1-1-2012

A linear time complexity solver for lattice quantum field theory computations

Dieter Beaven
University of Wollongong, djdb684@uow.edu.au

John Fulcher
University of Wollongong, john@uow.edu.au

Chao Zhang
University of Wollongong, czhang@uow.edu.au

A Linear Time Complexity Solver for Lattice Quantum Field Theory Computations

Dieter Beaven, John Fulcher, and Chao Zhang.

***Abstract*—Lattice Quantum Chromodynamics is used to investigate the behavior of quarks under the influence of the Strong Nuclear force. The computer implementation requires the solution of square sparse matrices with the number of rows up to the 100's of millions, and this represents the major computational factor with regards to overall runtime. In this paper we present verification of an algorithm that grows only linearly with respect to matrix size in terms of the computing resources required. Once realistically sized calculations can be done on commonly available hardware, this opens the door to the investigation of quantum fields in other areas such as condensed matter physics and nanotechnology.**

***Index Terms*—Linear Algebra, Algebraic Multigrid, Lattice Quantum Field Theory, High Performance Computing.**

I. Introduction

THE analysis of Quantum Chromodynamics, QCD, requires the solution of very large matrices derived from the discretization of the Dirac Equation over a four-dimensional spacetime lattice. Lattice QCD, LQCD, is a necessary tool for the understanding of the Strong Force within the Standard Model of Particle Physics since analytic techniques used for Quantum Electrodynamics fail to provide useful results due to the large coupling strengths involved.

LQCD was first successfully formulated in 1974 by K. G. Wilson[1], but only since the 1990's have supercomputers gained sufficient performance to provide meaningful results. Larger lattices with finer grid spacings are still sought after in order to improve the agreement with experiment, and in particular, investigate the low-mass region that may be able to provide a first-principles calculation of the proton mass [2].

Lattice sizes currently used are typically around 48 grid-points per dimension, which raised to the power of 4 to model the 4 dimensions of spacetime, and multiplied by the 24 wavefunction components per lattice site, can result in matrices with rank of over 100 million requiring solution.

Manuscript received December 8, 2011; revised Jan 28, 2012.
Dieter Beaven is with the School of Computer Science and Software Engineering, and with the School of Engineering Physics, at the University of Wollongong, Australia (phone: +61 2-4221-3606; fax: +612-4221-4843; e-mail: djdb684@ uowmail.edu.au).
John Fulcher is with the School of Computer Science and Software Engineering, at the University of Wollongong, Australia (e-mail: john@ uow.edu.au).
Chao Zhang is with the School of Engineering Physics, at the University of Wollongong, Australia (e-mail: czhang@ uow.edu.au).

The actual algorithm used to extract observable data is based upon the Hybrid Monte Carlo algorithm, and requires updates based upon the new configurations resulting from the solutions[3]. The Dirac-Wilson matrices are constructed from the following formula:

$$(4+m)\delta_{x,y}+\Sigma_{\mu}(P^{-\mu}\otimes U^{\mu}_{x,y}\delta_{x+\mu,y}+P^{+\mu}\otimes U^{\mu\dagger}_{x,y}\delta_{x-\mu,y})$$

$P^{-\mu}$ are constant projection matrices, δ are constant Dirac deltas, and $U^{\mu}_{x,y}$ are the color gauge fields that model the strong nuclear force. Each lattice site x uses one in each of the eight space-time directions μ to neighboring sites y. These are generated at random for each step in the Hybrid Monte Carlo algorithm. A mass value of m = -0.4 was used for this paper. Since matrix elements $A_{x,y}$ depend only on links to eight nearest neighbor sites, the matrix has a well defined and constant sparse structure.

II. Krylov Solvers

Non-stationary iterative Krylov methods are particularly suitable as solvers since they do not require the explicit construction of the entire matrix, it just needs an efficient algorithm for the BLAS level-2 Matrix-Vector product. The algorithm to solve Ax=b is shown below [4].

