Parallel Conjugate Gradient

Sourendu Gupta, TIFR,
April 2002, HRI, Allahabad

1. Motivation: lattice Fermion Monte Carlo
2. Formal problem: solve $Ax = b$
3. The conjugate gradient algorithm
4. Convergence and preconditioning
5. Parallelising the algorithm
6. Architecture dependence

Relativistic Fermions

The Dirac equation is a set of four first order differential equations which can be written as—

$$\gamma_\mu(\partial_\mu - ieA_\mu)\psi(x) + m\psi(x) = 0$$

where m is the mass of the Fermion, A_μ are gauge fields, ∂_μ denotes a first derivative with respect to the coordinate x_μ. ψ is a four component complex vector and γ_μ are 4×4 matrices which satisfy the conditions $\gamma_\mu\gamma_\nu + \gamma_\nu\gamma_\mu = 2g_{\mu,\nu}$.

I will work in Euclidean space-time, where $g_{\mu\nu}$ is diagonal and all component are unity. I will take the dimension of space-time to be $d = 4$.

I will work on a finite lattice and put either periodic or anti-periodic boundary conditions (PBC/ABC).

Free Relativistic Lattice Fermions

If the lattice spacing is a, then derivatives become

$$\partial_\mu \psi(x) \to \frac{1}{a}[\psi(x + \widehat{x}_\mu) - \psi(x)],$$

where $\widehat{x}_\mu$ is the unit vector in the direction μ.

I will choose length units such that $a = 1$.

At each point on the lattice there is a 4 component complex vector. Now put these together into a really big vector

$$\begin{aligned} \Psi \;=\; & (\psi(1), \psi(1+\widehat{x}_1), \cdots, \psi(1+N_1\widehat{x}_1), \\ & \psi(1+\widehat{x}_2), \cdots, \psi(1+N_2\widehat{x}_2), \cdots)^T \end{aligned}$$

The number of components is $N = 4 \times N_1 \times N_2 \times \cdots$.

The Dirac equation is $D\Psi = 0$, where each row of D has only $4(d+1)$ non-zero elements. The diagonal elements are $D(x,x) = m - d$. The only non-zero off-diagonal elements are $D(x, x+\widehat{x}_\mu) = 1$. The matrix is sparse.

Lattice Quantum Electrodynamics

The Monte Carlo procedure is meant to evaluate the integral

$$Z = \int \mathcal{D}U \, \mathrm{Det}D \, \exp[-S(U)].$$

I will take the example of $U(1)$ gauge theory, where each $U = \exp(ieA_\mu)$ is a complex number of unit modulus placed on the links of the lattice. This is a lattice version of quantum electrodynamics. The Fermions are electrons and their anti-particles, the positrons. The operator

$$(\partial_\mu - ieA_\mu) \rightarrow U(x, \hat{x}_\mu)\psi(x + \hat{x}_\mu) - \psi(x).$$

As a result, D has the same diagonal elements as the free Fermions. But the non-zero off diagonal elements are $D(x, x + \hat{x}_\mu) = U(x, \hat{x}_\mu)$.

We have neglected complications arising from Fermion doubling.

Fermion Monte Carlo

Now Z can be evaluated by a Metropolis algorithm. The weight factor of each configuration of U on the lattice is the integrand. To find the weight, we evaluate the determinant by the trick

$$\sqrt{\mathrm{Det}A} = \int \mathcal{D}\Phi \exp[-\Phi^\dagger A^{-1}\Phi],$$

where Φ are auxiliary Boson fields which are usually called 'pseudo-Fermion' fields.

In practice, we use

$$\mathrm{Det}DD^\dagger = \int \mathcal{D}\Phi (D^\dagger)^{-1} D^{-1}\Phi.$$

In order to do this, we select a random vector Φ and solve for X in

$$DX = \Phi.$$

This is the core numerical problem we solve.

