7532

p. 30

N91-22320

Serial and Parallel Computation of Kane's Equations for Multibody Dynamics

by

Amir Fijany

Jet Propulsion Laboratory, California Institute of Technology

Objective:

Analysis of the Efficiency of Algorithms resulting from Kane's Equation for Serial and Parallel Computation of Mass Matrix.

Overview:

*** Algorithms resulting from Kane's Equation and Modified Kane's Equation.**

*** Analysis of two Classes of Algorithms for Computation of Mass Matrix: The Newton-Euler Based Algorithms and the Composite Rigid-Body Algorithms.**

*** Analysis of the Efficiency of Different Algorithms for Serial and Parallel Computation.**

*** Conclusion**

Multibody Dynamics

Case Study: Rigid Multibody as Specialized to a Single Chain Robot Manipulator.

Multibody Dynamics: Solution for $\ddot{Q}$ from

$$A\ddot{Q} = \tau - b = \Gamma \tag{1}$$

A: nxn Symmetric Positive definite Mass Matrix
$\ddot{Q}$: nx1 Vector of Generalized Accelerations
τ: nx1 Vector of Applied Forces/Torques
b: nx1 Vector of nonlinear Terms (Bias vector)
Γ: nx1 Vector of Applied Inertia Forces/Torques

The O(n^3) Algorithms for Multibody Dynamics:

1) Computation of b and Γ.

2) Computation of Mass Matrix A.

3) Solution of Eq. (1) by Inversion of A.

Kane's Equation is widely used for Computation of Mass Matrix.

Kane's Method: Notation

Q: nx1 Vector of Generalized Coordinates

U: nx1 Vector of Generalized Speeds

Choice of U: $U_i = \sum_{j=1}^{n} A_{ij}\dot{Q}_j + B_i$

Angular and Linear Velocity of Body (Link) i

$$\underline{\omega}_i = \sum_{j=1}^{n} \underline{\omega}_{i(j)} U_j + \underline{\omega}_{i(t)}$$

$$\underline{V}^*_i = \sum_{j=1}^{n} \underline{V}^*_{i(j)} U_j + \underline{V}^*_{i(t)}$$

ω_i: Angular Velocity of Body i

$\omega_{i(j)}$: jth Partial Angular Velocity of Body i

$\omega_{i(t)}$: Angular Velocity of Remainder Terms

V^*_i: Linear Velocity of Center of Mass of Body i

$V^*_{i(j)}$: jth Partial Linear Velocity of Center of Mass of Body i

$V^*_{i(t)}$: Linear Velocity of Center of Mass of Body i Remainder Terms

Kane's Method: Notation

Partial Angular and Linear Momentum

$$\underline{N}_{i(j)} = \underline{\underline{I}}_i \underline{\omega}_{i(j)}$$

$$\underline{F}_{i(j)} = m_i \underline{V}_{i(j)}$$

$N_{i(j)}$: jth Partial Angular Momentum of Body i

$F_{i(j)}$: jth Partial Linear Momentum of Body i

Kane's Equation for Computation of Mass Matrix

The element a_{ij} of Mass Matrix A is Computed as

$$a_{ij} = \sum_{k=j}^{n} \underline{V}^*_{k(i)} \cdot \underline{F}_{k(j)} + \underline{\omega}_{k(i)} \cdot \underline{N}_{k(j)}$$

$$= \sum_{k=j}^{n} \underline{V}^*_{k(i)} \cdot m_k \underline{V}^*_{k(j)} + \underline{\omega}_{k(i)} \cdot \underline{\underline{I}}_k \underline{\omega}_{k(j)}$$

Kane's Equation: Analysis of General Case

For Analysis of the General Case, We Set $U_i = \dot{Q}_i$

$\underline{\omega}_{i(j)} = \underline{Z}_j$ and $\underline{\omega}_{i(t)} = 0$

$$\underline{\omega}_i = \sum_{j=1}^{i} \underline{Z}_j \dot{Q}_j$$

$\underline{V}^*_{i(j)} = (\underline{Z}_j \times \underline{P}_{i*,j})$ and $\underline{V}^*_{i(t)} = 0$

$$\underline{V}^*_i = \sum_{j=1}^{i} (\underline{Z}_j \times \underline{P}_{i*,j})\dot{Q}_j$$

$$\underline{N}_{i(j)} = \underline{\underline{I}}_i \underline{\omega}_{i(j)} = \underline{\underline{I}}_i \underline{Z}_j$$

$$\underline{F}_{i(j)} = m_i (\underline{Z}_j \times \underline{P}_{i*,j})$$

Kane's Equation can be written as

$$a_{ij} = \sum_{k=j}^{n} (\underline{Z}_i \times \underline{P}_{k*,i}) . m_k (\underline{Z}_j \times \underline{P}_{k*,j}) + \underline{Z}_i . \underline{\underline{I}}_k \underline{Z}_j$$

AN O(n^3) Algorithm Based on Kane's Equation

```
For i = 1, 2, ..., n
   For j = i, i+1, ..., n
      $a_{ij} = \sum_{k=j}^{n} (\underline{Z}_i \times \underline{P}_{k*,i}) \cdot m_k (\underline{Z}_j \times \underline{P}_{k*,j}) + \underline{Z}_i \cdot \underline{\underline{I}}_k \underline{Z}_j$
```

This Algorithm is Designated as Original Kane's Equation (OKE) Algorithm.

