Research Journal of Applied Sciences 2 (12): 1194-1199, 2007
ISSN: 1815-932X

Multitasking Algorithms for Optimization of Space Structures

K.V. Marthandan
Narayanaguru College of Engineering, Manjalumoodu, 629151, India

Abstract: Various multitasking approaches are investigated for optimization of large space structures. Judicious combination, macrotasking and autotasking is explored with the goal of achieves a vector zed and multitasked algorithm for optimization of large structure with maximum speedup performance. Speedup results are presented and compared for three space truss structures with 526, 1046 and 3126 members.

Key words: Multitasking algorithms, optimization, space structures, vectorization

INTRODUCTION

Optimization of large structures with a few thousands members such as space stations requires an inordinate amount of processing time if a sequential algorithm and code is used. Our goal is to develop efficient algorithms employing both vector processing and multitasking capabilities of multitasking capabilities of multiprocessor supercomputers (Adeli, 1992a, b).

Adeli and Kamal (1992a, b) presented parallel algorithms for optimization of structures through the use of the notion of cheap concurrency and the concept of threads. Hsu and Adeli presented a microtasking algorithm for optimization of structures on CRAY YMP 8/864. In this research, we explore judicious combination of various multitasking and autotasking with the goal of achieving a vectorized and multitasked algorithm for optimization of large structures with maximum speedup performance.

Computing environment: The computer used in this research is CRAY YMP 8/864. It is a shared-memory machine with eight processors, up to 32 M words of main memory, dual instruction mode for 32-bit addressing multiple memory ports and a 6-ns clock cycle. It supports vectorization and multitasking in FORTRAN and c computer languages, using the UNICOS operating system. UNICOS is derived from the AT and T UNIX system V operating system and is also based in part on the Fourth Berkeley Software (CARY, 1990).

The computer language used in this research is CRAY Standards C version 3.0 (1990). This is the first version of C that supports macrotasking, microtasking and autotasking. It allows macrotasking and microtasking to be combined. But it does not allow combination of autotasking with either macrotasking or microtasking.

Vectorization: On CRAY YMP 8/864 a single vector operation can produce a vector containing up to 64 values. Vectorization is performed on the innermost nested loops. The code segments may have to be rearranged in order to optimize the vectorization performance. Some complications in the loop structure may prevent loop vectorization.

Multitasking: Concurrent processing on Cray YAMP 8/864 is performed by macrotasking, microtasking and auto tasking.

Macro tasking: Macro tasking is performed at function level. Normally, major tasks that can be processed concurrently are macrotasked. Macrotasking is implemented by function calls and is suitable for tasks requiring large processing time because its overhead is large compared with that of microtasking. Macrotasked tasks should be identified when the general concurrent algorithm is developed.

Microtasking: Microtasking is parallel processing at the loop level. It is implemented by inserting compiler directives. Existing serial codes can be rather easily microtasked without creating new concurrent algorithms. But in most cases and for computation intensive jobs microtasking by itself does not yield high speedup; it should be combined with macrotasking in order to achieve maximum performance.

Auto tasking: Auto tasking is the automatic distribution of tasks to multiple processors by complier. It attempts to detect parallelism in the code automatically. Basically it combines vectorization and microtasking automatically.

Multitasking algorithms: In the finite element structural analysis most of the time is spent in setting up and

assembling the element stiffness matrices into the structure load vector and solving the system of linear equation for displacement degrees of freedom. Thus, we will concentrate on the functions "assemble" that assembles the structure stiffness matrix "load-vector" that assembles the structure load vector and "solve" that solves the resulting linear equations. Parallel algorithms are developed and compared using autotasking microtasking and macrotasking with the objectives of improving the performance of these functions.

In this research, three different multitasking algorithms are presented and compared. The first algorithm, called algorithm A, is based on the use of autotasking only (CARY, 1990). In the second algorithm, macrotasking directives are introduced in the loop level. No load balancing is used in this case. In the third algorithm, both microtasking and macrotasking are used with load balancing. The three algorithms are outlined in Table 1.