$r^{(0)} = b - \mathrm{A}\, x^{(0)}$
for $i = 1,2, \ldots$
  solve multigrid preconditioner $\mathrm{M}\, z^{(i-1)} = r^{(i-1)}$
  $\rho^{(i-1)} = r^{(i-1)T}\, z^{(i-1)}$
  if $i = 1$
    $p^{(1)} = z^{(0)}$
  else
    $\beta^{(i-1)} = \rho^{(i-1)} / \rho^{(i-2)}$
    $p^{(i)} = z^{(i-1)} + \beta^{(i-1)}\, p^{(i-1)}$
  endif
  $q^{(i)} = \mathrm{A}\, p^{(i)}$
  $\alpha^{(i)} = \rho^{(i-1)} / p^{(i)T}\, q^{(i)}$
  $x^{(i)} = x^{(i-1)} + \alpha^{(i)}\, p^{(i)}$
  $r^{(i)} = r^{(i-1)} - \alpha^{(i)}\, q^{(i)}$
  test residual, $r^{(i)}$, against required tolerance
end loop

Conjugate Gradients can be distributed across clusters fairly efficiently with mainly nearest neighbor communication. The primary bottleneck turns out to be the global scalar-

ISBN: 978-988-19251-9-0
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

reductions alpha and beta, required to calculate iteration updates. Each processor can calculate a sub-part of the scalar product relating to the matrix rows held locally, but those sub-parts need collecting and broadcasting back to all processors before the algorithm can further continue.

A particular feature of the Dirac Equation though, is that it is a non-symmetric matrix, so the simplest method of Conjugate Gradients is not directly applicable. Common variants for non-symmetric matrices are Generalised Minimal Residual (GMRES), Conjugate Gradient Normal Equations (CGNE, CGNR) or the BiConjugate Gradient methods (BiCG, BiCGStab) [5].

III. Research

The general time complexity of matrix solvers is $O(N^3)$ where N is the matrix rank. Conjugate Gradient converges on the exact solution after $O(N^3)$ iterations, but is normally stopped after sufficient accuracy has been achieved. In practice the time complexity achieved is $O(N^2)$. Convergence is further accelerated by preconditioning the linear system to reduce its spectral radius. Many different preconditioners are usable, with optimized preconditioners dependent on the eigenvalue structure of the matrix[6].

LQCD calculations are thus performed with a core algorithm that has an effective time-complexity of $O(N^2)$, and this has impeded progress with regards to increasing the lattice grid sizes and finer lattice spacings. MPI clusters of 100's or 1000's of CPU nodes are currently being used and may take months to run applications [7]. With large data sets distributed over a network, communication overhead also becomes a constraining factor.

Graphical Processing Units, GPUs, have been investigated as co-processor accelerators, and have shown promising results with regards to accelerating existing algorithms [8]. With hundreds of execution threads, GPUs are highly effective at floating-point parallel processing. However, they are limited by the size of their internal memory and the IO bandwidth for updates of matrix data.

Field Programmable Gate Arrays, FPGAs, also provide an alternative to commodity CPUs. Whilst FPGA speeds will never match GPU floating-point speeds nor even typical CPU's, they do offer the ability to customize the data-paths between fast embedded multipliers and directly adjacent Block-RAM. FPGAs can provide customized ALUs with directly connected Level 1 cache at a purchase cost as low as US$0.50 per multiplier, and with the additional benefit of low on-going operational costs [9].

In order to dramatically increase the lattice size for QCD, it becomes necessary to look at the algorithms being used, and options to tailor them for distributed architectures. The two key features that would enable even larger lattices are:

i) minimal communication between computing nodes;
ii) linear complexity algorithms for the matrix solver.

Matrix inversion, and hence solution of Ax = b, is an inherently global operation. Hence the default $O(N^3)$ time complexity and the difficulty with communication bandwidth for matrices that require data distributed across a computational cluster.

