3) = a(N0(kk+3))
      x1(kk+4) = a(N0(kk+4))
      x1(kk+5) = a(N0(kk+5))
    enddo
    enddo
    enddo
```
(14)

and the main calculation code is as follows:

```
      do j=1,1000,5
      do i=1,1000
        kk = 5*(i-1)
        kk1=kk+1
        kk2=kk+2
        kk3=kk+3
        kk4=kk+4
        kk5=kk+5
        a(i) = a(i) + b(kk1)*x1(kk1) &
     &              + b(kk2)*x1(kk2) &
     &              + b(kk3)*x1(kk3) &
     &              + b(kk4)*x1(kk4) &
     &              + b(kk5)*x1(kk5)
      enddo
      enddo
```
(15)

The computational performance of this code on various architectures are summarized in Table 3.

Table 3. Summary of factors and performance for fluid analysis

	Hitachi SR8000[1]						Alpha 21164	VPP5000[2]	
Factor	Pseudovector			Scalar					
(1) Pseudovectorization	Y	Y	Y	N	N	N	N	-	-
(2) Software pipelining	Y	Y	Y	Y	Y	N	N	-	-
(3) Outer loop expansion	N	Y	Y	Y	Y	Y	Y	N	Y
(4) Low load/store latency	N	Y	Y	Y	Y	Y	Y	N	Y
(5) Direct access	N	N	Y	Y	N	Y	Y	N	Y
Performance (Mflops)	68	174	129	145	85	45	83	1405	1881

[1]Each processor (not each node) is capable of 1 Gflops peak performance
[2]Each processor is capable of 9.6 Gflops peak performance

As can be seen from the table, a 3-fold performance increase was obtained using the present loop expansion coding and data structure on the SR8000 in pseudovector operations. A performance improvement of 70% was obtained using direct access in scalar operation, but reduced performance by 25% in pseudovector operation. Factors (3) to (5) increase performance on scalar architecture, but only if software pipelining is enabled. On the vector architecture (VPP5000), the performance was improved by 34% by applying factors (3), (4) and (5), reaching 20% of the peak performance of the vector architecture.

3.6 Evaluation of performance of proposed structure analysis code

Performance factors (1) and (2) were implemented in code (4). Factors (3) and (5) can be implemented in a similar manner to fluid analysis. Performance factor (4) will be implemented in the future. The reordering process for array a is as follows:

```
      do j=1,1000
      do i=1,1000
        k = i
        kk = N0(k)
        x2(i) = a(3*kk-2)
        x1(i) = a(3*kk-1)
        x0(i) = a(3*kk  )
      enddo
      enddo
```
(16)

The main calculation is given below.

```
 do j=1,1000
 do i=1,1000
 k=i
     a(3*i-2)= a(3*i-2) - b(9*k-8)*x2(i) - b(9*k-7)*x1(i)      &
&                                        - b(9*k-6)*x0(i)
     a(3*i-1)= a(3*i-1) - b(9*k-5)*x2(i) - b(9*k-4)*x1(i)      &
&                                        - b(9*k-3)*x0(i)
     a(3*i  )= a(3*i  ) - b(9*k-2)*x2(i) - b(9*k-1)*x1(i)      &
&                                        - b(9*k  )*x0(i)
 enddo
 enddo
                                                                        (17)
```

The computational performance of these codes on the various architectures is given in Table 4. A 28% improvement was obtained by applying loop expansion and direct access coding on the SR8000 in pseudovector operations. The performance of the reordering process was 122 Mops (M operation/sec), and the main calculation reached 335 Mflops, resulting in an overall performance of 226 Mflops. The scalar performance did not reach that of the pseudovector architecture due to the lack of low load/storage latency coding. The performance of the Fujitsu VPP5000 was not improved significantly by these implementations. Approximately 27% of the peak performance of vector architecture was obtained.

Table 4. Summary of factors and performance for fluid analysis

Factor	Hitachi SR8000[1]			Alpha	VPP5000[2]	
	Pseudovector		Scalar	21164		
(1) Pseudovectorization	Y	Y	N	N	-	-
(2) Software pipelining	Y	Y	Y	N	-	-
(3) Outer loop expansion	N	Y	Y	Y	N	Y
(4) Low load/store latency	N	N	N	N	N	N
(5) Direct access	N	Y	Y	Y	N	Y
Performance (Mflops)	177	226	149	123	2658	2727

[1]Each processor (not each node) is capable of 1 Gflops peak performance
[2]Each processor is capable of 9.6 Gflops peak performance

4. Matrix Assembly

4.1 Outline of Original code of Matrix assemble part

The loop structure for the original matrix assembly code is shown in Fig. 3. The 3 outermost loops are typewriter-scanned elements loops, referring to a consecutive rectangular sweep of nested loops. Jacobian calculation and element stiffness matrix calculation are enclosed in the 3 outermost loops. The element stiffness matrix calculation assembles the stiffness matrix, and consists of 2 loops of 8 elements. These 8 elements correpond to the number of nodes required for each hexahedral element. The 3 innermost loops correspond to integration of local coordinates for each element.

