Nationaal Lucht- en Ruimtevaartlaboratorium

National Aerospace Laboratory NLR

NLR TP 96575

Experiences with advanced CFD algorithms on NEC SX-4

H. van der Ven and J.J.W. van der Vegt

DOCUMENT CONTROL SHEET

	ORIGINATOR'S REF. NLR TP 96575 U		SECURITY CLASS. Unclassified

ORIGINATOR
National Aerospace Laboratory NLR, Amsterdam, The Netherlands

TITLE
Experiences with advanced CFD algorithms on NEC SX-4

PRESENTED AT
The 2nd International Meeting on Vector and Parallel Processing, Porto, Portugal, September 25-27, 1996.

AUTHORS H. van der Ven and J.J.W. van der Vegt	DATE 960919	pp 17	ref 5

DESCRIPTORS

Algorithms	Numerical integration
Computational fluid dynamics	Parallel programming
Computer systems performance	Programming environments
Eyler equations of motion	Run time (computers)
Finite volume method	Software development tools
Galerkin method	Supercomputers

ABSTRACT
In this paper topics related to parallel CFD simulations are discussed. The first topic is the shared memory parallelization of the unstructured adaptive flow solver Hexadap. The second topic discusses the performance results of this parallelization on a 16 processor NEC SX-4. The third topic combines the first two and concerns the CFD working environment ISNaS as developed by the National Aerospace Laboratory NLR.

Summary

In this paper three topics related to parallel CFD simulations are discussed. The first topic is the shared memory parallelization of the unstructured adaptive flow solver Hexadap. The second topic discusses the performance results of this parallelization on a 16 processor NEC SX-4. The third topic combines the first two and concerns the CFD working environment ISNaS as developed by the National Aerospace Laboratory NLR.

Contents

4 Tables

3 Figures

(17 pages in total)

1 Introduction

The National Aerospace Laboratory NLR has a long-standing tradition in the development and use of CFD software. The CFD software includes production solvers using proven technologies, research codes for developing and testing new algorithms and the support of both production and development of CFD software. This paper considers the latter two subjects.

NLR is developing a flow solver based on a discontinuous Galerkin discretization of the Euler or Navier-Stokes equations. The solver uses structured hexahedrons as initial mesh, which is followed by solution adaptive unstructured refinement on a hexahedron by hexahedron basis. Main incentive for the development of this solver is time-accurate flow simulation which will require the utmost of present and future computer hardware. Because of the computational complexity of large scale applications parallelization of the underlying algorithm is required. In June 1996 the NEC SX-3 at NLR has been replaced by a 16 processor NEC SX-4. A first parallelization has been performed on the NEC SX-4/16. This parallelization assesses the SX-4 capabilities and the parallelism of the flux integration scheme, which is the most computational intensive part of the algorithm. The parallelization is based on the shared memory paradigm.

To support the development and use of CFD software NLR has developed a working environment for CFD applications. This working environment appears to the user as a single, virtual computer, even though programs and data are located on different computers in a network. In addition to network transparency, the working environment provides facilities for the management of software, data, and documents.

The contents of this paper is as follows. In Chapter 2 the underlying algorithm of the adaptive, unstructured flow solver will be briefly described. In Chapter 3 the chosen parallelization strategy will be described. In Chapter 4 the results of the parallelization will be described and discussed. In Chapter 5 the working environment will be described. In Chapter 6 conclusions will be drawn.

2 Algorithm

Computational Fluid Dynamics is used for increasingly complicated problems. Many advanced applications of CFD can only be done with sophisticated grid adaptation algorithms and require significant computer resources. Capturing of shocks, vortical structures, and time-dependent changes in the flow pattern is still one of the key elements preventing efficient time-accurate simulation of problems in aerospace. Applications requiring time-accurate simulation are structural dynamics, aircraft maneuver, aircraft under large angle of attack, propeller-wing interaction and noise prediction. These applications can only be simulated efficiently with sophisticated grid adaptation techniques and require a numerical scheme which is accurate on highly irregular grids.

This was the main motivation to develop a new, discontinuous Galerkin finite element algorithm for the Euler/Navier-Stokes equations. The discontinuous Galerkin method (DG) uses a local polynomial expansion in each cell which results in a discontinuity at each cell face. This discontinuity can be represented as a Riemann problem which provides a natural way to introduce upwinding into a finite element method. The DG method can therefore be considered as a mixture of an upwind finite volume method and a finite element method. A unique feature of the DG method is that seperate equations for the flow gradients are used, which do not have to be determined from neighbouring cells.