It has been suggested that Domain Decomposition and Multigrid techniques may provide the answer [10]. Domain Decomposition would divide the matrix into sub-matrices and solve each in parallel, before recombining into a final solution. This paper will present results to suggest Multigrid on its own can provide an effective way forward, since a design can be provided that satisfies both counts above: O(N) time complexity with constant communication.

The sparse structure of the Dirac Equation can be exploited to allow O(N) time, memory, and communication increases with matrix order N.. The design must eliminate any $O(N^2)$ or higher complexities, since these would grow to swamp all other O(N) performances.

IV. Design

The idea behind Multigrid is to take advantage of the fact that stationary iterative solvers such as Jacobi, Gauss-Seidel and SOR are excellent at damping the high-frequency components of a solution's error. Specifically, repeated iterations will reduce the error in the initial guess by a factor of a component frequency's eigenvalue. For eigenvalues less than zero, the error will diminish, with larger frequencies that have smaller eigenvalues, disappearing faster.

Stationary iterative solvers may rapidly diminish high frequency components, but low frequency components, with eigenvalues near (or larger than) one, take longer, often resulting in $O(N^4)$ performance. These solvers quickly smooth out initial guesses, but are two slow to overall convergence.

Multigrid combines lattice data together to create coarser grids [11]. What may be a low frequency on a fine grid, will be a high frequency on a coarse grid. By applying a fixed number of smoother-iterations at each grid level, one ends up with a smoothing effect at all frequencies. Since a fixed number of iterations are applied at each grid-level, and each grid-level is smaller than the previous, the total work is a geometric sum that totals a value proportional to the original matrix size N. Multigrid is inherently a linear time-complexity algorithm.

Multigrid is often applied to a Krylov subspace method as a preconditioning step, and results in rapid convergence of the Krylov algorithm with an almost constant number of Krylov iterations. The combination of pre-conditioner smoothing and Krylov subspace traversal provides an extremely effective algorithm.

The results presented here will demonstrate the practical linearity of the multigrid preconditioned Krylov methods, but it is also important to note that communication between distributed processors can also effectively be made time-constant with respect to increasing matrix size. With P processors and N matrix rows, the method can be partitioned into N/P parallel segments. The Jacobi solver can be run completely in parallel for the preconditioning stages, whilst the Krylov loops only require nearest neighbor updates across the solution vector boundaries and

ISBN: 978-988-19251-9-0
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

some inner-product scalar-reduction. When the solution vector x is distributed into P processors, neighbors will need to communicate a fixed number of updates, but the communication overhead between nearest neighbors only, will not increase with the total number of processors: each processor will always have two nearest neighbors, and can remain oblivious to additional computing nodes.

With standard Krylov algorithms the global scalar reduce can become a major bottleneck since the algorithm is unable to proceed until all processors have contributed their segment to the global sum, and then have had the result broadcast back to them. With a multigrid preconditioned Krylov method we will produce results that show the number of Krylov loops is small and essentially constant, hence the global-reduce bottleneck is of fixed time-complexity, and actually becomes less significant as N increases.

With the multigrid preconditioner having linear time complexity, and the Krylov iterations fixed in number, the remaining time complexity is in the Krylov product Ax. Nominally Ax is $O(N^2)$ but the sparse structure of A reduces this to linear O(N): namely each row has a fixed number of elements k, so the actual Krylov product has time complexity O(kN) ~ O(N).

One aspect that bears attention is that in current typical implementations, it is usual that the matrix is never stored, but generated on-the-fly as the Ax terms are required. Generating the Dirac-Wilson matrix is in itself a O(N) algorithm, which could introduce an $O(N^2)$ dependency when the matrix is regenerated O(N) times. Whilst the multigrid algorithm just presented works on the basis of a fixed number of iterations, the design facilitates the one-off generation and storage of the Dirac-Wilson matrix. Since each processor only needs a subset number (N/P) of rows of the matrix elements that it uses for solution vector updates, the matrix storage is distributed over a cluster without the need to ever communicate matrix elements between processors.

