Neuberger's double-pass algorithm

Ting-Wai Chiu and Tung-Han Hsieh
Department of Physics, National Taiwan University Taipei, Taiwan 106, Taiwan
(Received 30 June 2003; revised manuscript received 29 September 2003; published 31 December 2003)

We analyze Neuberger's double-pass algorithm for the matrix-vector multiplication $R(H)\cdot Y$ [where $R(H)$ is $(n-1,n)$th degree rational polynomial of positive definite operator H], and show that the number of floating-point operations is independent of the degree n, provided that the number of sites is much larger than the number of iterations in the conjugate gradient. This implies that the matrix-vector product $(H)^{-1/2}Y \simeq R^{(n-1,n)}(H)\cdot Y$ can be approximated to very high precision with sufficiently large n, without noticeably extra costs. Further, we show that there exists a threshold n_T such that the double-pass is faster than the single pass for $n > n_T$, where $n_T \simeq 12$–25 for most platforms.

DOI: 10.1103/PhysRevE.68.066704 PACS number(s): 02.70.−c, 11.15.Ha, 11.30.Rd, 12.38.Gc

I. INTRODUCTION

In 1998, Neuberger proposed the nested conjugate gradient [1] for solving the propagator of the overlap-Dirac operator [2]

$$D = m_0\left(1+\gamma_5 \frac{H_w}{\sqrt{H_w^2}}\right), \tag{1}$$

with the sign function $\mathrm{sgn}(H_w) \equiv H_w(H_w^2)^{-1/2}$ approximated by the polar approximation

$$S(H_w) = \frac{H_w}{n}\sum_{l=1}^{n}\frac{b_l}{H_w^2+d_l} \equiv H_w R^{(n-1,n)}(H_w^2), \tag{2}$$

where $H_w = \gamma_5 D_w$, D_w is the standard Wilson-Dirac operator plus a negative parameter $-m_0$ $(0<m_0<2)$, and the coefficients b_l and d_l are

$$b_l = \sec^2\left[\frac{\pi}{2n}\left(l-\frac{1}{2}\right)\right], \quad d_l = \tan^2\left[\frac{\pi}{2n}\left(l-\frac{1}{2}\right)\right].$$

In principle, any column vector of $D^{-1} = D^{\dagger}(DD^{\dagger})^{-1}$ can be obtained by solving the system

$$DD^{\dagger}Y = m_0^2[2+\gamma_5 S(H_w) + S(H_w)\gamma_5]Y = \mathbb{I} \tag{3}$$

with conjugate gradient, provided that the matrix-vector product $S(H_w)Y$ can be carried out. Writing

$$S(H_w)Y = \frac{H_w}{n}\sum_{l=1}^{n} b_l Z^{(l)}, \tag{4}$$

one can obtain $\{Z^{(l)}\}$ by solving the system

$$(H_w^2 + d_l)Z^{(l)} = Y \tag{5}$$

with multishift conjugate gradient (CG) [3,4]. In other words, each (outer) CG iteration in Eq. (3) contains a complete (inner) CG loop (5), i.e., nested conjugate gradient.

Evidently, the overhead for the nested conjugate gradient is the execution time for the inner conjugate gradient loop (5) as well as the memory space it requires, i.e., $(2n+3)$ large vectors, each of $12N_{site}$ double complex numbers, where N_{site} is the number of sites, and $12=3$ (color) $\times$ 4 (Dirac) is the degree of freedom at each site for QCD. The memory storage becomes prohibitive for large lattices since n is often required to be larger than 16 in order to achieve a reliable approximation for the sign function. To minimize the memory storage for $\{P^{(l)}, Z^{(l)}\}$, Neuberger [5] observed that one only needs the linear combination $\Sigma_{l=1}^{n} b_l Z^{(l)}$ rather than each $Z^{(l)}$ individually. Since $Z_i^{(l)}$ and its conjugate vector $P_i^{(l)}$ at the ith iteration of the inner CG are linear combinations of their precedents $\{P_j^{(l)}, Z_j^{(l)}, j=0,\ldots,i-1\}$ in the iteration process, it is possible to obtain their updating coefficients $\{\alpha_i^{(l)}, \beta_i^{(l)}\}$ in the first pass, and then use them to update the sum $\Sigma_{l=1}^{n} b_l Z^{(l)}$ successively in the second pass, with memory storage of only five vectors, independent of the degree n of the rational polynomial $R^{(n-1,n)}$.

At first sight, the double-pass algorithm seems to be slower than the single-pass algorithm. However, in the test run [with SU(2) gauge field on the 8^3 lattice], Neuberger found that the double-pass actually ran faster by 30% than the single pass, and remarked that the speedup most likely reflects the cache usage in the testing platform, the SGI O2000 (with four processors, each with 4 MB cache memory).

In this paper, we analyze the number of floating-point operations (F_2) for the double-pass algorithm, and show that it is independent of the degree n of $R^{(n-1,n)}$, provided that the number of lattice sites (N_{site}) is much larger than the number of iterations (L_i) of the CG loop. The last condition is amply satisfied even for a small lattice (e.g., $N_{site} = 8^3 \times 24 = 12\,288$), since L_i is usually less than 1000 (after the low-lying eigenmodes of H_w^2 are projected out). On the other hand, the number of floating-point operations (F_1) for the single pass is a linearly increasing function of n. It follows that there exists a threshold n_F such that $F_2 \leqslant F_1$ for $n \geqslant n_F$, where the value of n_F depends on the implementation of the algorithms ($n_F \simeq 59$ for our codes). Corresponding to the number of floating-point operations, we also obtain the formulas for the CPU times (T_1 and T_2) for the single- and the double-pass algorithms. Further, we show that there exists a threshold n_T such that the double-pass is faster than the single pass[1] for $n > n_T$, where $n_T \simeq 12$–25 (for most platforms), which is quite smaller than the threshold $n_F \simeq 59$ for the number of floating-point operations. By timing the speed of each subroutine, we can account for the extra slow down in the single-pass algorithm, which is unlikely to be eliminated, due to the memory bandwidth, a generic weakness of any computational system. Thus, in general (for most vector or superscalar machines), one may find that the double-pass is faster than the single pass, for $n > n_T \simeq 12$–25. This explains why in Neuberger's test run, even at $n=32$, the double-pass is already faster than the single pass by 30%. In fact, we find that DEC alpha XP1000 and IBM SP2 SMP also have 30% speedup at $n=32$ (see Table IV), which agrees with the theoretical estimate (32) using the CPU time formulas (27) and (28).