A combination of local grid refinement and the discontinuous Galerkin finite element method is applied in the flow solver Hexadap (Ref. 3). This combination is capable of efficiently resolving local phenomena such as shear layers and shocks. This paper will be limited to inviscid flow in order to demonstrate the parallelism in the basic algorithm.

The DG method, combined with a face based data structure, is extremely local in nature and makes it a good candidate for parallel computing.

3 Parallelization strategy

The above described algorithm consists of two parts, namely grid adaptation and flow computation. The grid adaptation part, which consists predominantly of scalar operations, requires a domain decomposition for parallelization and is not considered in this paper. The flow computation has two main components: the calculation of cell face fluxes and a slope limiter.

The flow computation part of the solver consists of three kinds of loops:

face-face loops over faces, updating face values,
cell-cell loops over cells, updating cell values,
face-cell loops over faces, updating cell values.

An example of a face-face loop is the evaluation of the fluxes through cell faces. Examples of cell-cell loops are the loops in the Runge-Kutta scheme where the residuals are added to the flow states. An example of a face-cell loop is the update of the residual of a cell by the flux through one of its faces.

The loops over the faces, both face-face and face-cell, are split over different colours. This is done to be able to vectorize the face-cell loop. Within one colour all faces connect to cells with different cell addresses, hence within one colour there is no data dependency when updating cell values.

The parallelization of the face-cell loops is the most challenging. In general, the loop length of the face-face and face-cell loops is not sufficient for both vectorization and parallelization. The face-face loops can be parallelized over the colours, but this is prohibited for face-cell loops since data dependencies may occur.

The face-cell loops in the routine that calculates the residuals can, however, be parallelized by considering the number of variables for which a residual update is needed. For all independent variables, density, momentum components, and total energy, the residual has to be updated, and these updates are independent. For the second order space discretization also the three moment residuals need to be updated. Together with the time step update we thus have $5 + 3 \times 5 + 1 = 21$ independent updates. Since these updates have to be performed for both cells bounding the face, we have 42 independent updates within one colour. For a 16 processor machine this is sufficient parallelism. In Fig. 1 the structure of the flux integration is presented in pseudo-code.

In order to achieve load balance in the parallel loops over the colours, the number of colours is

```
           pardo for all colours
               do for all faces of a colour
face-face  |       calculate flux through face
               enddo
           enddo
           do for all colours
               pardo for all 42 variables
                   do for all faces of a colour
face-cell  |           add fluxes to residuals of cell based variables
                   enddo
               enddo
           enddo
```

Fig. 1 Pseudo-code representation of the flux integration. Parallelized do-loops are written as **pardo**. *Also shown are the kinds of the loops.*

adjusted to be a multiple of the number of processors, and the vector lengths of all colours are nearly equal.

The above treats the parallelization of the flux integration, the parallelization of the slope limiter is described below. The computation of the slope limiter consists of a number of min-max operations for each cell. The limiter is computed for each cell and for just the five independent variables, and hence the parallelization strategy for the residual updates cannot be followed. For the limiter the loops over the faces within one colour and the five independent variables are collapsed and both vectorized and parallelized. In Fig. 2 the structure of the slope limiter computation is presented in pseudo-code.

The remaining loops in the integration part of the solver are cell-cell loops, which are easily both parallelized and vectorized, sometimes collapsing them with the loop over the five independent variables. Eventually, the routines accounting for 99.5% of the integration time are parallelized.

The above parallelization strategy is different from the one proposed in Van der Vegt et al.(Ref. 3). The reason for the change is twofold. On the one hand, it was based on the assumption that a critical section was needed for the face-wise update of the cell-based residuals, which restricted the optimal speedup to 5. Using the strategy described above no critical section is needed. On the other hand, the flow solver has been improved and is now better suited for the above parallelization

```
                do for all colours
                    pardo for all faces and five independent variables
face-cell   |           compute mean, minimum, maximum
            |               of flow states over neighbouring cells
                    enddo
                enddo
                pardo for all colours
                    do for all faces
face-face   |           calculate limiter through cell face
                    enddo
                enddo
                do for all colours
                    pardo for all faces and five independent variables
face-cell   |           calculate minimum of slope limiter over all faces of a cell
                    enddo
                enddo
                pardo for all cells
                    correct flow status using slope limiter
cell-cell   |   enddo
```

Fig. 2 Representation of slope limiter in pseudo-code. Parallelized do-loops are written as **pardo***. Also shown are the kinds of the loops.*

strategy.

