TEMPUS SJEP N^{o} 08353-94

Activity N^{o} 6.

Digital Signal Processing

Realization of Parallel DSP Algorithms on the Wavefront Array Processors

Dr. Seebauer Marta,
Prof. Sergios Theodoridis, Nastou Panagiotis

Athens - 1995

Topics

Introduction

Hardware and Software Environment

Typical Topologies for the Parallel DSP Algorithms

- *Synchronous virtual link*
- *Asynchronous link*

Realization of Parallel DSP Algorithms

- *Convolution*
- *MA (FIR) filter*
- *AR (IIR) filter*
- *ARMA filter*

Conclusions

Convolution

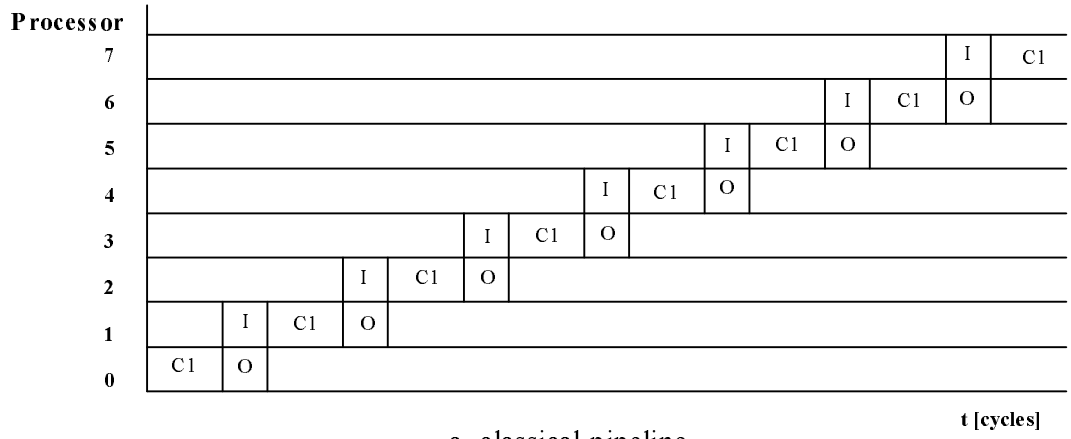
The algorithm for the convolution of two signals

$$u(i) \quad i = \overline{0, N-1}$$
$$w(i) \quad i = \overline{0, M-1} \qquad \text{and}$$

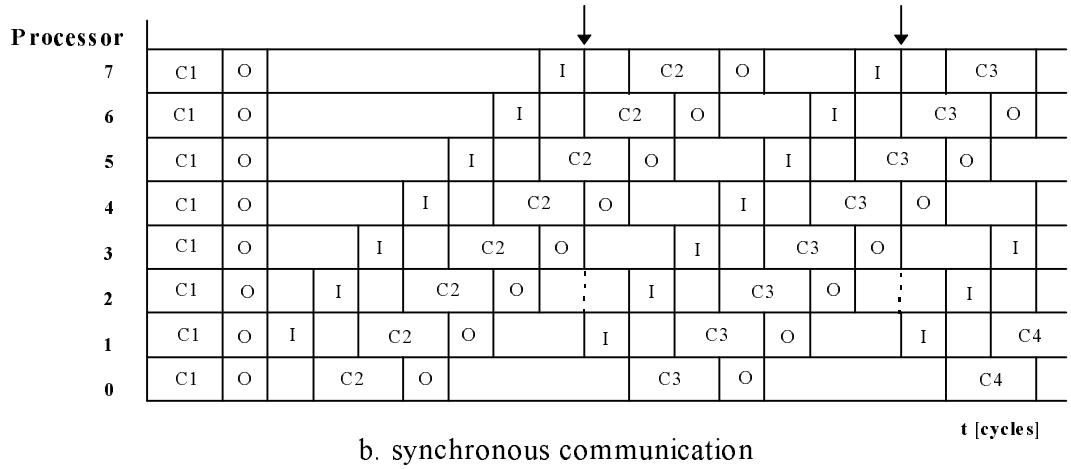
is given in formula 3.1.

$$y(i) = \sum_{j=0}^{i} u(j)w(i-j), \qquad if \quad i = \overline{0, M-1}$$
$$y(i) = \sum_{j=i-N+1}^{N-1} u(j)w(i-j), \qquad if \quad i = \overline{M, N+M-2} \qquad (3.1)$$