Nevertheless, the most interesting result is that the speed of the double-pass algorithm is almost independent of the degree n. This implies that the matrix-vector product $(H_w^2)^{-1/2} Y \simeq R^{(n-1,n)}(H_w^2) Y$ can be approximated to very high precision with sufficiently large n, without noticeably extra costs.

The outline of this paper is as follows. In Sec. II, we outline the single- and double-pass algorithms for the iteration of the (inner) CG loop (5), and analyze their major differences. In Sec. III, we estimate the number of (double precision) floating-point operations as well as the CPU time for the single- and double-pass algorithms, respectively, and show that there exists a threshold n_T such that the double-pass is faster than the single pass for $n > n_T$. In Sec. IV, we perform some tests. In Sec. V, we conclude with some remarks.

II. THE SINGLE- AND THE DOUBLE-PASS ALGORITHMS

In the section, we outline the single- and the double-pass algorithms for the inner CG loop (5), and point out their major differences.

For the single-pass algorithm, with the input vector Y, we initialize the vector variables $\{Z^{(l)}, P^{(l)}\}, R, A, B$ and the scalar variables $\alpha, \beta, \{\gamma^{(l)}\}$ as

$$Z_0^{(l)} = 0, \quad P_0^{(l)} = Y, \quad l = 1, \ldots, n,$$

$$R_0 = Y,$$

$$\alpha_{-1} = 1,$$

$$\beta_0 = 0,$$

$$\gamma_{-1}^{(l)} = \gamma_0^{(l)} = 1, \quad l = 1, \ldots, n.$$

Then we iterate ($j = 0, 1, \ldots$) according to

$$A_j = H_w P_j^{(1)}, \tag{6}$$

$$B_j = H_w A_j + d_1 P_j^{(1)} = (H_w^2 + d_1) P_j^{(1)}, \tag{7}$$

$$\alpha_j = \frac{\langle R_j | R_j \rangle}{\langle P_j^{(1)} | B_j \rangle}, \tag{8}$$

$$R_{j+1} = R_j - \alpha_j B_j, \tag{9}$$

$$\beta_{j+1} = \frac{\langle R_{j+1} | R_{j+1} \rangle}{\langle R_j | R_j \rangle}, \tag{10}$$

$$P_{j+1}^{(1)} = R_{j+1} + \beta_{j+1} P_j^{(1)}, \tag{11}$$

$$Z_{j+1}^{(1)} = Z_j^{(1)} + \alpha_j P_j^{(1)}, \tag{12}$$

together with the following updates for $l = 2, \ldots, n$:

$$\gamma_{j+1}^{(l)} = \frac{\gamma_j^{(l)} \gamma_{j-1}^{(l)} \alpha_{j-1}}{\alpha_j \beta_j (\gamma_{j-1}^{(l)} - \gamma_j^{(l)}) + \gamma_{j-1}^{(l)} \alpha_{j-1} [1 + \alpha_j (d_l - d_1)]}, \tag{13}$$

$$P_{j+1}^{(l)} = \gamma_{j+1}^{(l)} R_{j+1} + \beta_{j+1} \left(\frac{\gamma_{j+1}^{(l)}}{\gamma_j^{(l)}} \right)^2 P_j^{(l)}, \tag{14}$$

$$Z_{j+1}^{(l)} = Z_j^{(l)} + \alpha_j \frac{\gamma_{j+1}^{(l)}}{\gamma_j^{(l)}} P_j^{(l)}. \tag{15}$$

The loop terminates at the ith iteration if $\sqrt{\langle R_{i+1} | R_{i+1} \rangle / \langle Y | Y \rangle}$ is less than the tolerance (tol).

Since we are only interested in the linear combination $\sum_{l=1}^n b_l Z_{i+1}^{(l)}$, in which each $Z_{i+1}^{(l)}$ can be expressed in terms of $\{R_j, j = 0, \ldots, i\}$, we can write

$$\sum_{l=1}^{n} b_l Z_{i+1}^{(l)} = \sum_{j=0}^{i} c_j R_j, \tag{16}$$

where c_j can be derived as [5]

$$c_j = \sum_{m=0}^{i-j} \left[\alpha_{j+m} \delta_m \left(b_1 + \sum_{l=2}^{n} b_l \frac{\gamma_{m+j+1}^{(l)} \gamma_{m+j}^{(l)}}{\gamma_j^{(l)}} \right) \right], \tag{17}$$

with

$$\delta_m = \begin{cases} \prod_{k=1}^{m} \beta_{j+k} & \text{for } m > 0 \\ 1 & \text{for } m = 0. \end{cases} \tag{18}$$

Therefore, the right hand side (rhs) of Eq. (16) can be evaluated with the CG loop (6)–(11), requiring only the storage of five large vectors ($A, B, R, P^{(1)}, T = \Sigma c_j R_j$), provided that the coefficients $\{c_j, j = 0, \ldots, i\}$ are known. However, from Eq. (17), the determination of c_j at any jth iteration requires some values of $\{\alpha\}$, $\{\beta\}$, and $\{\gamma\}$ which can only be obtained in later iterations. Thus we have to run the first pass, i.e., the CG loop (6)–(11), to obtain all coefficients of $\{\alpha\}$ and $\{\beta\}$, up to the convergence point i, and then compute all $\{c_j, j = 0, \ldots, i\}$ according to Eqs. (17) and (13). Finally, we

[1] In this paper, we only consider the (faster) single-pass algorithm in which the vectors $P^{(l)}$ and $Z^{(l)}$ ($l = 2, \ldots, n$) are not updated after $Z^{(l)}$ converges.