V. Results

The code is written in C++ and was compiled and tested on two machines:

i) MSVC10, Win7 Desktop; Intel Duo E860, 3.33GHz.
ii) GCC, GNU Linux; AMD Operton 2356, 2.3Ghz.

The Windows7 machine could access 2GB of RAM, whilst the Linux machine had 16GB of real memory available. The results are for a single active processor running a single thread in order to investigate the linearity of the solver design. Once optimized for a single process, adding parallelization will provide the necessary additional speedup.

The current implementation builds a matrix in Compressed Row Storage format [4], then passes that into the Algebraic Multigrid system. The matrix is not limited to an LQCD matrix, but can be any general matrix, and any Krylov based algorithm can be selected for the main loop based upon knowledge about the structure of the matrix requiring solution. Alternative preconditioners are also available for comparison against multigrid.

Initial development was undertaken with a simpler 1-dimensional Laplacian partial differential equation. The discretization results in a symmetric tri-diagonal matrix, with diagonal elements a small fraction greater than the sum of the off-diagonals (i.e. just diagonally dominant).

$$\frac{d^2u}{dx^2} - 1000e^{20(x-0.5)^2}u(x) = -\sin(2\pi x)(4\pi^2 - 1000e^{20(x-0.5)^2})$$

Tables 1 and 2 compare the simple Aii^{-1} (inverse diagonals) preconditioning against solution of the Laplacian matrix with Multigrid Preconditioned Conjugate Gradient.

N	iterations	time/s
12,000	6001	1.2
24,000	11564	4.6
36,000	15855	9.4
48,000	21403	16.8
72,000	32534	53.0
96,000	41654	153.3
120,000	43843	196.7
144,000	44611	254.0

Table 1: Inverse Diagonals Preconditioning.

N	iterations	time/s
12,000	22	0.3
24,000	24	0.2
36,000	21	0.3
48,000	25	0.3
72,000	22	0.5
96,000	25	0.7
120,000	22	0.7
144,000	21	0.9

Table 2 :Multigrid Preconditioning.

It can be seen that the time complexity of the Conjugate Gradient solver with the simple inverse-diagonals pre-conditioner is approximately $O(N^2)$, which consists of O(N) for the number of row elements increasing, multiplied by O(N) for the number of loop-iterations taken for convergence: the number of iterations required is increasing at a rate of approximately ½ N. For dense matrices the third power of O(N) to give $O(N^3)$ comes from the increase in column elements, but for sparse matrices this is often a fixed number: here it is 3 from the tri-diagonal structure, for LQCD it is also fixed at 97 originating from the 8 nearest neighbors lattice sites.

The performance of Multigrid-preconditioned conjugate gradient is in stark contrast, where convergence is achieved within a small and constant number of iterative loops (i.e.

ISBN: 978-988-19251-9-0
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

within less than or equal to 25 loops for all matrix sizes presented). The only source of time-complexity growth is the number of rows that require evaluating: O(N). With a parallel implementation, these rows can be distributed across a cluster, and only nearest neighbors require communication to update overlapping vector solution value. The potential bottleneck of the global scalar-reduce required by the Conjugate Gradient algorithm is also kept under control since it is needed a couple of times per loop-iteration; so for this example no more than a constant 50 number of calls. For the Multigrid preconditioned Conjugate Gradient the sole time-complexity growth is the number of rows N of the matrix, and those N rows can be independently distributed over a cluster of P processors.

Table 3 shows the linearity of the algorithm for the Laplacian matrix up to rank 45 million. It can be seen that both the time taken and the memory usage both double as the matrix size N doubles, to give linear performance for both time and memory. The gradual increase in iterations required can be traced to the fact that as matrix size increases the number of individual element errors contributing to the total error (residual) is also steadily increasing in a linear way. That is, twice as many elements in the solution vector gives twice the overall residual error (even if the individual element error is the same). Thus an extra loop or so is required to get the residual error below the termination tolerance criterion.

