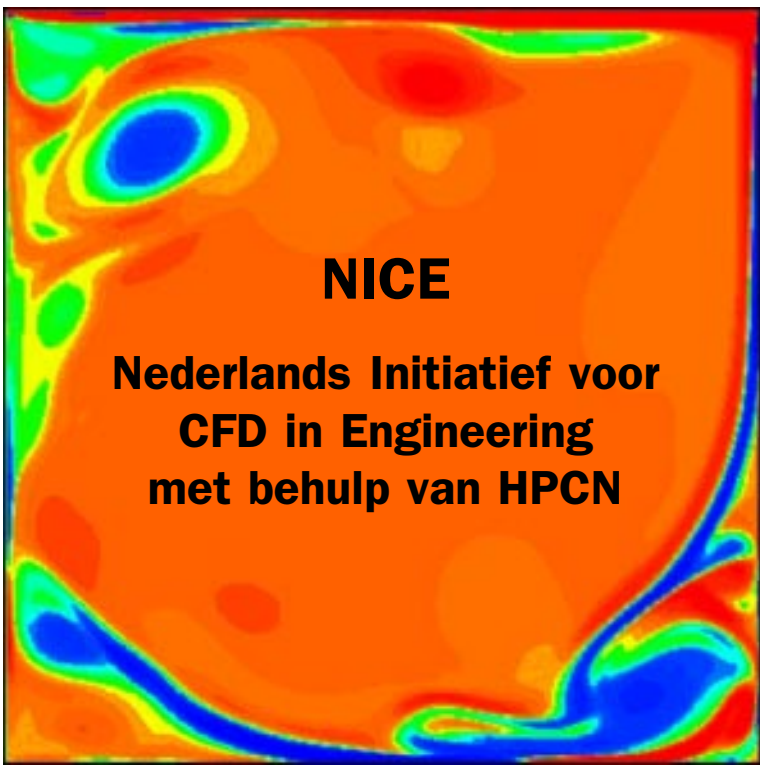
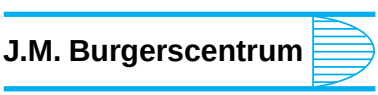
Nationaal Lucht- en Ruimtevaartlaboratorium
National Aerospace Laboratory NLR

NLR-TP-98380

Coarse-grain parallelization of a multi-block Navier-Stokes solver on a shared memory parallel vector computer

P. Wijnandts and M.E.S. Vogels





Nationaal Lucht- en Ruimtevaartlaboratorium
National Aerospace Laboratory NLR

NLR-TP-98380

Coarse-grain parallelization of a multi-block Navier-Stokes solver on a shared memory parallel vector computer

P. Wijnandts and M.E.S. Vogels

The work described in this report is partially supported by the Dutch Foundation HPCN in the project NICE under contract number 96009.

This report is based on a presentation held on the Vector and Parallel processing '98 Conference, Porto, Portugal, June 21-23, 1998.

Division:	Informatics
Issued:	September 1998
Classification of title:	unclassified

Summary

The coarse-grain, or block-loop parallelization of the multi-block Navier-Stokes flow solver ENSOLV on a NEC SX-4, a shared memory parallel vector computer, is discussed. The performance of the parallel code was tested by running the code on ten benchmark cases, provided by the ENSOLV user group. The performance is measured in terms of speed-up, memory usage and execution cost. The results of the benchmark cases are presented. The results are compared to those of the low-level DO-loop parallelization implemented earlier. The conclusion based on the comparison of the results, is that for all benchmark cases, except the single block, the block-loop parallelization gives better performance in terms of speed-up. Although block-loop parallelization requires more memory, it gives overall less execution cost.

Contents

(21 pages in total)

1 Introduction

The multi-block Navier-Stokes flow solver ENSOLV Ref. 2, Ref. 4, computes the solution of the steady 3D Euler and/or thin-layer Navier-Stokes equations in an arbitrary flow domain. The Euler and Navier-Stokes equations are given by five partial differential equations for the conservation of mass, 3D momentum and energy, extended by the perfect gas law. To solve the equations, an iterative procedure which resembles time integration is used. A number of techniques are employed to accelerate the convergence:

1. A multigrid scheme, which performs relaxations on different grid levels, is used as solution procedure. This accelerates the convergence on the finest grid level. As relaxation procedure, the explicit Runge-Kutta time stepping scheme is used;
2. The evaluation of the time step, needed for the Runge-Kutta scheme, is performed locally;
3. Implicit residual averaging with varying coefficients and enthalpy damping are used.

