**VECTOR ALGORITHMS AND ARCHITECTURES**

**VECTOR AND MATRIX ALGORITHMS**

Let’s consider

Y=A\*x

where y, x – are N-component vectors, A[N,N]

$$y\_{i}=\sum\_{j=1}^{N}A\_{ij}∙x\_{j}, i=\vec{1,N, }A=\left(A\_{1},..,A\_{N}\right)$$

$$y\_{i}=\left(A\_{i},x\right),i=\vec{1,N}$$

$$A\_{i}=\left(A\_{i1}, A\_{i2}, .., A\_{iN}\right), i=\vec{1,N}$$

$$\sum\_{i=1}^{N}X\_{i}∙Y\_{i}=\left(X,Y\right), X∙Y$$

It may be implemented by

For i:=1 step 1 until N begin//$ i=\vec{1,N, }$

 y[i]:=0;//initialization

 For j:=1 step 1 until N//summation j=1,N

 y[i]:=y[i]+A[i,j]\*x[j];

End;

**Program 3-1. Matrix-vector multiply with dot-product inner loop**

**O(N\*N)=O(N2)**

If exchange inner and outer loops, we obtain:

For i:=1 step 1 until N

 Y[i]:=0;

For j:=1 step 1 until N

 For i:=1 step 1 until N

 Y[i]:=y[i]+A[I,j]\*x[j];

Y1=Y1+A(1,1)\*X1; Y2=Y2+A(2,1)\*X1; Y3=Y3+A(3,1)\*X1

For j:=1 step 1 until N

 Y[i]:=y[i]+A[i,j]\*x[j], (1<=i<=N);

(Y1,Y2,Y3) =(Y1,Y2,Y3)+(A11,A21,A31)\*X1

Y=Y+A(1column)\*X

**Program 3-2. Matrix-vector multiply with SAXPY**

In this form, the outer loop is over columns of A, and the inner loop multiplies all components of a column of A by one element x and adds this vector of products to the partial result, y. The basic vector operations in this form are multiplication of a vector by a scalar and vector addition. This operation is called SAXPY after the mathematical operations aX plus Y, where X and Y are vectors.

**VECTOR AND MATRIX ALGORITHMS (CONT 1)**

Let’s consider

C=A\*B,

$$C\_{ij}=\sum\_{k=1}^{N}A\_{ik}∙B\_{kj}, i,j=\vec{1,N, }$$

For i:=1 step 1 until N

 For j:=1 step 1 until N begin

 C[I,j]:=0;

 For k:=1 step 1 until N

 C[I,j]:=c[I,j]+A[I,k]\*B[k,j];

 end

where A[N,N], B[N,N], C[N,N] – are two-dimensional matrices.

We can use kij form of matrix multiplication:

1. Initialize the result matrix C to zero
2. Form the N\*N outer product matrix of column k of A with row k of B (outer product of x[N] and y[N] is a 2-dimensional matrix, ij-th element of which is xi\*yj, i=1,..,N, j=1,..,N)
3. Add the N\*N matrix of product terms to C
4. Repeat steps 2 and 3 for all N values of k

This algorithm is implemented as follows:

For i:=1 step 1 until N

 For j:=1 step 1 until N

 C[I,j]:=0;

For k:=1 step 1 until N

 For i:=1 step 1 until N

 For j:=1 step 1 until N

 C[I,j]:=c[I,j]+A[I,k]\*B[k,j];

For k:=1 step 1 until N

 C[I,j]:=c[I,j]+A[I,k]\*B[k,j], (1<=I,j<=N);

Time=O(N^3)=>O(N)

**Program 3-3. Matrix-matrix in kij form**

This form makes explicit addition of N\*N matrices.

A\*B=E=B\*A

$$B=A^{-1}$$

$$\left(\begin{matrix}y1\\y2\\y3\end{matrix}\right)=\left(\begin{matrix}1&0&0\\0&1&0\\0&0&1\end{matrix}\right)\*\left(\begin{matrix}x1\\x2\\x3\end{matrix}\right)=\left(\begin{matrix}1\*x1+0\*x2+0\*x3\\0\*x1+1\*x2+0\*x3\\0\*x1+0\*x2+1\*x3\end{matrix}\right)=\left(\begin{matrix}x1\\x2\\x3\end{matrix}\right)$$

$$A^{-1}∙A=A∙A^{-1}=E$$

Y=A\*x

$$A^{-1}\*Y=A^{-1}\*\left(A\*x\right)=(A^{-1}\*A)\*x=E\*x=x$$

$$\left(\begin{matrix}y1\\y2\\y3\end{matrix}\right)=\left(\begin{matrix}1&a&b\\0&1&c\\0&0&1\end{matrix}\right)\left(\begin{matrix}x1\\x2\\x3\end{matrix}\right)=\left(\begin{matrix}1\*x1+a\*x2+b\*x3\\0\*x1+1\*x2+c\*x3\\0\*x1+0\*x2+1\*x3\end{matrix}\right)=\left(\begin{matrix}x1+a\*x2+b\*x3\\x2+c\*x3\\x3\end{matrix}\right)$$

$$x2=y2-c\*x3=y2-c\*y3$$

$x3$=y3

A1=B1

A2=B2

A3=B3

A1+A2=B1+B2

A1-A2=B1-B2

K\*A1=K\*B1

A1/K=B1/K

(Y1=Y1/a11)=~~a11/a11\*~~x1+a12/a11\*x2+a13/a11\*x3

Y2=a21\*x1+a22\*x2+a23\*x3

Y3=a31\*x1+a32\*x2+a33\*x3

A21\*Y1’=a21\*(x1+a12’\*x2+a13’\*x3)