N	iterations	time/s	memory/(% of 16GB)
50,000	16	0.5	0.1
100,000	17	1.0	0.2
200,000	18	2.2	0.4
400,000	19	4.6	0.9
800,000	20	9.7	1.8
1,600,000	22	20.6	3.5
3,200,000	23	42.2	7.0
6,400,000	25	92.5	14.0
12,000,000	25	175.0	26.2
24,000,000	25	359.8	52.3
36,000,000	24	538.0	78.5
45,000,000	23	670.3	97.3

Table 3: Multigrid upto matrix size N = 45 million.

Since multigrid is such as effective preconditioner, the residual often drops by several magnitudes per Conjugate Gradient loop, thus preventing the residual error from introducing a linear increase in the required iteration loops, as can potentially happen for other types of Conjugate Gradient algorithm. In contrast to stationary solvers such as Jacobi which tend to monotonically smooth-out errors, Conjugate Gradient is a search-algorithm and convergence can be seen to vary; sometimes stalling, sometimes finding a plateau, sometimes increasing again. It can be seen however, that for a given error tolerance in the result, multigrid preconditioning tends to need only a small number of CG loops, and within a fixed upper limit (25 in the case of figure 1).

Results for LQCD are now presented to show that the Multigrid method and its linearity performance also work for the more complicated Dirac-Wilson matrices. The Dirac-Wilson matrix elements represent probability amplitudes for the quantum wavefunction to propagate from spin-color states (4 times 3 complex numbers) at one site to the 12 complex components at each of the 8 adjacent lattice sites; plus diagonal terms related to mass. The complex numbers are split between even and odd rows, to make a total of ½ * (2 * 4 * 3 * 8) + 1, equals 97 elements per row. This is still very sparse in comparison to the total matrix order of millions, but is over 20 times larger than the previous tri-diagonal Laplacian.

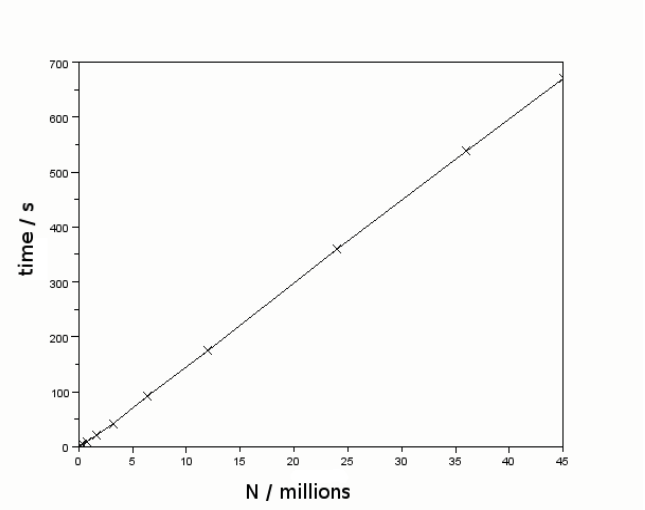

Figure 1: Multigrid solver time versus Laplace matrix size.

For the purposes of this investigation the non-sysmmetry of the Dirac-Wilson matrix will be handled by the simplest approach with respect to Krylov Subspace algorithms, namely the Conjugate Gradient on Normal Equations, the CGNR variant where the matrix system Ax = b is left-multiplied by the matrix transpose to give:

$$A^TAx = A^Tb$$

This is the simplest approach since A^TA is guaranteed to be symmetric for non-singular matrices, and thus the regular Conjugate Gradient algorithm can be applied. There are several drawbacks, including the extra computations required, and the sparsity is affected due to the cross-multiplications requiring more memory for storage of the resulting A^TA matrix. The most troublesome drawback though, is the fact that the convergence is reduced due to the A^TA matrix condition-number being the square of the original matrix.