The solver is based on multi-block structured grids. Multigrid is applied around multi-block, i.e. on each grid level a loop on the blocks is performed. The Runge-Kutta scheme is applied on a block-by-block basis. This means that a relaxation of all blocks consists of taking one complete Runge-Kutta time step for each block successively, keeping the flow states in the other blocks fixed. The flow solver ENSOLV is currently operational at NLR and industry.

Within the NICE[1] program, ENSOLV is being parallelized in order to reduce execution cost. Parallelization takes place on a 16-processor NEC SX-4 Ref. 9, a shared memory parallel vector computer, with a peak performance of 2 GFlop/s per processor. In Ref. 5, ten representative benchmark cases were defined by the ENSOLV user group, which constitute the benchmark for evaluating the parallelized version of the ENSOLV code. The performance of the parallel code is measured in terms of speed-up, memory usage and execution cost. At NLR, execution cost are expressed in a single number, so-called System Resource Units (SRU's). In the SRU's, the sum of all CPU-times, the amount of memory used and the time the memory is occupied, are accounted for; the formula reflects the cost price of the system elements Ref. 1. Note that the sum of all CPU-times is always larger when parallelization is applied. If the parallelized code will result in a *reduction* in real time, by the same factor as the *increase* in memory usage, the SRU's should stay constant. A detailed explanation of the SRU formula, as used for the calculations of the SRU's reported in this document, can be found in Ref. 13.

The *Data Parallelism* strategy for parallelizing ENSOLV was chosen Ref. 8. With this strategy, parallelism is obtained by splitting up the DO-loop's. Splitting up the DO-loop's is specif-

[1] Netherlands Initiative for Computational Fluid Dynamics in Engineering with HPCN

ically suited for shared memory computers, such as the shared memory parallel vector machine NEC SX-4, present at NLR.

There are different levels of DO-loop parallelization, two of which are:

1. *Low-level DO-loop parallelization*, parallelization of DO-loops in individual routines. A possible problem is the fine parallel grain size; the work per loop might not be enough to overcome the parallel overhead. Also, the parallelization has to be implemented on many loops in order to achieve an acceptable parallelization percentage;
2. *Block-loop parallelization*, parallelization of the DO-loop's over the blocks in the domain. This can be considered as high-level DO-loop parallelization. It results in the largest possible grain size. A possible problem is load imbalance. The ENSOLV code uses a multigrid algorithm, which is implemented around the multi-block algorithm. The operations of the multigrid algorithm are relaxation, restriction and prolongation. The routines performing these operations all contain block-loops. Therefore, this parallelization strategy is applicable.

Earlier, ENSOLV has been parallelized using the low-level DO-loop parallelization strategy. This parallelization is described in Ref. 11. The parallelization resulted in poor performance in terms of speed-up and execution cost, for most benchmark cases. For benchmark cases with a relatively high number of multigrid levels, combined with many small blocks in the grid, the poor performance was attributed to the large parallel overhead caused by the very fine grain size. It was decided that block-loop parallelization would be implemented. In Chapter 2, the block-loop parallelization of ENSOLV will be described briefly. Also, the system into which the resulting parallel code, along with tools for task estimation, task allocation and speed-up estimation, was integrated, will be described. In Chapter 3, the benchmark cases will be described and remarks are made about the expected performance of the parallel code for these benchmark cases. In Chapter 4, the results of testing the block-loop parallel ENSOLV code on the benchmark cases are presented and discussed. In Chapter 5, the final conclusions are given.

2 The Parallel ENSOLV System

In this section, the block-loop parallelization of ENSOLV is described briefly. A more extensive description of the parallelization can be found in Ref. 13. The resulting parallel code was integrated into a system including tools for task estimation, task allocation and speed-up estimation.

