Programming the Bi-CGSTAB matrix solver for HPC and benchmarking IBM SP3 and alpha ES40

Liu, P.; Li, K.

Programming the Bi-CGSTAB Matrix Solver for HPC and Benchmarking IBM SP3 and Alpha ES40

P. Liu and K. Li

***Abstract*—The panel method, or the boundary element method, has been widely used for calculating lifting flows in which forces on wing surfaces are primarily of interest. In panel method for a wind generation system with a number of propfans or for an oil platform motion control assembly with a number of thrusters/propellers, calling time for a matrix solver is the key factor for computing efficiency. This paper discusses the code development of an iterative matrix solver in MPI via C and benchmark analysis for a shared and a distributed memory UNIX machine, respectively.**

***Index Terms*— BiCGSTAB, Iterative matrix solver, HPC, Parallel computing**

I. Introduction

SURFACE panel method, or the boundary element method for lifting and non-lifting flows was initialized in the 1970s (Hess 1972). It has been widely used in aerodynamics and hydrodynamics computations. Simulation of single marine propeller using panel method was led by Hess and Valarezo (Hess and Valarezo 1985).

In a time-domain panel method with multi-body interactions such as in the case of ducted propellers, propeller with nozzle or rudder or both, or propeller with ice blockage, the influence coefficient matrix needs to be created at each time step. For a sudden accelerating propeller computation, at least three revolutions are needed to obtain stable results that can be used for design and performance evaluation. Each revolution consists of about 40 time steps, so the minimum required total time step for one run at one advance coefficient is n_t=120. At each time step, unsteady Kutta condition is applied at the trailing edge of a lifting foil section. An approximate zero-pressure difference at the trailing edge, resulted from the Kutta condition, is to be obtained by an iterative procedure. If the Newton-Raphson iteration procedure is used, about n_i=5 iterations are required to converge for each time step, which is dependant of load conditions and time step size. Each Newton-Raphson iteration requires creating a new Jacobian matrix of an order of $n \times n$, where n is a number of wake strips. For a 4-bladed propeller, it has about 40 wake strips. To find each row of the Jacobian matrix, the coefficient matrix needs to be inverted once. Therefore, each computational run requires to solve the linear system of equations for $N=n_t \times n_i \times n=24{,}000$ times. In the IMD in-house propeller code, it requires 3 square matrices and they are doublet, source and normalized/Kutta conditioned matrices. The size of these 3 matrices is small for single propellers (about 1,000x1000 each). For a group of propfans or the dynamic positioning (DP) system of a floating production, storage and offloading (FPSO) platform consisting of a group of 6 propellers with 4-6 blades each, the size of these 3 matrices is about 10,000 by 10,000 and 15,000 by 15,000, respectively, and the number of strips increases up to 360. For far wake velocity prediction up to 20-diameter downstream of a propeller, a total of 20 revolutions is needed for a propeller with a pitch-diameter ratio of 1.0. In this case, the number of calls to the matrix solver is about $40 \times 20 \times 5 \times 360=1440{,}000$ times. Increasing speed of the matrix solver is essential for total computing efficiency of the code.

As the code was designed to allocate memory for arrays dynamically, dynamic random access memory (DRAM) requirement becomes an issue. For double precision arrays, three matrices for a DP system require 15,000×15,000×3-*matrix*×8-*byte*=5.4 *GB* of memory. The DRAM requirement at double precision for the shed wake panels is 6-blade×(10×6)-strip×120-*step/rev*×20-*rev*×15,000-*body-panel*×8-*byte*=96 *GB*. This is the minimum requirement for the current panel method to predict the velocities at 20-diameter downstream of a propeller. For an NS solver for the same simulation, the required dynamic access memory would be in a range of tera to peta bytes, which is too high to be economically practical. Velocity prediction is the goal for the performance of propfans and for the momentum impact to risers by DP thrusters. This prediction requires solving a repeatedly created 15,000 by

Manuscript received October 9, 2002. This work was supported by the National Research Council Canada, Institute for Ocean Technology (formally Institute for Marine Dynamics, IMD for short).

Pengfei Liu is with the National Research Council Canada, Institute for Ocean Technology, 1 Kerwin Place and Arctic Avenue, St. John's, NL Canada A1B 3T5, Tel. 709-772-4575; Fax 709-772-2462; E-mail: Pengfei.Liu@nrc-cnrc.gc.ca.

