Computer Physics Communications 37 (1985) 195–203
North-Holland, Amsterdam

CYCLIC REDUCTION ON A BINARY TREE

S. Lennart JOHNSSON

Department of Computer Science, Yale University, New Haven, CT 06520, USA

Ensembles of large numbers of processors tightly coupled into networks are of increasing interest. Binary tree interconnect has many favourable characteristics from a construction point of view, though the limited communication bandwidth between arbitrary processors poses a potential bottleneck. In this paper we present an algorithm for odd–even cyclic reduction on a binary tree for which the limited bandwidth does not increase the order of the computational complexity, compared to an ideal parallel machine. The complexity is $2\log_2 N$ with respect to arithmetic operations, and $3\log_2 N$ with respect to communication. The communication complexity compares favourably with the best previously published result, $\mathcal{O}(\log_2^2 N)$. We also show that the benefits of truncated cyclic reduction are much greater for parallel reduction algorithms than for sequential algorithms. A reduction in the computational complexity proportional to the reduction in the number of reduction steps is possible.

1. Introduction

With the rapidly developing integrated circuit technology there is a growing interest in highly concurrent computations. Several 16-bit processors already fit on a chip in today's technology, Seitz et al. [1]. Within a few years on the order of a hundred such processors with local storage, each in the order of about 8–16 Kbytes, will fit on a single chip. With processors of a larger grain, such as 32-bit processors with floating-point arithmetic in hardware, and substantial local storage, say 128–256 Kbyte a processor and its local storage may demand one chip. A multiprocessor with a large number of processing elements is indeed feasible in the technology of the late eighties.

There are many possible models for multiprocessor architectures. One class consists of processors with small local storage connected to a global storage via a switching network. Examples of this kind of architecture are the HEP by Denelcor, the NYU Ultracomputer, Schwartz [2], Gottlieb et al. [3], and the TRAC machine of the University of Texas at Austin, Sejnowski et al. [4]. Another class of architectures is defined by machines consisting of processors with local storage interconnected in a network, with no shared resources. Binary tree interconnection, 2-dimensional mesh, and Boolean n-cube interconnection are some often proposed topologies. The Caltech tree machine, Browning [5], is a binary tree of processors with local storage. The Caltech Cosmic Cube [6] has the topology of a n-dimensional Boolean cube. The Cedar machine of the University of Illinois, Kuck et al. [7], is a multiprocessor with some global resources, as well as clusters of processors interconnected in networks.

All the machines mentioned so far are of the MIMD type, Flynn [8]. Each processor executes its own instructions. Other machines like the ILLIAC IV and the ICL DAP, both having processors interconnected in a 2-dimensional mesh, and the NON-VON machine of Columbia University, Shaw [9], with processors interconnected in a binary tree, are of the SIMD variety. All processors execute the same instruction, with the exception that some may perform a no-op.

A richer interconnect has fewer potential bottlenecks in interprocessor communication at the expense of increased cost of wiring the processors together. A Boolean n-cube is for a large number of processors significantly more difficult to build than a binary tree. In an n-dimensional cube of N processors there are $(N/2)\log_2 N$ communication paths. In a tree of N processors there are only N channels, yet the diameter of the tree, i.e., the maximum distance between two processors, is only $2(\log_2 N - 1)$, approximately twice the diameter of

0010-4655/85/$03.30

the cube. The diameter sets a lower bound for the time to perform computations requiring global communication. Solving a full rank, irreducible, linear system of equations requires communication between all processors among which the computations are divided. It is of interest to find efficient algorithms for sparsely interconnected ensembles of processors in order to simplify the construction problem.

The tree is a particularly interesting interconnection in that it scales well. With several processors per chip, the demands for interchip channels remain constant for the tree interconnection, regardless of what size tree is being costructed, and regardless of what number of processors with local storage fit on a chip, Leiserson [10]. For a 2-dimensional mesh with M processors per chip we know of no better growth rate of the number of interchip channels than $\sqrt{M}$. For an n-dimensional cube with an m-cube per chip, the corresponding growth rate is $(\log_2 N - \log_2 M)\log_2 M$ for an n-dimensional cube with an m-cube per chip ($N = 2^n$, $M = 2^m$).