2.1 Block-loop parallelization of ENSOLV

Implementing block-loop parallelization, in stead of low-level DO-loop parallelization, has some consequences that need to be examined:

1. To eliminate the dependency between time integration in the blocks, the Gauss-Seidel algorithm is replaced with the Jacobi algorithm. This means that when updating the flow state of one block, the flow states from the prior Runge-Kutta time step in the adjacent blocks are used, in stead of the most recent flow states. Implementing a different solution procedure will generally change both convergence and stability, but should result in the same final solution. However, in order to allow a fast evaluation of the block-loop parallelization of ENSOLV, a simplified implementation of the Jacobi algorithm was used, resulting in a slightly different final solution (in particular near block interfaces) Ref. 3. Results of the serial ENSOLV code using this implementation of the Jacobi algorithm, can be found in Tables 5-14;
2. A significant increase in memory usage is unavoidable; computing the blocks in parallel means that each processor needs its own scratch arrays. For all benchmark cases, except the single block benchmark case 02, the memory size is approximately doubled when run on eight processors;
3. Since blocks differ in the number of grid points, the model used, boundary conditions applied etc., a load balancing problem may occur. Implementing a load balancing, or task allocation tool will improve the load balance (Section 2.2.2);
4. The maximum speed-up that can be obtained, is limited to the number of blocks used, if the number of blocks is less than the number of processors. Also, if a case has one large block and many small blocks, the maximum speed-up is limited by the work load of the large block.

The block-loops were parallelized by splitting these single loops in double loops; the outer loop over the processors and the inner loop over the blocks assigned to that processor by the task allocation tool (Section 2.2). The outer loops were parallelized by inserting **odir* directives, recognizable only to the NEC Fortran compiler and therefore leading to a portable code. No message passing code is necessary, since the parallelization takes place on a shared memory computer. The NEC SX-4 preprocessor now generates the parallel code.

2.2 Integration of parallel ENSOLV

The code was integrated into a system, including tools for task estimation, task allocation and speed-up estimation. The current work was carried out by operating this system through a specific working environment, ISNaS Ref. 6, where the calculations can be started by simple drag-and-drop actions.

2.2.1 Task estimation

Initially, the work load, or the weight for each block was set equal to the number of grid points. This is reasonable under the assumption that the work in a block is proportional only to the number of grid points, and all blocks are active in all parallel parts of the code. With ENSOLV, this assumption proved to be incorrect; if two blocks have the same number of grid points, but not the same ordering of their dimensions in the grid, their work loads can be different, due to a difference in vectorization performance.

The present task estimation tool performs (at least) one iteration of the block-loop parallel ENSOLV code, including timing-commands. The work load for each block is then set equal to the time it spends in the block-loops.

2.2.2 Task allocation

In order to improve the load balance, a task allocation tool was implemented. This task allocation tool is a stand-alone partitioning tool, based on Ozturan algorithm Ref. 7, adapted for shared memory machines Ref. 12. The algorithm starts from an existing partitioning of the blocks. It then re-locates blocks until a satisfying (theoretically) load balance is reached, or there is no more improvement possible.

2.2.3 Speed-up estimation

An estimation of the maximal attainable speed-up can be made following task estimation and task allocation. An approximation of the parallel part of the code can be obtained by adding the work loads for all blocks. We can now calculate the maximal attainable speed-up using Amdahl:

$$S = \left(\frac{f \cdot \max_P W_P}{\sum_{P=1}^{N_P} W_P} + (1 - f)\right)^{-1} \tag{1}$$

where f equals the fraction representing the parallel part, N_P equals the number of processors and W_P equals the work assigned to processor P.

3 Settings for evaluation of parallel ENSOLV

In this section, the characteristics of the benchmark cases are given. The tools in the parallel ENSOLV system are used to calculate maximal attainable speed-ups.

3.1 Characteristics of the benchmark cases

For the performance tests on the NEC SX-4, a set of test problems has been defined Ref. 5. The characteristics of these benchmark cases can be found in Table 1. In Fig. 1, the benchmark cases are identified by configuration and number of blocks.

