Multicore Parallelization of Quad-Precision Eigensolvers

Yasuyo HATANO
Shigeyoshi YAMAMOTO

Abstract

Multicore parallelization is detailed, for a program named `MIPOWQ`, that determines a single eigenvalue and the corresponding eigenvector of a dense real symmetric matrix on the basis of Fortran quadruple precision (`REAL*16`) arithmetic. This program was developed by combining the inverse power method and the modified Cholesky decomposition, and it gives high parallel performance in the OpenMP environment. In particular, it achieves 7.79 times speedup in 8-thread parallel execution for the Frank matrix of dimension 10000. This enhanced performance is due to the simplicity of the inverse power method and the full use of processing units for quadruple precision arithmetic. A further program is provided, known as `JENNFQ`, which determines a subset of eigenvalues and eigenvectors by the Jennings simultaneous iteration method. These two programs would present good examples of parallelization by OpenMP although the adopted algorithms are rather classical.

Keywords: `REAL*16` • Eigenvalue problem • Inverse power method • Modified Cholesky decomposition • OpenMP • IEEE754

Program Summary

Program Title: `MIPOWQ`

Programming language: Fortran 2003

Licensing provisions: The MIT License

Computers: Any machine with a Fortran compiler supporting quadruple precision arithmetic (tested for Intel Fortran, Fujitsu Fortran, and gfortran)

Operating Systems: Linux (tested on CentOS 7.5, RedHat 6.5, and Fedora 24)

High speed storage required: Approximately $N^2 + 2N$ quadruple precision words, where N is the

matrix dimension.

No. of bits in a word: 32

No. of lines in combined program and test deck: 1942

No. of bytes in distributed program, including test data, etc.: 18973

Distribution format: tar.gz

Nature of physical problem: Eigenvalue problem

Method of solution: The inverse power method. The modified Cholesky decomposition.

Typical running time: The eigenvalue of smallest magnitude and the corresponding eigenvector of the 10000×10000 Frank matrix can be obtained within 620 sec wall-clock time on the Intel Xeon processor (3.7 GHz) via 8-thread parallel execution.

1. Introduction

Eigenvalue problems appear often in scientific computations. Accuracy greater than double precision is needed in some computations.[1–6] Programs tackling the eigenvalue problem exist as modules in almost all mathematical program libraries, but it is hard to find any that are capable of working in quadruple precision arithmetic (QP, IEEE754-2008 [7]). In particular, Intel Math Kernel Library (MKL[8]) does not release a quadruple precision version.

We address this need by using Fortran quadruple precision (`REAL*16`). If there is a source program which uses double precision, it is then easy to rewrite it to the quadruple precision version. However, quadruple precision arithmetic takes an order of magnitude greater computational time than double precision arithmetic.

To avoid this difficulty, double-double (DD) arithmetic was proposed by Bailey.[9] On the basis of DD arithmetic, a basic matrix operation program library (QPBLAS) and eigenvalue problem package (QPEigen) have been developed by Yamada et al.[10] They reported that computational times were reduced by a factor of 10 compared to the IEEE754 quadruple precision version. A similar DD program library (ASLQUAD[11,12]) has been provided by NEC Co. In addition, a DD and quad library named lis has been provided by Nishida's group.[13]

There are still cases in which the number of digits of the exponent is inadequate, however. The DD arithmetic does not work in such situations. Instead, we simply use Fortran `REAL*16`, which ensures the portability of the program and facilitates maintenance.

In this paper we present two programs. These are called `MIPOWQ` and `JENNFQ`. The `MIPOWQ` program is a new development. It determines an eigenvalue and the corresponding eigenvector of

a dense real symmetric matrix. JENNFQ has been developed by converting the JENNFD program contained in NUMPAC (Nagoya University Numerical Package) library[14,15] to quadruple precision, from double precision. (Y.H. is one of the authors of NUMPAC.) The JENNFQ program determines a subset of eigenvalues and eigenvectors. We have successfully gained high performance by multithreading. The advantages of these two programs will be made clear by comparing them with other eigenvalue programs based on other algorithms.

To date, we have used existing subroutine libraries of double/quadruple precision arithmetic contained in the open-source NUMPAC program library[14,15] to solve problems in physics.[1,2] In particular, we developed a program named SCELTO_gauss for single-center expansion of the $^2\Sigma_g$ wavefunction of the hydrogen molecular ion (H_2^+) using Laguerre-type orbitals (LTOs). We obtained wavefunctions with an error in the energy of less than 10^{-6} a.u.[16] by expanding the wavefunction with LTOs up to the quantum numbers n = 203 and l = 202. The Hamiltonian is a dense matrix of size 10404 × 10404. The proportion of elements having an absolute value greater than 10^{-30} is 92%. The diagonalization stage took 10.5 hours (38146 sec) on an Intel Core i7-3970X (3.50 GHz) processor. To reach convergence to the exact solution, quadruple precision arithmetic was necessary.

Much CPU time was taken in generating the Hamiltonian of size 10404 × 10404; this took 189 hours (682394 sec) in serial mode. We optimized the orbital exponent (ζ_l) by varying it about 10 times for each quantum number l while increasing l from 0 to (n–1). In this way we confirmed the convergence of the total energy to the exact value. Since we diagonalize the Hamiltonian many times during a solution, the eigenvalue problem took 50% of the entire computation. The SCELTO_gauss program uses HOBSVQ in NUMPAC as an eigensolver. The HOBSVQ subroutine is used for determining a subset of eigenvalues/eigenvectors by means of the Householder transformation and the bisection method. If the eigenvalue problem can be solved more efficiently, more LTOs can be used and more accurate results obtained. This is why we decided to develop more efficient eigensolvers. For parallelization, we use OpenMP.[17,18]

We have developed eigensolvers, based on the following points.

(1) As the number of significant digits in double precision (16 digits) proved insufficient for the quantum number n>20 for the H_2^+ calculations,[1,2] we require accuracy of significant digits over 20.

(2) The eigenvalue smallest in magnitude is determined for a dense real symmetric matrix.

(3) Multicore parallelization is to be implemented for speedup.