Because of memory restrictions on the NEC SX-3 the previous version of the flow solver made extensive use of the extended memory unit (XMU). The XMU is not part of the shared memory, and has to be addressed by explicit I/O. Due to hardware restrictions, this I/O is not parallelizable and the subsequent large serial sections in the code prohibit parallel efficiency. In the present version of the flow solver use of XMU is an option to the user, and for problems that fit in main memory no serial sections in the code occur.

4 Parallelization results and discussion

4.1 NEC SX-4 architecture

The code has been ported to the NLR NEC SX-4, using the above parallelization strategy and compiler directives. The processors in the SX-4 combine a powerful, balanced 2 Gflop/s vector unit with a state-of-the-art superscalar unit. Special synchronization, interprocessor communications and control hardware is implemented to maximize parallel processing efficiency.

The NEC SX-4 installed at NLR has 16 processors, 4 GB main memory and an extended memory unit of 8 GB. Each processor has eight vector pipes of length 256. The main memory consists of 16 modules, each having a bandwidth of 16 GB/s to the processors. Hence the bandwidth scales with the number of processors (with a peak of 256 GB/s), and a better parallel efficieny is expected than on the NEC SX-3. In the acceptance benchmark investigation (reported by Potma et al.,Ref. 1) the single processor NEC SX-4 reached a flop rate of 0.618 Gflop/s on the complete program Hexadap, which is an increase of 1.7 with respect to the performance of the NEC SX-3. On the SX-4 a speedup of 9.5 was obtained on 16 processors. These benchmark results were obtained using a previous version of the flow solver Hexadap, the present version has been developed further and is in principle more suited for parallel processing. Moreover, the reported speedups concern only the parallelized routines, where the results in this paper are for the complete flow solver.

4.2 Metrics

The prime metric for parallel processing is elapsed time. Therefore, in the discussion of the results the emphasis will lay on elapsed timings. Other relevant metrics are speedup and scalability.

Speedup is defined as the single processor execution time divided by the multi processor execution time. Here the single processor execution time is measured by running the parallel algorithm on one processor. To be able to measure the performance of parallel algorithms on large problems which do not fit in single processor memory *generalized speedup* (Ref. 2) has been introduced. This is defined as the quotient of sequential speed over parallel speed, where the two speeds may be obtained on different problem sizes.

Of interest to the industry is also the parallelization effort. To measure the effort the required man power is measured and the number of changed lines is counted.

4.3 Results

The parallelization effort has taken 3 manweeks extra in the CFD development effort. A total of 4500 lines has been changed (of the total of 50,000 lines) and 70 compiler directives have been

Table 1 Mesh characteristics for the four test cases. The work, measured in floating point operations, is defined as the number of floating point operations in the integration part of the solver. Memory is for single processor execution.

case	# cells	# faces	work [Gflop]	memory [GB]
`M6/1`	131,072	386,176	492.676	0.398
`M6/2`	262,190	778,257	1,000.951	0.904
`M6/3`	517,021	1,554,600	2,000.709	2.029
`M6/4`	865,623	2,613,670	3,355.267	2.586

added.

The tests used to assess the parallel performance are constructed from an initial structured mesh around the ONERA M6 wing. Subsequently the mesh has been adapted three times to obtain a series of meshes where each mesh is roughly twice the size of the previous mesh. See Table 1 for a characterization of the meshes. In this table, the work is defined as the number of floating point operations required in the integration part of the solver to advance the flow 100 time steps. The size of the last test case, `M6/4`, is restricted by memory requirements.

In Table 2 the elapsed timings of the flow integration part of the solver are presented for the four case and different number of processors. executed on a few processors since the timings are not relevant for the discussion. The NEC SX-4 at NLR is part of a multi-user environment and it is not possible to use the NEC SX-4 as a dedicated machine. Even though one can use all 16 processors, other users may use the computer interactively and system processes may run in the background. The performance of the flow solver does not increase on 16 processors, because other processors have to wait for results of the one processor which is (also) used for other processes. Therefore timings are only presented for up to 14 processors.