Modified Kane's Equation

$$a_{ij} = \sum_{k=j}^{n} (\underline{Z}_j \times \underline{P}_{k*,j}) \cdot m_k (\underline{Z}_i \times \underline{P}_{k*,i}) + \underline{Z}_j \cdot \underline{\underline{I}}_k \underline{Z}_i$$

$$= \sum_{k=j}^{n} \underline{Z}_j \cdot (\underline{P}_{k*,j} \times (m_k \underline{Z}_i \times \underline{P}_{k*,i}) + \underline{Z}_j \cdot \underline{\underline{I}}_k \underline{Z}_i$$

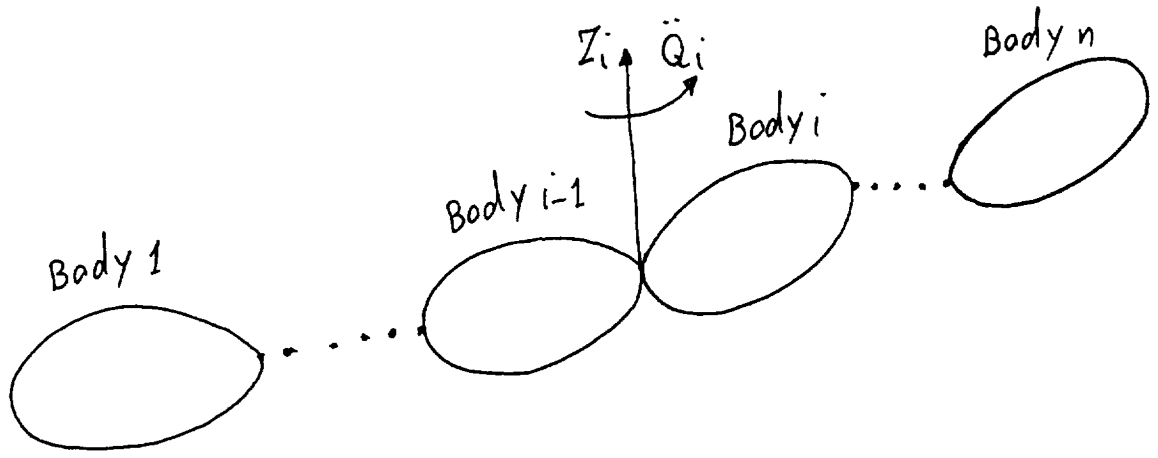
$$= \sum_{k=j}^{n} \underline{Z}_j \cdot ((\underline{P}_{k*,j} \times (m_k \underline{Z}_i \times \underline{P}_{k*,i}) + \underline{\underline{I}}_k \underline{Z}_i)$$

AN O(n^2) Algorithm Based on Kane's Equation

```
For i = 1, 2, ..., n
    For j = i, i+1, ..., n
        $\underline{P}_{j,i} = \underline{P}_{j-1,i} + \underline{P}_{j,j-1}$
        $\underline{P}_{j*,i} = \underline{P}_{j,i} + \underline{S}_j$
        $\underline{N}_{j(i)} = \underline{\underline{I}}_j \underline{Z}_i$
        $\underline{F}_{j(i)} = m_j(\underline{Z}_i \times \underline{P}_{j*,i})$
    For j = n, n-1, ..., i
        $\underline{f}_{j(i)} = \underline{F}_{j(i)} + \underline{f}_{j+1(i)}$
        $\underline{n}_{j(i)} = \underline{N}_{j(i)} + \underline{S}_j \times \underline{F}_{j(i)} + \underline{n}_{j+1(i)} + \underline{P}_{j+1,j} \times \underline{f}_{j+1(i)}$
        $a_{ij} = \underline{Z}_j \cdot \underline{n}_{j(i)}$
```

This Algorithm is Designated as Variant of Kane's Equation (VKE) Algorithm.

Algorithms for Computation of Mass Matrix



$A\ddot{Q} = \Gamma$ (1)

$a_{ij} = a_{ji} = \Gamma_i$ (2)

For the conditions given as

$\ddot{Q}_i = 1$ and $\dot{Q}_j = \ddot{Q}_{j \neq i} = 0$ For $j = 1, 2, \dots, n$ (3)

Two physical interpretations of Eqs. (2) & (3) lead to two classes of algorithms for computation of mass matrix:

1. The Newton-Euler Based (N-E B) Algorithms.
   Underlying Physical Concept: Propagation of acceleration among rigidly connected bodies.
   The Variant of Kane's Equation (VKE) Algorithm belongs to this class.

2. The Composite Rigid-Body (CRB) Algorithms.
   Underlying physical Concept: Propagation of force among rigidly connected bodies.

Algorithms for Computation of Mass Matrix

Clearly, the two physical interpretations are the same.
We have shown that the algorithms of the two classes can be transformed to one another.

From an algorithmic point of view, the main difference between the algorithms of the two classes is the presence of a two-dimensional recursion in Composite Rigid-Body Algorithms.

