Nationaal Lucht- en Ruimtevaartlaboratorium
National Aerospace Laboratory NLR

NLR TP 97592

High-level versus low-level do-loop parallelization: results for one testcase of a multi-block solver on a shared memory parallel vector computer

P. Wijnandts and M.E.S. Vogels

DOCUMENT CONTROL SHEET

	ORIGINATOR'S REF.		SECURITY CLASS.
	TP 97592 U		Unclassified

ORIGINATOR
National Aerospace Laboratory NLR, Amsterdam, The Netherlands

TITLE
High-level versus low-level do-loop parallelization: results for one testcase of a multi-block solver on a shared memory parallel vector computer

PRESENTED AT
the High Performance Computing and Networking Europe '98 Conference, Amsterdam, the Netherlands, April 21-23, 1998

AUTHORS	DATE	pp	ref
P. Wijnandts and M.E.S. Vogels	971126	10	4

DESCRIPTORS

Computational fluid dynamics	Grid generation (mathematics)
Computational grids	Multiblock grids
Computerized simulation	Multigrid methods
Computer programs	Navier-Stokes equation
Cost analysis	Parallel processing (computers)
Cost reduction	Performance tests

ABSTRACT
Within the NICE project, supported by the Dutch Foundation HPCN, the multi-block Navier-Stokes flow solver ENSOLV is being parallelized. In this article first results of the block-loop parallelization of ENSOLV are presented. We discuss the performance results of this parallelization on a 16-processor NEC SX-4, a shared memory parallel vector computer. The results are compared to those of the low-level DO-loop parallelization implemented earlier. The conclusion is that for the cases with higher number of multigrid levels, the block loop parallelization gives better speed-up, requires more memory, and gives overall less execution cost.

Summary

Within the NICE project, supported by the Dutch Foundation HPCN, the multi-block Navier-Stokes flow solver ENSOLV is being parallelized. In this article first results of the block-loop parallelization of ENSOLV are presented. We discuss the performance results of this parallelization on a 16-processor NEC SX-4, a shared memory parallel vector computer. The results are compared to those of the low-level DO-loop parallelization implemented earlier. The conclusion is that for the cases with higher number of multigrid levels, the block loop parallelization gives better speed-up, requires more memory, and gives overall less execution cost.

Contents

1 Table

3 Figures

(10 pages in total)

1 Introduction

The multi-block Navier-Stokes flow solver ENSOLV is currently operational at NLR and industry. Within the NICE project ENSOLV is being parallelized in order to reduce the turnaround time. Initially, two strategies for adding parallelism to the code were considered. The first was *Message Passing*, which would involve major changes to the code and is usually employed on a distributed memory computer. The second was *Data Parallelism*, where parallelism is added by splitting up the DO-loop's. This strategy costs little effort to apply and is specifically suited for shared memory computers such as the NEC SX-4 present at NLR. DO-loop's can be parallelized on different levels. Low-level DO-loop parallelization is 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. High-level DO-loop, or block-loop parallelization is parallelization of the DO-loop's over the blocks in the domain. This results in the largest possible grain size.

After parallelizing the flow solver ENSOLV using the low-level DO-loop parallelization strategy Ref. 2, it was clear that this strategy resulted in poor performance in speed-up and execution costs for cases with a relatively high number of multigrid levels (and hence very fine grain size). It was then decided that block-loop parallelization of ENSOLV would be tested.

For the evaluation one testcase was chosen from ten benchmark testcases described in Ref. 1. The testcase concerns a Navier-Stokes simulation about a wing-body-nacelle configuration on 105 blocks with 1.455 million grid points and 3 multi-grid levels. Because of the complexity of the configuration, there are several tens of small blocks of sizes $8\times8\times12$ - $8\times12\times16$, which causes the poor speed-ups in low-level DO-loop parallelization strategy. It was expected that the block-loop parallelized code would show an improvement in speed-up for this particular testcase, and, hopefully, an improvement in the execution cost.