The (traditional) speedups for case `M6/1` and `M6/2` are tabulated in Table 3. Using Amdahl's Law

Table 2 Elapsed timings for the flow integration part of the flow solver.

case	1	2	4	8	12	14
`M6/1`	663	341	177	93	67	58
`M6/2`	1441	726	377	200	146	125
`M6/3`	-	-	720	375	268	230
`M6/4`	-	-	-	652	462	397

Table 3 Speedups based on the elapsed timings for the flow integration part of the flow solver.

case	1	2	4	8	12	14
`M6/1`	1.0	1.94	3.75	7.13	9.89	11.43
`M6/2`	1.0	1.98	3.82	7.20	9.87	11.53

and given a parallelization ratio of 99.5% one would predict a speedup of 13.1 on 14 processors. Hence, the experimental speedups are close to the theoretical speedup. Differences most likely are caused by parallel overhead and small not parallelized sections in the parallelized routines.

On 14 processors the performance of the flow solver reaches 8.5, 8.0, 8.7 resp. 8.5 Gflop/s for the respective four cases. Surprising in the timing results is that a n-fold increase in the work, results in a roughly an n-fold increase in the elapsed time, even when running on several processors. This implies that for the given algorithm and given architecture traditional and generalized speedup coincide.

In Fig. 3 the reached speeds for the different cases and number of processors are shown. Here, speed is defined as the quotient of work (see Table 1) over elapsed time (see Table 2). It can be clearly seen that attained speed is nearly independent of the problem size.

The elapsed timings also show that the machine-algorithm combination scales well. An increase in the work and a likewise increase in the number of processors result in comparable elapsed timings.

A parameter in the algorithm determines the number of faces per colour, and hence the vector length in the flux integration and slope limiter routines. This parameter also determines the size of the scratch arrays in these routines, and therefore the size of scratch memory. In all the above tests the parameter is equal to 12,800. To assess the dependence of the performance on this parameter, test case `M6/2` on 8 processors is repeated with two different values of the parameter. Results are tabulated in Table 4. For small values of the parameter the performance degrades, because of decreased vector length. For memory critical runs the parameter may be reduced to limit memory requirements.

As mentioned in Chapter 3 it is possible to store and retrieve some scratch arrays on the extended memory unit (XMU). Since these stores and retrieves are essentially sequential, the use of XMU will degrade parallel performance. Test case `M6/2` is run on 8 processors using XMU. As can be seen in Table 4 execution time is almost doubled. The parameter determining the number of faces per colour is equal to 12,800. Memory use is reduced using XMU, but the memory use is the same as the run with the parameter equal to 4096 not using XMU.

Table 4 Timings and memory use for different values of the parameter determining the number of faces per colour; and comparison with a run using the extended memory unit (with parameter equal to 12800). All runs for case `M6/2` on 8 processors.

parameter value	4096	12800	25600	using XMU
elapsed time	210	200	200	343
memory use	818	1058	1236	809

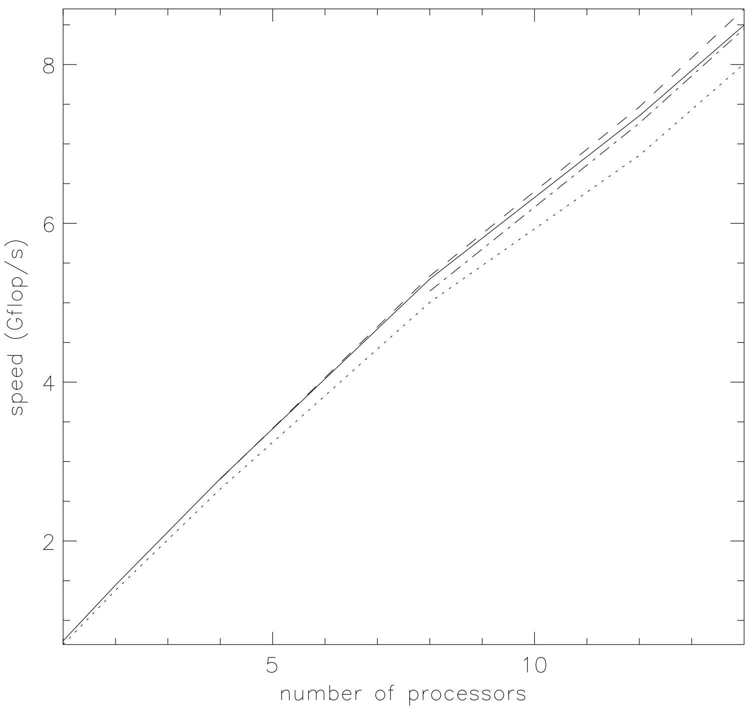

Fig. 3 Attained speeds (in Gflops/s) for the different cases and different number of processors.
— `M6/1`, · · · `M6/2`, - - - `M6/3`, – · – `M6/4`

4.4 Discussion

The present parallelization efforts have taken a limited time of three manweeks and produced a parallel code which has excellent parallel performance and good scalability. On the 16 processor NEC SX-4 a speedup of 11.5 and a speed of 8.5 Gflop/s is obtained by using 14 processors.