At the beginning of program development, we adopted the QD library[9] and achieved speedup by a factor of 3 compared to the serial `HOBSVQ`. (In QD, two double precision numbers are combined and the mantissa has $2 \times 2 = 4$ precision; the exponent is in the same range as a double precision number ($10^{\pm 308}$)). As the number of digits of the exponent is inadequate for calculation of the Hamiltonian, we decided to write a program without resorting to QD library. Regarding points (2) and (3) above, Intel MKL contains excellent eigensolvers such as `dsyevx`, `dsyevd`, but quadruple precision versions are not available at present. These reasons are why we developed a parallel eigensolver of quadruple precision (`REAL*16`).

The eigenproblem in this case is to obtain a vector $\mathbf{x}$ which satisfies $A\mathbf{x} = \lambda\mathbf{x}$ where A is a dense real symmetric matrix. Two methods are:

(1) `MIPOWQ`: The inverse power method.[19,20]

Using modified Cholesky decomposition (LDL^{T} factorization[20]), followed by repeated forward substitutions and backward substitutions, the eigenvalue smallest in magnitude, and the corresponding eigenvector, are obtained in a set. An arbitrary eigenvalue can be determined by subtracting its approximate value from the diagonal elements of the matrix.

(2) `JENNFQ`: The Jennings simultaneous iteration method.[21,22]

A limited subset of eigenvalues, including the smallest (or largest) in magnitude, and their corresponding eigenvectors, are determined. The subroutine `JENNFQ` was made by converting the subroutine `JENNFD`[14,15] in NUMPAC to quadruple precision, from double precision, and by parallelizing it by inserting an OpenMP directive.

For diagonalizing large sparse real symmetric matrices, other methods such as the Davidson method[23] have been developed especially for diagonal dominant cases. The Davidson method is applied to large configuration interaction (CI) calculations. Nowadays eigenvalue problems of dimension over one billion are solved in the area of molecular physics and chemistry.[24] By Sleijpen and van der Vorst,[25] the Davidson method was extended to more general cases and the convergence rate was improved. Their method is called the Jacobi-Davidson method.

2. Algorithm

The power method used in `MIPOWQ` and the simultaneous iteration method used in `JENNFQ` are both based on algorithms which give the eigenvalue largest in magnitude. To determine the

eigenvalue of least magnitude, an eigenvalue problem of the inverse matrix (A^{-1}) is to be solved, i.e., $A^{-1}\,\mathbf{x} = (1/\lambda)\,\mathbf{x}$. Then the inverse of the maximum absolute eigenvalue of A^{-1} gives an eigenvalue of A. An arbitrary eigenvalue can be determined by subtracting its approximate value (σ) from the diagonal elements of A in advance.

The detailed procedure is as follows. At first, setting an approximate value (σ), we obtain ρ_1 (the largest eigenvalue in magnitude) by solving the eigenvalue problem ($(A - \sigma I)^{-1}\,\mathbf{v} = \rho_1\,\mathbf{v}$). From the equation $A\,\mathbf{v} = (1/\rho_1 + \sigma)\,\mathbf{v}$, we obtain $(1/\rho_1 + \sigma)$ as the eigenvalue closest to σ, and $\mathbf{v}$ as the corresponding eigenvector.

(1) `MIPOWQ`

The power method associated with a Rayleigh quotient is employed to accelerate convergence. The matrix A is decomposed by the modified Cholesky decomposition. Then the inverse power method is applied; this is followed by the solution method for simultaneous linear equations, which is associated with forward substitutions and backward substitutions. Convergence is judged by fluctuation of the eigenvectors.

The numbers of arithmetic operations in `MIPOWQ` are $N^3/3 + N^2 - 5N/6$ for the modified Cholesky decomposition, and $2N^2 - N$ for the forward/backward substitutions. Thus the total number of operations becomes $N^3/3 + N^2/2 + \text{cycle} \times 2N^2 + \mathcal{O}(N)$, where 'cycle' denotes the number of iterations before convergence is attained.

In total, three OpenMP directives are inserted in the code. One is at the modified Cholesky decomposition, and the others are at the forward/backward substitutions.

`MIPOWQ` demands memory of minimum size, i.e., matrix A, eigenvector $\mathbf{v}$, and a 1-dimensional array of size N. The total memory required is $16(N^2 + 2N)$ bytes.

The procedure is described in detail in **Appendix B**.

(2) `JENNFQ`

Details of the algorithm are given in Ref.**21,22**. An OpenMP directive is inserted in the modified Cholesky decomposition portion. The large arrays required are the matrix A, eigenvector $\mathbf{v}$, and a 1D working area of size $3N$.

3. Performance

3.1 Test matrices

The Frank matrix[26] of dimension N is used for test calculations. Its elements are defined as

$a_{i,j} = N + 1 - \mathrm{MAX}(i, j)$. Its exact eigenvalues are given by the following formula.

$$\lambda_i = \frac{1}{4}\left[\sin^2\left(\frac{\pi}{2}\cdot\frac{2i-1}{2N+1}\right)\right]^{-1} \tag{1}$$

The error is represented by $\mathrm{err}(\lambda_N') = |\lambda_N' - \lambda_N|$, where λ_N' is the calculated value of the eigenvalue. We tested the cases N = 1000, 2000, and 10000. We assigned a value of 0.25 to σ (an approximate value for λ_N.

3.2 Computational environment

The host computer used for the benchmark test is a w2145. As detailed in **Table C.6**, it has a CPU unit comprising an Intel Xeon W-2145 (clock-rate 3.70 GHz) with 3rd cache (last level cache, LLC) of 11 MB, 8 cores, HTT (hyper-threading technology),[18] and TBT (turbo boost technology).[27,28] The Intel Fortran compiler is used, and the following compiler options are specified.

```
-O3 -xHost -mcmodel=small -qopenmp -ip
```

We bind OpenMP threads and hardware threads using the following environment variable, which is available under the Intel Fortran environment.

```
KMP_AFFINITY=verbose,granularity=fine,scatter
```

Details of the binding method will be described in **Subsection 3.5**.

3.3 Accuracy

In this subsection we discuss the accuracy[19,20] of the computational results. The errors ($\mathrm{err}(\lambda_N')$) in the eigenvalues, and errors ($\mathrm{err}(\mathbf{x}_N')$) in the eigenvectors are set out in **Table 1**.

The convergence criterion (EPS) is set at 10^{-25}. For MIPOWQ and JENNFQ, convergence is judged by the fluctuation of the eigenvector. For HOBSVQ, a tight convergence criterion (EPS = 10^{-34}) on the eigenvalue is adopted.