In the following we will describe an algorithm for odd–even cyclic reduction on a binary tree of processing elements. There is no global storage, no global control. Each processor executes its own instructions. Hence, the model architecture is of the MIMD type with no globally shared resources. Synchronization of computations in different processors is accomplished via message passing. The communication time between adjacent processors is denoted t_c. This time is sufficiently long to allow for the communication of the data required for the reduction computations on one equation. The time to perform the reduction computations on one equation is denoted t_a. The estimates of the computational complexity that are derived based on these assumptions can be refined with additional assumptions on the architecture of the processors and the communication protocol. In our complexity estimates we assume that computation and communication can take place concurrently. However, data being communicated during one time step can first be used in a computation, or another communication, during the next time step.

The main result presented in this paper is an $\mathcal{O}(\log_2 N)$ algorithm for cyclic reduction on a binary tree of processors. This algorithm represents an improvement by a factor of $\log_2 N$ over the previously best published binary tree algorithm, Presnell, Pargas [11]. Expressions for the computational complexity of the algorithm are derived below. The constants of proportionality are small. We also briefly discuss truncated cyclic reduction. The cases with multiple equations per node, and multiple independent problems are discussed in detail in ref. [12].

2. Odd–even cyclic reduction

Odd–even cyclic reduction for a tridiagonal system of N equations of full rank can be carried out in the following way, Hockney, Jesshope [13], assuming A is of full rank and of dimension $N = 2^n - 1$. Subscripts denote equation number and superscripts denote reduction and back substitution steps.

Reduction

$$a_i^j = e_i a_{i-2^{j-1}}^{j-1},$$
$$c_i^j = f_i c_{i+2^{j-1}}^{j-1},$$
$$b_i^j = b_i^{j-1} + e_i c_{i-2^{j-1}}^{j-1} + f_i a_{i+2^{j-1}}^{j-1},$$
$$y_i^j = y_i^{j-1} + e_i y_{i-2^{j-1}}^{j-1} + f_i y_{i+2^{j-1}}^{j-1},$$
$$e_i = -a_i^{j-1}/b_{i-2^{j-1}}^{j-1},$$
$$f_i = -c_i^{j-1}/b_{i+2^{j-1}}^{j-1},$$

where $i = \{2^j, 2*2^j, 3*2^j, \ldots, 2^n - 2^j\}$,
for reduction steps $j = \{1, 2, \ldots, n-1\}$.

The initial conditions are

$a_i^0 = a_i$, $b_i^0 = b_i$, $c_i^0 = c_i$, and $y_i^0 = y_i$.

After $n-1$ reduction steps only one equation remains

$$a_{2^{n-1}}^{n-1} x_0 + b_{2^{n-1}}^{n-1} x_{2^{n-1}} + c_{2^{n-1}}^{n-1} x_{2^n} = y_{2^{n-1}}^{n-1}.$$

A correct solution for $x_{2^{n-1}}$ is obtained with, $x_0 = x_{N+1} = 0$. Having solved for $x_{2^{n-1}}$ the other variables are obtained through back substitution.

Back substitution

$$x_{2^{n-1}} = y^{n-1}_{2^{n-1}} / b^{n-1}_{2^{n-1}}$$

$$x_i = \left(y_i^{j-1} - a_i^{j-1} x_{i-2^{j-1}} - b^{j-1} x_{i+2^{j-1}}\right)/b_i^{j-1},$$

where $i = \{2^{j-1}, 3*2^{j-1}, 5*2^{j-1}, \ldots, 2^n - 2^{j-1}\}$ and $j = \{n-1, n-2, \ldots, 1\}$.