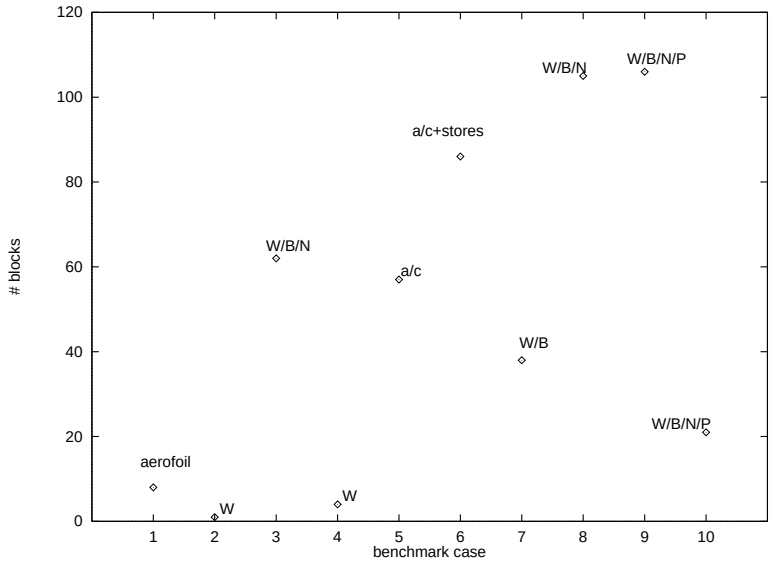


Fig. 1 Identification of benchmark cases by configuration and number of blocks

3.2 Maximal attainable speed-ups

In Tables 2 and 3, the task allocations calculated by the task allocation tool, discussed in Section 2.2.2, can be found. The work loads for benchmark case 05, measured by using one iteration only, were relatively small. This can lead to inaccuracies, e.g. when calculating the fraction f representing the parallel part. In order to reduce inaccuracies, the calculations for benchmark case 05 were done for the full 500 iterations. In Table 4, the maximal attainable speed-up calculated with Equation 1 can be found.

It is expected that only for benchmark case 02, a single block case, the block-loop parallelization will lead to significantly worse speed-up, compared to low-level DO-loop parallelization. For all other benchmark cases, block-loop parallelization is expected to lead to an improvement in

speed-up (Fig. 2).

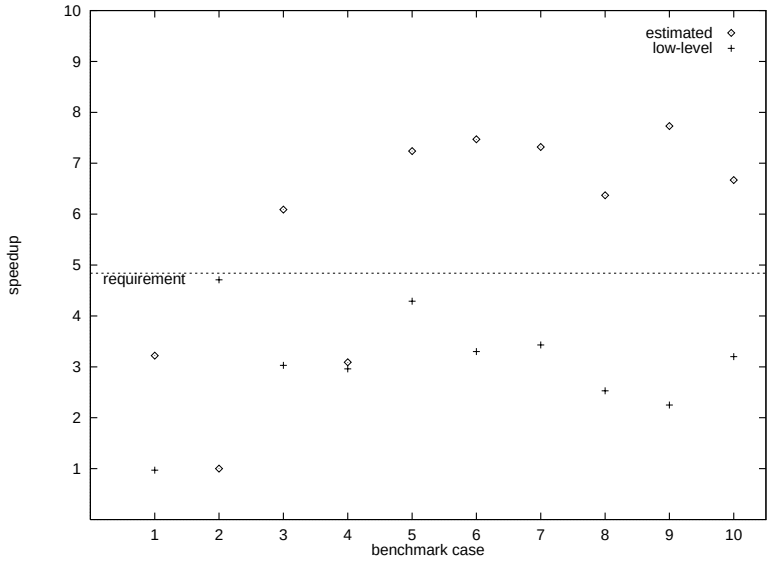


Fig. 2 Speed-up results for eight processors; estimated block-loop versus measured low-level DO-loop

4 Results

The block-loop parallelization results for all ten benchmark cases, for 1-, 4- and 8-processor runs, are shown in Tables 5-14 in Appendix A.

In Tables 5-14, the following definitions are used:

- The *Parallelization Overhead* is defined as the ratio of the real time needed by the parallel version run on 1 processor and the real time needed by the serial version;
- The *Speed-up for* N *processors* is defined as the ratio of the real time of the serial version and the real time of the parallel version run on *N* processors;
- The *Memory Overhead* is defined as the ratio of the amount of memory needed by parallel ENSOLV on *N* processors and the amount of memory needed by the serial version.