```
do kk1= 1, NZ
  do jj1= 1, NY
    do ii1= 1, NX
      |-jacobian calculate
      do ie= 1, 8
        do je= 1, 8
          do kpn= 1, 2
            do jpn= 1, 2
              do ipn= 1, 2
                |-stiffness matrix calculate
              enddo
            enddo
          enddo
        enddo
      enddo
    enddo
  enddo
enddo
```

Fig.3 Loop structure for original matrix assemble code

```
do loer = 1,50
  do lpn = 1,numlp+1
    do kp=1,2
      do jp=1,2
        do ip=1,2
          do ijk = 1,llnum
            |-jacobian calculate
          enddo
        enddo
      enddo
    enddo
    do ie= 1, 8
      do je= 1, 8
        do kpn= 1, 2
          do ijk = 1,llnum
            |-stiffness matrix calculate
          enddo
          do ijk = 1,llnum
            |-stiffness matrix calculate
          enddo
        enddo
      enddo
    enddo
  enddo
enddo
```

Fig.4 Loop structure for Modified matrix assemble code

4.2 Optimization

The strategy for optimization is to place the loop for array a elements as the innermost loop to obtain long vector length. If the loop for array a elements is a single loop, the process of adding a numerical value is assigned to a single node, resulting in dependency of data for the element loop.

The element loop was therefore divided into smaller loops of groups having no recursive reference. Each group was further divided to optimize memory storage for the Jacobian calculation results. The final structure is shown in Fig. 4.

Both Jacobian calculation and element stiffness matrix calculation were placed in a secondary loop. In the Jacobian calculation, the 3 loop for integration was added in a secondary loop. In the element stiffness matrix calculation, the outer loop is divided into 2 loops (8×8) to assemble the whole stiffness matrix. An integration loop for the ς coordinate was placed as an outer loop, and a third element loop was placed as an integration loop. In third element loop, 2 loops for integrating $\xi - \eta$ coordinates were replaced with 4 (2×2) operation lines.

4.3 Problem definision

The test problem is defined as follows: Elastic structure analysis for a cube with 50,000 elements and 164,000 degrees of freedom.

4.4 Performance

The computation performance and operation counts for each process are listed in Table 5. The computation time per Mflop for matrix assembly was 28.8 s/53.3 Mflops (SX-4: peak performance 2 Gflops) before modification, and 2.09 s/736.8 Mflops after optimization. The computation time per Mflop for the entire test code was 11.18 s/900.7 Mflops on the SX-4, and 4.88 s/2.06 Gflops on the VPP5000 (peak performance 9.6 Gflops).

Table 5. Performance and operation counts for each process

Process	Operation count (Gflops)	Dec		SX-4		VPP5000	
		Comp. time	Mflops	Comp. time	Mflops	Comp. time	Mflops
Boundary condition Matrix Assembly Solver Preprocess	1.54	0.54 11.85 0.07	123.6	0.19 1.73 0.17	736.8	0.04 0.76 0.11	1692.3
Solver	8.53	197.10	43.3	9.09	938.4	3.97	2148.6
Total	10.07	209.56	48.1	11.18	900.7	4.88	2063.5

5. Conclusion and Further Study

GeoFEM has the potential for very high parallel performance on various computer architectures, and significant development towards this goal has been completed by the GeoFEM team. In the present study, we focused on optimizing the performance on a single processor by defining a common data structure and coding scheme in order to achieve high cross-platform performance for GeoFEM. The components of GeoFEM have been prioritized for optimization based on relative importance and performance-enhancing capacity as follows:

(1) Solver for structure and fluid analysis
(2) Matrix assembly for structure analysis
(3) Matrix assembly for fluid analysis

A new data structure and coding scheme was developed and evaluated on scalar, vector and pseudovector architecture. By introducing direct access, the performance of the fluid analysis solver was increased by 17% on pseudovector architecture and 18% on vector architecture.

Direct access was introduced for the structure analysis solver, resulting in a 23% increase in performance on pseudovector architecture and 27% increase on vector architecture. The future introduction of a new data structure is expected to improve performance further.

Intra-element dependency has been removed by optimizing the loop structure for matrix assembly for structure analysis. The performance of the improved code was evaluated on vector and scalar machines, achieving 736.8 Mflops performance for the matrix assembly component on an NEC SX-4 and 900.7 Mflops for the entire test code. The preformance of the entire test code reached 2.06 Gflops on a Fujitsu VPP5000, and 124 Mflops was obtained for matrix assembly on an Alpha system (21164 533 MHz).

In future work, we intend to complete the evaluation of the data structure and coding scheme for matrix assembly for both structure and fluid analysis, and implement a new data structure for the structure analysis solver and a new data structure and direct access for existing GeoFEM code.

Acknowledgments

This research forms part of the Solid Earth Platform for Large Scale Computation project funded under the Special Promoting Funds for Science & Technology program of the Science and Technology Agency (STA) of Japan. The authors would like to thank the members of the Geo-FEM team for their helpful suggestions, and Professor Yasumasa Kaneda (Computing Center, University of Tokyo) for discussions on high-performance computing.

References

[1] K.Garatani,H.Nakamura,H.Okuda,G.Yagawa,GeoFEM:High Performance Parallel FEM for Solid Earth,Proceedings of 7th High-Performance Computing and Networking(HPCN Europe'99),LNCS-1593,133-140,1999.