The main issue is to determine the best algorithm(s) for serial and parallel computation.

The Original Kane's Equation Algorithm is the least efficient since its computational complexity is of $\mathcal{O}(n^3)$.

The computational complexity of both the Newton-Euler Based Algorithms and Composite Rigid-Body Algorithms is of $\mathcal{O}(n^2)$. However, the Composite Rigid-Body Algorithms, in general, are more efficient.

Algorithms for Computation of Mass Matrix

There are four major redundancies in the Original Newton-Euler Based Algorithm which can be removed by:

1) Optimizing the Newton-Euler Formulation for the conditions given in Eq. (3),

2) Using a variant of Newton-Euler Formulation ,

3) Choosing a better coordinate frame for projection of equations.

4) Introducing a two-dimensional recursion in the computation which transforms it to an equivalent Composite Rigid-Body Algorithm.

A Variant of Newton-Euler Based Algorithm

```
Step 1:

For $j = 1, 2, \ldots, n$

    For $i = j, j+1, \ldots, n$

        $\underline{\dot{\omega}}(i,j) = \underline{Z}(j)$

        $\underline{\dot{V}}(i,j) = \underline{\dot{V}}(i-1,j) + \underline{\dot{\omega}}(i,j) \times \underline{P}(i,i-1)$

        $\underline{F}(i+1,i,j) = m(i)\underline{\dot{V}}(i,j) + \underline{\dot{\omega}}(i,j) \times \underline{h}(i)$

        $\underline{N}(i+1,i,j) = \underline{\underline{k}}(i)\underline{\dot{\omega}}(i,j)$

Step 2:

    For $i = n, n-1, \ldots, j$

        $\underline{F}(n+1,n+1,j) = \underline{N}(n+1,n+1,j) = 0$

        $\underline{F}(n+1,i,j) = \underline{F}(i+1,i,j) + \underline{F}(n+1,i+1,j)$

        $\underline{N}(n+1,i,j) = \underline{N}(i+1,i,j) + \underline{N}(n+1,i+1,j) +$
                      $\underline{P}(i+1,i) \times \underline{F}(n+1,i+1,j)$

        $a_{ji} = \underline{Z}(i) . \underline{N}(n+1,i+1,j)$
```

This algorithm results from removing the first two redundancies of the O N-E B Algorithm. It is clearly equivalent to the $O(n^2)$ algorithm resulting from the Kane's Equation or the Variant of Kane's Equation (VKA) Algorithm.

m(i)	Mass of body i.
h(i)	First moment of mass of body i about point O_i.
k(i)	Second moment of mass of body i about point O_i.
Z(i)	Axis of joint i
P(i,j)	Position vector from point j to point i.
ω(i,j)	Angular acceleration of body i resulting from the unit acceleration of joint j.
V(i,j)	Linear acceleration of body i (point O_i) resulting from the unit acceleration of joint j.
F(k+1,i,j)	Force exerted on point O_i due to the acceleration of bodies i through k, i.e., the bodies contained between points O_i and O_{k+1}, resulting from the unit acceleration of joint j.
N(k+1,i,j)	Moment exerted on point O_i due to the acceleration of body i through k, resulting from the unit acceleration of joint j.

A Variant of Composite Rigid-Body Algorithm

```
Step 1:

For $i = n, n-1, \ldots, 1$

    $M(i) = m(i) + M(i+1)$

    $\underline{H}(i) = \underline{h}(i) + \underline{H}(i+1) + M(i+1)\underline{P}(i+1,i)$

    $\underline{\underline{K}}(i) = \underline{\underline{k}}(i) + \underline{\underline{K}}(i+1) - \underline{\underline{M}}(i+1)\hat{\underline{\underline{P}}}(i+1,i)\hat{\underline{\underline{P}}}(i+1,i) -$
            $\hat{\underline{\underline{P}}}(i+1,i)\hat{\underline{\underline{H}}}(i+1) - \hat{\underline{\underline{H}}}(i+1)\hat{\underline{\underline{P}}}(i+1,i)$

    $\underline{f}(i) = \underline{Z}(i) \times \underline{H}(i)$

    $\underline{n}(i) = \underline{\underline{K}}(i)\underline{Z}(i)$

    $a_{ii} = \underline{Z}(i) \cdot \underline{n}(i)$

Step 2:

    For $j = i-1, i-2, \ldots, 1$

        $\underline{f}(j) = \underline{f}(j+1)$

        $\underline{n}(j) = \underline{n}(j+1) + \underline{P}(j+1,j) \times \underline{f}(j+1)$

        $a_{ji} = \underline{Z}(j) \cdot \underline{n}(j)$
```

M(i)	Mass of composite rigid-body i composed of bodies i through n.
H(i)	First moment of mass of composite rigid-body i about point O_i.
K(i)	Second moment of mass of composite rigid-body i about point O_i.