Kun Li is with the Computer Engineering Department, Memorial University of Newfoundland, St. John's, NL A1B 3X5; E-mail: Kun@engr.mun.ca

15,000-*element* matrix for 1440,000 times and it is best to be done in a parallel computing environment.

II. Parallel Implementation of the Bi-CGSTAB Matrix solver

In panel methods, the influence doublet and source coefficient matrices are normally dense and non-symmetric. Larger size matrix requires iteration procedure to gain the speed of the solution. Among many available matrix solver algorithms, the Bi-Conjugate Gradient Stabilized (Bi-CGSTAB) is a suitable alternative. The Conjugate Gradient method to invert a symmetric and positive definite matrix appeared a long time ago (Jannings 1977). Bi-Conjugate gradient method was improved to handle non-symmetric matrix (Voevodin 1983 and Faber and Manteuffel 1984). The Bi-CGSTAB method was developed to avoid the irregular convergence patterns and to add some capability for ill-conditioned matrix (Freund et al. 1991 and Van de Vorst 1992). Barret et al. described implementation of a number of matrix solvers in a great extent and gave a pseudocode of the Bi-CGSTAB (Barret et al. 1994).

In code development for a larger matrix size than the amount of DRAM available, dynamic memory cannot be allocated so the panel method code computation cannot be performed. The solution was to store only one row of the matrix in DRAM and other rows in hard disk. The Bi-CGSTAB matrix solver was then rewritten in a row-by-row reduction form (Liu and Bose 1997). Therefore, the DRAM requirement for matrix size was reduced from n×n to n. However, using row-by-row reduction scheme was about 100 times slower than using DRAM to store the whole matrix elements; this is not computationally efficient for a group of propellers.

For a single propeller with a small number of body and wake panels, a modern PC can normally handle both the memory and speed requirement. On a Dell 866Mhz (OptiPlex GX200) machine it took about an hour for a 3-bladed bare propeller computation but it took about 3 whole weeks for a 4-bladed ducted propeller simulation. To simulate a group of propellers/propfans which requires 50 GB of DRAM and the solution to a 15,000×15,000 matrix 1440,000 times, HPC hardware is essential for an efficient computing calculation.

There have been many studies on Bi-CGSTAB for parallel computers. A package, ScaLAPACK is available online on the Netlib site. This package was designed for heterogeneous computing in Fortran 77. As the MPI 2.0 standard, which is C++ compliant, was not available on the Alpha Tru64 OS, the in-house panel method was written in C. Therefore, a concise MPI C function for the Bi-CGSTAB, working under homogenous environment was a preferred choice. This MPI C function was not available. Some other codes had different limitations such as only allowing 4 processors, etc. We then decided to develop an in-house parallel Bi-CGSTAB function. A guideline in the design of this parallelized Bi-CGSTAB C function is as follows:

- To work under a homogeneous computing environment on any platform using MPI,
- To be able to work in a distributed computing environment such as IBM SP, Alpha and Intel clusters as well as in a memory shared environment such as Alpha Server ES40, SGI derivatives and Intel multi-cpu machines, etc.,
- Easy to use with good scalability, and
- Executable can be run for any number of processors at run time without re-compilation of the source code.

The C version of the row-by-row reduction function was parallelized. Implementing the pseudocode by Barret et al. (1994) with the MPI library functionalities (Gropp et al. 1999), the pseudocode of this parallelized Bi-CGSTAB implementation is described as following:

Compute $r^{(0)} = b - Ax^{(0)}$ *for initial guess* $x^{(0)}$

Choose $\tilde{r}$ (*for example*, $\tilde{r} = r^{(0)}$)

for $i = 1, 2, \ldots$

$\rho_{i-1} = \tilde{r}^T r^{(i-1)}$

if $\rho_{i-1} = 0$ *method failed* (*MPI abort message*)