Table 1: Multitasking algorithms for optimization of space structures with multitasking and vectorization

Algorithm A: Autotasking and vectorization.
Algorithm B: Microtasking and vectorization.
Algorithm C: Macrotasking, microtasking and vectorization.
Set the Number of Processors (NP).
Read in the input data and the starting design variables.
Set 1/u =0.1, iteration= 1 and operation= 1, where operation is a factor to indicate whether this step is in the analysis stage (operation = 1) or in the redesign stage (operation=2).
Assemble the structure stiffness matrix.
Do concurrently:
I – Calculate element stiffness matrices.
A (autotasking) B (microtasking) C (macrotasking with load balancing).
II- Assemble element stiffness matrices into the structure stiffness matrix.
A (autotasking) B (microtasking with guarded regions) C(microtasking with guarded regions(
5. Assemble total load vector.
Do concurrently:
Assemble the nodal forces into
the total load vector.
A (autotasking and vectorization) B (microtasking and vectorization) C (microtasking with load balancing vectorization).
6. Apply boundary conditions.
Do concurrently:
I – Update the structure stiffness matrix.
A (autotasking) B(microtasking with guarded regions) C (microtasking with guarded regions)
II- Update total load vector.
A (autotasking and vectorization) B (microtasking and vectorization) C (microtasking with load balancing and vectorization).
7. Solve the linear equations.
Do concurrently:
I – Reduce the structure stiffness matrix.
A (vectorization) B (vectorization) C (vectorization).
II – Forward substitutions.
A (autotasking and vectorization) B (microtasking and vectorization) C (microtasking with load balancing vectorization).
III – Backward substitution.
A (autotasking and vectorization) B (microtasking and vectorization) C (microtasking with load balancing vectorization).
8. If operation = 1, calculate the member forces and stresses.
A (autotasking) B (microtasking) C (microtasking with load balancing).
If operation = 2, go to step 15.
Calculate the objective function (w)
A (autotasking) B (microtasking) C (microtasking with load balancing)
If the difference between the new and old objective functions is less than 0.1%, stop and print the results, otherwise, go to step 10.
Set operation = 2.
If there is no constrained displacements, set maximum displacement ratio = 1 and go to step 11. Otherwise calculate the maximum displacement ratio and go to step 11.
A (autotasking and vectorization) B (microtasking with guarded regions and vectorization) C (microtasking with guarded regions and vectorization)
11. Calculate the maximum stress ratio (stress ratio).
A (autotasking and vectorization)
B (microtasking with guarded regions and vectorization) C (microtasking with guarded regions and vectorization).
If there is no constrained displacement go to step 16. Otherwise go to step 12.
12. If iteration = 1, go to step 14. Otherwise if the value of the objective function (w) is less than that of the previous iteration divide the value of 1/u by two and go to step 14.
Find the active displacements, those
With in 0.1% of the allowable values.
A (autotasking) B (microtasking) C (microtasking with load balancing).
Apply unit loads in the directions of the most violated degrees of freedom one at a time each time go to step 6.

Table 1: Continue

Calculate the displacement gradients.
Do concurrently:
I-calculate the element stiffness matrices.
A (autotasking) B (microtasking) C (microtasking with load balancing)
II-Calculate the displacement gradients.
A(autotasking) B (microtasking) C (microtasking with load balancing).
Calculate the new design variables for the next iteration as follows:
If stress ratio > 1, modify the design variables using the optimality criteria recurrence formula, otherwise, modify the design variables using the optimality criteria recurrence formula.
A (autotasking) B (microtasking) C (microtasking with load balancing).
Calculate the new objective function and set iteration = + 1.
Go to step 4.