A careful count gives a total of $17N - 18n + 2$ arithmetic operations.

If the tridiagonal system is sufficiently diagonally dominant then the reduction can be terminated after $m < n$ steps, since the reduced system numerically can be considered as a diagonal system. The reduction in the number of reduction and back substitution steps is $2(n-m-1)$, and the reduction in the total number of operations is $16*(2^{n-m}-1) - 18(n-m) + 2$.

3. Cyclic reduction on a binary tree

In finding good mappings of computations on to a network of processors it is important to know the manner in which an algorithm makes use of the data, i.e., how a data structure is traversed. The access pattern can be illustrated by a computation graph in which nodes correspond to data and computations, and edges to information exchanges. A computation graph for odd–even cyclic reduction is shown in fig. 1. The edges are labelled with the step of the reduction algorithm during which the data exchange occurs. Nodes are labelled with the equation number.

In the mappings considered here nodes in the computation graph shall be mapped to distinct processors. From the computation graph in fig. 1 it is clear that no proximity preserving map on to a binary tree exists. The fanout of the centre node is $2(\log_2 N - 1)$. Some of the nodes adjacent to the centre node in the computation graph can, in the tree, be no closer to the node hosting the centre node than $\mathcal{O}(\log_2 \log_2 N)$. Presnell and Pargas mapped the nodes of the computation graph into the leaves of the binary tree. With their algorithm each step requires time $\mathcal{O}(\log_2 N)$, and the total time is $\mathcal{O}(\log_2^2 N)$.

We will now show that an inorder mapping of equations to nodes in the tree yields an algorithm that has a complexity of $3 \log_2 N t_c$, if $t_a \ll t_c$, and $2 \log_2 N t_a$ if $t_a \gg t_c$. Fig. 2 shows an inorder labelling of a binary tree.

For $N = 2^n - 1$ the distance between the centre node, i.e., the root, labeled 2^{n-1}, and the nodes $2^{n-1} \pm 1$, is $n-1$. Hence, the first reduction step cannot be completed in a time less than $(n-1)t_c$. However, unlike the mapping used by Presnell and Pargas, the inorder mapping allows successive steps of the reduction phase to be pipelined. The back substitution starts at the root and proceeds towards the leaves.

One can associate a wave front with each reduction step. The first wave front is initiated by the leaves. The tridiagonal system is solved when the wave front for the back substitution, initiated by the root, reaches the leaves. The required communication paths for reduction step 1 are indicated in fig. 3. The tree edges forming the path from the leftmost and rightmost leaf nodes to the root need only transmit one equation. Edges connecting leaf nodes with their respective parent node also only need to transmit one equation. All other edges have to transmit two equations in the first reduction step. The reduction computations are carried out in the nodes that store the equations subject to modification.

The algorithm described next in some detail

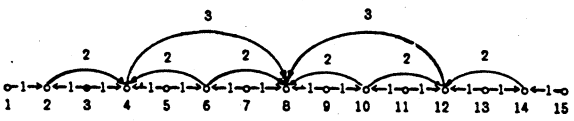

Fig. 1. A graph representation of odd–even cyclic reduction.

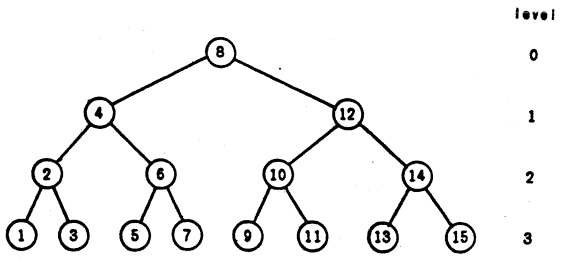

Fig. 2. Inorder labelling of a binary tree.