The performance is measured in terms of speed-up, memory usage and execution costs. At NLR, the execution costs are expressed in a single number, in so-called System Resource Units (SRU's). In the SRU's, the amount of CPU-time, memory and I/O are accounted for; the formula reflects the cost price of the system elements [Ref. 4].

2 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 closely. First, to eliminate the dependency between time integration in the blocks, the Gauss-Seidel algorithm is replaced by the Jacobi algorithm. This change has influence on the convergence speed, but not on the final solution.

Second, a significant increase in memory usage is unavoidable; computing the blocks in parallel means that each block will need its own scratch array for storing values. Even though memory usage and also CPU-time increases, this does not necessarily mean that the execution cost will increase, since the memory will be occupied for a shorter time compared to the single processor execution.

The third consequence involves load balancing. Since blocks differ in the number of grid points, the model used, boundary conditions applied etc., a load balancing problem may occur. For the current evaluation a task allocation, considering only the number of grid points per block, was implemented manually.

The block-loops were parallelized by inserting **odir* directives. No message passing code is necessary, since the parallelization takes place on a shared memory computer. The NEC SX-4/16 preprocessor now generated the parallel code.

3 Results and discussion

The results for the block-loop parallel ENSOLV code for the time integration part, performing 100 iterations are given on the Table below.

Table 1 Performance results for testcase 08

#proc	sequential or parallel	execution (real) time	S_p	Memory usage (Mb)	MFLOPS	SRU
1	sequential	1676	1.00	211	375	27197
1	parallel	1696	0.99	210	370	27500
4	parallel	546	3.07	280	1144	24805
8	parallel	326	5.14	380	1917	25764

Performance The attained speed-ups for both the block-loop and the low-level DO-loop parallelized code are given in Figure 1. The speed-up for the block-loop parallelized code is considerably better, as expected, because of the larger grain size.

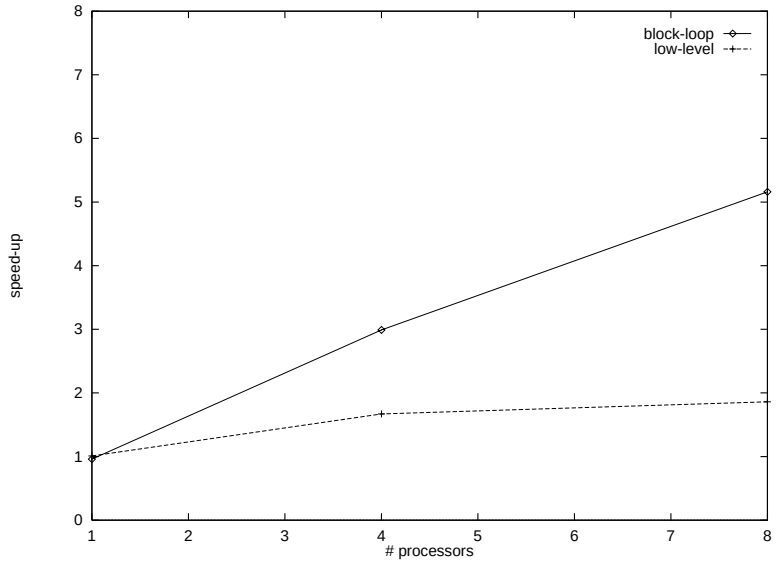

Fig. 1 Speedup results

Memory overhead Due to the fact that each processor needs its own scratch array to calculate the blocks, the memory overhead is larger (see Fig. 2).