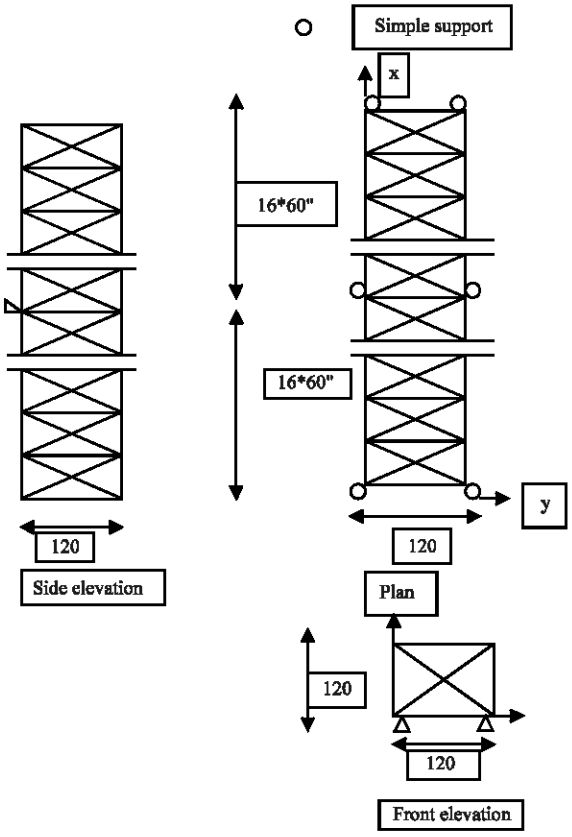

Fig. 1: Example 1 is a 526-members space truss

Examples: We solved three space structures using the algorithms outlined in Table 1. These examples model space station structures (Fig. 1).

It consists of 32 equal-span panels in the longitudinal direction and one square panel in the transverse directions. It has two simple supports at each end and 2 other supports at the middle of the span. Thus, it is a symmetric continuous two-span truss. The upper nodes at the middle of each span are loaded in the vertical y-direction by a 60-kip downward load and in x and z directions by 20-kip loads. The displacements of the nodes at the middle of each span in the vertical y-direction are restricted to 1/200 the of the span.

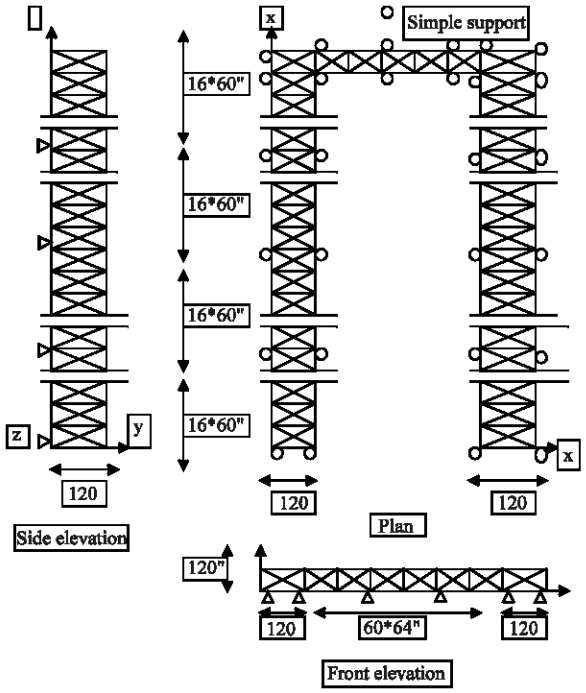

Fig. 2: Example 3 (3126-member space truss)

Example 2 is a 1046-member space truss. He geometry of this example is the same as that of example 1, but it has 64 panels (twice as many as example 1) and four supports at quarter points. The upper nodes at the middle of each span are loaded by a 60-kip downward load in the vertical y-direction and by 20-kip loads in the x and z directions. The displacements of the nodes at the middle of each span in the vertical y-direction are restricted to 1/200the of the span (Fig. 2).

Example 3 is a 3126-member space truss: The U-space truss has three wings. Longitudinal direction (similar to example 2). It has 30 simple supports as indicated in Fig. 3. The loading and displacement constraint are similar to those of example 2.

