Supercomputing on Massively Parallel Bit-Serial Architectures

Consider the idea that supercomputing is a synergy of generic algorithms, languages and architectures and that real breakthroughs in parallel computing will be achieved by considering all three together in a simulated software environment. Engineering tradeoffs could be made between performance, machine transparency, standardization and program portability before any new machines are actually built. Standardized languages could be developed for generic subclasses of parallel machines; languages that really give high performance and encourage free parallel expression and "thinking in parallel".

My own research on the Goodyear MPP (Massively Parallel Processor), suggests that high-level parallel languages are practical and can be designed with powerful new semantics that allow algorithms to be efficiently mapped to the real machines. For the MPP these semantics include parallel/associative array selection for both dense and sparse matrices, variable precision arithmetic to trade accuracy for speed, micro-pipelined "train" broadcast, and conditional branching at the PE control unit level.

The preliminary design of a FORTRAN-like parallel language for the MPP has been completed and is being used to write programs to perform sparse matrix array selection, min/max search, matrix multiplication, Gaussian elimination on single bit arrays and other generic algorithms. The MPP timing estimate for Gaussian elimination of a 4K by 4K single bit matrix is under one second -- the equivalent of approximately 64 billion scalar operations. Parallel Gauss-Jordan matrix inversion is also being investigated. The estimated time to invert a 128 X 128, 32 bit real matrix using full pivoting on the MPP is 50 msec. This is roughly equivalent to a 100 MFLOP scalar rate.

The MPP is a SIMD machine of 16384 single bit processors arranged in a 128 X 128 array. Individual PE's are interconnected with their four nearest neighbors. Each PE can address 1024 bits of its own local memory. A 32 bit shift register in each PE allows for micro-pipelining of long words and faster partial sum accumulation for multiplication. The machine can execute 160 billion micro-instructions per second which translates to 800 GOPS for some instructions. Operations include single bit logical, shift, and add as well as column I/O and one or two dimensional routing in a spiral, cylinder, or torus. All operations can be directly or indirectly masked. The logical "or" of one bit per PE (SUMOR) can be used to pass array information back to the PE control unit for broadcast to other PE's, scalar I/O or conditional branching. If a second MPP were ever built, it might look considerably different than the current MPP. For example, it would certainly have greater memory depth -- at least 64K bits per PE. It might also have a reconfigurable bit/byte serial ALU, staged PE's for table lookup arithmetic, and pipelined SUMOR logic.

Ken Iobst
4/15/85
SUPERCOMPUTING ON MASSIVELY PARALLEL BIT-SERIAL ARCHITECTURES

- SUPERCOMPUTING DOMAIN
- NEW DIMENSIONS IN PARALLEL COMPUTING
- SOME GENERIC ALGORITHMS
- THE GOODYEAR MPP
- SOME MPP SPECIFIC ALGORITHMS CODED IN A FORTRAN-LIKE BIT-SERIAL PROGRAMMING LANGUAGE
- WHAT MIGHT A SECOND GENERATION MPP LOOK LIKE?
NEW DIMENSIONS IN PARALLEL COMPUTING

BIT-serial

DIVISION BY $2^n - 1$

MIN/MAX SEARCH
MATRIX MULTIPLICATION
COLUMN BROADCAST
GAUSSIAN ELIMINATION

LINEAR RECURRENCE

PARALLEL
PERFORMANCE WITH INTEGER OPERANDS

128 x 128 ARRAY
10 MHz CLOCK RATE

ADD ARRAY TO ARRAY
SUBTRACT ARRAY FROM ARRAY
MULTIPLY ARRAY BY ARRAY

MILLIONS OF OPERATIONS PER SECOND

OPERAND WORDLENGTHS, BITS

10,000
1,000
100
10
12603
6550
5851
3340
1862
2240
1680
1350
1120
970
850
546
253
146
82
58
44
34

OPERAND WORDLENGTHS, BITS

0 8 16 24 32 40 48 56 64

1-150
DIVISION BY $2^n \pm 1$ EXAMPLE

FROM THE BINOMIAL THEOREM,

$$\frac{1}{1 \pm x} = 1 \pm x + x^2 \pm x^3 + \cdots \; (x^2 < 1)$$

BY A CHANGE OF VARIABLE $y = \frac{1}{x}$ THEN

$$\frac{1}{y \pm 1} = \frac{1}{y} \pm \frac{1}{y^2} \pm \frac{1}{y^3} \pm \cdots \; (y^2 > 1)$$

NOW LET $y = 2^n$ AND DIVISION BY $2^n \pm 1$ REDUCES TO A SHORT SEQUENCE OF BINARY SHIFTS AND ADDS (AND/OR SUBTRACTS),

$$\frac{v}{2^n \pm 1} = \frac{v}{2^n} + \frac{v}{2^{2n}} + \frac{v}{2^{3n}} + \cdots$$

