QED on the Connection Machine

Clive F. Baillie
Caltech Concurrent Computation Project,
California Institute of Technology,
Pasadena, CA 91125, USA

S. Lennart Johnsson[†] *and Luis Ortiz*
Thinking Machines Corporation,
245 First Street,
Cambridge, MA 02142, USA

G. Stuart Pawley
Department of Physics, University of Edinburgh,
King's Buildings, Mayfield Road,
Edinburgh, EH9 3JZ, UK

NA88-1
January 1988

Abstract

Physicists believe that the world is described in terms of gauge theories. A popular technique for investigating these theories is to discretize them onto a lattice and simulate numerically by a computer, yielding so-called lattice gauge theory. Such computations require at least 10^{14} floating-point operations, necessitating the use of advanced architecture supercomputers such as the Connection Machine made by Thinking Machines Corporation. Currently the most important gauge theory to be solved is that describing the sub-nuclear world of high energy physics: Quantum Chromo-dynamics (QCD). The simplest example of a gauge theory is Quantum Electro-dynamics (QED), the theory which describes the interaction of electrons and photons. Simulation of QCD requires computer software very similar to that for the simpler QED problem. Our current QED code achieves a computational rate of 1.6 million lattice site updates per second for a Monte Carlo algorithm, and 7.4 million site updates per second for a microcanonical algorithm. The estimated performance for a Monte Carlo QCD code is 200,000 site updates per second (or 5.6 Gflops/sec).

[†] *also: Departments of Computer Science and Electrical Engineering, Yale University, P.O. Box 2158, New Haven, CT 06520*

1. Introduction

Universally accepted for some time has been the belief that testing the fundamental interaction laws of physics will demand enormous computing resources. This is a consequence of the fact that the gauge theories involved do not permit an analytic solution, and large-scale Monte Carlo, or alternatively microcanonical, simulations have to be performed. These computer simulations generate configurations which then have to undergo extensive analysis before yielding the desired results. Depending on the form of the gauge field being simulated, the configurations may correspond to either QED (quantum electro-dynamics) or QCD (quantum chromo-dynamics).

The configurations are of a space-time region containing the gauge fields. The interaction of these fields is governed by a coupling parameter, which is analogous to temperature. It is possible to represent the four dimensions involved as four Euclidean dimensions, and the region is bounded in these four dimensions by cyclic repetition. The gauge fields within this region are initially randomized and then allowed to attain thermodynamic equilibrium following either Monte Carlo or microcanonical algorithms.

2. Connection Machine

The Connection Machine [6] is configured as 1 - 4 quarter sized units, each consisting of 128 Mbytes of memory using 256 kbits memory chips, or 512 Mbytes using chips of 1 Mbits each. The fully configured machine has 64k processors. The Connection Machine can also be equipped with hardware support for floating-point operations giving it a peak performance of several Gflops/sec (for example, 15.5 Gflops/sec for 4x4 matrix multiplication). The communication system in the Connection Machine includes a router, largely implemented in hardware. The router allows for efficient access to any part of memory, which is partitioned into 8 kbytes/processor. There are 16 processors to a Connection Machine chip, and a maximum of 4k such chips interconnected as a 12-dimensional Boolean cube. The communication bandwidth depends on the access pattern, with a peak memory bandwidth of 50 Gbytes/sec, and a bandwidth of 1.6 - 16 Gbytes/sec for emulation of lattices, 7 - 16 Gbytes/sec for emulation of butterfly networks, and about one fifth of that for random permutations.

The Connection Machine requires a host, which currently can be either a VAX with a BI-bus, or a Symbolics 3600 series Lisp Machine. The programming languages for the Connection Machine are currently C* [12] and *Lisp [14], with *Fortran [13] becoming available later this year. C* is strongly influenced by C++, *Lisp is an extension of Common Lisp, and *Fortran implements the array extensions proposed for Fortran 8X [9]. The modifications essentially amount to one new data type, known as *poly* in C* and *parallel*

variables (*pvars*) in *Lisp. In C* *domains* are used to identify objects with the same data structure. In C* the difference between *poly* and *mono* variables, and operations upon them, is entirely transparent to the programmer, and so is communication. The programming and debugging environment is that provided by the language environment on the host machine. The Connection Machine is mapped into the address space of the host. The host fetches the program instructions from its memory, and passes on to the Connection Machine instructions that apply to variables in the Connection Machine memory. The host provides the scalar processing capability. Instructions for the Connection Machine are decoded and executed by a microcontroller that manages it.