All real time results are timings of the iteration part of the solver, output to the ENSOLV output file *OUT*.

In the following sections, the speed-up, memory usage and execution cost are compared to low-level DO-loop parallelization results. Not all the results of low-level DO-loop parallelization are listed here, the reader is referred to Ref. 11.

4.1 Speed-up results

For all benchmark cases, except the single block benchmark case 02, block-loop parallelization shows better performance in terms of speed-up, compared to low-level DO-loop parallelization.

The remaining differences in speed-up estimations and measurements are attributed to the fact that the task estimation tool uses only one iteration. For benchmark case 05 the full 500 iterations were used, and the differences are minimal. The required speed-up of 4.8 for eight processors, defined by the ENSOLV user group, is attained by seven of the ten benchmark cases (Fig. 3).

4.2 Memory usage

As expected, the memory usage increases considerably for all benchmark cases, except the single block benchmark case 02 (Tables 5-14). Of course, the memory usage does not further increase when the number of processors is larger than the number of blocks. Benchmark case 10 shows the largest increase in memory usage. For all benchmark cases, the memory usage was smaller than the maximal available memory on the NEC SX-4.

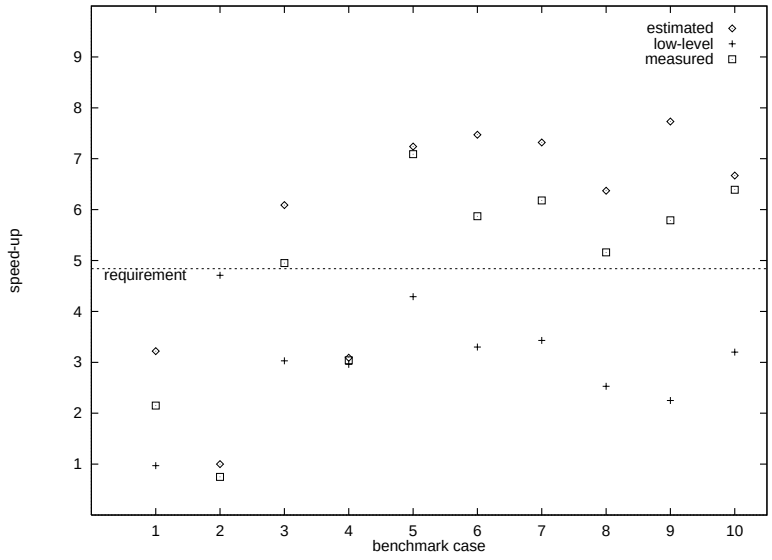


Fig. 3 Speed-up results for eight processors; measured block-loop versus estimated block-loop and measured low-level DO-loop

4.3 Execution cost

The execution cost for block-loop parallelization are considerably lower for all benchmark cases, except for the single block benchmark case 02.

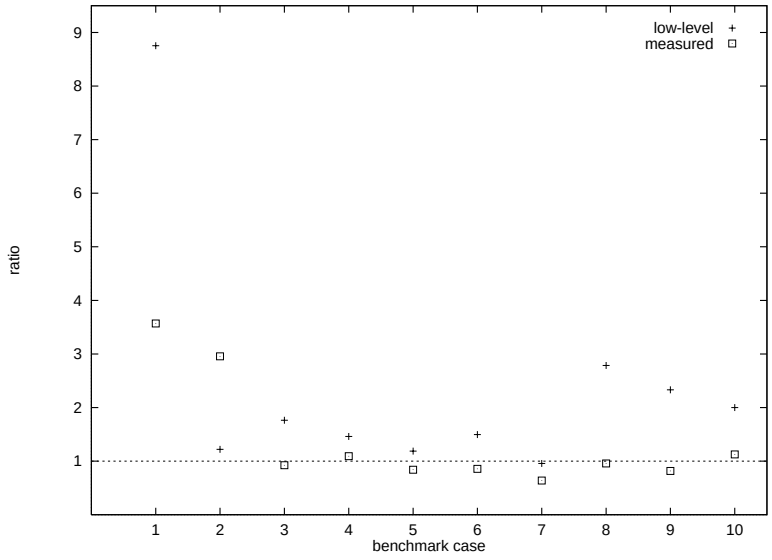


Fig. 4 Ratio of execution cost on eight processors and cost of sequential execution; measured block-loop versus measured low-level DO-loop