Table 1. Errors in the eigenvalues and eigenvectors of the Frank matrices diagonalized by MIPOWQ / JENNFQ. (The degree of parallelism is 1.)

	MIPOWQ (EPS = 10^{-25})			JENNFQ (EPS = 10^{-25})		
N	Iter.*	$\mathrm{err}(\lambda_N')$	$\mathrm{err}(\mathbf{x}_N')$	Iter.*	$\mathrm{err}(\lambda_N')$	$\mathrm{err}(\mathbf{x}_N')$
1000	30	1.54E-31	2.00E-30	29	1.54E-31	2.58E-30
2000	38	1.66E-31	1.99E-29	26	1.66E-31	1.10E-29
10000	37	3.84E-30	2.37E-28	29	3.84E-30	3.24E-28

* Number of iterations until convergence.

Below, the exact eigenvalues for N = 1000, 2000, and 10000 are listed up to 35 digits.

λ_{1000} = 0.25000 06162 34899 77511 48137 94229 54161

λ_{2000} = 0.25000 01541 35554 74188 04357 05940 25200

λ_{10000} = 0.25000 00061 67886 04811 39811 77862 77643

The results of MIPOWQ and HOBSVQ (Householder-Bisection) are as follows.

λ_{1000}' = 0.25000 06162 34899 77511 48137 94229 69545

λ_{2000}' = 0.25000 01541 35554 74188 04357 05940 08631

λ_{10000}' = 0.25000 00061 67886 04811 39811 77858 93636

λ_{1000}' = 0.25000 06162 34899 77511 48137 94229 57252

λ_{2000}' = 0.25000 01541 35554 74188 04357 05940 16898

λ_{10000}' = 0.25000 00061 67886 04811 39811 77862 73877

The calculated eigenvalues of MIPOWQ are identical to the exact values up to 29 digits, and the errors are less than 10^{-30}. Compared to the rounding unit (10^{-34}) of quadruple precision defined by IEEE754,[7] the loss of precision is 5 digits. The eigenvector has an accuracy of more than 28 decimal digits. The number of iterations in JENNFQ is less than that with MIPOWQ, indicating that the vector acceleration in JENNFQ works effectively.

3.4 Speed and parallel performance

Wall-clock times (elapsed times) taken to determine an eigenvalue and the corresponding eigenvector of the Frank matrices are set out in **Table 2** (for MIPOWQ) and **Table 3** (for JENNFQ).

We term the ratio of the wall-clock time of a 1-thread job to that of a multithread job the *performance ratio* (PR). In the case of MIPOWQ (N = 10000), the PRs are 1.99 for 2-thread, 3.78 for 4-thread, and 7.45 for 8-thread. In the case of JENNFQ they are 1.95 for 2-thread, 3.57 for 4-thread, and 6.52 for 8-thread. (Only a single eigenpair is evaluated even for JENNFQ.) Thus the PR increases almost linearly with the number of threads. The PR of MIPOWQ is greater than that of JENNFQ. The PR values of MIPOWQ are shown as solid lines in **Fig. 1**. Dotted lines in **Fig. 1** show PRs with TBT correction. The TBT correction will be explained in the next subsection.

Table 2. Wall-clock time (sec) for diagonalizing the Frank matrices by MIPOWQ on w2145.

	Number of threads							
N	1	2	3	4	5	6	7	8
1000	5.74	3.05	2.23	1.77	1.50	1.30	1.18	1.07
2000	41.55	21.30	15.02	11.46	9.37	7.94	6.92	6.16
10000	4610.87	2321.39	1621.15	1219.89	979.68	819.20	704.66	619.31

Table 3. Wall-clock time (sec) for diagonalizing the Frank matrices by JENNFQ on w2145.

	Number of threads							
N	1	2	3	4	5	6	7	8
1000	5.52	3.27	2.60	2.21	1.97	1.82	1.71	1.61
2000	40.39	22.08	16.67	13.49	11.57	10.30	9.39	8.70
10000	4620.30	2371.41	1687.53	1295.48	1060.05	904.74	792.65	708.80

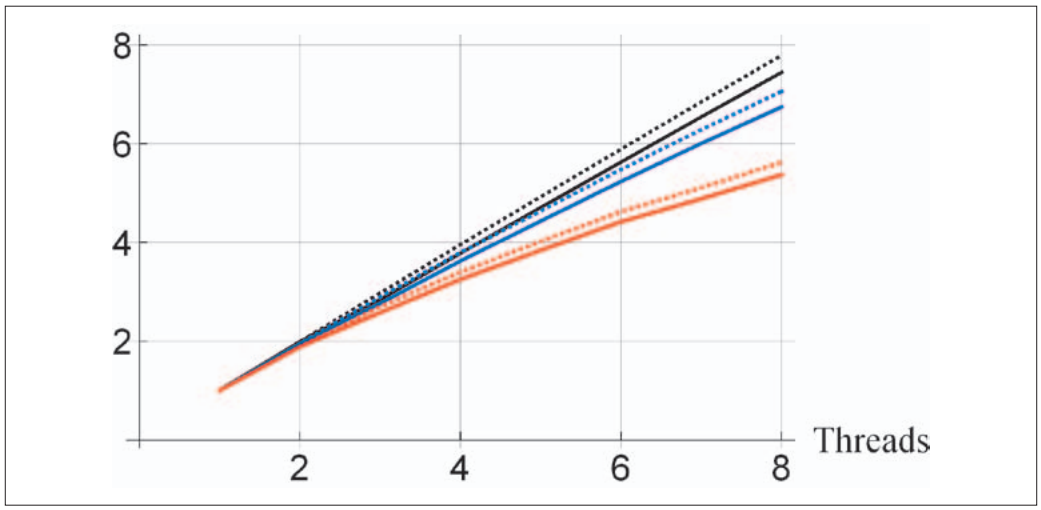

Fig. 1. Parallel performance of MIPOWQ on w2145 for diagonalizing the Frank matrices. Red, blue, and black lines (in the order from the lowermost to the uppermost) refer to N = 1000, 2000, and 10000, respectively. Solid/dotted lines illustrate performance ratios without/with TBT correction.