All languages on the Connection Machine support a "data parallel" programming model. For many problems in science and engineering it is often convenient to express data structures and computations in the original problem domain, like four dimensional lattices for QED and QCD simulations, and two and three dimensional lattices for solving many partial differential equations in engineering. The order of the parallelism with suitable algorithms is often the same as the order of the data set. Expressing algorithms at this level is simple, and the programming languages on the Connection Machine support it, for instance by providing primitives for relative addressing in multi-dimensional lattices. Another important feature is that of *virtual processors*, which allows the programmer to focus on the data structures in the problem domain by associating a virtual processor with each "atomic" object, like a lattice point. The mapping to real Connection Machine processors is currently made at compile time, and is transparent to the user. Each virtual processor allocated to a real processor occupies a part of that processors memory. The virtual processors assigned to a real processor time-share that processor. The scheduling is transparent to the user.

3. Physics

In constructing a lattice gauge theory the gauge symmetry must be kept explicit in the lattice formulation so that in the continuum limit (when the lattice spacing tends to zero) the original gauge theory is recovered. The simplest such formulation is the original one due to Wilson [15] in which the action S for the gauge fields U is local, involving only the product of the gauge fields around elementary squares, called plaquettes, on the lattice. This is written as follows

$$S(U) = \beta E(U) = \beta \sum_p \left(1 - \frac{1}{N} ReTrU_p\right), \tag{1}$$

with

$$U_p = U_\mu(n) U_\nu(n+\hat{\mu}) U_\mu^\dagger(n+\hat{\nu}) U_\nu^\dagger(n), \tag{2}$$

and is illustrated in Figure 1. The gauge fields $U_\mu(n)$ are elements of the gauge group of the theory being studied ($U(1)$ for QED, $SU(3)$ for QCD). They are associated with links on the four dimensional lattice joining sites n and $n+\hat{\mu}$, where $\hat{\mu}$ is a unit lattice vector in the μ-direction. $U_\mu(n)$ is a directed variable: in going from $n+\hat{\mu}$ to n we use $U_\mu^\dagger(n)$. The parameter β determines the coupling (interaction strength) or "temperature" of the theory; the constant N is the dimensionality of the group elements (1 for QED, 3 for QCD).

As we are primarily concerned with QED here, it is worth giving slightly more detail for this particular theory. The gauge group for QED is U(1) and the elements of this group may be represented by complex numbers with unit modulus:

$$U_\mu(n) \in U(1) = e^{i\theta_\mu(n)}. \tag{3}$$

Hence $U_\mu^\dagger(n) = e^{-i\theta_\mu(n)}$ and Eqn. (1) can be rewritten as

$$S(\theta) = \beta \sum_p \Big[1 - cos(\theta_\mu(n) + \theta_\nu(n+\hat{\mu}) - \theta_\mu(n+\hat{\nu}) - \theta_\nu(n))\Big], \tag{4}$$

which is the form implemented in the QED code.

There are several algorithms which can be used for the computer simulation of lattice gauge theory. These algorithms fall into two categories: stochastic or deterministic. The most popular stochastic method is Monte Carlo, as exemplified by the Metropolis algorithm [10]; the deterministic method used is usually a microcanonical algorithm [1]. The Monte Carlo algorithm changes the energy of a system while keeping its temperature constant, whereas the microcanonical algorithm conserves the total energy while allowing temperature to vary. We make use of both in our QED code. The Monte Carlo method is used first to bring the lattice gauge theory into equilibrium at a specified temperature (coupling), then the (faster) microcanonical algorithm is used to evolve the system for measurements of its properties. During the latter phase we periodically switch back to the Monte Carlo algorithm in order to obtain ergodicity [2].