Y2=a21=0\*x1+a22\*x2+a23\*x3

Y3=a31\*x1+a32\*x2+a33\*x3

Y2-a21\*Y1/a11= a21\*x1+a22\*x2+a23\*x3- (a21\*x1+ a21\*a12’\*x2+ a21\*a13’\*x3)=a22\*x2+a23\*x3- a21\*a12’\*x2- a21\*a13’\*x3=

(a22- a21\*a12/a11)\*x2+(a23 - a21\*a13/a11)\*x3 .. (alk-alj\*ajk/ajj)\*xk

(aij+(-aij /akk) \*akj)\*xj

J=k

aik+(-aik /akk) \*akk= aik-aik =0

Aij=aij+q \*akj

Let’s consider solving of a system of linear algebraic equations by Gaussian elimination. The algorithm performs order of N3 operations on N\*N matrix. Usually it is made with selection of the order of operations to be performed, pivoting, to prevent round off error from destroying the accuracy of the result.

**VECTOR AND MATRIX ALGORITHMS (CONT 2)**

Let’s consider the simplified version without pivoting: Forward step of Gaussian elimination

For k:=1 step 1 until N-1 begin //over diagonal elements

 P:=1/a[k,k];

 A[k,k]:=p;

 For i:=k+1 step 1 until N begin//over rows below k-th diagonal element - //pivot

 Q:=-a[I,k]\*p; //a[I,k]/a[k,k]

 A[I,k]:=q;

 For j:=k+1 step 1 until N//over elements of i-th row

 A[I,j]:=a[i,j]+q\*a[k,j];

 End;

End;//end of loop on k

**Program 3-4. Row wise form of Gaussian elimination without pivoting**

We can get another form of Gaussian elimination by reordering of loops:

For k:=1 step 1 until N begin//over diagonal elements

 P:=1/a[k,k];

 A[k,k]:=p;

 For i:=k+1 step 1 until N//over rows below k-th diagonal element - //pivot

 A[I,k]:=-a[I,k]\*p;

 For j:=k+1 step 1 until N begin

 Q1:=a[k,j];

 For i:=k+1 step 1 until N

 A[I,j]:=a[I,j]+q1\*a[I,k];

 End;

End;

~~For i:=k+1 step 1 until N~~

 A[I,j]:=a[I,j]+q\*a[I,k], (k+1<=i<=N);k=N-1; k+1=N

**Program 3-5. Column-wise form of Gaussian elimination without pivoting**

**VECTOR AND MATRIX ALGORITHMS (CONT 3)**

Pivoting may be included in such a way:

Int N=10, m;

Real a(N,N);

For k:=1 step 1 until N begin

 /\* Index of maximum absolute value in column k\*/

 m:=idamax(a,k,N); idamax(a,k,10)

 piv[k]:=m;

 swap(a,k,m,N); /\*Exchange row k with row m\*/

 /\* program is identical from here on\*/

 p:=1/a[k,k];

 …

end;

**Program 3-6. Modifications to Gaussian elimination to handle pivoting**

Let’s consider used above functions:

Integer function idamax(a,k,N);

Int N;

Real a(N,N);

 M:=k;

 S:=abs(a[k,k]);

 For i:=k+1 step 1 until N

 If abs(a[I,k]) > s then begin

 M:=I;

 S:=abs(a[I,k]);

 End;

 Return m;

End function;

**Program 3-7. Search for the maximum absolute value element**

**VECTOR AND MATRIX ALGORITHMS (CONT 4)**

Procedure swap(a,k,m,N);

 For j:=k step 1 until N begin

 Tmp:=a[k,j];

 A[k,j]:=a[m,j];

 A[m,j]:=tmp;

 End;

End procedure;

**Program 3-8. Procedure to swap portions of rows to the right of the diagonal**

Let’s consider linear recurrence.

**Definition**: An m-th order linear recurrence system of n equations, R(n,m), is



J=max(1,i-m)

where . The case m=n-1 is called a general linear recurrence.

The recurrence can be written as a vector-matrix equation

x=c+A\*x,

where elements of matrix A satisfy the restriction Aij=0 if either i<=j or i>j+m. j<i-m => aij=0It means that matrix A has the lower triangular form with no more than m non-zero elements in each row.

We can solve linear recurrence by sequential computations:

X1=c1

X2=c2+A21\*x1=c2+a21\*c1

X3=c3+A31\*x1+A32\*x2

M=2; n=5

X4=c4+a42\*x2+a53\*x3

X5=c5+a53\*x3+a54\*x4

…

..

**VECTOR AND MATRIX ALGORITHMS (CONT 5)**

It means that all x components through x(i-1) must be known before xi can be computed. Let’s consider SIMD-style column sweep algorithm:

X[i]=c[i], (1<=i<=n); /\*initialize the x vector making x1 correct\*/

For j:=1 step 1 until n-1

 X[i]:=x[i]+A[I,j]\*x[j], (j+1<=i<=min(j+m,n));

/\*do all column j multiplies and add to vector x, completing x[j+1] \*/

**Program 3-9. Column sweep form of a linear recurrence solver**

Two interesting cases:

1. m=n-1

 - minimal number of processors providing least time of execution







1. m<<n , 