Here we compare wall-clock times between HOBSVQ (serial version) in NUMPAC used in Ref.**1,2**, and MIPOWQ. Wall-clock times for HOBSVQ for N = 1000, 2000, and 10000 are 26.73 sec, 226.33 sec, and 29395.27 sec. MIPOWQ is 6.4 times faster than HOBSVQ for 1-thread, and 47.5 times faster for 8-thread in the case N = 10000.

The number of arithmetic operations for the Householder transformation[19,20] in HOBSVQ is of the order of $(4/3)N^3$ (See Ref.**20**). The operations in the modified Cholesky decomposition in MIPOWQ

number about $(1/3)N^3$. Thus for large N, the number of operations will be about 4 times greater in HOBSVQ than in MIPOWQ. HOBSVQ takes more wall-clock time than expected from the number of arithmetic operations alone. This is because there is an array reference with a long stride in the code of HOBSVQ.

Here we compare the wall-clock time of double precision program MIPOWD derived from MIPOWQ and that of MIPOWQ. For N = 10000, they are 96 sec and 4611 sec, respectively. Therefore the ratio of the cost between double precision and quadruple precision is 48. This suggests that MIPOWQ is CPU-bound, that is, arithmetic operations in the cores consume more time than data transfer between the cores and the main memory.

For N = 10000, the performance of MIPOWQ is 550 MFLOPS. Dongarra's group[29] obtained high performance for Cholesky decomposition in a similar computational environment by exploiting the tiled (or blocked) algorithms,[29] which are optimal for the modern cache architecture. We provisionally implemented the blocked version of the modified Cholesky decomposition (dsytrs) and forward/backward substitutions (dsytrf) in LAPACK[30] with MIPOWQ and evaluated its performance. The speedup was not so great (33% for 1-thread, 10% for 8-thread). For a higher degree of parallelism, the PR of the modified program is decreased. This is because the quadruple precision arithmetic in the present program is CPU-bound. We provide the programs coded without blocking because we prefer higher parallel performance and the simplicity of the code.

We conclude that, for evaluating a single eigenpair, MIPOWQ (the inverse power method) is superior to the Householder-Bisection method. To determine a limited subset of eigenpairs, JENNFQ (the simultaneous iteration method) is advantageous. In the case of MIPOWQ, for large N, the modified Cholesky decomposition becomes the dominant step in the computation, and the number of operations is approximately $(1/3)N^3$.

The wall-clock time for diagonalizing the H_2^+ Hamiltonian described in our **Introduction** has been reduced from 38146 sec (serial HOBSVQ) to 7374 sec by 1-thread execution (MIPOWQ), and to 1350 sec by 6-thread.

3.5 Scalability

We now consider the scalability of the MIPOWQ program by exploiting TBT.[27,28] TBT dynamically varies the clock-rates of cores. When only a single core is working, TBT increases the clock-rate over the base frequency, and when multiple cores are working it slows their clock-rates to reduce heat generation. TBT is available on the Intel Xeon W-2145 of the computer systems used (w2145).

The measured parallel performance (solid lines in **Fig. 1**) shows deceleration of growth as

the number of threads increases. This is not easy to interpret, because TBT contributes to this deceleration. Consequently, we estimate the parallel performance assuming that the clock-rate is constant even for multithread execution. We call this compensation for parallel performance *TBT correction*. TBT correction will now be discussed.

Under the Intel Fortran environment, a binding method between OpenMP threads and hardware threads can be specified by the environment variable `KMP_AFFINITY`. If "`scatter`" is specified for this variable, the threads are distributed as evenly as possible across the entire system. For example, in the 2-thread parallel mode, the first thread is bound to the first core of the first CPU unit, and the second thread is bound to the first core of the second CPU unit. The frequency[31] of the Intel Xeon W-2145 is listed in **Table 4**; the cores operate at 4.5 GHz until there are 2 threads, and at 4.3 GHz for 3 threads or more. Assuming that the performance of each core is proportional to the frequency, we compensate for the ostensible deceleration of performance by multiplying by 1.0 for 1–2 thread, (4.5/4.3) for 3–8 threads. **Table 5** gives the PRs without/with TBT correction. The dotted lines in **Fig. 1** show the corrected PRs. The corrected PR of `MIPOWQ` reaches 7.79 in the 8-thread case, suggesting high scalability of `MIPOWQ`.

Table 4. Turbo frequency (GHz) of the Intel Xeon W-2145.

	Number of working cores		
	1	2	3–8
Turbo frequency	4.5	4.5	4.3

Cited from Ref.**31**.

Table 5. Performance ratio of `MIPOWQ` (N = 10000) corrected by taking TBT into account.

	TBT	Number of threads							
Hostname	correction	1	2	3	4	5	6	7	8
w2145	Without	1	1.99	2.84	3.78	4.71	5.63	6.54	7.45
w2145	With	1	1.99	2.98	3.96	4.93	5.89	6.85	7.79
CX2	Needless	1	1.99	2.97	3.95	4.93	5.88	6.85	7.79

We have measured the performance on a different computer system (CX2). The specifications of this system are listed in **Table C.6** of **Appendix C**. The Intel Xeon E5-2697V2 is mounted on the CX2, with the Intel Xeon Phi 3120P co-processor attached. Wall-clock times for `MIPOWQ` are listed in **Table D.7** in **Appendix D**, and plotted in **Fig. D.2**. The wall-clock times for `JENNFQ` on CX2 are given in **Table D.8**. For `MIPOWQ` the PR for the 8-thread case and N = 10000 is 7.79. Amdahl's law[18,32] is expressed by the following formula:

$$\mathrm{PR} = \frac{1}{1-\alpha+\dfrac{\alpha}{p}} \tag{2}$$

By substituting 7.79 for PR and 8 for p (the degree of parallelism) into Eq.(2), the value of α (the fraction of the parallelized code) is calculated to be 0.9961. This large value of α implies high scalability of MIPOWQ. Since the Phi 3120P co-processor does not have TBT, its PRs can be compared directly with those for w2145. The PR value of 7.79 for CX2 is very close to the PR value (7.79) with TBT correction in the 8-thread case on w2145. This strongly suggests that the TBT correction we have proposed is appropriate and that MIPOWQ has high scalability.