TABLE I. The average CPU time (in units of nanosecond) per floating-point operation (FPO) for four different kinds of matrix-vector operations in the single- and double-pass algorithms. The CPU is Pentium 4 (2.53 GHz), with 1 Gbyte Rambus (PC800 or PC1066).

			CPU time (ns) per FPO			
			PC800		PC1066	
	Operation	No. of FPO	SSE2 on	SSE2 off	SSE2 on	SSE2 off
(a)	$\|A\rangle = c_1\|A\rangle + c_2\|B\rangle$	$72N_{site}$	3.720	3.721	2.977	3.016
(b)	$\|V\rangle = \|A\rangle + c\|B\rangle$	$48N_{site}$	5.521	5.522	4.330	4.429
(c)	$\alpha = \langle V\|V\rangle$	$36N_{site}$	4.249	4.251	3.312	3.340
(d)	$\|A\rangle = H_w\|B\rangle$	$1644N_{site}$	0.764	1.535	0.686	1.440

run the second pass, i.e., going through Eqs. (6), (7), (9), and (11), and adding $c_j R_j$ to the rhs of Eq. (16), successively from $j=0$ to the convergence point i.

Evidently, all operations in Eqs. (6)–(12), (14), and (15) are proportional to the number of lattice sites, N_{site}, times the number of iterations, L_i. On the other hand, the computations of the coefficients $\{\gamma\}$ [Eq. (13)] and $\{c_j\}$ [Eq. (17)] do not depend on N_{site}, but only on L_i (up to a small term proportional to L_i^3). Thus, for $N_{site} \gg L_i$, we can neglect the computation of $\{c_j\}$ [Eq. (17)], and focus on the major difference between the single pass and the double-pass, namely, the number of operations in Eqs. (14) and (15), which is proportional to $(n-1)N_{site}L_i$, versus the number of operations in Eqs. (6), (7), (9), and (11) plus the vector update in the rhs of Eq. (16), which is proportional to $N_{site}L_i$. Obviously, the number of floating-point operations in the single pass is a linearly increasing function of n, while that of the double-pass is independent of n, thus it follows that the double-pass must be faster than the single pass for sufficiently large n.

In the following section, we estimate the number of floating-point operations as well as the CPU time, for the single pass, and the double-pass respectively. Even though our countings are based on our codes, they serve to illustrate the general features of the single- and the double-pass algorithms, which are valid for any software implementations and/or machines.

III. THE CPU TIME AND THE NUMBER OF FLOATING-POINT OPERATIONS

For our codes, the number of floating-point operations for the single pass is

$$F_1 = N_{site}L_i[3552 + 120(n-1)p] + N_{site}(288n_{ev} + 48n + 1776), \tag{19}$$

while for the double-pass it is

$$F_2 = 6888N_{site}L_i + N_{site}(288n_{ev} - 1656) + \left[\frac{L_i^3}{6} + L_i^2(2n-1) + L_i\left(13n - \frac{73}{6}\right) - 7n + 7\right]q, \tag{20}$$

where N_{site} is the number of sites of the lattice, L_i is the number of iterations of the CG loop, n_{ev} is the number of projected eigenmodes of H_w^2, and n is the degree of the rational polynomial $R^{(n-1,n)}$. In the single pass, Eq. (19), $(n-1)p$ is the effective number of the $(n-1)$ updates in Eqs. (14) and (15), since $P^{(l)}$ and $Z^{(l)}$ are not updated after $Z^{(l)}$ converges. The value of p depends on the convergence criteria as well as the rational polynomial $R^{(n-1,n)}$ and its argument. Similarly, in the double-pass, the sum in Eq. (17) only includes the terms which have not yet converged at the iteration j, and the reduction in the number of floating-point operations can be taken into account by the factor q in Eq. (20). (The value of q is about 0.95 for convergence up to zero in the IEEE double precision representation.)

Taking into account different speeds of various floating-point operations, we estimate the CPU time for the single pass and the double-pass as follows:

$$T_1 = N_{site}L_i[192t_b + 72t_c + 3288t_d + (48t_b + 72t_a)(n-1)p] + N_{site}(288n_{ev}t_e + 48nt_b + 24t_a + 108t_c + 1644t_d), \tag{21}$$

$$T_2 = N_{site}L_i(240t_b + 72t_c + 6576t_d) + N_{site}(288n_{ev}t_e + 24t_a - 144t_b + 108t_c - 1644t_d) + q\left[\frac{L_i^3}{6} + L_i^2(2n-1) + L_i\left(13n - \frac{73}{6}\right) - 7n + 7\right]t_f, \tag{22}$$

where t_a, t_b, t_c, and t_d denote the average CPU time per floating-point operation (FPO) for the four different kinds of vector operations (a)–(d) listed in Table I, respectively, t_e the average CPU time per FPO for constructing the complementary vector from the projected eigenmodes of H_w^2, and t_f the time for computing the coefficients (17) in the double-pass. Note that setting $t_a = t_b = t_c = t_d = t_e = t_f = 1$ in Eqs. (21) and (22) reproduces Eqs. (19) and (20), respectively.

It should be emphasized that the numerical values of the constants and coefficients in Eqs. (19)–(22) may vary slightly from one implementation to another, however, the number of different terms and their functional dependences on the variables (N_{site}, L_i, n, n_{ev}, p, q, t_a, t_b, t_c, t_d, t_e, and t_f) should be the same for any codes of the single- and double-pass algorithms.