if $i = 1$

$p_{i-1} = r^{(r-1)}$

else

$\beta_{i-1} = (\rho_{i-1} / \rho_{i-2})(\alpha_{i-1} / \omega_{i-1})$

$p_i = r^{i-1} + \beta_{i-1}(p^{i-1} - \omega_{i-1} v^{(i-1)})$

endif

solve $M\hat{p} = p^{(i)}$

$v^{(i)} = A\hat{p}$, *do this with each process by taskid* *

MPI broadcast $v^{(i)}$

$a_i = p_{i-1} / \tilde{r}^T v^{(i-1)}$

$s = r^{(i-1)} - \alpha_i v^{(i)}$

check s; *if small enough*: *set* $x^{(i)} = x^{(i-1)} + \alpha_i \hat{p}$ *and stop*

solve $M\hat{s} = s$

$t = A\hat{s}$

$\omega_i = t^T s / t^T t$

$x^{(i)} = x^{(i-1)} - \alpha_i \hat{p}^{(i)} + \omega_i \hat{s}$

$r^{(i)} = s - \omega_i t$

check convergence; *continue if necessary*

for continuation it is necessary that $\omega_i != 0$

end

The total number of rows of the matrix was divided into n_m blocks, $n_m = \frac{n + n_p - n\%n_p}{n_p}$. Each block holds n_p rows, where n is the number of rows of the square matrix; n_p is the number of processors at run time. Each processor holds $n_{pr} = (i-1)n_p + taskid_{np} + 1$ rows, where $i=1...n_m$, and $taskid_{np}$ is processor ID starting with 0.

III. Results and Disccussion

This section presents a comparison of computing efficiency in terms of matrix solver algorithms, between the Bi-CGSTAB and the Gauss Elimination methods, the scalability of the parallelized Bi-CGSTAB solver on the IBM SP3 and Alpha ES40 machines, and NAS Benchmark analysis on these two machines.

A. Serial Benchmarks of the Bi-CGSTAB and the Gauss Elimination methods

The C version of the Bi-CGSTAB and the C version of the Gauss Elimination (Press et al. 1992) were run on a HP PC, a Dell PIII Xeon PC (used only one CPU), the IBM SP3 at UNB over the C3 grid, and the recently purchased Alpha ES40 at IMD.

Table 1 shows the elapsed CPU time and machine specifications and Figure 1 shows the comparison via plot.

In serial computation, Bi-CGSTAB gave about 50 times better efficiency than the Gauss Elimination method. The larger the matrix size, the higher the ratio of Bi-CGSTAB to the Gauss Elimination. In this test, only a small matrix size of 4344x4344 was used because of the limitation of the amount of the dynamic memory on the PCs.

Table 2 and Figure 2 show the computing efficiency comparison between the serial code Bi-CGSTAB and Gauss Elimination methods for a matrix size of 9984x9984.

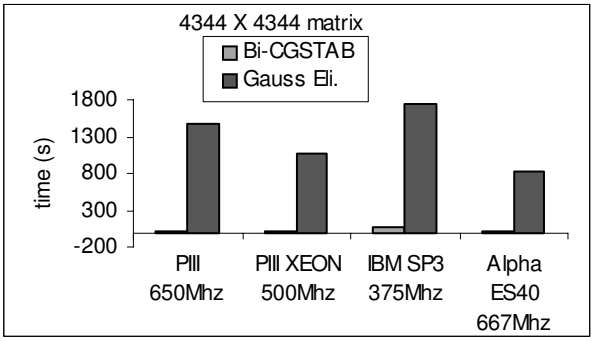

Fig. 1. Comparison of the computing efficiency between the serial code Bi-CGSTAB and Gauss Elimination methods for a matrix size of 4344.

TABLE II
Computing Efficiency Comparison Between The Serial Code Bi-CGSTAB And Gauss Elimination Methods For A Matrix Size Of 9984×9984.

	Bi-CGSTAB (s)	Gauss Eli. (s)	Time Ratio
Dell XEON	121.5	12310.7	101
Alpha ES40	98.7	1498.4	15

TABLE I
Comparison Of The Computing Efficiency Between The Serial Code Bi-CGSTAB And Gauss Elimination Methods For A Matrix Size Of 4344.