Here we give other results of another machine (cluster) of 32 cores in total. The specifications of cluster is given in **Table C.6** of **Appendix C** and the turbo frequencies in **Table E.9** of **Appendix E**. The wall-clock times of MIPOWQ and JENNFQ are set out in **Tables E.10** and **E.11**. PRs are plotted against number of threads in **Fig. E.3**. High scalability is shown also on cluster. Since the actual number of cores is 32, the PR descends sharply at 33-thread as shown in **Fig. E.3**. The PR for 64 threads is only slightly better than that of 32 threads, which can be attributed to HTT. It is recommended to run the program with the number of threads less than or equal to that of cores equipped actually.

3.6 Portability

We have tested the portability of the programs by executing them under the following environments: Intel Fortran version 19.0.1, Fujitsu Fortran version 1.2.0, and gfortran version 7.3.1 (Intel Core i7-3970X, Linux Fedora 24).

4. Concluding remarks

We have developed two eigensolvers having quadruple precision (Fortran `REAL*16`) for dense real symmetric matrices, known as MIPOWQ and JENNFQ. MIPOWQ specifically determines a single eigenvalue (the smallest in magnitude), and the corresponding eigenvector is found by combining the inverse power method and the modified Cholesky decomposition. MIPOWQ exhibits high multithread parallelism in the OpenMP environment. The calculated eigenvalue has an accuracy of more than 28 decimal digits. For determining single eigenpairs, the inverse power method is superior to the Householder-Bisection method. JENNFQ determines a limited subset of eigenpairs, and has been developed by parallelizing the existing double precision JENNFD program in NUMPAC. Because

of the scarcity of quad-precision eigensolvers and the high parallel performance of the programs, we provide MIPOWQ and JENNFQ. The source program of MIPOWQ is listed in **Appendix F**. The full set of the source programs and the output files is available upon request to the corresponding author.

Author information

Professor Emeritus, Dr. Yasuyo HATANO

Institute for Advanced Studies in Artificial Intelligence (IASAI), Chukyo University, Toyota 470-0393, Japan.

ORCID: 0000-0002-7914-8846

Corresponding author: E-mail address: hatano@sist.chukyo-u.ac.jp

Professor, Dr. Shigeyoshi YAMAMOTO

School of International Liberal Studies, Chukyo University, 101-2 Yagoto Honmachi, Showa-ku, Nagoya 466-8666, Japan

Institute for Advanced Studies in Artificial Intelligence (IASAI), Chukyo University, Toyota 470-0393, Japan.

ORCID: 0000-0003-3780-534X

Acknowledgements

We used computers at IASAI of Chukyo University, and the computer system (CX2) at the Information Technology Center of Nagoya University. We thank Professor Emeritus Hiroshi Tatewaki of Nagoya-City University, Professor Emeritus Takemitsu Hasegawa of Fukui University, and Professor Hiroshi Sugiura of Nanzan University for critical reading of the manuscript. We are grateful to Professor Akiumi Hasegawa at Chukyo University for making the computers (cluster in particular) at IASAI available to us.

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Appendix A. User's manual for MIPOWQ and JENNFQ

```
call MIPOWQ(A, KA, VV, N, EIG, EPS, ITER, MITER, G, ILL)
call JENNFQ(A, KA, N, NE, NV, VV, EIG, C, W, EPS, ITER, ILL)
```

Name	Type | Input&Output | Contents
A	real*16 | Input&Output | A 2D array. A dense real symmetric matrix. σ should be subtracted from the diagonal elements in advance. For MIPOWQ, components of the modified Cholesky decomposition are to be stored in the upper triangle.
KA	integer | Input | The adjustable dimension of A/VV.
VV	real*16 | Input&Output | A 1D/2D array for MIPOWQ/JENNFQ. Input initial values of eigenvectors.
N	integer | Input | The dimension of A.
NE	integer | Input | Input the number of eigenvalues to be evaluated by |NE|. NE > 0 (NE < 0) indicates that eigenvalues are to be stored in descending (ascending) order of the absolute values.
NV	integer | Input | The number of initial vectors.
EIG	real*16 | Output | An eigenvalue (eigenvalues) is (are) to be stored. For JENNFQ, a 1D array of size |NE|.
C	real*16 | Work | A 1D working array of size NV × NV.
W	real*16 | Work | A 1D working array of size 3N.
EPS	real*16 | Input | A convergence criterion. 1.Q-25 is recommended.
ITER	integer | Input&Output | For JENNFQ, input an iteration limit. The number of actual iterations is to be stored.
MITER	integer | Input | An iteration limit.
G	real*16 | Work | A 1D working array of size N.
ILL	integer | Output | For normal end, zero is to be stored.

By inputting A, where σ is subtracted from the diagonal elements in advance, whatever eigenvalue closest to σ can be obtained as (σ + EIG).

Appendix B. Procedure of MIPOWQ

1. Set an initial vector $\mathbf{v}^{(0)}$, a shift parameter σ, and a convergence criterion ε.
2. Set $k = 0$. Perform the modified Cholesky decomposition[20] so that $B = A - \sigma I = LDL^{\mathrm{T}}$, where L is a lower triangular matrix. The eigenvalue problem to be solved is $B^{-1}\,\mathbf{v} = \rho\,\mathbf{v}$. To obtain $\mathbf{v}^{(k+1)} = B^{-1}\,\mathbf{v}^{(k)}$, solve the simultaneous equation $B\,\mathbf{v}^{(k+1)} = \mathbf{v}^{(k)}$; in other words, solve $B\,\mathbf{v}^{(k+1)} = \mathbf{v}^{(k)} \rightarrow LDL^{\mathrm{T}}\,\mathbf{v}^{(k+1)} = \mathbf{v}^{(k)}$.
3. By forward substitutions, solve $L\,\mathbf{y} = \mathbf{v}^{(k)}$, and obtain $\mathbf{y}$.
4. By backward substitutions, solve $(DL)^{\mathrm{T}}\,\mathbf{v}^{(k+1)} = \mathbf{y}$, and obtain $\mathbf{v}^{(k+1)}$.
5. Obtain $r^{(k+1)}$ from $(\mathbf{v}^{(k)},\mathbf{v}^{(k+1)})$ (an inner product of vectors).
6. Obtain normalized $\mathbf{v}^{(k+1)}$ as $\mathbf{v}^{(k+1)} / \sqrt{\left(\mathbf{v}^{(k+1)},\, \mathbf{v}^{(k+1)}\right)}$.
7. If $\sum_i \left| \left(v_i^{(k+1)}\right)^2 - \left(v_i^{(k)}\right)^2 \right| \le N\varepsilon$ then go to step 8; otherwise set $k + 1 \rightarrow k$ and go to step 3. Here, N is the dimension of matrix A.