For the double-pass, it is clear that the first term in the rhs of Eq. (20) is the most significant part, since the number of lattice sites (N_{site}) is usually much larger than the number of iterations (L_i) of the CG loop such that the second and the third terms in the rhs of Eq. (20) can be neglected. For example, $N_{site}=8^3\times 24$, $L_i=1000$, $n=16$, $n_{ev}=32$, and $q=0.95$, then the first term is $6888 N_{site} L_i \simeq 8.5\times 10^{10}$, while the sum of the second and the third terms only gives $\sim 2.8\times 10^8$. Thus we can single out the most significant part of F_2,

$$F_2 \simeq 6888 N_{site} L_i, \tag{23}$$

which comes from the first pass, Eqs. (6)–(11), and the second pass, Eqs. (6), (7), (9), and (11), plus the vector update in the rhs of Eq. (16). Similarly, for the single pass, the most significant part of F_1 is the first term in the rhs of Eq. (19),

$$F_1 \simeq N_{site} L_i [3552 + 120(n-1)p], \tag{24}$$

which comes from the operations in Eqs. (6)–(12), (14) and (15).

Evidently, from Eqs. (24) and (23), F_1 is a linearly increasing function of n while F_2 is independent of n. Thus it follows that there exists a threshold n_F such that $F_2 < F_1$ for $n > n_F$. From Eqs. (24) and (23), we obtain the threshold n_F,

$$n_F = 1 + \frac{139}{5p}, \tag{25}$$

where the value of p depends on the convergence criterion for removing $\{P^{(l)}, Z^{(l)}\}$ from the updating list, as well as the rational polynomial $R^{(n-1,n)}$ and its argument. For our codes and the tests in the following section, $p \simeq 0.48$, thus we have

$$n_F \simeq 59. \tag{26}$$

Assuming $N_{site} \gg L_i$, we obtain the most significant parts of the CPU times (21) and (22) as

$$T_1 \simeq N_{site} L_i [192 t_b + 72 t_c + 3288 t_d + (48 t_b + 72 t_a)(n-1)p], \tag{27}$$

$$T_2 \simeq N_{site} L_i (240 t_b + 72 t_c + 6576 t_d). \tag{28}$$

Obviously, from Eqs. (27) and (28), there exists a threshold

$$n_T = 1 + \frac{2 t_b + 137 t_d}{(2 t_b + 3 t_a) p} \tag{29}$$

such that $T_2 < T_1$ (the double-pass is faster than the single pass) for $n > n_T$.

Even though the countings in Eqs. (21) and (22) are based on our codes (for $R^{(n-1,n)}$ with argument H_w^2), the essential features of Eqs. (21) and (22) should be common to all implementations of the single- and the double-pass algorithms. In other words, the numerical coefficients in Eqs. (27) and (28) may change from one implementation to another, however, the existence of a threshold n_T must hold for any implementation.

TABLE II. Similar to Table I, except for the platforms IBM SP2 SMP (Power 3 at 375 MHz) with 4 Gbyte memory and DEC alpha XP1000 (21264A at 667 MHz) with 1.5 Gbyte memory.

	Operation	No. of FPO	CPU time (ns) per FPO IBM	CPU time (ns) per FPO DEC
(a)	$\|A\rangle = c_1\|A\rangle + c_2\|B\rangle$	$72 N_{site}$	5.269	7.232
(b)	$\|V\rangle = \|A\rangle + c\|B\rangle$	$48 N_{site}$	10.98	12.91
(c)	$\alpha = \langle V\|V\rangle$	$36 N_{site}$	6.209	7.684
(d)	$\|A\rangle = H_w\|B\rangle$	$1644 N_{site}$	2.379	3.054

Now it is interesting to compare n_T with n_F. From Eqs. (25) and (29), one immediately sees that $n_T < n_F$ if

$$19 t_d < 11 t_a + 7 t_b \tag{30}$$

is satisfied.[2]

In practice, it turns out that $t_a/t_d > 2$ and $t_b/t_d > 3$ for most systems (Tables I and II). Thus, $n_T \simeq 12$–25, which is quite smaller than $n_F \simeq 59$.

The speedup of the double-pass with respect to the single pass (for $n > n_T$) can be defined as

$$S = \frac{T_1 - T_2}{T_2} \tag{31}$$

which is estimated to be

$$S \simeq \frac{(3 t_a + 2 t_b) p}{10 t_b + 3 t_c + 274 t_d}(n - n_T), \tag{32}$$

where Eqs. (27)–(29) have been used.

In Table I, we list our measurements of t_a, t_b, t_c, and t_d for four different hardware configurations of Pentium 4, i.e., two different Rambuses of faster/slower (PC1066/PC800) speed, and with/without SSE2 (the vector processing unit of Pentium 4) codes.

Substituting the values of t_a, t_b, and t_d into (29), we obtain the theoretical estimates for the threshold n_T,

$$n_T \simeq \begin{cases} 12, & \text{Pentium 4, PC800, with } \textit{SSE2} \\ 22, & \text{Pentium 4, PC800} \\ 13, & \text{Pentium 4, PC1066, with } \textit{SSE2} \\ 25, & \text{Pentium 4, PC1066,} \end{cases} \tag{33}$$

where $p \simeq 0.48$ has been used.