3.1. The Monte Carlo algorithm

Monte Carlo algorithms, such as Metropolis [10], cycle through all the gauge field links of the lattice by a random procedure changing their values until they settle down into physically correct configurations, C. These are such that, when statistical equilibrium is reached, the probability of finding any one of them is proportional to its Boltzmann factor $e^{-S(C)}$, where S is the action of the gauge theory. A sufficient condition for statistical equilibrium to be attained is that, at each step of the Monte Carlo algorithm, the

probability of changing a configuration C into a new one C' is the same as the probability of changing C' back to C. This is called "detailed balance" and it has important consequences for parallel computer implementations - in order to preserve detailed balance one cannot simultaneously update gauge field links which interact with one another. As the action involves interactions around plaquettes, one can therefore update only half the links in any one dimension simultaneously and preserve detailed balance, as shown in Figure 2 (in two dimensions for simplicity). On a parallel computer full processor utilization is obtained by observing that there are two plaquettes to be calculated for each dimension and link update, and scheduling half of the processors to calculate the "positive plaquettes" and half to calculate the "negative plaquettes", Figure 3.

The way the Metropolis algorithm proceeds computationally is as follows. In order to change a configuration C into another C' the change in action (β times the change in energy) is computed:

$$\Delta S = S(C') - S(C). \tag{5}$$

If $\Delta S \leq 0$, the change is accepted and C replaced with C'; otherwise if $\Delta S > 0$, the new configuration is accepted with the probability $e^{-\Delta S}$. In practice this is done by generating a pseudo-random number r in the interval $(0, 1]$ with uniform distribution. If $r < e^{-\Delta S}$, the change is accepted; otherwise it is rejected. With the condition $\Delta S < log_e r$ and Eqns. (4) and (5), it is relatively straightforward to write the pseudo-code in Appendix A for one Monte Carlo update. The operations required are given in Table 1.

3.2. The Microcanonical algorithm

Monte Carlo algorithms move through configuration space stochastically, whereas the microcanonical algorithm moves deterministically. Here, one enlarges the configuration space by adding fictitious momenta canonical to the degrees of freedom one wants to study, then solves Newton's equations [1]. Computationally this means solving coupled partial differential equations using difference approximations. The gauge theory action $S = \beta E$ is interpreted as β times the potential energy for a classical dynamics governed by Newton's law:

$$\ddot{U}(t) = -E(U). \tag{6}$$

Then the canonical momentum $\Pi = \dot{U}$ and the Hamiltonian $H(\Pi, U) = \Pi^2/2 + E(U)$ are introduced. From the resulting kinetic energy term T one obtains the temperature of the system as

$$\beta^{-1} = 2T/N_{indep}, \tag{7}$$

where N_{indep} is the number of linearly independent variables amoung the N gauge field links U. $N_{indep} < N$ because of the gauge symmetry in the theory. In the thermodynamic

limit ($N \rightarrow \infty$) the number of linearly independent variables in d dimensions for the $U(1)$ gauge theory is

$$N_{indep} = [(d-1)/d]N = (d-1)L^d, \tag{8}$$

for a hypercube with L lattice sites along each side.

Again, it is worth going into slightly more detail for QED. We write $U_\mu(n) = e^{ip_\mu(n)}$ and use Eqn. (4) to turn Eqn. (6) into:

$$\begin{aligned} \dot{p}_\mu(n) = -\sum_{\nu \neq \mu} \Big[& sin(\theta_\mu(n) + \theta_\nu(n+\hat{\mu}) - \theta_\mu(n+\hat{\nu}) - \theta_\nu(n)) \\ & -sin(\theta_\mu(n-\hat{\nu}) + \theta_\nu(n+\hat{\mu}-\hat{\nu}) - \theta_\mu(n) - \theta_\nu(n-\hat{\nu})) \Big]. \end{aligned} \tag{9}$$

This is pseudo-coded to perform one microcanonical update in Appendix B, yielding the operation counts listed in the third column of Table 1.