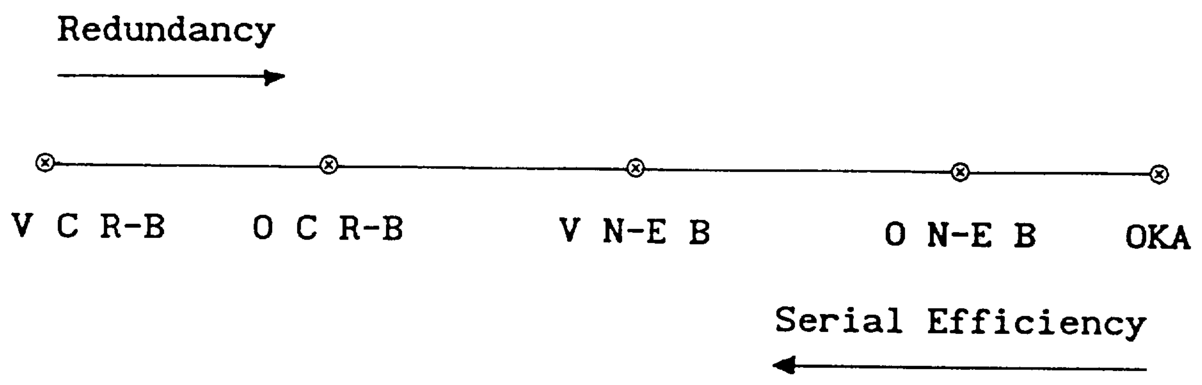
Comparison of Serial Efficiency of Different Algorithms

In order to study the relative efficiency of the algorithms, the optimal choice of coordinate frame(s) for projection of the Equations should be carefully analyzed.

For the Variant of Newton-Euler Algorithm, projection of all equations onto any fixed frame leads to maximum computational efficiency; It requires $\mathcal{O}(n)$ transformations. Projection onto the body frame leads to copmputational inefficiency; it requires $\mathcal{O}(n^2)$ transformations!

For the Variant of Composite Rigid-Body Algorithm, projection of Step 1 onto body frame and Step 2 onto any fixed frame leads to maximum computational efficiency; It requires $\mathcal{O}(n)$ transformations.
Projection of both steps onto the body frame leads to copmputational inefficiency; it requires $\mathcal{O}(n^2)$ transformations!

Comparison of Serial Efficiency of Different Algorithms



OKEA: Original Kane's Equation Algorithm.

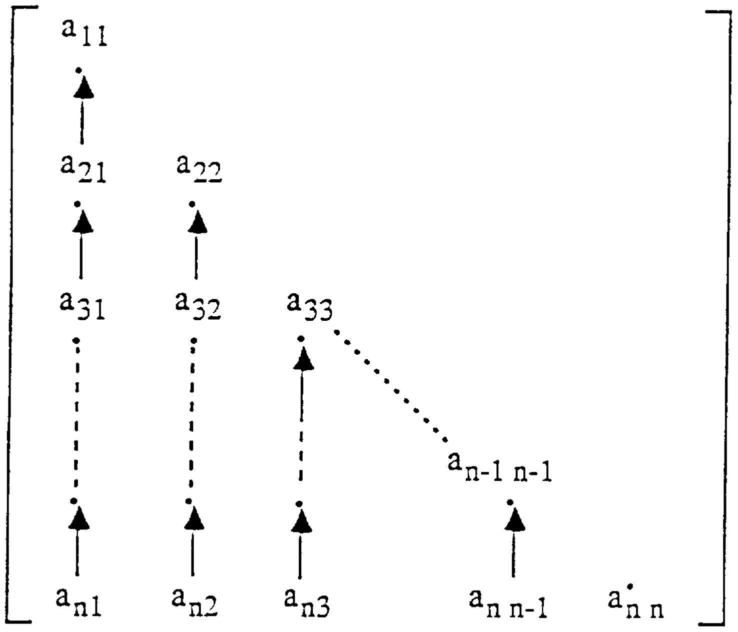
O N-E B: Original Newton-Euler Based Algorithm.

V N-E B: Variant of Newton-Euler Based Algorithm.

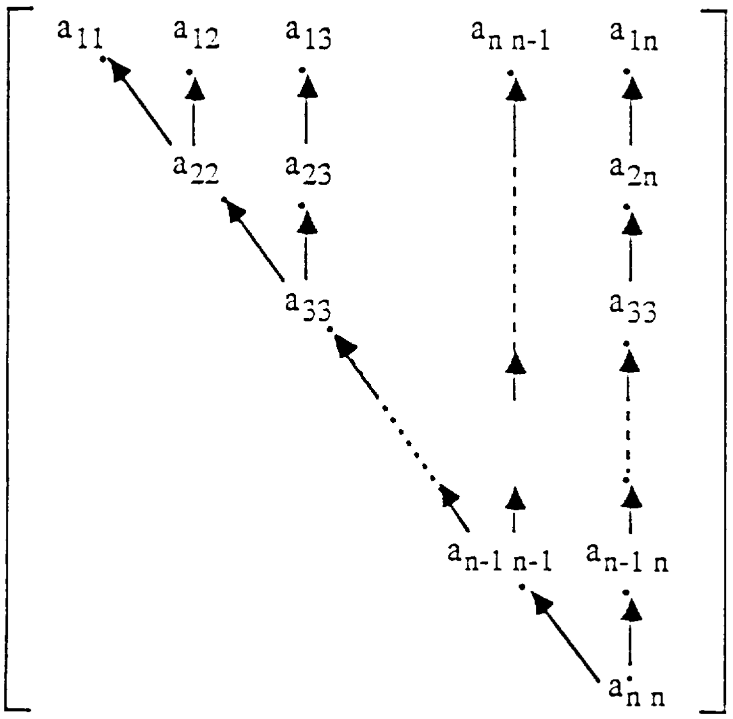
O C R-B: Original Composite Rigid-Body Algorithm.