	CPU (MHz)	DRAM	Nodes	OS	Bi-CGSTAB	Gauss Eli.	Ratio
PIII	650	256 MB	1	Win 2000	26.02 sec.	1481.48 sec.	57
Dell XEON	2 * 500	1GB/CPU	1	NT 4.0	19.20 sec.	1063.90 sec.	55
IBM SP3	16 * 375	1GB/CPU	4	AIX 4.3.3	57.60 sec.	1748.50 sec.	30
Alpha ES40	4 * 667	4GB/CPU	1	Tru64	19.40 sec.	826.10 sec.	43

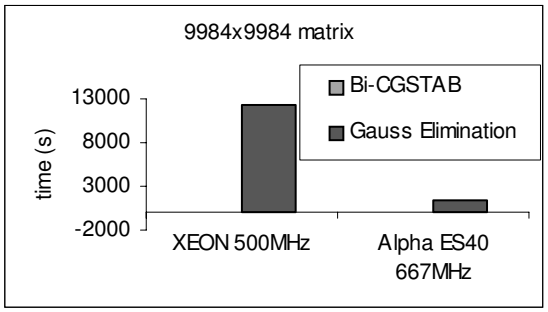

Fig. 2. Computing efficiency comparison between the serial code Bi-CGSTAB and Gauss Elimination methods on a Dell XEON PC and Alpha ES40 Server, for a matrix of 9984 by 9984.

In table II, on a PC with Windows operation system the computing efficiency of the Bi-CGSTAB is much higher than on the Alpha server.

B. Parallel Benchmark using the Bi-CGSTAB

Both the serial and the parallelized Bi-CGSTAB MPI C function were used to run on the IBM SP3 and Alpha ES40 Server. The matrix size was 4344×4344. A comparison is shown in figure 3. A larger matrix size cannot be used because the disk space to hold the matrix elements was limited on the IBM SP3 at UNB.

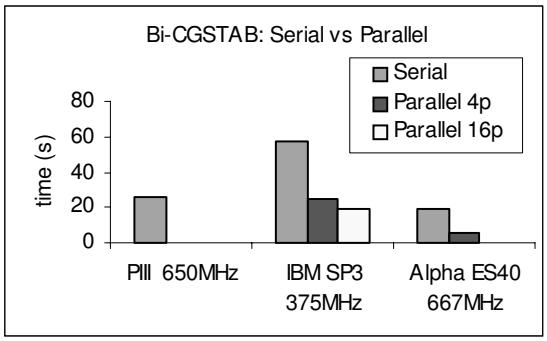

Fig. 3. Comparison of the serial and parallel Bi-CGSTAB solver on three machines.

From figure 3 it can be seen that the serial code computing efficiency is roughly proportional to the CPU clock speed among Intel PIII, IBM and Alpha processors, regardless CISC or RISC processor architecture.

For the parallel Bi-CGSTAB code, the 4-node IBM SP3 machine performed poorly. The communication speed across the 4-nodes was not the best (*300 MBps* bi-directional, 1.2 μsec latency). The 1-node Alpha Server, however, gave a relatively good scalability in this case (1:4 processors vs 20:6 seconds). The new crossbar memory architecture of the Alpha ES40 had a peak throughput of *5.2 GBps* so that it had much less communication drag. In terms of computing efficiency using the serial Bi-CGSTAB code, the IBM SP3 gave about the same computing efficiency as that of an inexpensive Intel desktop with a PIII processor.

For heavy matrix inversion computing using the current Bi-CSGTAB code, high network/channel throughput over clustered computers (distributed memory) or high memory bandwidth over a multi-processor machine (shared memory) is essential for a computing efficiency. PC clusters equipped with fast Ethernet card (*100 MBps*) are deemed not suitable for such above-mentioned computation.

C. NAS Benchmarking the IBM SP3 and Compaq Alpha ES40

The computing power of a parallel system to execute parallel application programs may be measured by NAS parallel benchmarks. A similar test was performed for a 24-node Alpha powered PC cluster (Syms 2001). A detailed procedure that describes the installation, compile and the run the NAS benchmark programs for IBM SP3 and the Compaq Alpha ES40 machine was written recently (Li 2001).

NAS application benchmarks include LU, SP and BT (Bailey et al. 1995). Benchmark LU requires 2^i processors, with $i=1, 2, \ldots N_{pt}$, where $N_{pt}+1$ is the number of available points on a curve. Benchmarks SP and BT require i^2 processors, with $i=1, 2, \ldots N_{pt}$. Running the serial code, which usually takes shorter time than running the parallel code with one processor, creates an extra point. The serial results were obtained with a flag of 0 processor. Figures 4, 5 and 6 show the execution time and Mflop/s of the class *A* version of the benchmarks on IBM SP3 and Alpha ES40 machines.