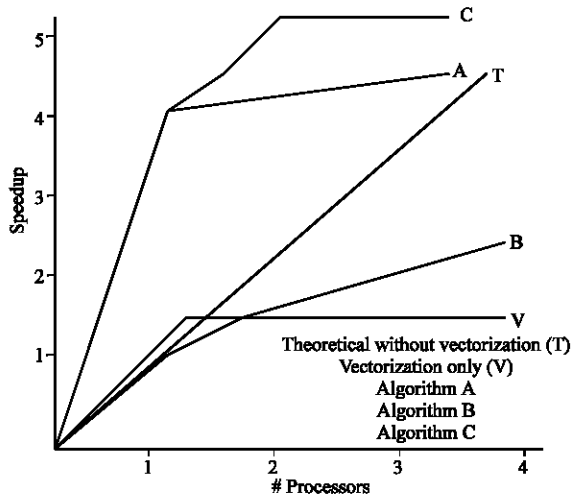

Fig. 3: Speedup comparisons for the function "load = vector" in example 1

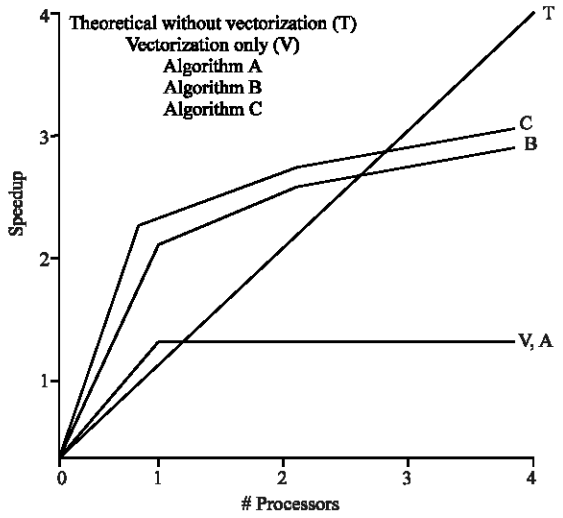

Fig. 4: Speedup comparisons for the function "assemble" in example 1

SPEEDUP RESULTS

We study the speedup due to vectorization and multitasking for the three examples described in the previous section using the algorithms a, b and c. The theoretical speedup due to multitasking is definer as the ratio of the execution time spent by a task in a sequential program to that spent in a concurrent program (CARY, 1987).

Speedup results due to vectorization and various types of multitasking are presented for three main functions of the algorithms "assemble", "load-vector" and solve. The last function is divided into 2 parts: Reduction of the stiffness matrix into the product of a lower triangular and an upper triangular matrix and forward and backward substitutions considered together. At the end of the study we present speedup results for complete solution of the optimization problem.

Figure 3-5 show the speedup results for setting up the load vector, setting up the structure stiffness matrix and the forward and backward substitutions for computing the nodal displacement vector for example 1, respectively.

Figure 3 shows that auto tasking improves the speedup for the function load-vector substantially because this function consists of only nested loops without any dependencies among matrix elements and function calls within loops. Microtasking does not improve the performance of this function substantially. This is due to the relatively large overhead needed in microtasking (compared with the amount of the work done) and poor load balancing which in turn deteriorates

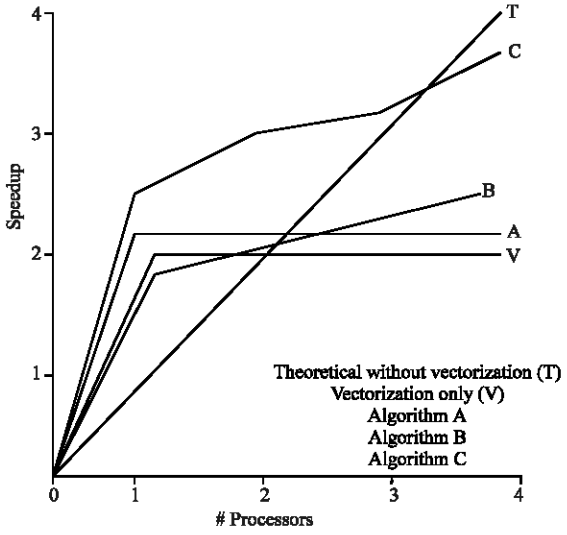

Fig. 5: Speedup comparisons for the forward/backward substitutions in the function "solve" in example 1

the speedup due to vectorization. In algorithm C where microtasking is combined with macrotasking, uniform, load balancing is achieved. Thus, the effect of stripmining is achieved. Thus, the effect of stripmining is maximized in algorithm C.