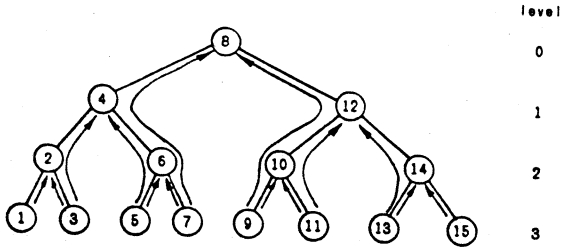

Fig. 3. Communication for reduction step 1.

distinguishes only between the root, the intermediate level nodes, and the leaf nodes. The nodes on the path from node 1 and node N to the root are executing the same code as other intermediate level nodes for reasons of simplicity and clarity. No loss of performance occurs. The time for the additional communication is masked by the time required for communication in other parts of the tree.

The algorithm starts and terminates in the leaf processors. They send their equations to their respective parent processors. These nodes (at level $n-2$ from the root) send the two equations they receive to their parents, and perform the computations for reduction step 1 on the equation they store. The processors at level $n-3$ receive two equations from each of their two children processors. One of the equations a processor receives from each of its children processors is to be forwarded to its own parent processor, one is used in reduction step 1 for the equation the processor stores. Hence, processors at level $n-3$ send two equations to their parents, precisely as nodes at level $n-2$. After $n-1$ steps equations $2^{n-1} \pm 1$ have propagated to the root. Assuming that a data item can first be used during the time step following the one during which it was received, one additional step is needed to complete reduction step 1. The leaf nodes do not participate in reduction step 2. The effective tree height is reduced by 1 per reduction step.

By pipelining the reduction steps a new reduction step can be initiated every other time step. Associating a wave front with each reduction step, wave fronts are spaced one level apart. There is a maximum of $n/2$ wave fronts in the tree during the reduction phase. Pipelining occurs naturally in a system in which synchronization is accomplished via message passing. As soon as a processor has finished the communication actions, and computations associated with a reduction step, it proceeds to the next reduction step. If the partner processor in a communication is not ready to participate, the requesting processor has to wait until the partner is ready.

On completion of the last reduction step the root processor computes the variable $x_{2^{n-1}}$ and sends it to its two children processors. Those compute the variables $x_{2^{n-2}}$ and $x_{3*2^{n-2}}$. For the sake of program uniformity we also have the root processor send x_0 to the left child and x_{N+1} to the right child. By so doing each node will receive two x-values from its parent in the back substitution phase. There is one wave front propagating from the root towards the leaves in the back substitution phase. There are n computations and $n-1$ communication steps in sequence.

A concise description of odd–even cyclic reduction on a binary tree is given in the form of pseudo-code in the appendix. The progression of the computations is for $N = 15$ illustrated in figs. 4–9. Superscripts refer to reduction step. The numbers on edges and nodes denotes the equation number. A few time steps of the back substitution computations are shown in fig. 10–13.

The computational complexity is:

Reduction $(n-2)(\max(t_a, t_c) + t_c) + t_a + t_c$, $n > 1$;

Back substitution $(n-1)t_c + nt_a$.

In the estimates t_c is the maximum time required for a communication, and t_a is the maximum time required for arithmetic operations on one equation. For each reduction step, except the last, the required time is $t_c + \max(t_a, t_c)$. If $t_a \gg t_c$, then the total time for cyclic reduction with one equation per node is $(2n-1)t_a$. With $t_c \gg t_a$ the time is $(3n-4)t_c$. This computational complexity is of the minimum order for a full rank, irreducible system of equations. The constant of proportionality is lower than the constant for more versatile interconnection networks such as the perfect shuf-

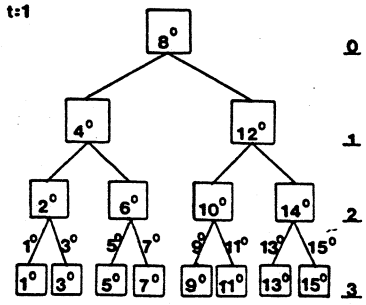

Fig. 4. Cycle 1 of the reduction phase.