8. Obtain the eigenvalue ρ of B^{-1} as $r^{(k+1)}$, and the eigenvector as $\mathbf{v}^{(k+1)}$. Then $(\sigma + 1/\rho)$ is obtained as the eigenvalue of A closest to σ.

Appendix C. Specification table of computer systems

The specifications of the computer systems used is listed in **Table C.6**.

Table C.6 Specifications of computer systems.

	Hostname		
	w2145	CX2 (CX400/270)	cluster
CPU	Intel Xeon W-2145	Intel Xeon E5-2697V2 + Intel Xeon Phi 3120P (co-processor)	Intel Xeon E5-2683V4
Architecture	Skylake	IvyBridgeEP / KnightsCorner	Broadwell
Clock-rate	3.70 GHz	2.70 GHz / 1.10 GHz	2.10 GHz
#Core/CPU	8	12 / 57	16
#CPU	1	2+1	2
LLC	11 MB	30 MB / 28.5 MB	40 MB
TBT	Max 4.50 GHz	Max 3.50 GHz / No	Max 3.00 GHz
HTT	Yes	Yes	Yes
AVX2	Yes	Yes	Yes
AVX-512	Yes	No	No
Main memory	128 GB	128 GB	64 GB
OS	Linux	Linux	Linux
(distribution)	CentOS 7.5	RedHat 6.5	CentOS 7.3
Fortran compiler	Intel Fortran 19.0.1 (OpenMP 4.5)	Intel Fortran 15.0.5 Fujitsu Fortran 1.2.0	Intel Fortran 18.0.1

Appendix D. Performance on CX2

The wall-clock times for `MIPOWQ` and `JENNFQ` on CX2 are set out in **Tables D.7** and **D.8**, respectively. The PR of `MIPOWQ` is shown in **Fig. D.2**.

Table D.7 Wall-clock time (sec) for diagonalizing the Frank matrices using `MIPOWQ` on CX2.

	Number of threads					
N	1	4	8	12	18	24
1000	10.58	3.05	1.91	1.50	1.27	1.15
2000	75.21	19.84	10.81	7.81	5.88	4.90
10000	8586.31	2174.26	1101.77	745.01	506.29	388.90

Table D.8 Wall-clock time (sec) for diagonalizing the Frank matrices using JENNFQ on CX2.

	Number of threads					
N	1	4	8	12	18	24
1000	10.02	3.77	2.77	2.39	2.16	2.05
2000	72.22	22.71	14.62	11.94	10.17	9.32
10000	8467.85	2283.24	1251.82	907.07	675.96	567.21

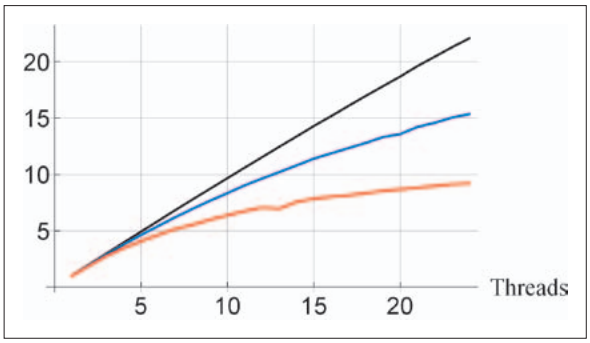

Fig. D.2 Parallel performance of MIPOWQ on CX2 for diagonalizing Frank matrices. Red, blue, and black lines are for N = 1000, 2000, and 10000, respectively.

Appendix E. Performance on cluster

The wall-clock times for MIPOWQ and JENNFQ on cluster are set out in **Tables E.9** and **E.10**, respectively. The PR of MIPOWQ is shown in **Fig. E.2**.

Table E.9 Turbo frequency (GHz) of the Intel Xeon E5-2683V4 of cluster.

	Number of working cores			
	1–2	3	4	5–16
Turbo frequency	3.0	2.8	2.7	2.6

Table E.10 Wall-clock time (sec) for diagonalizing the Frank matrices using MIPOWQ on cluster.

	Number of threads					
N	1	4	8	16	32	64
1000	9.11	2.81	1.86	1.29	1.03	1.27
2000	64.72	18.00	10.70	6.20	4.06	4.54
10000	7403.38	2098.07	1097.60	562.26	298.01	288.45

Table E.11 Wall-clock time (sec) for diagonalizing the Frank matrices using JENNFQ on cluster.

	Number of threads					
N	1	4	8	16	32	64
1000	8.67	3.45	2.73	2.23	1.97	3.23
2000	62.41	20.67	14.29	10.18	8.14	10.39
10000	7301.91	2076.75	1222.99	706.78	456.98	436.69

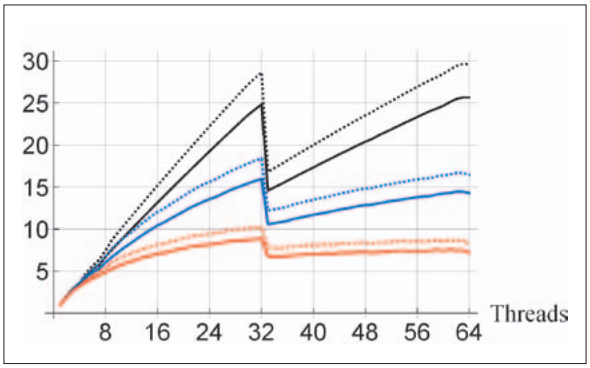

Fig. E.3 Parallel performance of MIPOWQ on cluster for diagonalizing Frank matrices. Red, blue, and black lines are for N = 1000, 2000, and 10000, respectively. Solid/dotted lines illustrate performance ratios without/with TBT correction.