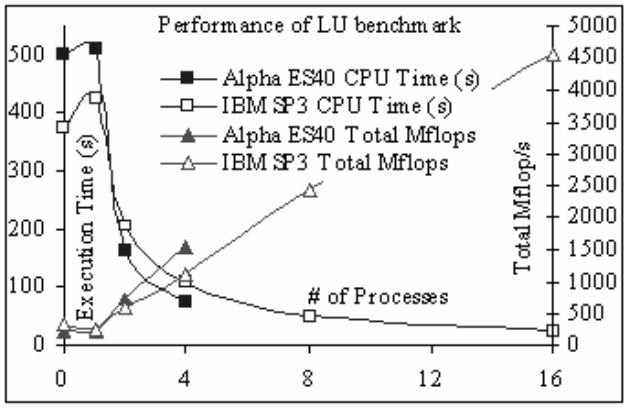

Fig. 4. Performance of LU benchmark.

From figure 4, it can be seen that the scalability of SP3 was good, with 4:16 processors vs 427:107 seconds. The Alpha Server also had a better scalability indication with 1:4 processors vs 512:77 seconds. In one processor case, the communication overhead and multi-user access might be the reason for an extra long computing time of 512 seconds.

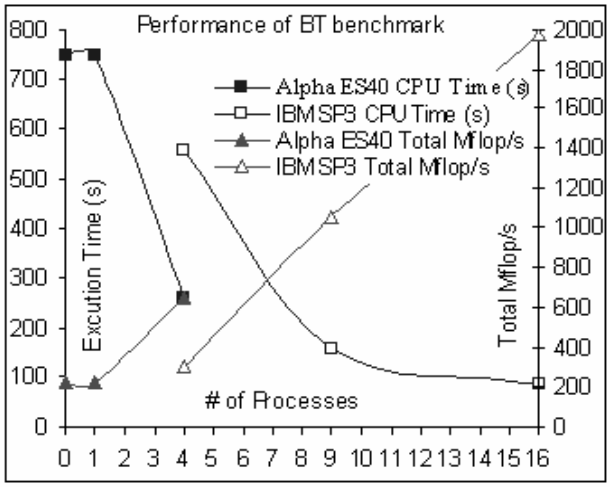

Fig. 5. Performance of BT benchmark.

The BT version could not be executed in serial mode with one processor on the IBM SP3 machine, because the memory was not enough (1.2GB minimum for each execution). The scalability of the SP3 shows a ratio of 4:16 processors with 555:86 seconds.

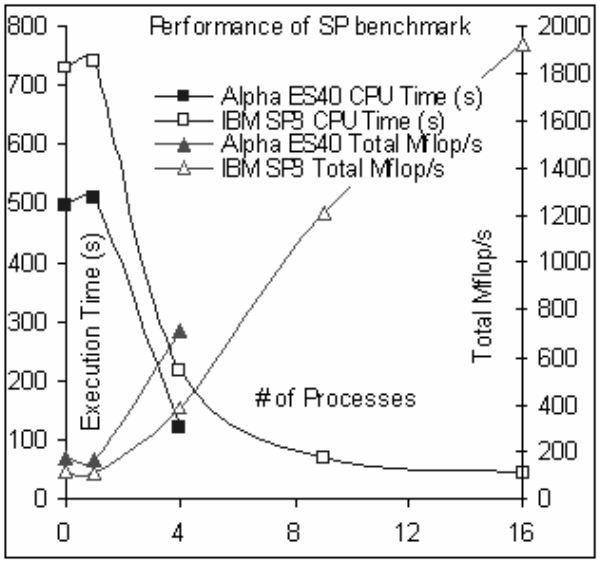

Fig. 6. Performance of SP benchmark.