V C R-B: Variant of Composite Rigid-Body Algorithm.

Algorithm	General		n = 6		
	Mul.	Add.	Mul.	Add.	Total
V N-E B	$(39/2)n^2+$ $(195/2)n-95$	$19n^2+$ $55n-66$	1192	948	2140
V C R-B	$(9/2)n^2+$ $(231/2)n-181$	$4n^2+$ $88n-137$	644	535	1179



(a)



(b)

Computational Structure of and Data-Dependency in Algorithms for Mass Matrix
a) The Newton-Euler Based Algorithms
b) The Composite Rigid-Body Algorithms

Algorithmic Choice for Parallel Computation of Mass Matrix

Parallelism in Computation of Mass Matrix: Time and Processors Bouds

We have shown that the time lower bound in computation of mass matrix is of $\mathcal{O}(\log_2 n)$ and can be achieved by using $\mathcal{O}(n^2)$ processors.

The Original Kane's Equation Algorithm might seem very suitable for parallel computation since all elements of the mass matrix can be computed totally in parallel.
The computation of each element of mass matrix can be performed in $\mathcal{O}(\log_2 n)$ steps by using $\mathcal{O}(n)$ processors. Hence, in order compute all the elements in parallel and achieve the time lower bound of $\mathcal{O}(\log_2 n)$, $\mathcal{O}(n^3)$ processors are required!

Using both the Newton-Euler Based Algorithms and the Composite Rigid-Body Algorithms, the mass matrix can be computed in $\mathcal{O}(\log_2 n)$ steps with only $\mathcal{O}(n^2)$ processors.

Algorithmic Choice for Parallel Computation of Mass Matrix

The Newton-Euler Based Algorithms are more suitable for parallel computation due to their regular computational structure and a lesser degree of data-dependency in their computation.

1) They provide a high degree of coarse grain parallelism:
   The columns of the mass matrix can be computed in parallel.

2) They are more regular and have a finer grain:
   A higher degree of parallelism in computation of the elements of each column can be exploited

3) Their parallel computation on a two-dimensional processor array requires simpler communication and synchronization mechanisms.

Choice of Coordinate Frame for Parallel Computation on a two-dimensional processor array:

For the Variant of Newton-Euler Based Algorithm it is more efficient to project the equations of onto the End-effector (Body n) frame while for the Variant of Composite Rigid-Body Algorithm it is more efficient to project the equations onto the base frame!

Conclusion

* For recursive serial computation, the Variant of Composite Rigid-Body Algorithm is significantly more efficient than the Variant of Newton-Euler and the Variant of Kane's Equation Algorithms.

* For parallel computation with $\mathcal{O}(n^2)$ processors, i.e., maximum exploitation of parallelism, the Variant of Newton-Euler and the Variant of Kane's Equation Algorithms are not only significantly more efficient than the Variant of Composite Rigid-Body Algorithm but they also require much simpler architectural features.

* For parallel computation with $\mathcal{O}(n)$ processors, i.e., limited exploitation of parallelism, the Variant of Composite Rigid-Body Algorithm is more efficient than the Variant of Newton-Euler and the Variant of Kane's Equation Algorithms

Comparison of Two Classes of Serial and Parallel Algorithms for Computation of Mass Matrix

Algorithm		Computation Cost (General)	Computation Cost ($n = 6$)	SP	Proc.
SA	VCR-B	$((9/2)m+4a))n^2+((231/2)m+88a))n-(181m+137a)$	$644m+535a$	-	1
	VN-EB	$((39/2)m+19a)n^2+((195/2)m+55a)n-(95m+66a)$	$1192m+948a$	-	1
PA	VCR-B	$(48m+63a)\lceil \log_2 n\rceil+(100m+65a)$	$244m+254a$	2.40	$n(n+1)/2$
	VN-EB	$(33m+33a)\lceil \log_2 n\rceil+(109m+89a)$	$208m+188a$	2.98	$n(n+1)/2$
PPA	VCR-B	$(9m+8a)n+(48m+63a)\lceil \log_2 n\rceil+(58m+24a)$	$256m+261a$	2.32	n
	VN-EB	$(39m+38a)n+(27m+18a)\lceil \log_2 n\rceil+(25m-2a)$	$340m+280a$	1.90	n

SA: Serial Algorithm.

PA: Parallel Algorithm with $O(n^2)$ processors.

PPA: Parallel Algorithm with $O(n)$ processors.