If the multigrid design can survive these significant drawbacks, the authors are optimistic that the more sophisticated variants will mark further improvement. In intial testing multigrid preconditioned BiCGStab with Symmetric-SOR smoothing has also demonstrated converge with linear time complexity and requires only three Krylov iterations for the same level of accuracy.

The convergence of the Dirac-Wilson matrix was found to be very sensitive to the relaxation parameter used. The

ISBN: 978-988-19251-9-0
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

stationary smoother used for the multigrid pre-conditioner was the over-relaxed variant of the Jacobi Iteration. Over-relaxing does not affect the embarrassingly parallel nature of the Jacobi algorithm and a relaxation parameter of around 0.17 (1.0/6.0) was found to obtain convergence. Unlike the Laplacian examples, the Dirac-Wilson matrices typically fail the diagonal dominance criteria, so un-relaxed Jacobi (omega = 1) did not converge at all.

Lattice Width	Matrix Rank	time/s	RAM/(% of 16GB)	CG Loops	Output Residual
4	6,144	9.9	0.8	40	1.0E-06
5	15,000	24.0	1.7	40	1.1E-05
6	31,104	47.7	3.5	40	6.5E-05
7	57,624	96.9	6.5	40	2.0E-04
8	98,304	163.1	11.1	40	2.6E-04
9	157,464	255.3	17.8	40	2.5E-04
10	240,000	371.3	27.1	40	3.1E-04
12	497,664	834.2	56.1	40	1.8E-04

Table 4: Dirac-Wilson with multigrid (fixed iterations).

In table 4 one can see that both the memory usage and time taken increase linearly with respect to matrix rank. For test 4 the number of Conjugate Gradient iterations was set to 40 for all matrices, whereas the following table 5 shows similar results when the result-tolerance was used as the terminating criterion (more realistic in practice).

Lattice Width	Matrix Rank	time/s	RAM/(% of 16GB)	CG loops	Output Residual
4	6,144	7.5	0.8	23	2.3E-04
5	15,000	20.8	1.7	28	2.7E-04
6	31,104	45.6	3.5	32	3.1E-04
7	57,624	86.6	6.5	36	3.3E-04
8	98,304	166.6	11.1	39	2.6E-04
9	157,464	255.6	17.8	37	3.4E-04
10	240,000	382.1	27.1	38	3.5E-04
12	497,664	768.8	56.1	36	3.1E-04

Table 5: Dirac-Wilson with multigrid (fixed tolerance).

The memory performance is identical as one would expect, and again the time performance is linear, with a slight improvement since 40 loops were generally not required for the specified terminating tolerance of 1e-08. Note the terminating tolerance criterion for preconditioned Conjugate Gradient is generally related to the magnitude of the preconditioned residual, which is typically of order of the square of the actual output residual.

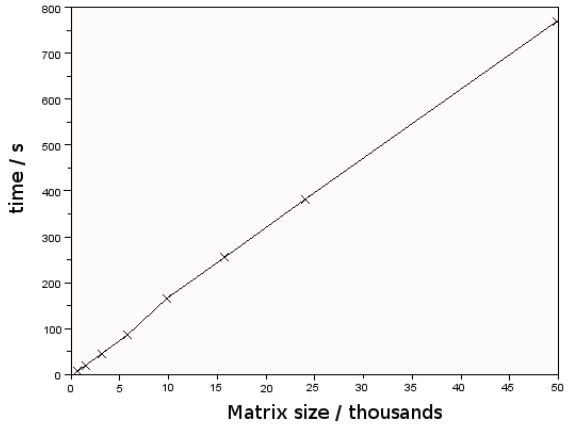

Figure 2: Multigrid solver time versus LQCD matrix size.