The Conjugate Gradient Algorithm

To solve the linear system $Ax = b$ where A is a Hermitean matrix, we use the following algorithm:

```
 1. Choose x_0 and compute r_0 = b - Ax_0
 2. Compute p_0 = 0 and rho_{-2} = 1
 3. for i = 1,2,... do
 4.       rho_{i-1} = (r_i,r_i)
 5.       beta_{i-1} = rho_{i-1} / rho_{i-2}
 6.       p_i = r_{i-1} + beta_{i-1} p_i
 7.       q_i = Ap_i
 8.       alpha_i = rho_{i-1}/(p_i,q_i)
 9.       x_i = x_{i-1} + alpha_i p_i
10.       r_i = r_{i-1} - alpha_i q_i
11.       Check convergence; continue?
12. end
```

Algorithmic Complexity (1)

Let the vectors have dimension N.

Every $u^\dagger v$ requires $\mathcal{O}(N)$ operations.

Every Av requires $\mathcal{O}(N)$ operations.

Each iteration through the loop requires $\mathcal{O}(N)$ operations.

Problem 1: Count the exact number of operations in each iteration.

Problem 2: Count the exact number of operations for Gauss elimination.

Linear system = quadratic minimum

The quadratic form

$$f(x) = \frac{1}{2}x^\dagger Ax - x^\dagger b + c,$$

where A is a matrix, b a vector and c a constant, has an unique minimum if all eigenvalues of A are positive. Finding the position of the minimum is equivalent to solving

$$Ax = b$$

We will discuss iterative methods for finding the minimum, *i.e.*, methods of defining a sequence x_0, x_1, x_2, $\cdots$ which converge to the minimum.

The vectors $e_i = x - x_i$ are called the errors.

The vectors $r_i = b - Ax_i = Ae_i$ are called the residuals.

Usual stopping criteria on iterative methods is to have $|r_i| \leq \epsilon$, for some pre-fixed ϵ.

Steepest Descent

To solve the linear system $Ax = b$ where A is a Hermitean matrix, we use the following algorithm:

```
1. Choose x_1
2. for i = 1,2,... do
3.       r_i = b - Ax_i
4.       alpha_i = (r_i,r_i)/(r_i,Ar_i)
5.       x_{i+1} = x_i + alpha_i r_i
6.       Check convergence; continue?
7. end
```

With $f(y)$ being the quadratic form, $f'(y) = b - Ay = r$. Hence the change in x_i is always along the direction of the gradient, *i.e.*, along the direction of maximum change.

In this direction (as in any other) the function is parabolic, and the step is chosen to bring us to the minimum of this parabola.

The steps are repeated until we are sufficiently close to the minimum.

Convergence of steepest descent

Let the eigenvalues of A be λ_α and eigenvectors be v_α. Assume all the vectors are orthonormal.

If $r_i = \sum_\alpha r_i^\alpha v_\alpha$, then

$$\alpha_i = \frac{\sum_\alpha \left(r_i^\alpha\right)^2}{\sum_\alpha \lambda_\alpha \left(r_i^\alpha\right)^2} = \left\langle \frac{1}{\lambda} \right\rangle$$

As a result,

$$r_{i+1} = (1 - \alpha_i A) r_i = \sum_\alpha \left[1 - \lambda_\alpha \left\langle \frac{1}{\lambda} \right\rangle\right] r_i^\alpha v_\alpha.$$

The longest components of r_i decrease at each step, but some of the shorter components can increase.

Problem 3: What is the convergence rate of steepest descent?

Search Directions and Conjugacy

In steepest descent, the successive search directions (for minimisation) are the r_i. The successive r_i are not orthogonal and therefore an infinite number of steps may be necessary for convergence.

If the search directions d_i are conjugate, *i.e.*, $d_i^\dagger A d_j = 0$ then we set $d_i^\dagger A e_{i+1} = 0$. Now let $e_{i+1} = e_i + \alpha_i d_i$. Then

$$\alpha_i = -\frac{d_i^\dagger A e_i}{d^\dagger i A d_i} = -\frac{d_i^\dagger r_i}{d^\dagger i A d_i}.$$

Knowing the search directions, we can then set up an iterative procedure. This converges in N steps, as we now show.