4. Communication Structure

The communication structure for QED and QCD calculations is a regular, periodic, four-dimensional lattice. It is well known [3,8] that multidimensional lattices with sides being powers of 2 can be mapped into Boolean cubes preserving adjacency by using a Gray code [4]. Gray codes by definition differ in a single bit for consecutive numbers. Nodes in a Boolean cube can be assigned addresses such that adjacent nodes have addresses that differ in precisely one bit. The most frequently used Gray code for lattice embedding in Boolean cubes is a *binary-reflected* Gray code [11]. This Gray code is periodic. Grids of arbitrary shapes can be embedded preserving proximity, but not adjacency, with minimal expansion [5,7], that is, the number of cube nodes need not be greater than the smallest number of dimensions required to provide at least as many nodes in the cube as the total number of nodes in the lattice. We only consider lattices with sides being powers of two here.

The address field of the Connection Machine is divided into three parts:
$(off-chip|on-chip|virtual\ processors)$. Configuring the Connection Machine as a lattice automatically provides a binary-reflected Gray code encoding of the nodes in each dimension of the lattice. What bits, or dimensions, in the Connection Machine address field are used for a lattice dimension can be selected by the programmer. In order to minimize the total communication the surface for a given volume should be minimized, with the same frequency of communication in all directions. Hence, for the QED and QCD calculations the subfield $|on-chip|virtual\ processors)$ is divided as equally as possible between the four dimensions. The length of the on-chip field is 4 bits, and if there is one

virtual processor per real processor a Connection Machine chip will simply hold a 2x2x2x2 sublattice. Our current code allows for 64 virtual processors for a Connection Machine equipped with 256 kbits memory chips. The sublattice on chip is in this case configured as a 8x8x4x4 lattice (4+6 bits). With one virtual processor the number of lattice points that communicate off-chip in any direction is 8, and the number of on-chip communications is 8. With 64 virtual processors the number of off-chip communications is 256 in two directions and 128 in two directions, and the number of on-chip communications is 768 and 896 respectively. On-chip communications are equivalent to moves (memory to memory copies).

5. Timings

Table 2 provides measured times for all essential operations used in the QED code implemented on a Connection Machine with hardware support for 32-bit IEEE format, floating-point operations. The timings have all been made on early models operating at a clock rate of 6.4 MHz and projected to the specified 8 MHz rate. The predicted times required for one complete update of each algorithm - Monte Carlo and microcanonical - for QED with one and 64 lattice sites per processor are listed in Table 3, and the actual measured times and corresponding millions of site updates per second are listed in Tables 4 and 5, respectively. The figures for site updates per second, and total floating-point rate are projected to a full machine from smaller configurations (8k and 16k configurations). The measured times are longer than the predicted times because the predictions ignore some infrequent operations and the overhead of the host broadcasting instructions to the processors.

In computing the floating-point rate in Table 6 the operations for the random number generation are neglected. The implementation of the trigonometric functions uses 34 floating-point operations, and operates at 11 - 11.5 Gflops/sec. The logarithm evaluation uses 14 floating-point operations. The trigonometric functions and the logarithm evaluation are part of a library coded for efficent use of the floating-point unit. The random number generator has not yet been coded for efficiency. A considerable improvement is expected. The communication time can be improved by a factor of 2 for computations on a lattice of 64^4 (16 million) lattice sites. Such a simulation would require 1 Mbits memory chips. By simulating lattices of this size and performing code improvements a performance gain by a factor of 1.5 - 2 is expected.

6. Conclusions

We have successfully implemented a QED simulation on the Connection Machine as a first step towards implementation of a QCD simulation. The performance obtained on a

fully configured Connection Machine with 512 Mbytes of memory and hardware support for floating-point operations is of the order of a million site updates per second for the current Monte Carlo routine, and 5 - 7 million site updates per second for the microcanonical routine for QED (Table 5). The floating-point rate is in the range 2.3 - 3.6 Gflops/sec in single precision IEEE format. A performance improvement by a factor of 1.5 - 2 is expected by improving the code efficiency, and by simulating lattices of size 64^4 on a Connection Machine with 2 Gbytes of memory. Such a lattice would fit in the memory, and a single update of the entire lattice would require about 8 seconds for the Monte Carlo routine and 1.8 seconds for the microcanonical routine, with the current code. Code improvements should make feasible update rates of about 3 million sites per second for the Monte Carlo routine, and rates in excess of 10 million sites per second for the microcanonical routine.