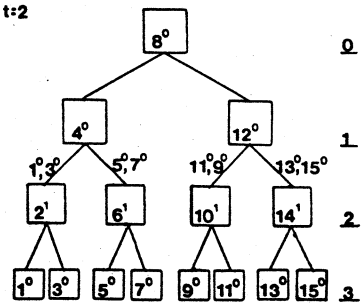

Fig. 5. Cycle 2 of the reduction phase.

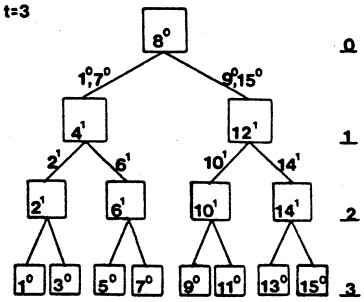

Fig. 6. Cycle 3 of the reduction phase.

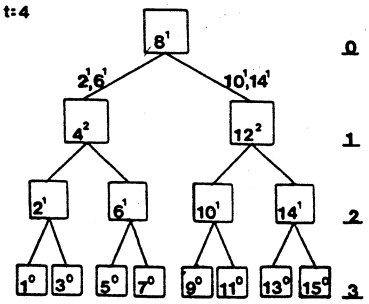

Fig. 7. Cycle 4 of the reduction phase.

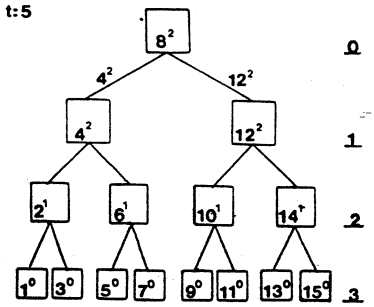

Fig. 8. Cycle 5 of the reduction phase.

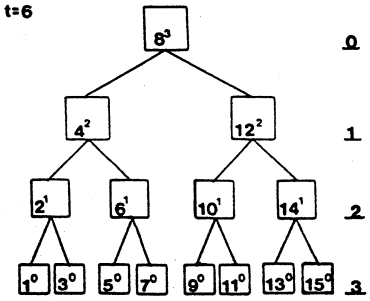

Fig. 9. Cycle 6 of the reduction phase.

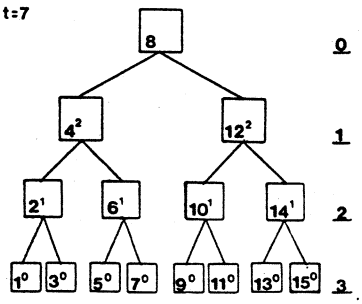

Fig. 10. Cycle 1 of the back substitution phase.

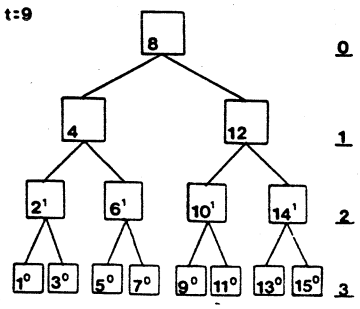

Fig. 11. Cycle 2 of the back substitution phase.

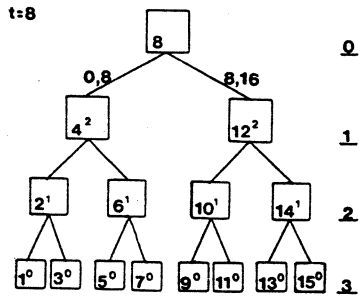

Fig. 12. Cycle 3 of the back substitution phase.

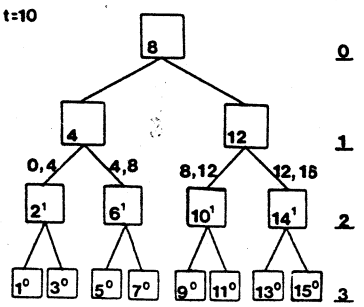

Fig. 13. Cycle 4 of the back substitution phase.

fle and the Boolean n-cube, under the same computational model, [12].