Figure 4 shows that vectorization and autotasking do not improve the speedup in the function "assemble" because of the existence of function calls and the guarded regions. Microtasking improves the speedup in the function "assemble" because the amount of work done in setting up the element stiffness matrices and assembling

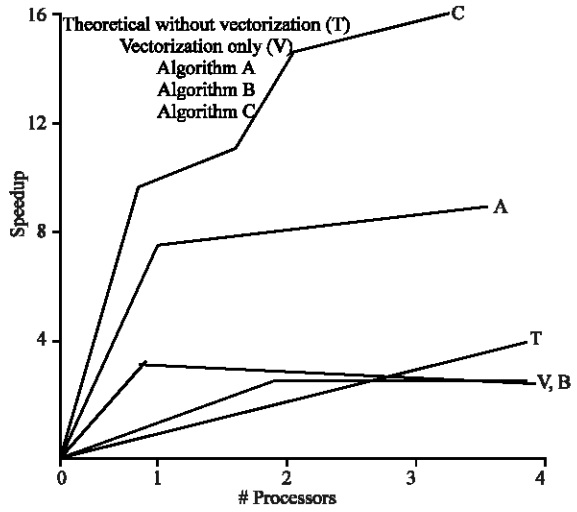

Fig. 6: Speedup comparisons for the "function "load vector" in example 3

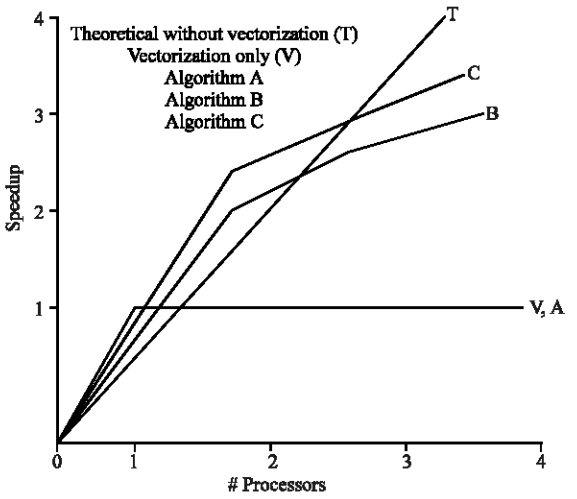

Fig. 7: Speedup comparisons for the function "assemble" in example 3

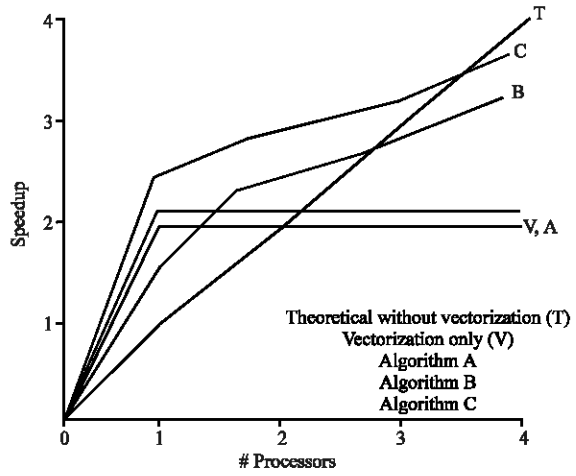

Fig. 8: Speedup comparisons for the forward and backward substitutions in the function "solve" in example 3

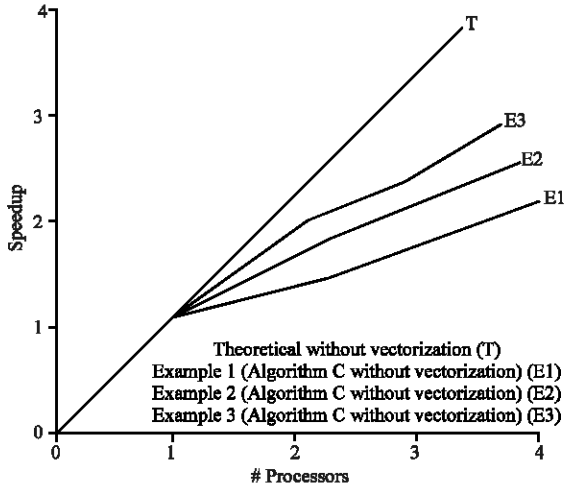

Fig. 9: Overall speedups

them into the structure stiffness matrix is relatively large compared with the overhead required in microtasking. Algorithm C produces higher speedup than both algorithms A and B.