Appendix F. Source program

Table F.12 Source program of MIPOWQ.

```
001       subroutine MIPOWQ(A, KA, X, N, EIG,
002      &  EPS, ITER, MITER, Y, IND)
003   !-----------------------------------------------------------------------
004   !     Calculates the minimum absolute eigenvalue of the matrix A by the
005   !     modified power method.
006   !
007   !     Written by Yasuyo Hatano 2018-09-08
008   !     Modified by Shigeyoshi Yamamoto 2018-12-08 at Chukyo Univ.
009   !     Modified by Shigeyoshi Yamamoto 2018-12-09
010   !       4-factor loop unrolling for J.
011   !     Modified by Shigeyoshi Yamamoto 2018-12-10
012   !       The OpenMP directive is moved to just in front of the inner
013   !       loop of I.
014   !     Modified by Shigeyoshi Yamamoto 2018-12-15
```

```
015  !     Modified by Yasuyo Hatano 2018-12-26
016  !-----------------------------------------------------------------------
017  !   Explanation of the arguments.
018  !     A     : A 2D matrix.  The lower triangle (except for the
019  !             diagonals) are kept unchanged.
020  !     KA    : The extent of the 1st dimension of the matrix A.
021  !     X     : Eigenvectors.  The initial vectors are to be input and
022  !             the resultant vectors are to be output.
023  !     N     : The length of the 1st index of the matrix A that is
024  !             actually referred.
025  !     EIG   : Eigenvalues are to be output.
026  !     EPS   : A constant for judging convergence.
027  !     ITER  : The number of iterations is to be output.
028  !     MITER : The limit for the iteration number is to be input.
029  !             This is initialized by 100 in the subroutine TESTSUB.
030  !     Y     : A 1D working array.
031  !     IND   : Resultant status is to be output.
032  !               IND=0     : Normal end
033  !               IND=30000 : Incorrect input values of the arguments.
034  !               IND=K     : Abnormal end because the pivot approaches
035  !                           zero at the K-th stage of elimination.
036  !-----------------------------------------------------------------------
037        include 'MIT_license.txt'
038        implicit none
039        integer, intent(in) :: KA, N, MITER
040        integer, intent(out) :: ITER, IND
041        real(kind=16) :: EPS, EIG
042        real(kind=16), intent(inout) :: A(KA,KA), X(KA)
043        real(kind=16) :: Y(KA)
044  !-----------------------------------------------------------------------
045        integer :: I, J, K
046        integer :: JEND
047        real(kind=16) :: SUMTMP, SUMM, ANORME, W
048        real(kind=16) :: AKJ, AKJ1, AKJ2, AKJ3
049  !-----------------------------------------------------------------------
050  !   unrolling factor
051        integer :: JFACTOR = 4
052        real(kind=16) :: EPSZ = 1.q-34
053        intrinsic abs, max, sqrt
054  !-----------------------------------------------------------------------
055  !     The original loop structure of the Modified Cholesky
056  !     decomposition (LDL^T) before loop unrolling.
057  !
058  !#    do K = 1, N
059  !#      do I = 1, K-1
060  !#        W = A(I,K)
061  !#        A(I,K) = W / A(I,I)
062  !#        A(K,K) = A(K,K) - A(I,K) * W
```

```
063   !#      end do
064   !#      do J = K+1, N
065   !#        AKJ = A(K,J)
066   !# $OMP parallel do reduction(-:AKJ)
067   !#        do I = 1, K-1
068   !#          AKJ = AKJ - A(I,K) * A(I,J)
069   !#        end do
070   !#        A(K,J) = AKJ
071   !#      end do
072   !-----------------------------------------------------------------------
073         ITER = 0
074         if( N > KA .or. N < 1 .or. EPS <= 0.q0 ) then
075           IND = 30000
076           return
077         end if
078   !
079         SUMM = 0.q0
080         do J = 1, N
081           SUMTMP = 0.q0
082           do I = 1, N
083             SUMTMP = SUMTMP + abs(A(I,J))
084           end do
085           SUMM = max(SUMM,SUMTMP)
086         end do
087         ANORME = SUMM * EPSZ
088
089         SUMTMP = 0.q0
090         do I = 1, N
091           SUMTMP = SUMTMP + X(I) * X(I)
092         end do
093         SUMM = SUMTMP
094   !
095         SUMTMP = 1.q0 / sqrt(SUMTMP)
096         do I = 1, N
097           X(I) = X(I) * SUMTMP
098         end do
099         IND = 0
100   !
101   !     ...The Modified Cholesky decomposition (LDLT) starts.
102   !
103         do K = 1, N
104           do I = 1, K-1
105             W = A(I,K)
106             A(I,K) = W / A(I,I)
107             A(K,K) = A(K,K) - A(I,K) * W
108           end do
109   !
110           if( A(K,K) < ANORME ) then
```

```
111           IND = K
112           write(6,"('A is ill-conditioned.  IND=',I5)") IND
113           return
114         end if
115 !
116         JEND = JFACTOR * ((N-K)/JFACTOR) + K
117 !$OMP parallel do
118         do J = K+1, JEND, JFACTOR
119           AKJ  = A(K,J)
120           AKJ1 = A(K,J+1)
121           AKJ2 = A(K,J+2)
122           AKJ3 = A(K,J+3)
123           do I = 1, K-1
124             AKJ  = AKJ  - A(I,K) * A(I,J)
125             AKJ1 = AKJ1 - A(I,K) * A(I,J+1)
126             AKJ2 = AKJ2 - A(I,K) * A(I,J+2)
127             AKJ3 = AKJ3 - A(I,K) * A(I,J+3)
128           end do
129           A(K,J)   = AKJ
130           A(K,J+1) = AKJ1
131           A(K,J+2) = AKJ2
132           A(K,J+3) = AKJ3
133         end do
134 !
135 !$OMP parallel do
136         do J = JEND+1, N, 1
137           AKJ = A(K,J)
138           do I = 1, K-1
139             AKJ = AKJ - A(I,K) * A(I,J)
140           end do
141           A(K,J) = AKJ
142         end do
143 !
144       end do ! K
145 !
146 !     ...End of the Modified Cholesky decomposition
147 !
148       do I = 1, N
149         Y(I) = X(I)
150       end do
151 !
152       do K = 1, MITER
153 !
154 !     ...Forward substitution
155 !
156         do I = 1, N
157           SUMTMP = 0.q0
158 !
```

```
!$OMP parallel do reduction(+:SUMTMP)
          do J = 1, I-1
            SUMTMP = SUMTMP + A(J,I) * Y(J)
          end do ! J
          Y(I) = Y(I) - SUMTMP
        end do ! I
!
!     ...Backward substitution
!
        do I = N, 1, -1
          Y(I) = Y(I) / A(I,I)
!
          SUMTMP = 0.q0
!$OMP parallel do reduction(+:SUMTMP)
          do J = I+1, N
            SUMTMP = SUMTMP + A(I,J) * Y(J)
          end do
          Y(I) = Y(I) - SUMTMP
        end do ! I
!
        SUMTMP = 0.q0
        do I = 1, N
          SUMTMP = SUMTMP + X(I) * Y(I)
        end do
!
        SUMM = SUMTMP
!
        SUMTMP = 0.q0
        do I = 1, N
          SUMTMP = SUMTMP + Y(I) * Y(I)
        end do
!
        SUMTMP = 1.q0 / sqrt(SUMTMP)
        do I = 1, N
          Y(I) = Y(I) * SUMTMP
        end do
!
        SUMTMP = 0.q0
        do I = 1, N
          SUMTMP = SUMTMP + abs(X(I) * X(I) - Y(I) * Y(I))
          X(I) = Y(I)
        end do
!
        if( SUMTMP < (EPS*N) ) then
           ITER = K
           go to 1000
        end if
!
```

```
      end do  ! K
      IND = MITER
      EIG = 1.q0 / SUMM
      write(6,*)' ITER greater than  MITER '
      return
 1000 continue
      EIG = 1.q0 / SUMM
      IND = 0
      return
      end subroutine MIPOWQ
```