For Monte Carlo QCD simulations we predict a computational rate of about 200,000 site updates per second, or 5.6 Gflops/sec, with 64 virtual processors. Almost all of the arithmetic operations are in the form of multiplication of 3x3 complex matrices. We estimate that these operations can be performed at a rate of 13 Gflops/sec with kernels that make efficient use of the floating-point unit. At this rate approximately 42% of the time is spent for matrix multiplication, 10% for miscellaneous operations, and 48% for communication. The operation counts for the Monte Carlo algorithm for QCD are given in Table 7 and our prediction of the time required using efficient matrix kernels is given in Table 8. (We cannot easily predict the time for the microcanonical algorithm because it is much more complicated for QCD.)

References

[1] D.J.E. Callaway and A. Rahman, Phys. Rev. Lett. **49**, 613 (1982); J. Polonyi and H.W. Wyld, Phys. Rev. Lett. **51**, 2257 (1983)

[2] S. Duane, Nucl. Phys. **B257**, 652 (1985)

[3] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon and D. Walker, *Solving Problems on Concurrent Processors*, Prentice-Hall, Englewood Cliffs, N.J. (1987)

[4] M. Gardner, *Mathematical Games*, Scientific American, 106, Aug. 1972; E.N. Gilbert, Bell System Technical Journal **37**, 815 (1958)

[5] D. S. Greenberg, "Minimum Expansion Embedding of Meshes in Hypercubes", Yale University report, YALEU/DCS/TR-535 (1987)

[6] W. Daniel Hillis, *The Connection Machine*, MIT Press (1985)

[7] C.-T. Ho and S.L. Johnsson, "On the Embedding of arbitrary Meshes in Boolean cubes", International Conference on Parallel Processing, pp. 188-191 (1987)

[8] S.L. Johnsson, "Communication Efficient Basic Linear Algebra Computations on Hypercube Architectures", Journal of Parallel and Distributed Computing **4**, pp. 133-172 (1985)

[9] M. Metcalf and J. Reid, *Fortran 8X Explained*, Oxford Science Publishers (1987)

[10] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller and E. Teller, J. Chem. Phys. **21**, 1087 (1953)

[11] E.M Reingold, J. Nievergelt and D. Deo, *Combinatorial Algorithms*, Prentice-Hall, Englewood Cliffs, N.J. (1977)

[12] J. Rose and G.L. Steele, "C*: An Extended C Language for Data Parallel Programming", Second International Conference on Supercomputing (1987)

[13] P. Rosenblum, "Using the Fortran 8X style of Programming", Thinking Machines Corporation (1987)

[14] Thinking Machines Corporation, "*Lisp Release Notes", (1987)

[15] K.G. Wilson, Phys. Rev. **D10**, 2445 (1974)

Appendix A
Pseudo-code for Monte Carlo algorithm

```
mask = chessboardmask   % black and white squares
do i = 1,4   % loop over 4 links
{
  linki = link[i]
  linktry = twopi * random()   % possible new value for link
  query = log( random() )      % for Metropolis
  do parity = 1,2  % loop over black then white squares
  {                % can only update one set of squares simultaneously
    where (mask) linknew = linktry   % black is positive plaquettes
    else    linknew = linki          % white is negative plaquettes
    deltaE = 0.0
    do j = 1,4   % loop over 3 plaquettes in positive ij-plane
    {
      if (j == i) skip
      linkj = link[j]
      shlinkj = shift( linkj, i, + )
      shlinki = shift( linki, j, + )
      shlinknew = shift( linknew, j, + )
      plaqdiff =
        cos( linknew + shlinkj - shlinknew - linkj )   % new - old
      - cos( linki + shlinkj - shlinki - linkj )  % positive plaquette
      shplaqdiff = shift( plaqdiff, j, - )        % negative plaquette
      deltaE = deltaE + plaqdiff + shplaqdiff     % change in energy
    }
    where ( (beta * deltaE > query) && mask )   % Metropolis accept
      linki = linknew
    mask = !mask   % swap black and white
  }
  link[i] = linki
}
```