Parallel VNEB algorithm

Step 1:

1) Parallel compute $R(j+1,j)$ by all processors of Row j.

```
For $j = 1, 2, \ldots, n$
  For $i = 1, 2, \ldots, j$
    $PR_{ji}$ : $R(j+1,j)$
```

2) Parallel compute $R(n+1,j)$ by processors of Column i.

```
For $i = 1, 2, \ldots, n$
  For $j = i, i+1, \ldots, n$
    For $\eta$ = 1 step 1 until $\lceil \log_2(n+1-i) \rceil$, Do
      $R(j+2^{\eta},j) = R(n+1,j)$
                                $j+2^{\eta} > j+2^{\eta-1} \geq n+1$
      $R(j+2^{\eta},j) = R(n+1,j) = R(n+1,j+2^{\eta-1})R(j+2^{\eta-1},j)$
                                $j+2^{\eta} \geq n+1 > j+2^{\eta-1}$
      $R(j+2^{\eta},j) = R(j+2^{\eta},j+2^{\eta-1})R(j+2^{\eta-1},j)$
                                $n+1 > j+2^{\eta} > j+2^{\eta-1}$
    End_Do
```

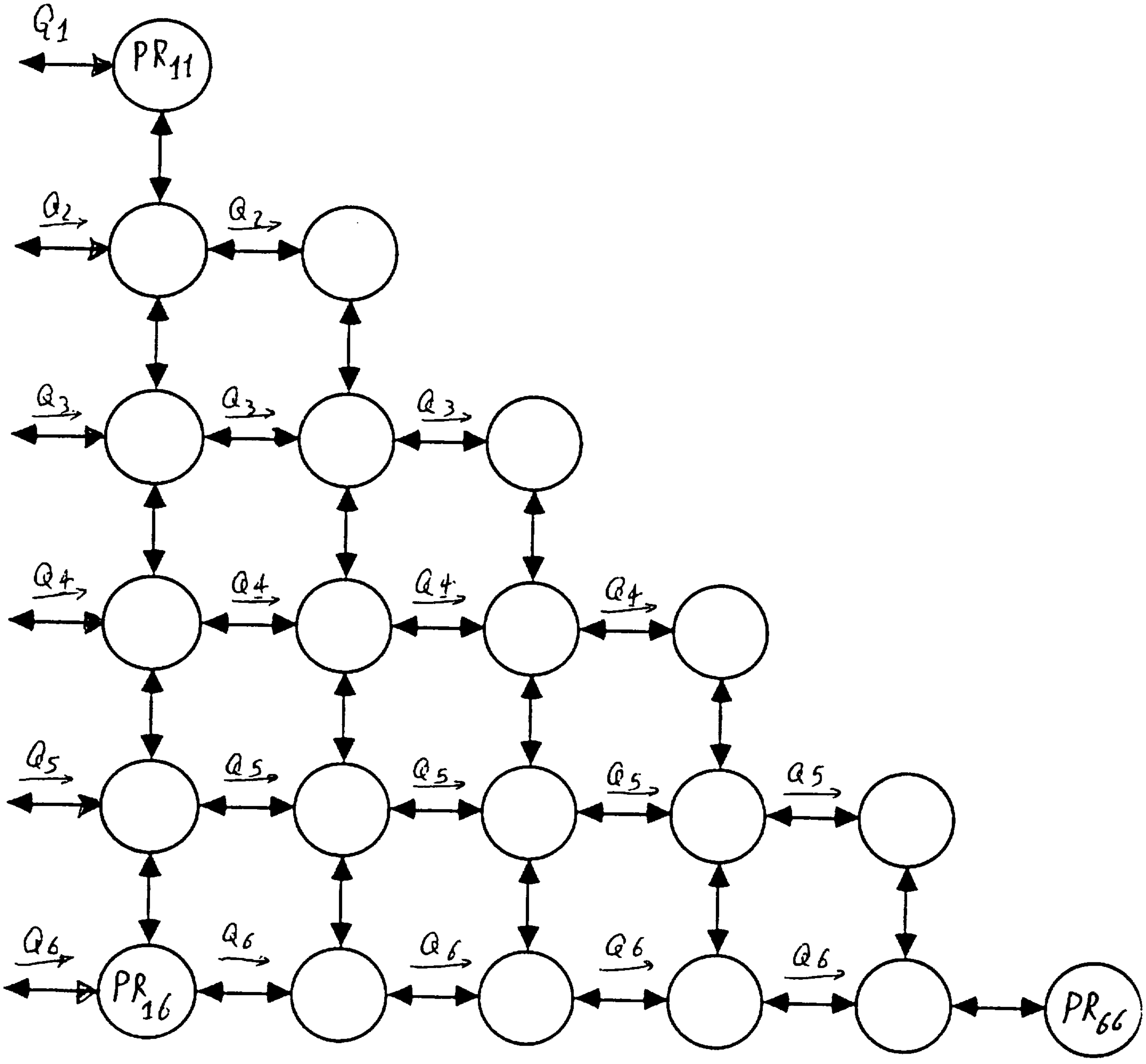
3) Shift $R(n+1,j+1)$ by processors of Row j+1 to the processors of Row j.

```
For $j = 1, 2, \ldots, n$
  For $i = 1, 2, \ldots, j$
    $PR_{ji}$: $R(n+1,j+1)$
    with $R(n+1,n+1) = U$ (Unit Matrix)
```