Let $e_0 = \sum_j \epsilon_j d_j$. Since the search directions are conjugate

$$\epsilon_j = \frac{d_j^\dagger A e_j}{d^\dagger j A d_j} = -\alpha_j.$$

Thus each step cuts out one component of e_0. Hence N steps suffice.

Orthogonality of Residuals + Conjugacy of Search Directions = Conjugate Gradient

Since $r_i = Ae_i$, and $d_i^\dagger A e_{i+1} = 0$, therefore

$$d_i^\dagger r_{i+1} = 0.$$

If we build the search direction from the successive residuals, then, at step i, they span the space $\mathcal{D}_i = \{r_0, r_1, \cdots, r_{i-1}\}$. Since the search direction $d_i \in \mathcal{D}_i$, the new residual is orthogonal to all the old search directions.

Problem 4: In the steepest descent method the residuals are the search directions, Prove that these are orthogonal.

Problem 5: Prove that $\mathcal{D}_i$ are Krylov spaces:

$$\mathcal{D}_i = \text{span}\{r_0, Ar_0, A^2 r_0, \cdots, A^{i-1} r_0\}.$$

Convergence of CG (1)

Since $\mathcal{D}_i$ is a Krylov space, $e_i = P(A)e_0$, where $P(A)$ is some polynomial in A. If we write $e_0 = \sum_\alpha \epsilon_\alpha v_\alpha$, then

$$e_i = \sum_\alpha \epsilon_\alpha P(\lambda_\alpha) v_\alpha, \quad r_i = \sum_\alpha \epsilon_\alpha \lambda_\alpha P(\lambda_\alpha) v_\alpha.$$

The CG algorithm minimises the maximum possible value of

$$||e_i|| = e_i^\dagger A e_i = \sum_\alpha \epsilon_\alpha^2 \lambda_\alpha P^2(\lambda_\alpha).$$

This minimax polynomial is known to be the Tchebychev polynomial,

$$T_i(z) = \frac{1}{2}[\left(z + \sqrt{z^2-1}\right)^i + \left(z - \sqrt{z^2-1}\right)^i].$$

It can be proved that

$$P_i(\lambda) = \frac{T_i(n(\lambda))}{T_i(d(\lambda))},$$

where

$$n(\lambda) = \frac{\lambda_{max} + \lambda_{min} - 2\lambda}{\lambda_{max} - \lambda_{min}}, \quad d(\lambda) = \frac{\lambda_{max} + \lambda_{min}}{\lambda_{max} - \lambda_{min}}.$$

Convergence of CG (2)

Since $|T_i(n(\lambda))| \leq 1$, the convergence rate is given by the denominator. Writing the condition number,

$$\kappa = \frac{\lambda_{max}}{\lambda_{min}},$$

it is easy to see that

$$||e_i|| \leq T_i\left(\frac{\kappa+1}{\kappa-1}\right)||e_0||.$$

Using the given expression for the Tchebychev polynomials, and noting that the second term vanishes with increasing i, it can be shown that

$$||e_i|| \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i ||e_0||.$$

Problem 6: Prove the above formula.

Dependence on Eigenvalue Distributions

CG builds a minimal polynomial whose zeroes are the eigenvalues of the matrix A—

$$||e_n|| = 0 = \sum_\alpha \epsilon_\alpha^2 \lambda_\alpha P^2(\lambda_\alpha).$$

The order of the polynomial is n.

We have proved that in the most generic case, $n = N$. However, from the above, if some of the roots are repeated, then $n < N$.

Problem 7: For free Fermions on a L^4 lattice with anti-periodic boundary conditions, how many steps are required for convergence of the CG algorithm in exact arithmetic?

Algorithmic Complexity (2)

Exact solution of the problem (in exact arithmetic) is $\mathcal{O}(N^2)$.