Note that for each hardware configuration in Table I, the average CPU time per FPO of the simple vector operations (a)–(c) is much longer than that of (d), Wilson matrix times

[2]Note that the inequality (30) is more restrictive than $685 t_d < 417 t_a + 268 t_b$.

vector. A simplified explanation[3] is as follows. Since all these four vector operations involve long vectors, the CPU and its cache cannot hold all data at once. Thus it is necessary to transfer the data from/to the memory successively, every time the CPU completes its operations on a portion of the vectors. However, for any system, the memory bandwidth is limited. Thus, there is a time interval between consecutive sets of data transferring to/from the CPU. Therefore, if the CPU finishes a computation before the next set of data is ready, then it would waste its cycles in idling. Since any one of the vector operations (a)–(c) is rather simple, the CPU finishes a computation at a speed faster than that of transferring data from/to the memory, thus the CPU ends up wasting a significant fraction of time in idling. On the other hand, for the vector operation (d), the number of FPO is much more than that of any one of (a)–(c), thus when the CPU completes its operations on a portion of the vectors, the next set of data might have been ready, so the CPU does not waste much time in the memory I/O. This explains why the average CPU time per FPO of (a)–(c) is much longer than that of (d). Further, this simple picture also explains why turning on SSE2 of Pentium 4 (see Table I) doubles the speed of (d) but has no speedups for (a)–(c), since the bottleneck of (a)–(c) is essentially due to the memory bandwidth rather than the speed of the CPU.

If the memory bandwidth is the major cause for the inefficiency of the simple vector operations (a)–(c), then using faster memories would increase the speeds of (a)–(c) more significantly than that of (d). From Table I, we can compare the speedups of these four vector operations as the (slower) PC800 is replaced with (faster) PC1066. We find that the speedup for (a)–(c) is 27%, but that for (d) is only 11%. Thus the speedups are consistent with above picture.

Obviously, the inefficiency of vector operations (a)–(c) should exist in any platform, not only for the Pentium 4 systems. To check this, we measure t_a, t_b, t_c, and t_d for IBM SP2 SMP (Power 3 at 375 MHz) and DEC alpha XP1000 (21264A at 667 MHz), respectively. The results are listed in Table II, which give

$$n_T \simeq \begin{cases} 21, & \text{DEC alpha XP1000} \\ 20, & \text{IBM SP2 SMP.} \end{cases} \quad (34)$$

Although it is impossible to go through all platforms and measure the values of t_a, t_b, and t_d individually, it is expected that $t_a/t_d > 1$ and $t_b/t_d > 1$ [such that the inequality (30) is amply satisfied] is a common feature of most systems. In other words, we expect that the double-pass is faster than the single pass for $n > n_T \simeq 12$–25, at least for most platforms.

Recall that in Neuberger's test run with SGI O2000, at $n=32$, the double-pass is faster than the single pass by 30% [5]. This is not a surprise at all, in view of similar speedups of other systems at $n=32$. For example, for IBM SP2 SMP or DEC alpha XP1000, substituting the values of t_a, t_b, t_c, and t_d (from Table II) into Eq. (32), we find that $S = T_1/T_2 - 1 \simeq 30\%$ at $n=32$, which also agrees with the actual measurements given in the following section (see Table IV). Thus, the speedup S of the double-pass for $n > n_T$ with n_T quite smaller than n_F is a generic feature of any platform, stemming from the fact that the vector operations in the second pass is more efficient than those, Eqs. (14) and (15), in the single pass (i.e., $t_a > t_d$ and $t_b > t_d$).

Nevertheless, the salient feature of Eqs. (23) and (28) is that the number of floating-point operations and the CPU time for the double-pass are almost independent of n. Thus one can choose n as large as one wishes, with only a negligible overhead. For example, for the $16^3 \times 32$ lattice, with $L_i = 1000$, $n_{ev} = 20$, and $q = 0.95$, the increment of T_2 from $n=16$ to $n=200$ is less than 0.05%. In other words, one can approximate $(H_w^2)^{-1/2} Y$ (i.e., preserve the chiral symmetry) to any precision as ones wishes, without noticeably extra costs. This is the virtue of Neuberger's double-pass algorithm, which may have been overlooked in the last five years.

IV. TESTS

In this section, we perform several tests on the single- and the double-pass algorithms, and compare the theoretical thresholds n_T, Eq. (29), and n_F, Eq. (25), with the measured values.

In Table III, we list the number of floating-point operations and the CPU time for computing one column of the inverse of

$$D(m_q) = m_q + (m_0 - m_q/2)[1 + \gamma_5 S(H_w)],$$

i.e., $D^{-1}(m_q) = D(m_q)^\dagger Y$, where Y is solved from

$$D(m_q) D^\dagger(m_q) Y = \{m_q^2 + (m_0^2 - m_q^2/4) \times [2 + (\gamma_5 \pm 1) S(H_w)]\} Y = \mathbb{I} \quad (35)$$

with multimass (outer) conjugate gradient for a set of 16 bare quark masses ($0.02 \leqslant m_q \leqslant 0.3$), while the inner CG (5) is iterated with the single pass, and the double-pass respectively. The tests are performed on the $8^3 \times 24$ lattice with SU(3) gauge configuration generated by the Wilson gauge action at $\beta = 5.8$. Other parameters are $m_0 = 1.30$, $n_{ev} = 32$ (the number of projected eigenmodes), $\lambda_{max}/\lambda_{min} = 6.207/0.198$ (after projection), and the tolerances for the outer and inner CG loops are 1.0×10^{-11} and 2.0×10^{-12}, respectively. The total number of iterations, L_o, for the outer CG loop is around 100–103, while the average number of iterations for the inner CG loop is ~ 287.

With the formulas (19)–(22), we can estimate the number of floating-point operations and the CPU time for computing one column of $D^{-1}(m_q)$ for a number n_q of bare quark masses. For the number of floating-point operations, our results are

$$G_k = (L_o + n_q) F_k + N_{site}(60 L_o n_q + 84 L_o + 66 n_q) + 16 L_o n_q - 13 L_o + 18 n_q + 2, \quad (36)$$

[3] It should be emphasized that the mechanism of the interactions between the CPU and the RAM is a rather complicated process, which is beyond the scope of this paper.

TABLE III. The number of floating-point operations and the CPU time (in units of second) for Pentium 4 (2.53 GHz) with 1 Gbyte Rambus (PC1066) to compute one column of $D^{-1}(m_q)$ for 16 quark masses versus the degree n of the rational polynomial $R^{(n-1,n)}$ in polar approximation (2).