The estimates can be refined somewhat with the following assumptions:

- the time for communication is proportional to the number of values communicated;
- the time for arithmetic is proportional to the number of arithmetic operations;
- communication and computation can overlap in a pipelined fashion.

In the reduction phase 2 equations, i.e., 8 variables are communicated between processor pairs. In the back substitution phase 2 variables are communicated. In the reduction phase 2 multiplications, 4 multiply-and-add operations, and 2 divisions are covered by t_a. In the back substitution phase 2 multiply-and add operations, and 1 division are performed.

If t_c^1 denotes the time for communication of 1 variable, and t_a^1 denotes the time for a multiply-and-add operation, assuming that division can be accomplished in this time as well, the following alternate estimates can be derived.

$$\text{Reduction } (n-3)\left(t_c^1 + 8\max\left(t_a^1, t_c^1\right)\right) + 2t_c^1 + 4\max\left(2t_a^1, t_c^1\right) + 3\max\left(2t_a^1, t_c^1\right) + 2t_a^1 \quad n > 3,$$

$$2t_c^1 + 4\max\left(2t_a^1, t_c^1\right) + 3\max\left(2t_a^1, t_c^1\right) + 2t_a^1 \quad n = 2;$$

$$\text{Back substitution } (n-1)\left(2t_a^1 + t_c^1\right) + \max\left(t_a^1, t_c^1\right) \quad n > 1,$$

used. The processors along the paths from the leftmost and rightmost leaves to the root also only have to send 4 values to their respective parent processor in the reduction phase. This property results in special treatment of trees with $n < 3$.

4. Truncated cyclic reduction

The reduction in the total time for cyclic reduction on a binary tree that is possible if the reduction process can be truncated after $m < n$ steps, depends on the ratio between t_a and t_c. If the time for communication is insignificant, then a reduction step effectively takes place in the entire tree at once, and the total computation time is proportional to the number of reduction steps. However, if $t_c \gg t_a$ then there is a propagation time proportional to the height of the tree regardless of how few reduction steps are actually taken, disregarding the degenerate case with A being a diagonal matrix. Hence, the complexity of the tree algorithm is reduced by truncating the reduction process, but the order remains the same. The complexity of truncated cyclic reduction on the tree is

$$\text{Reduction } (n-1)t_c + (m-1)\max(t_a, t_c) + t_a,$$

$$\text{Back substitution } (n-1)t_c + (m+1)t_a.$$

The reduction in the computational time is directly proportional to the reduction in the number of reduction steps. Hence, the effect of truncating the reduction process is much more significant on this kind of architecture than on a sequential machine. On a sequential machine most of the time is spent in the first few reduction steps. On such a machine approximately half the time is

devoted to the first reduction step, and last back-substitution step.

5. Parallel cyclic reduction

Parallel cyclic reduction [13] allows a tridiagonal system to be solved in $\log_2 N$ steps on an ideal parallel machine with no restrictions on the communication bandwidth. The cyclic reduction algorithm used above needs $2 \log_2 N$ steps, and has an operations count of $\mathcal{O}(N)$. Parallel cyclic reduction has an operations count of $\mathcal{O}(N \log_2 N)$. Parallel cyclic reduction can be performed in $\log_2 N$ steps also on some processor networks such as the perfect shuffle, and the n-cube [12] even under the constraint of limited communication bandwidth.

The inorder map of equations to processors that we use to obtain a running time proportional to $\log_2 N$ both with respect to communication and computation, does not fit parallel cyclic reduction well. With the inorder map the leaves contain all the odd equations (labeling equations from 1 to N), and only odd equations. In parallel cyclic reduction arithmetic operations are performed on odd equations during every step. After the first step these operations involve communication with nodes storing other odd equations. Hence, the first reduction step in Hockney's parallel algorithm requires root to leaf communication, and the remaining step leaf to leaf communication, some of which have to go through the root. Each step has a communication time proportional to $\log_2 N$, and the total time complexity is $\mathcal{O}(\log_2^2 N)$. Successive steps cannot be pipelined to decrease the time for communication through the root.