For eight of the ten benchmark cases, the cost for the parallel execution of ENSOLV on eight processors are equal to or less than the cost for serial execution of ENSOLV (Fig. 4). For the large memory benchmark case 07, the cost of the parallel runs are considerably lower than the cost of the serial run. This is due to the construction of the SRU formula Ref. 13.

5 Conclusions and future work

Block-loop parallelization has been used for parallelizing the multi-block Navier-Stokes flow solver ENSOLV. The parallel code was integrated into a system, including tools for task estimation, task allocation and speed-up estimation. Future users will be able to operate this system through a specific working environment, ISNaS Ref. 6, where the calculations can be started by simple drag-and-drop actions.

The block-loop parallelized code was tested on ten benchmark cases. The performance was measured in terms of speed-up, memory usage and execution cost, and compared to the performance of the low-level DO-loop parallelized code implemented earlier.

All benchmark cases, except the single block benchmark case 02, show better performance in terms of speed-up compared to low-level DO-loop parallelization. For seven of the ten benchmark cases, the speed-up for eight processors is higher than the the user required value of 4.8.

For all benchmark cases, except the single block benchmark case 02, memory usage increases considerably when using block-loop parallelization in stead of low-level DO-loop parallelization, as was foreseen.

The block-loop parallelization gives better or comparable performance in terms of execution cost, than the low-level DO-loop parallelization, for all benchmark cases, except the single block benchmark case 02. For six of the ten benchmark cases, the execution cost for parallel runs is lower than or comparable to the execution cost for the sequential run.

Based on the results, it was decided not to implement a single parallelization approach, combining both previously applied parallelization strategies; low-level DO-loop parallelization for larger blocks, block-loop parallelization for several smaller blocks.

6 References

1. Hameetman, G.: Private communications (1997)
2. Kok, J.C., Boerstoel, J.W., Kassies, A., Spekreijse, S.P.: *A robust multi-block Navier-Stokes flow solver for industrial applications*, NLR Technical Publication TP 96323 L (1996)
3. Kok, J.C.: Private communications (1998)
4. Kok, J.C.: *An industrially applicable solver for compressible, turbulent flows*, Ph.D. thesis, Delft University of Technology (1998)
5. Laban, M.: *Parallelized ENSOLV User Requirements*, NLR Technical Report TR 96353 L (1996)
6. Loeve, W., van der Ven, H., Vogels, M.E.S., Baalbergen, E.H.: *Network Middelware illustrated for enterprise enhanced operation,* NLR Technical Report TR 97224 L (1997), in CAPE'97 proceedings
7. Ozturan, C., deCougny, H.L., Shephard, M.S., Flaherty, J.E.: *Parallel adaptive mesh refinement and redistribution on distributed memory computers*, Comp. Methods Appl. Mech. Eng., Vol. 119 (1994) 123-137
8. Potma, K., Sukul, A.R.: *Preliminary ENFLOW Parallelization for the definition of a Parallelization Strategy for the NEC SX-4/16*, NLR Technical Report TR 96410 L (1996)
9. Schoenmaker, M.: *NLR SX-4/16 Vector/Parallel Supercomputer*, http://super.nlr.nl:8080/ (1998)
10. Sukul, A.R.: *Preliminary Parallelization Results of ENSOLV on the NEC SX-4*, NLR Technical Report TR 96725 L (1996)
11. Sukul, A.R.: *Predesign of ENSOLV Parallelization on the NEC SX-4*, NLR Technical Report TR 96726 L (1996)
12. van der Ven, H.: *Partitioning and parallel development of an unstructured, adaptive flow solver on the NEC SX-4*, NLR Technical Publication TP 97329 L (1997)
13. Wijnandts, P.: *Evaluation of block-loop parallelization of ENSOLV on the NEC SX-4/16*, NLR Technical Report TR 97344 L (1997)

This page is intentionally left blank.