	Double pass			Single pass			
	No. of FPO	CPU time (s)		No. of FPO	CPU time (s)		σ
n	G_2	V_2	Measured	G_1	V_1	Measured	Polar
12	2.90×10^{12}	2456	2451	1.68×10^{12}	2342	2241	6×10^{-5}
13	2.90×10^{12}	2456	2452	1.71×10^{12}	2429	2372	3×10^{-5}
14	2.90×10^{12}	2456	2454	1.75×10^{12}	2515	2520	1×10^{-5}
16	2.90×10^{12}	2456	2454	1.81×10^{12}	2689	2714	3×10^{-6}
32	2.90×10^{12}	2458	2456	2.25×10^{12}	4097	4089	3×10^{-11}
34	2.90×10^{12}	2458	2458	2.30×10^{12}	4273	4278	7×10^{-12}
40	2.90×10^{12}	2458	2456	2.45×10^{12}	4803	4819	1×10^{-13}
56	2.90×10^{12}	2460	2460	2.86×10^{12}	6218	6261	2×10^{-14}
59	2.90×10^{12}	2460	2460	2.93×10^{12}	6483	6491	2×10^{-14}
60	2.90×10^{12}	2460	2461	2.96×10^{12}	6572	6604	2×10^{-14}
64	2.90×10^{12}	2460	2461	3.06×10^{12}	6926	6965	2×10^{-14}

where L_o is the number of iterations of the outer CG loop (35), the subscript $k=1$ (2) stands for the single (double) pass. Obviously, the most significant part of G_k is the first term in the rhs of Eq. (36), thus

$$G_k \simeq (L_o + n_q) F_k, \quad k=1,2. \tag{37}$$

Similarly, the most significant part of the CPU time is

$$V_k \simeq (L_o + n_q) T_k, \quad k=1,2 \tag{38}$$

where T_1 and T_2 are given in Eqs. (21) and (22).

In Table III, the estimated CPU times V_1 and V_2 are in good agreement with the measured CPU times (the deviation is always less than 5%). By comparing the CPU times for the single pass and the double-pass, we see that the double-pass becomes faster than the single pass at $n\simeq13$, in agreement with the theoretical estimate (33) for $p=0.48$, where p is obtained by measuring the effective number of the $(n-1)$ vector pairs $\{P^{(l)}, Z^{(l)}, l=2, \dots, n\}$ which are updated before $Z^{(l)}$ converges.

Further, comparing G_2 and G_1, we see that $G_1 \simeq G_2$ at $n_F \simeq 59$, in agreement with the theoretical estimate (26) for $p=0.48$.

Also, in Table III, the remarkable feature of the double-pass algorithm is demonstrated: the number of floating point operations (G_2) and the CPU time are almost independent of n. Thus n can be increased to 64 or any higher value such that the chiral symmetry is preserved to any precision as one wishes. The chiral symmetry breaking or the error of the rational approximation $R^{(n-1,n)}$ due to a finite n can be measured by

$$\sigma = \max_Y \left| \frac{W^\dagger W}{Y^\dagger Y} - 1 \right|, \quad W = S(H_w) Y, \tag{39}$$

which is shown in the last column of Table III.

To check the theoretical estimates for the threshold n_T in Eq. (34), we repeat the tests of Table III for Pentium 4 (PC800), IBM SP2 SMP, and DEC alpha XP1000, respectively. The results are listed in Table IV. Obviously, in each case, the double-pass is faster than the single pass for $n > 20$–22, in good agreement with the theoretical estimates in Eq. (34). Further, at $n=32$, the speed of the double-pass is faster than the single pass by 25%, 31%, and 31% for these three platforms, respectively, compatible with what Neuberger found in his test run with SGI O2000 [5]. Note that

TABLE IV. The CPU time (in units of second) for the single- and the double-pass algorithms to compute one column of $D^{-1}(m_q)$ for 16 quark masses versus the degree n of the rational polynomial $R^{(n-1,n)}$ in polar approximation (2).

n	P4 PC800		IBM SP2 SMP		DEC alpha XP1000	
	Double pass	Single pass	Double pass	Single pass	Double pass	Single pass
20	4922	4627	7701	7674	9921	9868
21	4930	4794	7711	7881	9924	10197
22	4918	4940	7710	8090	9931	10531
24	4921	5166	7705	8529	9929	11125
26	4920	5433	7710	8990	9929	11599
32	4918	6167	7718	10138	9926	13043

TABLE V. The number of floating-point operations and the CPU time (in units of second) for Pentium 4 (2.53 GHz) with one Gbyte Rambus (PC1066) to compute 1 column of $D^{-1}(m_q)$ for 16 quark masses versus the degree n of the Zolotarev rational polynomial $R_Z^{(n-1,n)}$.

	Double pass			Single pass			
	No. of FPO	CPU time(s)		No. of FPO	CPU time(s)		σ
n	G_2	V_2	Measured	G_1	V_1	Measured	Zolotarev
12	2.90×10^{12}	2456	2450	1.72×10^{12}	2309	2274	7×10^{-11}
13	2.90×10^{12}	2456	2452	1.75×10^{12}	2398	2404	8×10^{-12}
14	2.90×10^{12}	2456	2455	1.78×10^{12}	2485	2463	1×10^{-12}
16	2.90×10^{12}	2456	2455	1.83×10^{12}	2659	2638	3×10^{-14}
32	2.90×10^{12}	2458	2458	2.25×10^{12}	4058	4068	3×10^{-14}
34	2.90×10^{12}	2458	2458	2.30×10^{12}	4233	4245	3×10^{-14}
40	2.90×10^{12}	2458	2460	2.45×10^{12}	4759	4795	3×10^{-14}
56	2.90×10^{12}	2460	2462	2.86×10^{12}	6159	6180	3×10^{-14}
59	2.90×10^{12}	2460	2462	2.92×10^{12}	6423	6459	3×10^{-14}
60	2.90×10^{12}	2460	2460	2.95×10^{12}	6510	6544	3×10^{-14}
64	2.90×10^{12}	2460	2462	3.05×10^{12}	6860	6903	3×10^{-14}

for Pentium 4, using SSE2 code increases the speedup of the double-pass to 66% at $n=32$ (see Table III), thus making the double-pass algorithm even more favorable for P4 clusters.