Appendix B
Pseudo-code for Microcanonical algorithm

```
do i = 1,4   % loop over 4 links
{
  link[i] = link[i] + p[i] * deltat   % advance links
}
do i = 1,4   % loop over 4 links
{
  sumpdot = 0.0
  linki = link[i]
  do j = 1,4   % loop over 3 plaquettes in positive ij-plane
  {
    if (j == i) skip
    linkj = link[j]
    shlinkj = shift( linkj, i, + )
    shlinki = shift( linki, j, + )
    pdot[i] = sin( linki+shlinkj-shlinki-linkj )  % positive plaquette
    shpdot = shift( pdot[i], j, - )               % negative plaquette
    sumpdot = sumpdot - pdot[i] + shpdot   % derivative of momentum
  }
  pdot[i] = sumpdot
}
do i = 1,4   % loop over 4 links
{
  p[i] = p[i] + pdot[i] * deltat   % advance momenta
}
```

Operation	Monte Carlo	Microcanonical
arithmetic (+,-,×)	244	76
sin	0	12
cos	48	0
log	4	0
random	8	0
move	48	20
communication	96	36

Table 1: Operation counts for one complete update of a lattice site for QED.

Operation	1 vp	64 vp
arithmetic (+,-,×)	36	25
sin	200	195
cos	200	195
log	120	115
random	950	940
move	22	20
communication	208	120

Table 2: Measured operation times in μ-seconds for 32-bit operations (IEEE format) on a CM with floating-point unit (fpu) option using 1 and 64 virtual processors (vp).

Operation	1 vp	64 vp
Monte Carlo	0.0475	2.30
Microcanonical	0.0131	0.57

Table 3: Predicted times in seconds for one complete lattice update for QED with one site per vp.

Operation	1 vp	64 vp
Monte Carlo	0.054	2.61
Microcanonical	0.013	0.57

Table 4: Measured times in seconds for one complete lattice update for QED a on CM with fpu option.

Operation	1 vp	64 vp
Monte Carlo	1.21	1.61
Microcanonical	5.04	7.36

Table 5: Measured millions of site updates per second for QED on a CM with fpu option.

Operation	1 vp	64 vp
Monte Carlo	2.34	3.11
Microcanonical	2.44	3.56

Table 6: Measured Gflops rates for QED on a CM with fpu option.

Operation	Monte Carlo
arithmetic (+,-,×)	28736
sin	0
cos	0
log	4
random	12
move	864
communication	1320

Table 7: Operation counts for one complete update of a lattice site for QCD.

Operation	1 vp	64 vp
Monte Carlo	1.34	21.2

Table 8: Predicted times in seconds on a CM with fpu option for one complete lattice update for QCD with one site per vp.