Since the time complexity performance is linear, extrapolating to a lattice width of 24 for a matrix rank of 8 million, the estimate is 12,300 seconds (3½ hours). Compared to the GPU results of 13.6 seconds in table 6, this appears slow, but then consider this algorithm is almost 100% scalable. Assuming 90% scalability and using 200 processors (192 in a GPU), the multigrid estimate reduces to 68 seconds. Assuming 90% scalability with the 9000 processors as available to some full scale systems, we now have an estimated time to solution of 1.5 seconds.

Lattice Width	Matrix Rank	Memory/MB	time/s	CG loops
8	98,304	1	0.5	155
12	497,664	5	0.9	177
16	1,572,864	15	2.4	248
20	3,840,000	36	5.4	263
24	7,962,624	74	13.6	317

Table 6: GPU Performance.

A key feature of multigrid, also investigated by the authors of the GPU code[2], is that multigrid is robust at critical lighter masses, where the ordinary Conjugate Gradient algorithms have problems converging. A key feature of the multigrid program presented here is that it is row orientated to enable embarrassingly parallel scalability.

Figure 3 below shows the output of the multigrid algorithm for the case of lattice width equal to 12 with fixed iteration count (from table 3a).

```
Dirac.rank() = 497664
A.non_zeros  = 48273408
A.per_row_av = 97
Relative: 0.0   Order: 497664   Level: 0
Relative: 0.00628537    Order: 3128     Level: 1
Relative: 0.0249361     Order: 78       Level: 2
Relative: 0.025641      Order: 2        Level: 3
Relative: 0.5           Order: 1        Level: 4
Relative: 1             Order: 1        Level: 5
-------------------------------------------------
          Conjugate Gradient Run
-------------------------------------------------
INPUT RESIDUAL-NORM: 1
MgPCGNR Loop: 10   dot(mr,r) = 0.000125487
MgPCGNR Loop: 20   dot(mr,r) = 1.94623e-06
MgPCGNR Loop: 30   dot(mr,r) = 9.0124e-08
MgPCGNR Loop: 40  dot(mr,r) = 2.68616e-09
ITERATIONS: 40
OUTPUT RESIDUAL-NORM: 0.000180507

 x = [ 0.276214; 4.39952e-08; -0.00183957; -0.00117994;

 b = [ 1; 0; 0; 0;
Ax = [ 1; 4.45647e-08; -5.38577e-07; -6.51208e-08;

Residual = 0.000180507


real    13m54.164s
user    13m47.589s
sys     0m6.422s
```

Figure 3. Showing the program's output log.

The size of the Dirac matrix constructed is reported, followed by statistics for the multigrid grid-level construction. Relative refers to the ratio of grid-sizes between levels, with order being the matrix rank at the given level. The first few values of the solution vector x are displayed, along with b and a sanity check of Ax to confirm a solution was found within the range indicated by the reported residuals.

ISBN: 978-988-19251-9-0
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

VI. Memory

Memory usage is far greater than actually required for several reasons, the main one being the creation of the A^TA matrix required for the Conjugate Gradient Normalized algorithm variant. Table 7 shows the time and memory performance of the Dirac-Wilson matrix generator component of the code alone.

Lattice Width	Matrix Rank	time/s	RAM/(% of 16GB)
10	240,000	6.4	2.3
12	497,664	13.6	4.8
14	921,984	25.1	9.0
16	1,572,864	42.8	15.3
20	3,840,000	104.5	37.3
24	7,962,624	215.7	77.3
25	9,375,000	259.7	91.1

Table 7: Dirac-Wilson Matrix Generator Performance.

The code has been optimized to have linear behavior in both time complexity and memory requirements, as well as being 100% scalable with regards to parallelization. With the matrix rows distributed across P processors, each processor needs only to generate the N/P rows of matrix elements it requires. Even with matrix ranks of a billion, the storage requirements would be relatively modest for a cluster with 1000's of processors and Terabytes of RAM.