References

[1] Y. Hatano and S. Yamamoto, J. Comput. Chem., Jpn -Int. Ed. 2, 2016-0003 (2016). doi: 10.2477/jccjie.2016-0003.

[2] S. Yamamoto, Y. Hatano, and H. Tatewaki, Comp. Theor. Chem. 1103, 17–24 (2017).

[3] J. S. Sims and S. A. Hagstrom, J. Phys. B: At. Mol. Opt. Phys. 37, 1519–1540 (2004).

[4] L.-Y. Tang, Y.-B. Tang, T.-Y. Shi, and J. Mitroy, J. Chem. Phys. 139, 134112 (2013).

[5] Y. Scribano, G. Parlant, and B. Poirier, J. Chem. Phys. 149, 021101 (2018).

[6] A. M. Ferrenberg, J. Xu, and D. P. Landau, Phys. Rev. E 97, 043301 (2018).

[7] IEEE Computer Society, (August 29, 2008), IEEE Standard for Floating-Point Arithmetic, IEEE. doi: 10.1109/IEEESTD.2008.4610935.

[8] Intel MKL, Intel Math Kernel Library, 2018. [Cited 2018-10-9] URL https://software.intel.com/en-us/mkl.

[9] D. H. Bailey, QD (A C++/Fortran-90 double-double and quad-double package). [Cited 2018-10-9] URL http://crd-legacy.lbl.gov/~dhbailey/mpdist/.

[10] S. Yamada, T. Ina, N. Sasa, Y. Idomura, M. Machida, and T. Imamura, 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 1418, IEEE Com. Soc. (2017).

[11] R. Ogata, Y. Kubo, and T. Takei, NEC Technical Journal 3, 59–63 (2008). URL https://www.nec.com/en/global/techrep/journal/g08/n04/pdf/080412.pdf.

[12] R. Ogata, Y. Kubo, and T. Takei, J. Soc. Instrum. Control Engnr. 49, 319–322 (2010). (mainly in Japanese).

[13] A. Nishida, “Experience in Developing an Open Source Scalable Software Infrastructure in Japan”, Lecture Notes in Computer Science 6017, Springer, 448–462 (2010). doi: https://doi.org/10.1007/978-3-642-12165-4_36.

[14] I. Ninomiya and Y. Hatano, “Performance of the Vector Version of NUMPAC”, in: High Performance Computing: Research and Practice in Japan, R. Mendez (Ed.), John Wiley & Sons, Chichester (1992).

[15] T. Hasegawa, T. Sandoh, Y. Sato, Y. Hatano, and I. Ninomiya, Memoirs of the Faculty of Engineering, Fukui University, 48, 193–213 (2000). http://hdl.handle.net/10098/3316 [Cited 2018-10-9] URL http://www2.itc.nagoya-u.ac.jp/center/ja/numpac/index.html.

[16] J. M. Peek, J. Chem. Phys. 43, 3004–3006 (1965).

[17] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, and R. Menon, “Parallel Programming in OpenMP”, Academic Press, San Diego, CA, USA (2001).

[18] S. Akhter and J. Roberts, “Multi-Core Programming”, Intel Press, Hillsboro, OR, USA (2006).

[19] A. N. Ralston and P. R. Rabinowitz, “A First Course in Numerical Analysis”, 2nd Ed., Dover, N.Y. (2001).

[20] G. H. Golub and C. F. van Loan, “Matrix Computations”, 4th Ed., Johns Hopkins University Press, Baltimore (2013).

[21] M. Clint and A. Jennings, Comput. J. 13, 76–80 (1970).
[22] A. Jennings, “Matrix Computation for Engineers and Scientists”, John Wiley & Sons, London (1977).
[23] E. R. Davidson, J. Comput. Phys. 17, 87–94 (1975).
[24] J. Olsen, P. Jørgensen, and J. Simons, Chem. Phys. Letters, 169, 463–472 (1990).
[25] G. L. G. Sleijpen and H. A. van der Vorst, SIAM Review, 42, 267–293 (2000).
[26] W. L. Frank, J. Soc. Indust. Appl. Math. 6, 378–392 (1958).
[27] Intel Corporation. “Intel Turbo Boost Technology in Intel Core Microarchitecture (Nehalem) Based Processors.”, Whitepaper, Intel Corporation, November 2008.
[28] J. Charles, P. Jassi, N. S. Ananth, A. Sadat, and A. Fedorova, 2009 IEEE International Symposium on Workload Characterization (IISWC), 188–197 IEEE Com. Soc. (2009).
[29] A. YarKhan, J. Kurzak, P. Luszczek, and J. Dongarra, Int. J. Parallel Prog. 45, 612–633 (2017).
[30] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, LAPACK Users’ Guide, Release 3.0, SIAM, Philadelphia (1999). [Cited 2018-10-9]
URL http://www.netlib.org/lapack/.
[31] WikiChip, Intel Xeon W-2145. [Cited 2018-12-19]
URL https://en.wikichip.org/wiki/intel/xeon_w/w-2145.
[32] G. M. Amdahl, AFIPS Conference Proceedings (30) 483–485 (1967).
doi: 10.1145/1465482.1465560.