Linear scalability was also obtained for the SP benchmark on SP3 with 4:16 processors vs 219:44 seconds. From the above three graphs, it can be seen that with the increase of the number of processors, the execution time decrease and the total Mflop/s increase. The same observation as Syms (2001) was examined, of which one processor in parallel mode takes more time to run than the serial mode (communication overhead). The computing efficiency of the IBM SP3 and Alpha ES40 in terms of Mflop/s/process rating is shown in Figure 7.

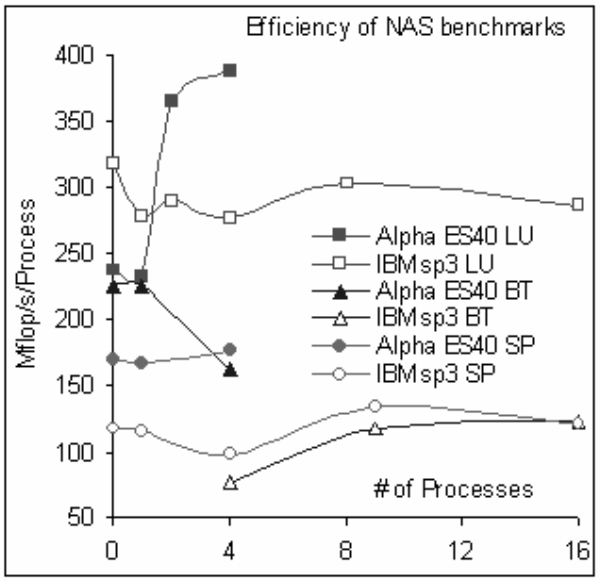

Fig. 7. Efficiency of NAS benchmarks.

The Alpha server outperformed SP3 substantially within its range of number of processors. The computing efficiency for both machines showed a relatively smooth, straight curve.

IV. Conclusions

Parallel programming with the Bi-CGSTAB was done. The serial versions of the solver code were tested on a Windows PC and the two multi-node UNIX machines. The parallel version of the solver was then executed on both the SP3 and the ES40. On the Windows PC with the serial code, for a matrix size of $10{,}000 \times 10{,}000$ the CPU time taken for the Gauss elimination solver was about 50 times for the Bi-CGSTAB solver. The serial version of the Bi-Conjugate Gradient Stabilized matrix inverter took substantially less time than Gauss Elimination, regardless CISC or RISC CPUs and Windows or UNIX platforms, which means that proper algorithms are essential for computing efficiency and in cases they may save more execution time than a powerful HPC hardware system.

Scalability was also obtained from IBM SP3 and Alpha ES40 by using both the NAS software and the parallelized matrix solver. A parallel benchmark application program by NASA (Advanced Supercomputing Division) was used to obtain some benchmarks on these two machines, i.e., a 16-processor IBM SP3 at UNB and a 4-CPU Alpha ES40 that was acquired recently at IMD. The Alpha ES40 showed a rough trend because it has only 4 nodes; at least a 16-processor configuration is required for a reasonable analysis. For a small matrix with a size of 4344×4344, the scalability of the Bi-CGSTAB matrix solver code was not as good as that of the NAS Parallel Benchmarks on the IBM SP3 machine. However, the scalability of the matrix solver was much higher on the Alpha ES40 machine because of its high-speed crossbar memory architecture. Higher performance and scalability analysis also require a parallel computing system to have at least 16 processors. For intensive matrix inversion using current Bi-CGSTAB code, fast I/O is essential for the integrated computing performance of a parallel system.

Acknowledgement

The authors thank to the National Research Council Canada for its support. They are grateful to Dr. Eric Aubanel of the Advanced Computational Research Laboratory (ACRL) at UNB and Mr. Gilbert Wong at IMD for their assistance. They are also thankful to the C3 and the ACRL for their availing the computing resources.

References

[1] J.L. Hess, “Calculation of potential flow about arbitrary three-dimensional lifting bodies”, Report of Douglas Aircraft Company, McDonnell Douglas Corporation, 3855 Lakewood Blvd., Long Beach, California, MDC J5679/01, 90846, pp. 1-160, 1972.

[2] J.L. Hess and W.O. Valarezo, "Calculation of steady flow about propellers by means of surface panel method", Proceedings, Research and Technology Douglas Aircraft Company, Long Beach, CA, 1985, pp. 1-8.