At this point, it may be interesting to repeat the tests of Table III, but replacing the polar approximation (2) with the Zolotarev optimal rational approximation,

$$S_{opt}(H_w)=h_w\sum_{l=1}^{n}\frac{b_l'}{h_w^2+c_{2l-1}'}\equiv H_w R_Z^{(n-1,n)}(H_w^2),$$

$$h_w=H_w/\lambda_{min}, \tag{40}$$

where

$$R_Z^{(n-1,n)}(H_w^2)=\frac{d_0'}{\lambda_{min}}\frac{\prod_{l=1}^{n-1}(1+h_w^2/c_{2l}')}{\prod_{l=1}^{n}(1+h_w^2/c_{2l-1}')}$$

$$=\frac{1}{\lambda_{min}}\sum_{l=1}^{n}\frac{b_l'}{h_w^2+c_{2l-1}'}, \tag{41}$$

and the coefficients d_0', b_l' and c_l' are expressed in terms of Jacobian elliptic functions [6–8] with arguments depending only on n and $\lambda_{max}^2/\lambda_{min}^2$ (λ_{max} and λ_{min} are the maximum and the minimum of the eigenvalues of $|H_w|$). The results are listed in Table V.

Comparing Table III with Table V, it is clear that for the single pass with $n<32$, Zolotarev optimal approximation is better than the polar approximation, in terms of the precision of the approximation (σ). However, for the double-pass, the polar approximation seems to be as good as the Zolotarev approximation since the degree n can be pushed to a very large value, with negligible extra CPU time. In other words, with the double-pass algorithm, it does not matter which rational approximation one uses to compute $D^{-1}(m_q)$ min a gauge background. This seems to be a rather unexpected result.

V. CONCLUDING REMARKS

So far, we have restricted our discussions to the sign function with argument H_w. However, it is clear that the salient features of the double-pass algorithm are invariant for other choices of the argument, e.g., improved Wilson operator. In general, the double-pass algorithm is a powerful scheme for the matrix-vector product $R(H^2)\cdot Y$, where R can be any rational polynomial R with argument H^2 (positive definite Hermitian operator), not just for $(H^2)^{-1/2}$.

The virtue of Neuberger's double-pass algorithm is its constancy in speed and memory storage for any degree n of the rational approximation, where its constancy in speed is valid under a mild condition ($N_{site}\gg L_i$) which can be fulfilled in most cases. Further, the double-pass is faster than the single pass even for n as small as 12 (Pentium 4), and it is expected that the threshold $n_T\simeq 12$–25 for most systems. Thus, it seems that there is not much room left for the single-pass algorithm with Zolotarev approximation, unless the number of inner CG iterations is exceptionally large, which could happen if the low-lying eigenmodes of H_w^2 are not projected out and treated exactly.

Note that H_w^2 can be tridiagonalized by the conjugate gradient (6)–(11), with the unitary transformation matrix U formed by the normalized residue vectors $\{\hat{R}_j, j=0,\ldots,i\}$, and the elements of the tridiagonal matrix expressed in terms of the coefficients $\{\alpha_j,\beta_j,j=0,\ldots,i\}$ [9] (up to the tolerance of the conjugate gradient), i.e.,

$$U^{\dagger}H_w^2U\simeq\mathcal{T}, \tag{42}$$

where

TABLE VI. The number of floating-point operations and the CPU time (in units of second) for Pentium 4 (2.53 GHz) with 1 Gbyte Rambus (PC1066) to compute one column of $D^{-1}(m_q)$ versus different algorithms.

	Double-pass algorithm		Lanczos (CG) algorithm	
	Polar ($n=128$)	Zolotarev ($n=16$)	Lanczos	CG
FPO	9.49×10^{13}	9.49×10^{13}	9.54×10^{13}	9.51×10^{13}
Time (total)	94543	94632	97824	94722
Time (second pass)	46281	46303	46353	46174
σ	1×10^{-14}	1×10^{-14}	1×10^{-14}	1×10^{-14}

$$U_{kj}=\frac{(R_j)_k}{\sqrt{\langle R_j|R_j\rangle}}, \tag{43}$$

and $\mathcal{T}$ is a symmetric tridiagonal matrix with nonzero elements,

$$\mathcal{T}_{jj}=\frac{\beta_j}{\alpha_{j-1}}+\frac{1}{\alpha_j}, \tag{44}$$

$$\mathcal{T}_{j+1,j}=\mathcal{T}_{j,j+1}=-\frac{\sqrt{\beta_{j+1}}}{\alpha_j}, \quad j=0,\ldots,i. \tag{45}$$

Thus, after running the first pass of the CG loop (6)–(11), $\mathcal{T}$ can be constructed from the coefficients $\{\alpha_j,\beta_j\}$, and diagonalized by an orthogonal transformation

$$\mathcal{T}=O\Lambda\tilde{O}. \tag{46}$$

Then the matrix-vector product $(H_w^2)^{-1/2}Y$ can be evaluated as

$$\frac{1}{\sqrt{H_w^2}}Y\simeq UO\frac{1}{\sqrt{\Lambda}}\tilde{O}U^\dagger Y=\sum_{j=0}^{i} l_j R_j, \tag{47}$$

where

$$l_j=\sum_{m=0}^{i} O_{jm}\frac{1}{\sqrt{\lambda_m}}O_{0m}\sqrt{\frac{\langle R_0|R_0\rangle}{\langle R_j|R_j\rangle}}. \tag{48}$$

Here the summation in the rhs of Eq. (47) is obtained by running the second pass of the CG loop {Eqs. (6),(7),(9), and (11)}, and adding l_jR_j to the sum successively from $j=0$ to i.