Appendices

A Tables

Table 1 Characteristics of the benchmark cases (w=wing, w/b=wing-body, w/b/n=wing-body-nacelle, w/b/n/p=wing-body-nacelle-pylon, BL=Baldwin-Lomax, CS=Cebeci-Smith, JK=Johnson-King)

Case	ident	Config.	2D/3D	Blocks	Mcells	Mgrid	Euler/TLNS	Tur. Mod.	Iter.
01	RAE2822	aerofoil	2D	8	0.010	3	T(j)	BL	400
02	Delta	w	3D	1	0.369	3	T(k)	BL	200
03	AS28g	w/b/n	3D	62	1.556	2	E	-	500
04	Onera M6	w	3D	4	0.786	4	T(j)	CS	80
05	F16	a/c	3D	57	2.084	1	E	-	500
06	F16	a/c+stores	3D	86	2.084	2	E	-	360
07	VTP4	w/b	3D	38	6.636	4	T(j)	BL	100
08	VTP4	w/b/n	3D	105	1.455	3	T(j)	JK	100
09	Model 10	w/b/n/p	3D	106	2.211	3	T(i,j,k)	CS	100
10	Duprin	w/b/n/p	3D	21	0.577	2	E	-	100

Table 2 Task allocations for four processors, with W_P equal to the work load of processor P, the maximum given in bold

case	01	02	03	04	05	06	07	08	09	10
W_1	0.17	**36.20**	11.35	**29.76**	707.30	18.19	110.63	**42.50**	57.12	5.25
W_2	0.16	0	11.27	29.69	715.86	17.90	110.90	42.33	**57.69**	4.64
W_3	**0.17**	0	11.30	17.43	**726.24**	17.97	**112.44**	42.49	57.63	**5.34**
W_4	0.11	0	**11.44**	17.41	717.94	**18.21**	111.62	42.29	57.48	4.36
total	0.61	36.20	45.36	94.29	2867.34	72.27	445.59	169.61	229.92	19.59

Table 3 Task allocations for eight processors, with W_P equal to the work load of processor P, the maximum given in bold

case	01	02	03	04	05	06	07	08	09	10
W_1	**0.17**	**36.20**	5.44	**29.76**	354.89	9.00	56.12	20.50	28.79	2.57
W_2	0.16	0	5.49	29.69	**383.44**	**9.19**	55.27	20.31	**29.00**	2.70
W_3	0.07	0	**7.10**	17.43	355.95	8.96	54.23	20.45	28.89	2.08
W_4	0.10	0	5.47	17.41	352.33	8.99	56.11	21.13	28.75	2.21
W_5	0.03	0	5.54	0	353.48	8.91	**58.24**	20.32	28.61	2.59
W_6	0.01	0	5.46	0	351.00	9.07	54.72	20.27	28.54	**2.76**
W_7	0.04	0	5.41	0	366.18	9.02	56.89	**26.00**	28.47	2.49
W_8	0.03	0	5.45	0	350.07	9.13	54.01	20.63	28.87	2.19
total	0.61	36.20	45.36	94.29	2867.34	72.27	445.59	169.61	229.92	19.59

Table 4 Maximal attainable speed-ups for four and eight processors, with f the fraction representing the parallel part of the code

case	01	02	03	04	05	06	07	08	09	10
f	0.9394	0.9981	0.9908	0.9890	0.9949	0.9922	0.9932	0.9957	0.9962	0.9894
S_4	3.08	1.00	3.86	3.09	3.89	3.88	3.88	3.94	3.94	3.57
S_8	3.22	1.00	6.09	3.09	7.24	7.47	7.32	6.37	7.73	6.67

Table 5 Parallel performance for case 01, 400 iterations

#processors	**sequential or parallel**	**execution (real) time**	**parallel overhead**	**speed-up**	**MFLOPS**	**memory usage (MB)**	**memory overhead**	**SRU**
1	sequential	118	-	1.00	237	24	-	1475
1	parallel	124	1.05	0.95	226	25	1.04	1553
4	parallel	52	-	2.27	533	40	1.67	2543
8	parallel	55	-	2.15	508	54	2.25	5266