[3] A. Jennings, *Matrix computation for engineers and scientists*, John Wiley & Sons, 1977

[4] V. Voevodin, "The problem of non-self-adjoint generalization of the conjugate gradient method is closed," *U.S.S.R. Comput. Maths. and Math. Phys*., vol. 23, pp. 143-144, 1983.

[5] V. Faber and T. Manteuffel, "Necessary and sufficient conditions for the existence of a conjugate gradient method," *SIAM J. Numer. Anal.*, vol. 21, pp. 315-339, 1984.

[6] R. Freund, G. Golub and N. Nachtigal, "Iterative solution of linear systems," *Acta Numerica*, 1991, pp. 50-100.

[7] H. van der vorst, "Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetrical linear systems", *SIAM J. Sci. Statist. Comput.*, vol. 13, pp. 631-644, 1992.

[8] R. Barret, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. van der Verst, (1995, November 20). *Templates for the solution of linear systems: Building blocks for iterative methods (2nd ed.)*, [electronic file in html and postscript format]. Available: http://www.netlib.org/linalg/html_templates/Templates.html

[9] P. Liu and N. Bose, Propulsive performance from oscillating propulsors with spanwise flexibility, *Proc. R. Soc. Lond. Vol.* 453(1963), pp. 1763-1770, 1997.

[10] W. Gropp, E. Lusk and A. Skjellum A., *Using MPI, Portable Parallel Programming with the Message-Passing Interface*, 2nd ed, MIT Press, 1999.

[11] W. Press, S. Teukolsky, W. Vetterling and B. Flannery, *Numerical recipes in C*, 2nd ed., Cambridge University Press, 1992.

[12] G. Syms, Parallel Performance of Aerodynamic CFD Codes, *Canadian Aeronautics and Space Journal*, vol. 1(March), pp. 25-31, 2001.

[13] K. Li, "Parallel programming/computing with IBM SP3 and Compaq Alpha ES40", National Research Council Canada, Institute for Marine Dynamics, Laboratory Memorandum, LM 2001-09, 2001

[14] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo and M. Yarrow, "The NAS Parallel Benchmarks 2.0", Report NAS-95-020, 1995.

Pengfei Liu was born in Rongcheng city of Shandong province, China, on April 18, 1955. He obtained a bachelor degree in naval architecture and ocean engineering at Wuhan University of Technology in 1982, a master's and PhD degree in 1991 and 1996, respectively in naval architecture and ocean engineering at Memorial University of Newfoundland (MUN), Canada.

After high school in 1972, he worked in a shipyard for a few months then served for the Chinese Army for three years as a MORSE CODE OPERATOR. In 1976 he returned to the shipyard and completed one-year training as a SHIP DESIGN TECHNOLOGIST. After graduated from Wuhan University of Technology (formally Wuhan Institute for Water Transportation Engineering), he worked as a NAVAL ARCHITECT and then SENIOR OFFICER at Chinese Central State Government for 6 years. Pengfei entered Canada in 1988. After his master's degree in 1991, he taught at a college as an instructor for 8 years. He joined the National Research Council Canada, Institute for Ocean Technology as a Research Officer in 1999. Three of his and coauthored recent publications are:

[1] P. Liu, "Propulsive performance of a twin-rectangular-foil propulsor in a counter-phase oscillation", *Journal of Ship Research*, to be published.

[2] P. Liu, "Design and Implementation for 3D unsteady CFD Data Visualization Using Object-Oriented MFC with OpenGL", *International CFD Journal of Japan*, vol. 11, no. 3, p335-345, October 2002.

[3] P. Liu and K. Li, "Performance Analysis of a BiCGSTAB Solver for Multiple Marine Propeller Simulation with Several MPI Libraries and Platforms," in *High Performance Scientific and Engineering Computing, Hardware/Software Support*, L. Yang and Y. Pan, Ed., Boston/Dordrecht/London: Kluwer Academic Publishers, 2004, pp. 63-77.

His research interests cover CFD, numerical and experimental hydrodynamics and marine propulsion, computer graphics and visualization and high performance computing algorithms and application to engineering.

Dr. Liu is also a professional engineer, a member of the Society of Naval Architects and Marine Engineers (SNAME), a member of the board of directors of CFD Society of Canada and, a member and secretary of the 24th ITTC Azimuth Podded Propulsion Technical Committee.