It is well known that (any positive definite Hermitian matrix) H_w^2 can be tridiagonalized by Lanczos iteration [9,10] as well as the conjugate gradient. The connection between the Lanczos iteration and the conjugate gradient for the tridiagonalization of a positive definite Hermitian matrix has been well established [9], and both have almost the same performance in practice. In Ref. [11], the Lanczos approach was proposed for the matrix-vector product $(H_w^2)^{-1/2}Y$, and its variant (replacing Lanczos iteration with the conjugate gradient) was used in Ref. [12].

The only difference between the Lanczos (CG) algorithm and Neuberger's double-pass algorithm is the diagonalization of the tridiagonal matrix $\mathcal{T}$ and the computation of the coefficients $\{l_j\}$, Eq. (48), in the former versus the computation of the coefficients $\{c_j\}$, Eq. (17), in the latter. Since the number of floating-point operations for the diagonalization of a symmetric tridiagonal matrix $\mathcal{T}$ is $\simeq 3L_i^3$ (where L_i is the number of iterations of the inner CG loop, or the size of $\mathcal{T}$), it is compatible with that of computing the coefficients $\{c_j\}$, i.e., the last term on the rhs of Eq. (20). Thus we expect that the performance (speed and accuracy) of Lanczos (CG) algorithm and Neuberger's double-pass algorithm are compatible.

In Table VI, we compare the Lanczos (CG) algorithm with Neuberger's double-pass algorithm, by computing one column of $D^{-1}(m_q)$ (for 16 bare quark masses) on the $16^3\times32$ lattice with SU(3) gauge configuration generated by the Wilson gauge action at $\beta=6.0$. Other parameters are $m_0=1.30$, $n_{ev}=20$ (the number of projected eigenmodes), $\lambda_{max}/\lambda_{min}=6.260/0.215$ (after projection), and the tolerances for the outer and inner CG (Lanczos) loops are 1.0×10^{-11} and 2.0×10^{-12}, respectively. The number of iterations for the outer CG loop is $L_o=347$, while the average number of iterations for the inner CG loop is ~300. Evidently, these seemingly different algorithms have almost the same speed as well as accuracy (σ).

Thus, for quenched lattice QCD, one has several compatible options to compute the quenched quark propagator,

$$(D_c+m_q)^{-1}=(1-rm_q)^{-1}[D^{-1}(m_q)-r], \quad r=\frac{1}{2m_0}, \tag{49}$$

even though we have chosen Neuberger's double-pass algorithm to solve $D^{-1}(m_q)$ in our recent investigation [13]. Nevertheless, for lattice QCD with dynamical quarks, the quark determinant $\det D(m_q)$ could not be computed directly with existing algorithms and computers. If $\det D(m_q)$ is incorporated through the dynamics of $2n$ pseudofermion fields (where n can be regarded as the degree n in the rational polynomial $R^{(n-1,n)}$), then an additional degree of freedom (or the fifth dimension with $N_s=2n$ sites) has to be introduced. Thus a relevant question is how to reproduce $\det D(m_q)$ accurately with the minimal N_s. A solution has been presented in Ref. [14]. On the other hand, it would be interesting to see whether there is an algorithm to drive the dynamics of these N_s pseudofermion fields such that the cost is almost independent of $N_s=2n$.

ACKNOWLEDGMENT

T.W.C. would like to thank Herbert Neuberger and Rajamani Narayanan for a brief but stimulating discussion at the Institute for Advanced Study, during a visit in November 2002. This work was supported in part by the National Science Council, ROC, under Grant No. NSC91-2112-M002-025.

[1] H. Neuberger, Phys. Rev. Lett. **81**, 4060 (1998).

[2] H. Neuberger, Phys. Lett. B **417**, 141 (1998).

[3] A. Frommer, B. Nockel, S. Gusken, T. Lippert, and K. Schilling, Int. J. Mod. Phys. C **6**, 627 (1995).

[4] B. Jegerlehner, e-print hep-lat/9612014.

[5] H. Neuberger, Int. J. Mod. Phys. C **10**, 1051 (1999).

[6] N.I. Akhiezer, *Theory of Approximation* (Dover, New York, 1992); *Elements of the Theory of Elliptic Functions*, Translations of Mathematical Monographs Vol. 79 (American Mathematical Society, Providence, RI, 1990).

[7] J. van den Eshof, A. Frommer, T. Lippert, K. Schilling, and H.A. van der Vorst, Nucl. Phys. B, Proc. Suppl. **106A**, 1070 (2002).

[8] T.W. Chiu, T.H. Hsieh, C.H. Huang, and T.R. Huang, Phys. Rev. D **66**, 114502 (2002).

[9] See, for example, Eq. (10. 2-14) on page 371 of G.H. Golub and C.F. Van Loan, *Matrix Computations*, (Johns Hopkins University Press, Baltimore, 1983).

[10] See, for example, J. Cullum and R.A. Willoughby, *Lanczos Algorithms for Large Symmetric Eigenvalue Computations* (Birkhauser, Boston 1985), Vol. I.

[11] A. Borici, J. Comput. Phys. **162**, 123 (2000).

[12] R.V. Gavai, S. Gupta, and R. Lacaze, Phys. Rev. D **65**, 094504 (2002).

[13] T.W. Chiu and T.H. Hsieh, Nucl. Phys. B **673**, 217 (2003).

[14] T.W. Chiu, Phys. Rev. Lett. **90**, 071601 (2003); Phys. Lett. B **552**, 97 (2003); e-print hep-lat/0303008.