Table 6 Parallel performance for case 02, 200 iterations

#processors	sequential or parallel	execution (real) time	parallel overhead	speed-up	MFLOPS	memory usage (MB)	memory overhead	SRU
1	sequential	717	-	1.00	485	195	-	11297
1	parallel	865	1.21	0.83	436	212	1.09	12121
4	parallel	712	-	1.01	484	203	1.04	14052
8	parallel	962	-	0.75	364	212	1.09	33396

Table 7 Parallel performance for case 03, 500 iterations

#processors	sequential or parallel	execution (real) time	parallel overhead	speed-up	MFLOPS	memory usage (MB)	memory overhead	SRU
1	sequential	2256	-	1.00	666	249	-	38296
1	parallel	2327	1.03	0.97	652	266	1.07	34873
4	parallel	603	-	3.74	2488	376	1.51	33385
8	parallel	456	-	4.95	3291	465	1.87	35373

Table 8 Parallel performance for case 04, 80 iterations

#processors	sequential or parallel	execution (real) time	parallel overhead	speed-up	MFLOPS	memory usage (MB)	memory overhead	SRU
1	sequential	753	-	1.00	436	208	-	12170
1	parallel	760	1.01	0.99	433	208	1.00	12266
4	parallel	250	-	3.01	1307	398	1.91	11628
8	parallel	248	-	3.04	1324	406	1.95	13300

Table 9 Parallel performance for case 05, 500 iterations

#processors	sequential or parallel	execution (real) time	parallel overhead	speed-up	MFLOPS	memory usage (MB)	memory overhead	SRU
1	sequential	2873	-	1.00	644	241	-	48331
1	parallel	2882	1.00	1.00	643	241	1.00	48412
4	parallel	768	-	3.74	2402	357	1.48	40759
8	parallel	405	-	7.09	4541	502	2.08	40594

Table 10 Parallel performance for case 06, 360 iterations

#processors	sequential or parallel	execution (real) time	parallel overhead	speed-up	MFLOPS	memory usage (MB)	memory overhead	SRU
1	sequential	2600	-	1.00	618	282	-	45689
1	parallel	2648	1.02	0.98	608	300	1.06	40254
4	parallel	684	-	3.80	2324	434	1.54	38595
8	parallel	443	-	5.87	3584	584	2.07	39109

Table 11 Parallel performance for case 07, 100 iterations

#processors	sequential or parallel	execution (real) time	parallel overhead	speed-up	MFLOPS	memory usage (MB)	memory overhead	SRU
1	sequential	4430	-	1.00	621	859	-	129283
1	parallel	4593	1.04	0.96	617	859	1.00	130170
4	parallel	1270	-	3.49	2161	1555	1.81	91093
8	parallel	717	-	6.18	3825	1944	2.26	82421

Table 12 Parallel performance for case 08, 100 iterations

#processors	sequential or parallel	execution (real) time	parallel overhead	speed-up	MFLOPS	memory usage (MB)	memory overhead	SRU
1	sequential	1676	-	1.00	375	211	-	27197
1	parallel	1752	1.05	0.96	362	228	1.08	25539
4	parallel	560	-	2.99	1117	281	1.33	24390
8	parallel	325	-	5.16	1923	346	1.64	26005

Table 13 Parallel performance for case 09, 100 iterations

#processors	sequential or parallel	execution (real) time	parallel overhead	speed-up	MFLOPS	memory usage (MB)	memory overhead	SRU
1	sequential	2294	-	1.00	395	285	-	40452
1	parallel	2326	1.01	0.99	389	302	1.06	35627
4	parallel	681	-	3.37	1320	357	1.25	33209
8	parallel	396	-	5.79	2269	436	1.53	33046

Table 14 Parallel performance for case 10, 100 iterations

#processors	sequential or parallel	execution (real) time	parallel overhead	speed-up	MFLOPS	memory usage (MB)	memory overhead	SRU
1	sequential	198	-	1.00	570	104	-	2784
1	parallel	195	0.98	1.02	571	104	1.00	2776
4	parallel	64	-	3.09	1715	183	1.76	3028
8	parallel	31	-	6.39	3487	249	2.39	3135