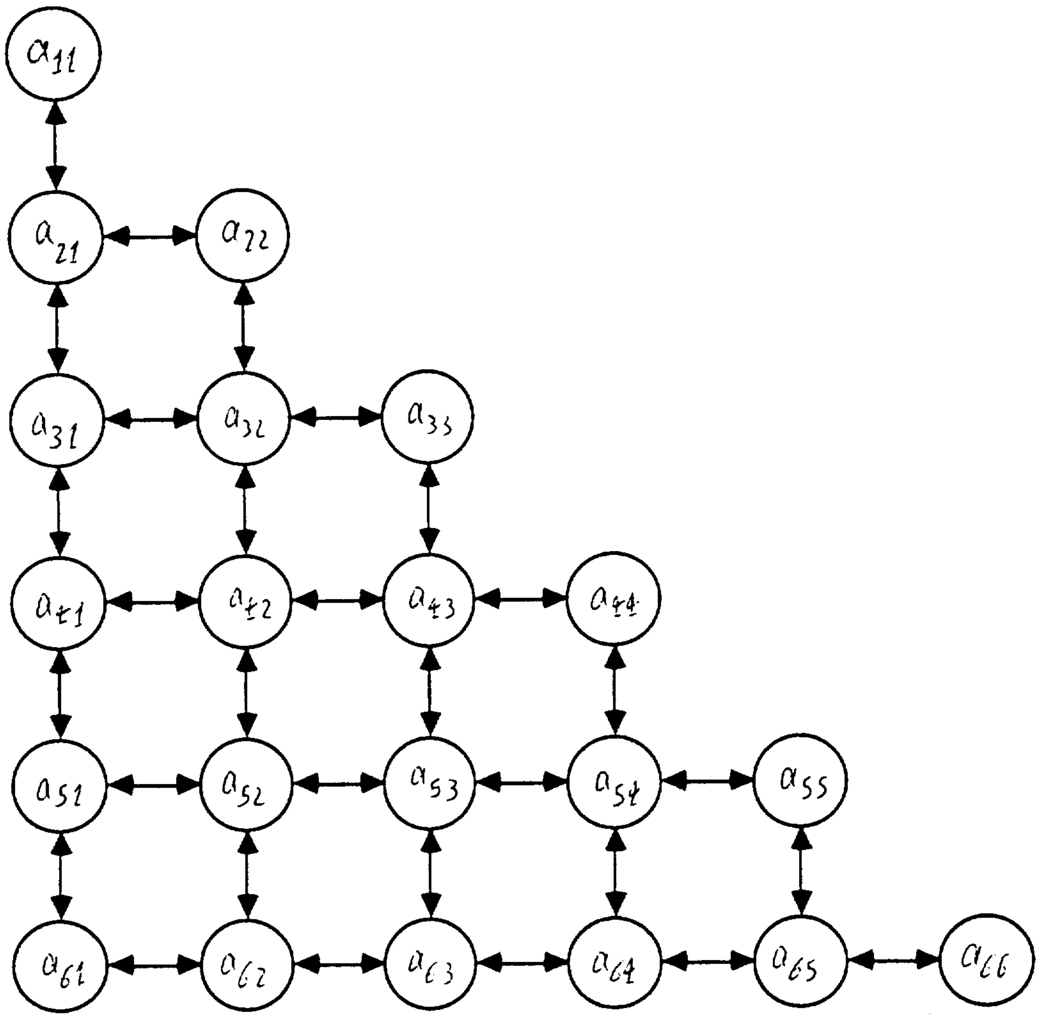
```
4) Parallel compute ${}^{n+1}Z(j)$, ${}^{n+1}P(j+1,j)$, and ${}^{n+1}H(j)$ by all
   processors of Row j.
For j = 1, 2 ,..., n
  For i = 1, 2, ..., j
    a) $PR_{ji}$ : ${}^{n+1}Z(j) = R(n+1,j)\,{}^{j}Z(j)$
       with ${}^{j}Z(j) = [0\ 0\ 1]^t$

    b) $PR_{ji}$: ${}^{n+1}P(j+1,j) = R(n+1,j+1)\,{}^{j+1}P(j+1,j)$

    c) $PR_{ji}$: ${}^{n+1}S(j) = R(n+1,j+1)\,{}^{j+1}S(j)$

    d) $PR_{ji}$: ${}^{n+1}H(j) = M(j)\,{}^{n+1}S(j)$

Step 2:
1) Parallel compute $P(j+1,i)$ and $\omega(j,i)$ by processors of Column i.
For i = 1, 2, ..., n
  For j = i, i+1, ..., n
    For $\eta$ = 1 step 1 until $\lceil \log_2(n+1-i) \rceil$, Do
        $\omega(j+2^{\eta},i) = \omega(j+2^{\eta-1},i) = Z(i)$
        $P(j+2^{\eta},j) = P(j+1,i)$
                                          $j+2^{\eta}>j+2^{\eta-1}\geq n+1$
        $P(j+2^{\eta},j) = P(j+1,i) = P(j+2^{\eta},j+2^{\eta-1})+P(j+2^{\eta-1},j)$
                                          $j+2^{\eta}\geq n+1>j+2^{\eta-1}$
        $P(j+2^{\eta},j) = P(j+2^{\eta},j+2^{\eta-1})+P(j+2^{\eta-1},j)$
                                          $n+1>j+2^{\eta}>j+2^{\eta-1}$

    End_Do
```

```
2) Parallel compute V(j,i), F(j+1,j,i), and N(j+1,j,i) by
   processors of Column i.
For i = 1, 2, ..., n
  For j = i, i+1, ...n
   a) $PR_{ji}$: $V(j,i) = \omega(j,i) x P(j+1,i) = Z(i) x P(j+1,i)$
   b) $PR_{ji}$: $F(j+1,j,i) = \omega(j,i) x H(j)+M(j)V(j,i)$
   c) $PR_{ji}$: $N(j+1,j,i) = R(n+1,j+1)\left({}^{j+1}K(j)R(j+1,n+1)^{n+1}\omega(j,i)\right)+ H(j) x V(j,i)$

Step 3:
1) Parallel compute F(n+1,j,i) processors of Column i.
For i = 1, 2, ..., n
  For j = i, i+1, ..., n
    For $\eta$ = 1 step 1 until $\lceil \log_2(n+1-j) \rceil$, Do
      $F(j+2^{\eta},j,i) = F(n+1,j,i)$
                                        $j+2^{\eta}>j+2^{\eta-1}\geq n+1$
      $F(j+2^{\eta},j,i) = F(n+1,j,i) = F(j+2^{\eta},j+2^{\eta-1},i)+F(j+2^{\eta-1},j,i)$
                                        $j+2^{\eta}>n+1>j+2^{\eta-1}$
      $F(j+2^{\eta},j,i) = F(j+2^{\eta},j+2^{\eta-1},i)+F(j+2^{\eta-1},j,i)$
                                        $n+1>j+2^{\eta}>j+2^{\eta-1}$
    End_Do
```

```
2) Shift $F(n+1,j+1,i)$ by processors of Row $j+1$ to processors of
   Row $j$.
For $j = 1, 2, \ldots, n$
  For $i = 1, 2, \ldots, j$
    $PR_{ji}$: $F(n+1,j+1,i)$
3) Parallel compute $N(n+1,j,i)$ by processors of Column $i$.
For $i = 1, 2, \ldots, n$
  For $j = i, i+1, \ldots, n$
 a) $PR_{ji}$: $N(j+1,j,i) = N(j+1,j,i)+P(j+1,j)xF(n+1,j+1,i)$
 b) For $\eta = 1$ step 1 until $\lceil \log_2(n+1-j) \rceil$, Do
      $N(j+2^{\eta},j,i) = N(n+1,j,i)$
                                        $j+2^{\eta}>j+2^{\eta-1}\geq n+1$
      $N(j+2^{\eta},j,i) = N(n+1,j,i) = N(n+1,j+2^{\eta-1},i)+N(j+2^{\eta-1},j,i)$
                                        $j+2^{\eta}\geq n+1>j+2^{\eta-1}$
      $N(j+2^{\eta},j,i) = N(j+2^{\eta},j+2^{\eta-1},i)+N(j+2^{\eta-1},j,i)$
                                        $n+1>j+2^{\eta}>j+2^{\eta-1}$
    End_Do
2) Parallel compute $a_{ji}$ by $PR_{ji}$.
For $i = 1, 2, \ldots, n$
  For $j = i, i+1, \ldots n$
    $PR_{ji}$: $a_{ji} = Z(j).N(n+1,j,i)$
```