FOR EXAMPLE, LET $V = 237658$ AND $N = 10$ THEN

$$\frac{v}{2^n} = \frac{237658}{1023} = 232.315$$

AFTER 3 SHIFTS AND 2 ADDS
THE GOODYEAR MPP

- SIMD MACHINE OF 16384 SINGLE BIT PROCESSORS ARRANGED IN A 128 X 128 ARRAY
- NEAREST NEIGHBOR INTERCONNECTIVITY
- 1024 BITS OF MEMORY PER PE
- 32 BIT SHIFT REGISTER ALLOWS FOR MICRO-PIPELINING AND FASTER MULTIPLICATION
- EXECUTION SPEED OF 160 BILLION MICRO-INSTRUCTIONS PER SECOND WHICH TRANSLATES TO 800 GOPS FOR SOME INSTRUCTIONS
- OPERATIONS INCLUDE SINGLE BIT LOGICAL, SHIFT, AND ADD AS WELL AS COLUMN I/O AND ONE OR TWO DIMENSIONAL ROUTING IN A SPIRAL, CYLINDER, OR TORUS
- ALL OPERATIONS CAN BE DIRECTLY OR INDIRECTLY MASKED
- THE LOGICAL "OR" OF ONE BIT PER PE (SUMOR) CAN BE USED TO PASS ARRAY INFORMATION BACK TO THE PE CONTROL UNIT FOR BROADCAST, SCALAR I/O, OR CONDITIONAL BRANCHING
PARALLEL/ASSOCIATIVE ARRAY SELECTION

REAL S(8:24), A[64,256](8:24)

S = SUMOR(A[64,256])
MAXIMUM OF 32 BIT INTEGER ARRAY
(OF UNIQUE VALUES)

BIT MAX[ ]
INTEGER A[128,128](0:32)
MAX=1
DO 1 I=1,32
IF (SUMOR(A[MAX](I))) MAX=A[MAX](I)
1 CONTINUE

; DECLARE MAX AS BIT MASK OVER ALL PE'S
; DECLARE A AS A 128 X 128 UNSIGNED INTEGER ARRAY
; INITIALIZE MAX TO 1 OVER ALL PE'S
; SCAN BITS IN A FROM MOST TO LEAST SIGNIFICANT BITS
; REPLACE MAX WITH A NEW SUBSET OF MAXIMUM VALUES FOR EACH NON ZERO BIT PLANE OF A

MAXIMUM OF 32 BIT INTEGER ARRAY
(GENERAL CASE)

BIT MAX[ ],T[ ](46),INDEX[ ](14)
INTEGER A[128,128](0:32)
COMMON /INIT/ INDEX

MAX=1
T=A.CON.INDEX
DO 1 I=1,46
IF (SUMOR(T[MAX](I))) MAX=T[MAX](I)
1 CONTINUE

; SAME ALGORITHM AS BEFORE EXCEPT A ARRAY IS FIRST CONCATENATED WITH THE PE ADDRESS FIELD TO INSURE UNIQUENESS OF RESULT
MATRIX MULTIPLICATION EXAMPLE

REAL A[8,16,128](8:32), B[8,16,128](8:32),
& C[8,16,128](8:32), T[8,16,128](8:32)

READ A[,,1], B[1,,]
T=A[,,1...]*B[1...,,]
C=T[,,+,,]
PRINT C[,,1,]
COLUMN BROADCAST EXAMPLE

\[ A_{i,j} = \]

\[ A_{i,j} = A_{*,j} \]

REAL \[ A[128,128](8:32) \]
\[ A=A[ , , J\ldots] \]

OR

REAL \[ A[128,128](8:32) \]
BIT \[ M[ ] \]
\[ M=[128,128; ,J] \]
\[ A=A[\cdot NOT\cdot M][ ,128 \rightarrow] \]
PROBLEM: TO BROADCAST A COLUMN OF FLOATING POINT NUMBERS ACROSS THE MPP ARRAY

SOLUTION #1: WITH PE'S INTERCONNECTED IN AN E/W CYLINDER; LOAD, SHIFT AND STORE THE 32 BIT VALUES ACROSS THE ARRAY. THIS TAKES APPROXIMATELY $3 \times 32 \times 128 = 12288$ CYCLES.

SOLUTION #2: WITH PE'S INTERCONNECTED IN AN E/W CYLINDER; "TRAIN" BROADCAST THE 32 BIT VALUES ACROSS THE ARRAY. THIS CAN BE VIEWED AS A MICRO-Pipelining OPERATION AND TAKES ONLY 207 CYCLES. THE ALGORITHM IS AS FOLLOWS:

- GET "TRAIN" OF 1 STOP BIT + 32 BIT VALUES OUT ONTO THE E/W PE CHANNEL ($\approx 33$ CYCLES)

- CIRCULATE "TRAIN" ONCE AROUND ($\approx 128$ CYCLES). DURING THIS PROCESS INDIVIDUAL PE'S WILL STORE THE "TRAIN" IN THEIR SHIFT REGISTERS. SHIFTING STOPS WHEN THE STOP BIT ENTERS THE CONDITIONAL MASK REGISTER OF EACH PE.