The attained speed is independent of the problem size.

The code has been restructured slightly, and the parallel version is part of version pipeline ensuring that future extensions of the functionality will use the parallel structure, which is achieved using the software repository in the CFD working environment.

5 CFD working environment

The parallelization and development of the above algorithm is supported by (tools in) the working environment ISNaS (Ref. 5) for CFD applications. Within the NICE project this working environment serves as a prototype for the Dutch HPCN Center for Flow Simulation, HFS. HFS supports cooperative work across the network consisting of the combined networks of the partners with the NLR NEC SX-4 in its center.

CFD development and use takes place on such diverse systems as mainframes, parallel and/or vector computers and graphical work stations. ISNaS is designed to make the network transparent to the user. Together with the management of data and documentation, processing of CFD codes in a production environment is greatly facilitated by ISNaS.

ISNaS also supports the developers of CFD software during the development phase. In large projects where several disciplines cooperate, software configuration management and information exchange support the software engineers. The use of the working environment enforces quality control. The Informatics division of NLR is certified for ISO-9001.

To support the development of parallel codes the parallelization tools of the NEC SX-4 are integrated in ISNaS. Two main support tools for parallelization are the parallelizer and analyzer tool. The integrated tools allow users easy access to this tools without the need to read all details in the manuals. Options of the tools which are considered to be generic to all problems are presented to the user as clear text. For the analyzer tool, for example, the user can choose between static analysis, execution time analysis of the entire program and do-loop analysis of specified loops. In this way, the tools are made accessible in a user-friendly way and it is expected that the parallel use of the NEC SX-4 will increase. Moreover, job control and assignment of processors are handled by the working environment.

ISNaS is a so-called instantiation of SPINE, the general tool developed at NLR to create application area specific working environments. ISNaS or related products can be installed on any UNIX network.

6 Conclusions

In this paper the use of the NEC SX-4 for advanced CFD processing is described.

First, a hexahedron based, flow solver with unstructured grid adaptation is parallelized on the 16 processor NEC SX-4 using the shared memory paradigm. Roughly 10% of the code is restructured to achieve an efficient algorithm which reaches 8.5 Gflop/s on 14 processors. The obtained speed is independent of the problem size. This proves the capabilities of the NEX SX-4 as a high performance computing platform. The present parallelization only concerns the flow solver part of the algorithm, for the parallelization of the adaptation part a grid partitioning is required. This constitutes future work.

Second, the efficient use of the NEC SX-4 is supported by the working environment ISNaS for CFD applications. Support tools for the analysis and shared memory parallelization of algorithms have been integrated in the working environment. This integration supplies easy access to the tools for first-time users. In the production phase, ISNaS supports users of the CFD software in job control, version consistency and assignment of processors.

Summarizing, the NEC SX-4 allows for good parallel efficiency using the easy-to-use shared memory paradigm. Application to production codes will reduce computing time and costs.

Acknowledgment The authors want to thank both G.A. van der Velde (NEC), for his idea of independent residual updates, and K. Potma and M.C.Z. Schoemaker (NLR) for their support in the use of the NEC SX-4.

7 References

1. K. Potma, G.J. Hameetman, W. Loeve and G. Poppinga, *Early benchmark results on the NEC SX-4*, NLR Technical Publication TP96464L, 1996, presented at Parallel CFD'96, Capri.
2. Xian-He Sun and J. Gustafson, *Toward a better parallel performance metric*, Parallel Computing 17 (1991) 1093-1109.
3. J.J.W. van der Vegt, *Anisotropic grid refinement using an unstructured discontinuous Galerkin method for the three dimensional Euler equations of gas dynamics*, AIAA Paper 95-1657, 1995.
4. J.J.W. van der Vegt and H. van der Ven, *Hexahedron based grid adaptation for future Large Eddy Simulation*, AGARD symposium 'Progress and challenges in CFD methods and algorithms', Seville, October 1995.
5. M.E.S. Vogels and W. Loeve, *Development of ISNaS: an information system for flow simulation in design*, NLR TP89025, 1989. Also see *http://www.nlr.nl/public/fac/fac-isna/isnas.html*