Q_1
PR_{11}
Q_2
Q_2
Q_3
Q_3
Q_3
Q_4
Q_4
Q_4
Q_4
Q_5
Q_5
Q_5
Q_5
Q_5
Q_6
PR_{16}
Q_6
Q_6
Q_6
Q_6
PR_{66}


a_{11}
a_{21}
a_{22}
a_{31}
a_{32}
a_{33}
a_{41}
a_{42}
a_{43}
a_{44}
a_{51}
a_{52}
a_{53}
a_{54}
a_{55}
a_{61}
a_{62}
a_{63}
a_{64}
a_{65}
a_{66}

209

Algorithm-To-Architecture Mapping

Determination of an Algorithmically-Speciaslized Parallel Architecture for Efficient Implementation of the Algorithm.

1) Processors Interconnection and Communication Complexity

For perfect mapping:

a) The required interconnection among processors of each column is Shuffle Exchange augmented with Nearest-Neighbor (SENN).

b) The required interconnection among processors of each row is Nearest-Neighbor.

The perfect mapping leads to a communication complexity of $\mathcal{O}(\log_2 n)$. Mapping on an array with nearest-neighbor interconnection leads to the communication complexity of $\mathcal{O}(n)$.

2) Synchronization Mechanism

Exploitation of parallelism at two computational levels:

a) Coarse grain parallelism in computing columns of mass matrix, and

b) Fine grain parallelism in computing the elements of each column.

Global Clock-Based Synchronization Mechanism (similar to Systolic Array) for processors of each column, and Local Data Driven (similar to Wavefront Array) for processor of each row.