Figure 5 shows the speedup for the part of the function "solve"that performs the forward and backward substitutions for computing the nodal displacement vector. Both vectorization and multitasking are used in this part. Algorithm C produces substantially higher speedup compared with the algorithms A and B.

The speedup results for setting up the load vector, setting up the structure stiffness matrix and the forward and backward substitutions for computing the nodal displacement vector for example 3. Comparing Fig. 6-8 with the corresponding Fig. 3-5 for example 1, we observe that the speedups due to both vectorization and multitasking increase with the size of the structure. The increase due to multitasking is specially substantial. These figures also demonstrate the superiority of algorithm C where we employed a judicious combination of vectorization, microtasking and macrotasking with load balancing.

The overall speedup results from the algorithm C for the complete optimization of three space structure examples without and with vectorization are presented in Fig. 9 and 10.

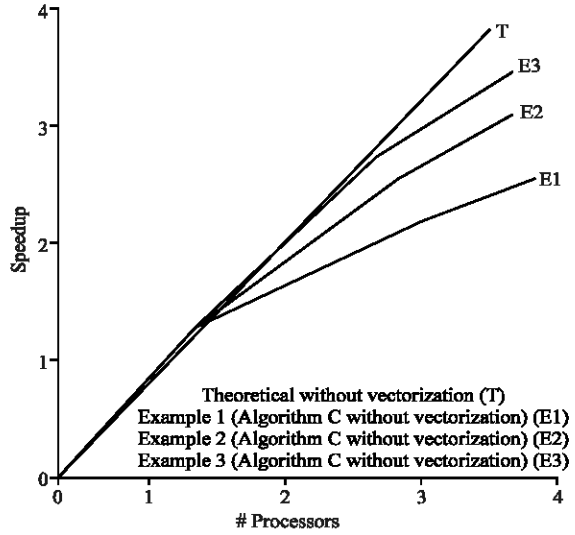

Fig. 10: Overall speedups, respectively. It is seen that the speedup increases substantially with the size of the structure

CONCLUSION

Based on this investigation, a number of general conclusions may be drawn:

- Autotasking works best in programs where most of the code consists of nested loops. Autotasking can not be performed when loop iterations contain inter dependent array elements or when there is a function call. Most of the speedup due to autotasking is achieved by vectorization. Basically it is effective only when the program has a simple structure.
- Microtasking is simple to implement but does not improve the speedup substantially in complex problems without combining it with macrotasking.
- Judicious combination of vectorization, microtasking and macrotasking is required in order to develop an efficient vectorized and multitasked algorithm. Algorithm C presented in this study is an example of such algorithm for optimization of large structures where substantial processing power is required even on high performance machines.
- The processing time required for optimization of large structures increase exponentially with the size of the structure (number of design variables). Example 3 of this study has 3126 members and 2226 displacement degrees of freedom. Development of efficient concurrent algorithms utilizing the unique architecture and capabilities of high-performance computers results in substantial reduction in the overall execution time.

ACKNOWLEDGEMENT

This research has been partially supported by a grant from Cray Research, Inc. Computing time was provided by the Ohio super computer center.

REFERENCES

Adeli, H., 1992a. Supercomputing in Engineering Analysis. Marcel Dekker, New York.

Adeli, H., 1992b. Parallel Processing in Computational Mechanics. Marcel Dekker, New York.

Adeli, H. and O. Kamal, 1989. Parallel Structural Analysis Using Threads. Microcomputers in Civil Eng., 4: 133-147.

Adeli, H. and O. Kamal, 1992b. Concurrent Optimization of Large Structures. Part I. Algorithms. J. Aerospace Eng. ASCE., 5: 79-90.

Adeli, H. and O. Kamal, 1992b. Co current Optimization of Large Structures. Part II. Applications. J. Aerospace Eng. ASCE., 5: 91-110.

CRAY, 1990. Cray standard C programmers References Manual, SR-2074 3.0, Cray Research Inc.

CRAY, 1987. Cray Y-MP Multitasking Programmers Reference Manual, SR-0222, Cray Research Inc.