- STORE ALL SHIFT REGISTERS ($\approx 32$ CYCLES).
GAUSSIAN ELIMINATION EXAMPLE

SINGLE BIT MATRIX

\[
\begin{pmatrix}
1 & 2 \\
10 & 11 & 12 & 13 \\
4_{0,0} & 4_{0,1} & 4_{0,2} & 4_{0,3}
\end{pmatrix}
\]

4000 x 4000

1 OF 1000 BIT PLANES

...MPP ARRAY...

128 x 128
GAUSSIAN ELIMINATION EXAMPLE

BIT A[4000,4](1000),M[4000,4],USED(4000)
INTEGER PIVOT(4000,0:14),J1(0:2),J2(0:12),J(0:14)
EQUIVALENCE (J1,J(1)),(J2,J(3))

READ A
DO 1 I=1,4000
     USED(I)=0
1 CONTINUE
DO 7 I=1,4000
   DO 2 J2=1,1000
      IF (SUMOR(A[I,J2])) GO TO 3
2 CONTINUE
   GO TO 8
3 CONTINUE
   DO 4 J1=1,4
      IF (SUMOR(A[I,J1](J2))) GO TO 5
4 CONTINUE
   5 CONTINUE
   PIVOT(1)=J
   USED(J)=1
   M=A[ ](J2).AND..NOT.[4000,4;I,J1] ; SAVE PIVOT COLUMN IN NEW MATRIX M, ZEROING THE PIVOT ROW VALUE
   DO 6 J2=1,1000
      A[ ](J2)=A[ ](J2).XOR.M[,J1...]
6 CONTINUE
7 CONTINUE
8 CONTINUE

; Read in array
; Initialize history matrix
; Search for a 1 in row I in steps of 4 columns
; Row of all 0's - exit
; Find which column of 4
; Save history information
; Eliminate 4 columns at a time by broadcasting the pivot column across the M array
GAUSS-JORDAN MATRIX INVERSION

WITH FULL PIVOTING
PARALLEL DATA STRUCTURES

REAL ARRAYS

\[ U = \begin{bmatrix} A \\ I \end{bmatrix} \] AUGMENTED MATRIX

\[ V = \begin{bmatrix} : \end{bmatrix} \] WORKING ARRAY

\[ W = \begin{bmatrix} I \end{bmatrix} \] WORKING ARRAY

BIT MASKS

\[ X = \begin{bmatrix} \bar{I} \\ 0 \end{bmatrix} \] PIVOTED ROW/COLUMNS

\[ Y = \begin{bmatrix} \bar{I} \\ \bar{I} \end{bmatrix} \] PIVOT ROW

WHERE \( I \) IS THE IDENTITY MATRIX

\( \bar{I} \) IS THE UNITY MATRIX

\( 0 \) IS THE ZERO MATRIX
OTHER DATA STRUCTURES

SCALARS

\[ \text{DET} = 1 \quad \text{PIVOT} \]
PARALLEL APPROACH TO MATRIX INVERSION

REPEAT FOLLOWING STEPS N TIMES

○ FIND NEXT PIVOT
○ UPDATE DETERMINATE (OPTIONAL)
○ ZERO PIVOT ROW AND COLUMN IN X
○ ZERO PIVOT ROW IN Y
○ NORMALIZE PIVOT ROW IN U
○ BROADCAST PIVOT ROW N TIMES INTO V
○ BROADCAST PIVOT COLUMN 2N TIMES INTO W
○ PERFORM PARALLEL ROW OPERATIONS FOR A SINGLE PIVOT
○ RESET PIVOT ROW IN Y

THEN REORDER ROWS IN U TO FORM

\[ U = [ I : A^{-1} ] \]
PARALLEL MATRIX INVERSION ALGORITHM

FOR I = 1 TO N
Pivot = Max |U| PER X
DET = DET * PIVOT

\[
X = 0
\]

\[
Y = 0
\]

\[
U = U / PIVOT
\]

\[
V = U
\]

\[
W = I
\]

\[
U = U - V * W PER Y
\]

\[
Y = 1
\]

END I

FOR J = 1 TO N
FOR I = 1 TO N
IF U[I, J] = 1 THEN V[J, *] = U[I, *]
END I

END J

U = V
MPP II: WHAT MIGHT IT LOOK LIKE?

- Much greater memory depth: at least 64K bits per PE, with at least one level of indirect addressing.
- Reconfigurable bit/nibble/byte serial ALU
- Staged PE's for table lookup arithmetic. How many tables? What size? RAM or ROM?
- Pipelined sumor logic