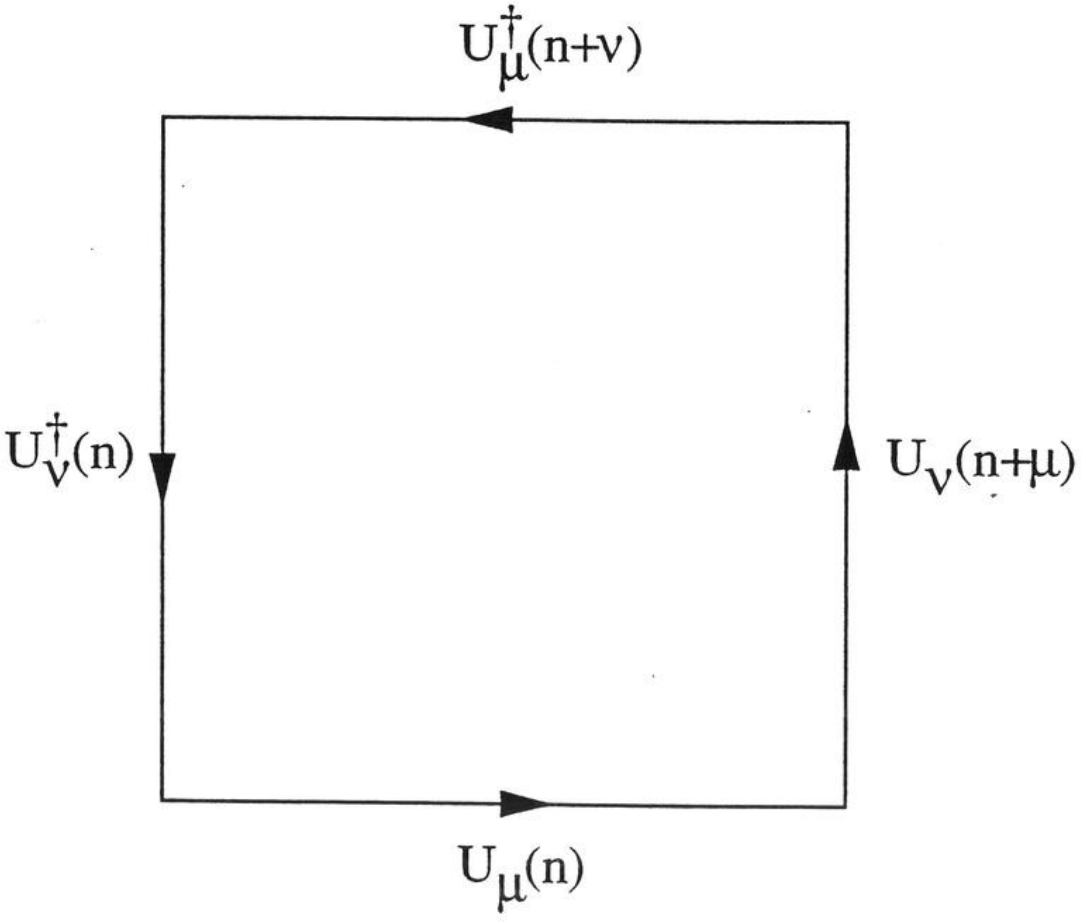

Figure 1

Illustration of plaquette calculation

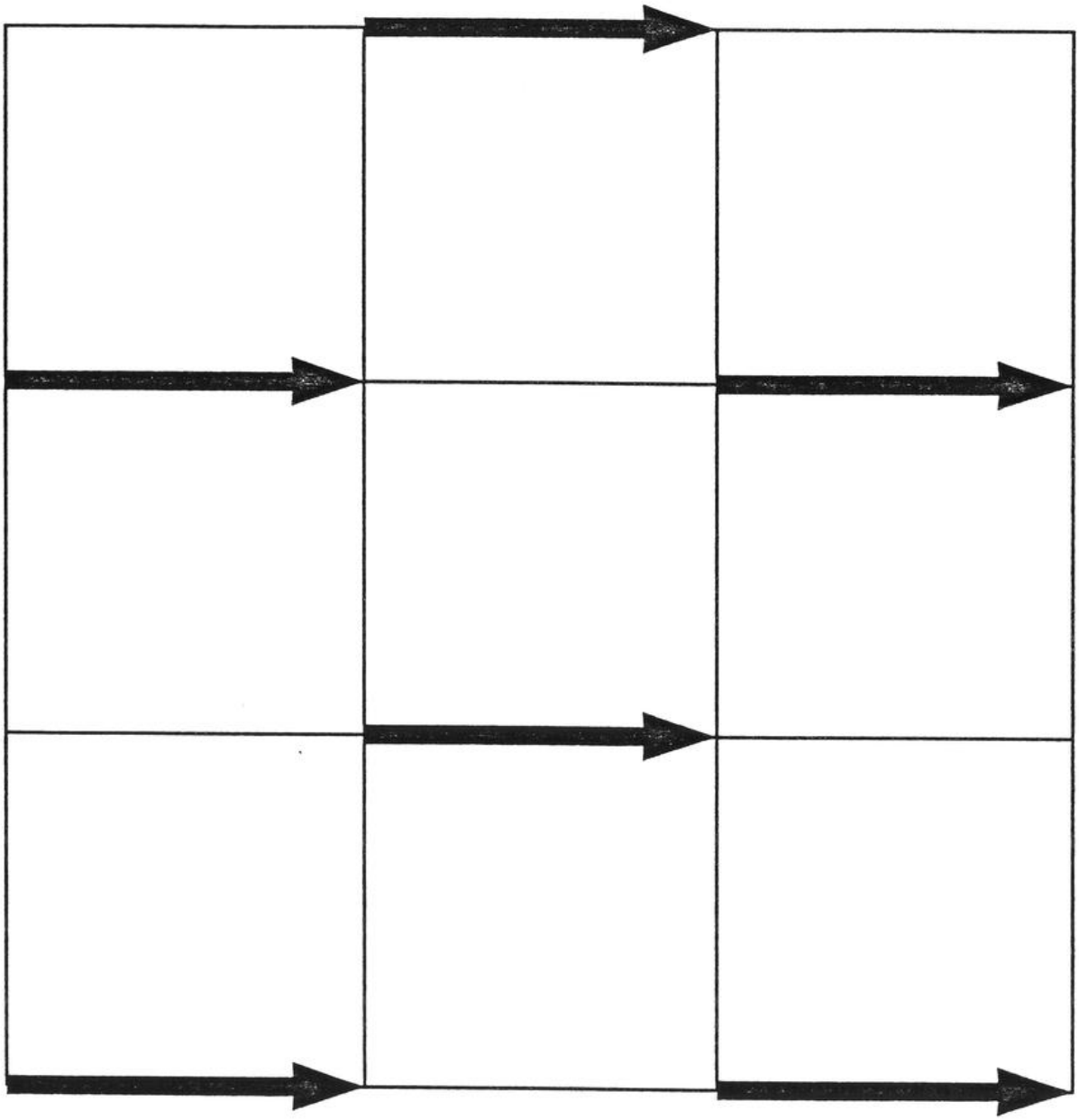

Figure 2