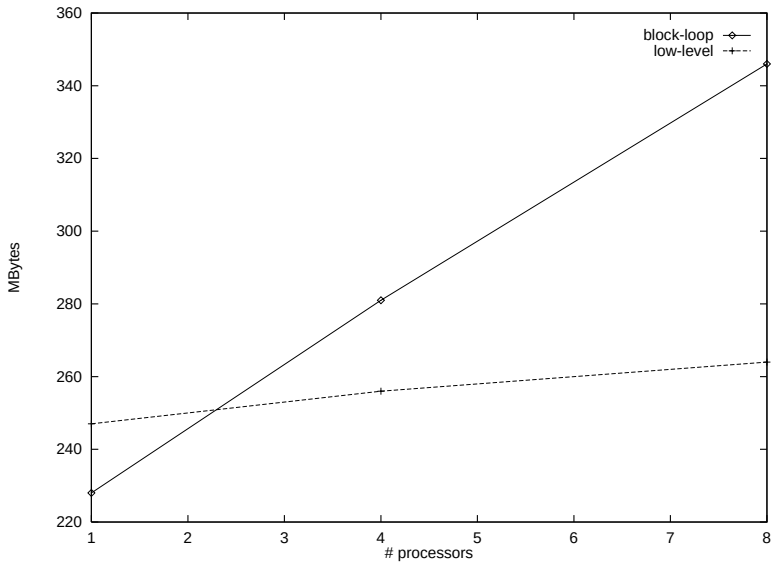

Fig. 2 Memory usage

SRU costs The SRU costs proved to be invariant for the number of processors studied; the improvement in performance is not canceled by the deterioration in memory usage (see Fig. 3).

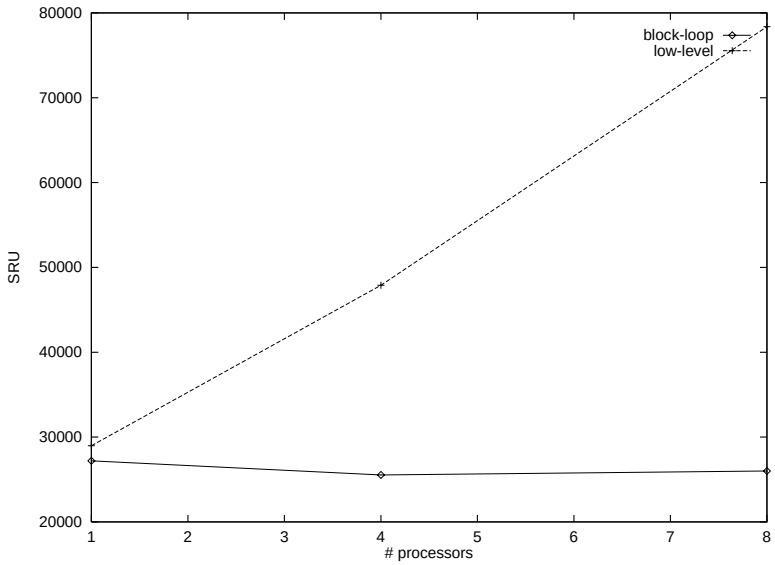

Fig. 3 SRU costs

4 Conclusions and future work

Block-loop parallelization has been used for parallelizing ENSOLV 2.00. The resulting code was tested on one benchmark testcase with 105 blocks of varying sizes, and 3 multi-grid levels. Compared to low-level DO-loop parallelization, block-loop parallelization showed great improvement in the speed-up, a high increase in memory usage, and almost invariant SRU costs.

Next, the code will be integrated into a system, including performance estimation and task allocation. This system will be tested for all ten benchmark testcases, while at the same time, improving the performance estimation.

5 References

1. M. Laban, *Parallelized ENSOLV User Requirements*, NLR Technical Report TR96353L, 1996.
2. A.R. Sukul, *Predesign of ENSOLV Parallelization on the NEC SX-4*, NLR Technical Report TR96726L, 1996.
3. P. Wijnandts, *Evaluation of block-loop parallelization of ENSOLV on the NEC SX-4/16*, NLR Technical Report TR97344L, 1997.
4. G.J. Hameetman. Private communication.