6. Summary and conclusions

A concurrent algorithm for odd–even cyclic reduction on a binary tree of processing elements is presented. The running time of the algorithm is proportional to $\log_2 N$, both in terms of the time for basic arithmetic operations, and in terms of the time for communication of single data items. The constant of proportionality is 2–3, and indeed slightly smaller than the constants of proportionality for perfect shuffle and Boolean n-cube networks.

The computational complexity of the binary tree algorithm also compares favourably with algorithms for architectures with processors interconnected to storage via a switch network, such as the TRAC and the Ultracomputer. The time to traverse the switch is proportional to $\log_2 N$ in those architectures. Pipelining of the switch can improve the bandwidth of the switch, but the sequential dependencies of the cyclic reduction algorithm are such that effective use cannot be made of such a feature. Hence, the complexity is $\mathcal{O}(\log_2^2 N)$ for cyclic reduction on these architectures.

The relative decrease in running time rendered by truncating the reduction process after $m < n$ steps depends on t_c/t_a. With $nt_c \ll mt_a$ the reduction is proportional to $(n-m)/n$, and with $t_c \gg t_a$ the reduction is proportional to $(n-m)/3n$. The payoff in reduced computation time is much more significant than on a uniprocessor.

The inorder map of equations to processors does not fit Hockney's parallel cyclic reduction algorithm well. Indeed, with an inorder map the communication time is proportional to $\mathcal{O}(\log_2^2 N)$.

Our computational model is of the MIMD type, and in our algorithm different processors execute different programs. However, there is a high degree of uniformity. Only three different types of code are employed. The root has one type, the intermediate level nodes another type, and the leaf nodes the third type. The appendix contains pseudo code for the algorithm, and serves to illustrate the simplicity of the code in each node, as well as a programming style for systems with synchronization via message passing.

Appendix. A binary tree algorithm

Root processor(i):

$x_0 := 0; x_{N+1} := 0$
$k := 1$
$m := i/(2k)$

# Reduction computations
while m is even do
receive $(al, bl, cl, yl, a_{i-k}, b_{i-k}, c_{i-k}, y_{i-k})$
  from the left child
receive $(a_{i+k}, b_{i+k}, c_{i+k}, y_{i+k}, ar, br, cr, yr)$
  from the right child
$e_i := -a_i/b_{i-k}$
$f_i := -c_i/b_{i+k}$
$a_i := e_i a_{i-k}$
$c_i := f_i c_{i+k}$
$b_i := b_i + e_i c_{i-k} + f_i a_{i+k}$
$y_i := y_i + e_i y_{i-k} + f_i y_{i+k}$
$k := 2k$
$m := m/2$
enddo
# The last reduction step
for m odd do
receive $(a_{i-k}, b_{i-k}, c_{i-k}, y_{i-k})$
  from the left child
receive $(a_{i+k}, b_{i+k}, cd_{+k}, y_{i+k})$
  from the right child
$e_i := -a_i/b_{i-k}$
$f_i := -c_i/b_{i+k}$
$a_i := e_i a_{i-k}$
$c_i := f_i c_{i+k}$
$b_i := b_i + e_i c_{i-k} + f_i a_{i+k}$
$y_i := y_i + e_i y_{i-k} + f_i y_{i+k}$
enddo
# Back substitution
$x_i := y_i/b_i$
send (x_{i-2k}, x_i) to the left child
send (x_i, x_{i+2k}) to the right child

Intermediate level processor(i):