If it was undesirable to store the Dirac-Wilson matrix, with calls to Ax limited to a constant factor of the number of Conjugate Gradient loops, a linear matrix generator would not cause an $O(N^2)$ blowout in time complexity for the overall calculation. With non-multigrid conjugate gradient, the $O(N^2)$ can be related to the fact that Ax products are O(N), and with iterations to convergence are also being O(N). The tradeoff between saving memory and saving time is configurable here, since both behave in a linear manner. For GPU and FPGA implementations saving memory is probably more important, but CPU clusters would probably have sufficient RAM and prefer the savings in time.

The storage requirement for multigrid is again a geometric sum, since each coarser sub-grid requires some fraction less in space than the previous finer grid. In the tri-diagonal Laplacian case, each sub-grid has exactly half the number of grid-points of the previous level: the geometric sum is exactly twice the storage of the original matrix alone. For LQCD, the Algebraic Multigrid algorithm coarsens the grid much faster, as can be seen from the data in figure 7 (initially by a factor of over 100 corresponding to the fact there are 97 elements per row).

VII. Conclusion

The time complexity and memory usage of the Multigrid Preconditioned Conjugate Gradient algorithm have been empirically verified to be linear in agreement with theoretical predictions, and results shown to converge for the Dirac-Wilson matrix as required.

At all levels of the software design efforts have been made to keep the code embarrassingly parallel by selecting, as far as possible, algorithms that are inherently scalable without compromising the linear memory usage and time complexity.

Potentially expensive communication bottlenecks that might degrade scalability for the parallel version have been kept under control by implementing algorithms that are matrix-row orientated, and allowing solution vector updates that are almost entirely independent of any other row. For the limited areas of inter-row dependencies, the communication requirements have been designed to be at a constant bandwidth, effectively independent of matrix size.

Having verified and validated the sequential version of the algorithm for LQCD, the next step will be to implement the parallel versions for both CGNR and BiCGStab variants with multigrid preconditioning.

References

[1] Wilson, K. G. (1974). "Confinement of quarks." Physical Review D 10(8): 2445-2459.
[2] Brannick, J., R. C. Brower, et al. (2008). "Adaptive Multigrid Algorithm for Lattice QCD." Physical review letters 100(4).
[3] Schaefer, S. (2011). Algorithms for lattice QCD: Progress and challenges, American Institute of Physics.
[4] R. Barrett, M. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. van der Vorst, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, SIAM, Philadelphia, PA (1994).
[5] H. A. van der Vorst. 1992. BI-CGSTAB: a fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems. SIAM J. Sci. Stat. Comput. 13, 2 (March 1992), 631-644.
[6] Saad, Y. and H. A. van der Vorst (2000). "Iterative solution of linear systems in the 20th century." Journal of Computational and Applied Mathematics 123(1-2): 1-33.
[7] T. Streuer and H. Stuuben, Simulations of QCD in the Era of Sustained Tflop/s Computing, in C. Bischof, M. Bruckner, P. Gibbon, G. Goubert, T. Lippert, B. Mohr, F. Peters (Eds.), Parallel Computing: Architectures, Algorithms and Applications, NIC Series, Vol. 38 (2007), 535-542.
[8] M. A. Clark, R. Babich, K. Barros, R. C. Brower, and C. Rebbi, "Solving Lattice QCD systems of equations using mixed precision solvers on GPUs," Comput. Phys. Commun. 181 (2010) 1517—1528
[9] Asano, S., T. Maruyama, et al. (2009). Performance comparison of FPGA, GPU and CPU in image processing. International Conference on Field Programmable Logic and Applications, 2009. FPL 2009.
[10] M. Luscher, "Solution of the Dirac equation in lattice QCD using a domain decomposition method," Comput.Phys.Commun. 156 (2004) 209—220
[11] Brandt, A., McCormick, S., and Ruge, J.: Algebraic multigrid (AMG) for sparse matrix equations. In Evans, D., editor, Sparsity and Its Applications. Cambridge University Press, (1984).

ISBN: 978-988-19251-9-0
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)