Showing that only half the links can be updated in any one dimension simultaneously

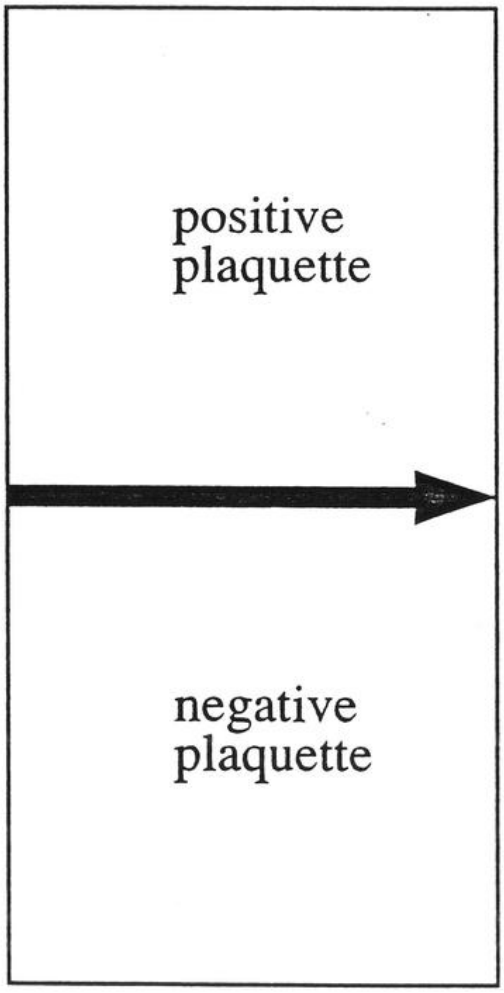

Figure 3

Illustration of positive and negative plaquettes