$k := 1$
$m := i/(2k)$
# Reduction computations
while m is even do
receive $(al, bl, cl, yl, a_{i-k}, b_{i-k}, c_{i-k}, y_{i-k})$
  from the left child
receive $(a_{i+k}, b_{i+k}, c_{i+k}, y_{i+k}, ar, br, cr, yr)$
  from the right child
send $(a_{i-k}, b_{i-k}, c_{i-k}, y_{i-k}, a_{i+k}, b_{i+k}, c_{i+k}, y_{i+k})$
  to the parent
$e_i := -a_i/b_{i-k}$
$f_i := -c_i/b_{i+k}$
$a_i := e_i a_{i-k}$
$c_i := f_i c_{i+k}$
$b_i := b_i + e_i c_{i-k} + f_i a_{i+k}$
$y_i := y_i + e_i y_{i-k} + f_i y_{i+k}$
$k := 2k$
$m := m/2$
enddo
# The last reduction step for node i
for m odd do
receive $(a_{i-k}, b_{i-k}, c_{i-k}, y_{i-k})$ from the left child
receive $(a_{i+k}, b_{i+k}, c_{i+k}, y_{i+k})$ from the right child
send $(a_{i-k}, b_{i-k}, c_{i-k}, y_{i-k}, a_{i+k}, b_{i+k}, c_{i+k}, y_{i+k})$
  to the parent
$e_i := -a_i/b_{i-k}$
$f_i := -c_i/b_{i+k}$
$a_i := e_i a_{i-k}$
$c_i := f_i c_{i+k}$
$b_i := b_i + e_i c_{i-k} + f_i a_{i+k}$
$y_i := y_i + e_i y_{i-k} + f_i y_{i+k}$
send (a_i, b_i, c_i, y_i) to the parent
enddo
# Back substitution
receive (x_{i-2k}, x_{i+2k}) from the parent
$x_i := (y_i - a_i x_{i-2k} - c_i x_{i+2k})/b_i$
send (x_{i-2k}, x_i) to the left child
send (x_i, x_{i+2k}) to the right child

Leaf processor(i):

send (a_i, b_i, c_i, y_i) to the parent
Back substitution
receive (x_{i-1}, x_{i+1}) from the parent
$x_i := (y_i - a_i x_{i-1} - c_i x_{i+1})/b_i$

References

[1] C. Lutz, S. Rabin, C.L. Seitz, D. Speck, in: Proc. Conf. on Advanced Research in VLSI (Artech House, 1984) p. 1–10.

[2] J.T. Schwartz, ACM Trans. on Programming Languages and Systems 2 (1980).

[3] A. Gottlieb, R. Grishman, C.P. Kruskal, K.P. McAuliffe, L. Rudolph and M. Snir, IEEE Trans. Comput. C-32 (1983) 175.

[4] M.C. Sejnowski, E.T. Upchurch, R.N. Kapur, D.P.S. Charlu and G.J. Lipovski, in: Proc. National Computer Conf. IEEE (1980) p. 631.

[5] S.A. Browning, Technical Report 1980: TR: 3760, Computer Science, California Institute of Technology (January 1980).

[6] C.L. Seitz, The Cosmic Cube, CACM (1985).

[7] D. Kuck, D. Lawrie, R. Cytron, A. Sameh and D. Gajski, The Architecture and Programming of the Cedar System, in: LASL Workshop on Vector and Parallel Processing (August 1983).

[8] M.J. Flynn, Proc. of the IEEE 12 (1966) 1901.

[9] D. Shaw, The NON-VON Supercomputer, Technical Report, Dept. of Computer Science, Columbia University (August 1982).

[10] C.E. Leiserson Area-Efficient VLSI Computation (MIT Press, Cambridge MA, 1982).

[11] H.A. Presnell and R.P. Pargas, in: Proc. 1981 Conf. on Functional Programming Languages and Computer Architecture, ACM (1981) p. 107.

[12] S.L. Johnsson, YALEK/CSD/RR-339 (October 1984).

[13] R.W. Hockney and C.R. Jesshope, Parallel Computers (Adam Hilger, London, 1981).