The number of steps required to reach the limit $||e_i|| \leq \epsilon ||e_0||$ is

$$N_{step} \leq \frac{1}{2}\sqrt{\kappa}\log\left(\frac{2}{\epsilon}\right).$$

The solution requires time of $\mathcal{O}(N\sqrt{\kappa})$ and memory space of $\mathcal{O}(N)$.

Problem 8: Find the complexity of the steepest descent method.

Preconditioning

If M is a Hermitean matrix that approximates A in some appropriate sense, then the problem

$$M^{-1}Ax = M^{-1}b$$

is easier to solve, provided M is easy to invert. However, $M^{-1}A$ is not necessarily Hermitean.

If we can find any E such that $M = EE^\dagger$, then we can solve

$$E^{-1}A(E^{-1})^\dagger y = E^{-1}b, \quad \text{where} \quad y = E^\dagger x.$$

This is a Hermitean problem.

Problem 8: Prove that the eigenvalues of $E^{-1}A(E^{-1})^\dagger$ are the same as those of $M^{-1}A$.

Such preconditioning works by changing the condition number of the problem and the clustering of eigenvalues. The simplest preconditioner is the diagonal matrix M obtained by setting all off-diagonal elements of A to zero.

Perfect Parallelisation

If a job can be done totally in parallel, and the CPU time on a serial machine is S, then on P processors, the usage time, U, will be

$$U = S/P.$$

An example of such a problem is the addition of two vectors (assuming that the vector components are already divided among the processors).

Parallelising never decreases CPU time usage.

We define the speedup

$$\Sigma = \frac{S}{U}.$$

For a perfectly parallelisable problem, $\Sigma = P$.

If parallelisation involves inter-processor communications, then it increases user time and decreases the speedup.

Imperfect Parallelisation

Take the computation of a dot product of two vectors u and v. Assume that the vectors are equally divided among P processors. Each processor computes its part of the dot product, sends this value to all other processors, adds the P numbers together to get the dot product. If each communication takes time C and blocks the channel, then

$$U = \frac{S}{P} + CP \quad \text{and} \quad \Sigma = \frac{P}{1 + (C/S)P^2} \leq P$$

The overhead due to communications prevents us from gaining in U by breaking the problem into smaller and smaller pieces. For a given value of C/S, the optimum is

$$P_o = \sqrt{\frac{S}{C}}, \quad \text{and} \quad \Sigma_o = \frac{1}{2}\sqrt{\frac{S}{C}} = \frac{1}{2}P_o.$$

The optimum speedup is always half the perfect speedup!

Parallel CG

The simplest parallelisation of CG is to divide all vectors and A among the processors. This may take some initialisation time, which we do not take into account.

There are two dot products to be evaluated. Each of these is a point of synchronisation of the processors. The parallelisation of a dot product has already been examined. Hence, a straightforward parallelisation of CG would at best give half the perfect speedup.

Problem 9: How does the optimum number of processors depend on the size, N, of the matrix being inverted?

Problem 10: How does the speedup of CG scale with N? How does the memory requirement of parallel CG scale with N?

Improving Parallel CG

The main bottleneck in CG are the two synchronisation points. There are several possible ways to cut down user time—

1. If the architecture allows simultaneous computation and communication, the we can try to rearrange the CG to mask the communication wait by a computation.

2. There are methods to re-arrange the computation and reduce the number of synchronisation points to one. The stability of such methods need study.

3. It is possible to generate several Krylov space vectors in parallel and simultaneously orthogonalise them by an explicit Gram-Schmidt procedure.

Problem 11: Investigate the parallel version of Gram-Schmidt orthogonalisation.

Other Issues

In the general case we would like to investigate the parallelisation of matrix-vector products Av. However, in the lattice Fermion case this does not seem to be a bottleneck.

Preconditioning is not always efficiently parallelisable. For the lattice Fermion problem on a hypercubic lattice, a degree of parallelisation can be achieved by utilising the even-odd decomposition of the lattice.

For an overview see the netlib article by Jack Dongarra (1995)—

`http://www.netlib.org/linalg/html_templates/node105.html#SECTION00940000000000000000`