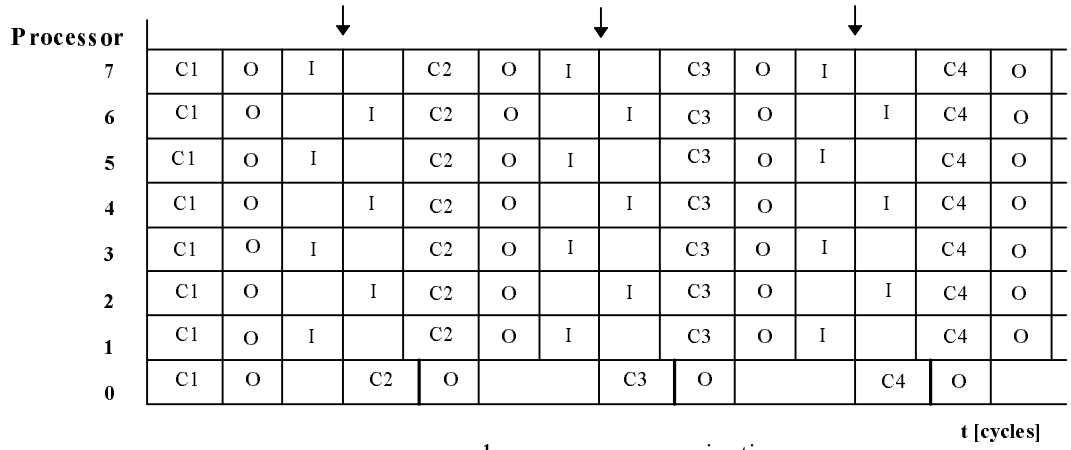
Processor activity diagram for program of convolution



a. classical pipeline



b. synchronous communication



c. asynchronous communication

Ci - count
O - output of data
I - input of data
- wait
- begin of next cycle

Classical version of the program for realization of convolution of two signal sequences with synchronous communication between the processor elements

```
#include <stdio.h>
#include <virt_top.h>
#include <sys/comm.h>
#include <sys/time.h>

#define     BACKWARD 0      /*virtual link from previous processor*/
#define     FORWARD  1      /*virtual link to next processor*/
#define     N       1000    /*number of input signal*/
#define     M        256    /*number of processor element*/

int main (void)
{
 int        first, second;
 double       work_time;

 int        i;
 int        u[N];
 int        w[M];
 int        y = 0;

 int        id;

 int        topId;
 int        reqId = 0;
 PipeData_t    *pipeData = NULL;

 int        pipe_len = M;

 topId = MakePipe(reqId, pipe_len, MINSLICE, MAXSLICE,
                  MINSLICE, MAXSLICE, MINSLICE, MAXSLICE);
 pipeData = GetPipe_Data(topId);

 id = pipeData->id;

 /* assignment for signal sequences u[i] and w[i] */
 for (i = 0; i < N; ++i)
     u[i] = i;
 for (i = 0; i < pipe_len; ++i)
     w[i] = i;
```

```
 /* convolution */

    switch(pipeData->status){
          case PIPE_NONE:
              break;

        case PIPE_HEAD:
                for (i = 0; i < N+pipe_len-1; ++i){
                    if (i > N-1)
                        y = 0;
                else
                        y = w[id] * u[i];
                    Send(topId, FORWARD, &y, sizeof(int));
             }
                 break;

        case PIPE_IN:
                for (i = 0; i < N+pipe_len-1; ++i){
                    Recv(topId, BACKWARD, &y, sizeof(int));
                    if (i < id || i > N+id-1 )
                        y = y + 0;
                    else
                        y = y + w[id] * u[i-id];
                    Send(topId, FORWARD, &y, sizeof(int));
                }
                break;

           case PIPE_TAIL:
                first = TimeNowHigh();
                for (i = 0; i < N+pipe_len-1; ++i){
                    Recv(topId, BACKWARD, &y, sizeof(int));
                    if (i < id)
                        y = y + 0;
                    else
                        y = y + w[id] * u[i-id];
                    printf("%6d\n", y);
                 }
                 second = TimeNowHigh();
                 work_time = (double) (second-first)/ (double) CLK_TCK_HIGH;
                 printf("\nWork time: %6.24f sec\n", work_time);
                 break;
      }
FreePipe(topId);
return 0;
}
```

Program for realization of convolution of two signal sequences with synchronous communication between the processor elements

```
#include <stdio.h>
#include <virt_top.h>
#include <sys/comm.h>
#include <sys/time.h>

#define     BACKWARD 0      /*virtual link from previous processor*/
#define     FORWARD  1      /*virtual link to next processor*/
#define     N       1000    /*number of input signal*/
#define     M        256    /*number of processor element*/

int main (void)
{
  int         first, second;
  double        work_time;

  int         i;
  int         u[N];
  int         w[M];
  int         y_send = 0;
  int         y_recv = 0;

  int         id;

  int         topId;
  int         reqId = 0;
  int         prev;
  PipeData_t    *pipeData = NULL;
  int         pipe_len = M;

  topId = MakePipe(reqId, pipe_len, MINSLICE, MAXSLICE,
                   MINSLICE, MAXSLICE, MINSLICE, MAXSLICE);
  pipeData = GetPipe_Data(topId);

  id = pipeData->id;

  /* assignment for signal sequences u[i] and w[i] */

  for (i = 0; i < N; ++i)
      u[i] = i;
  for (i = 0; i < pipe_len; ++i)
      w[i] = i;
```

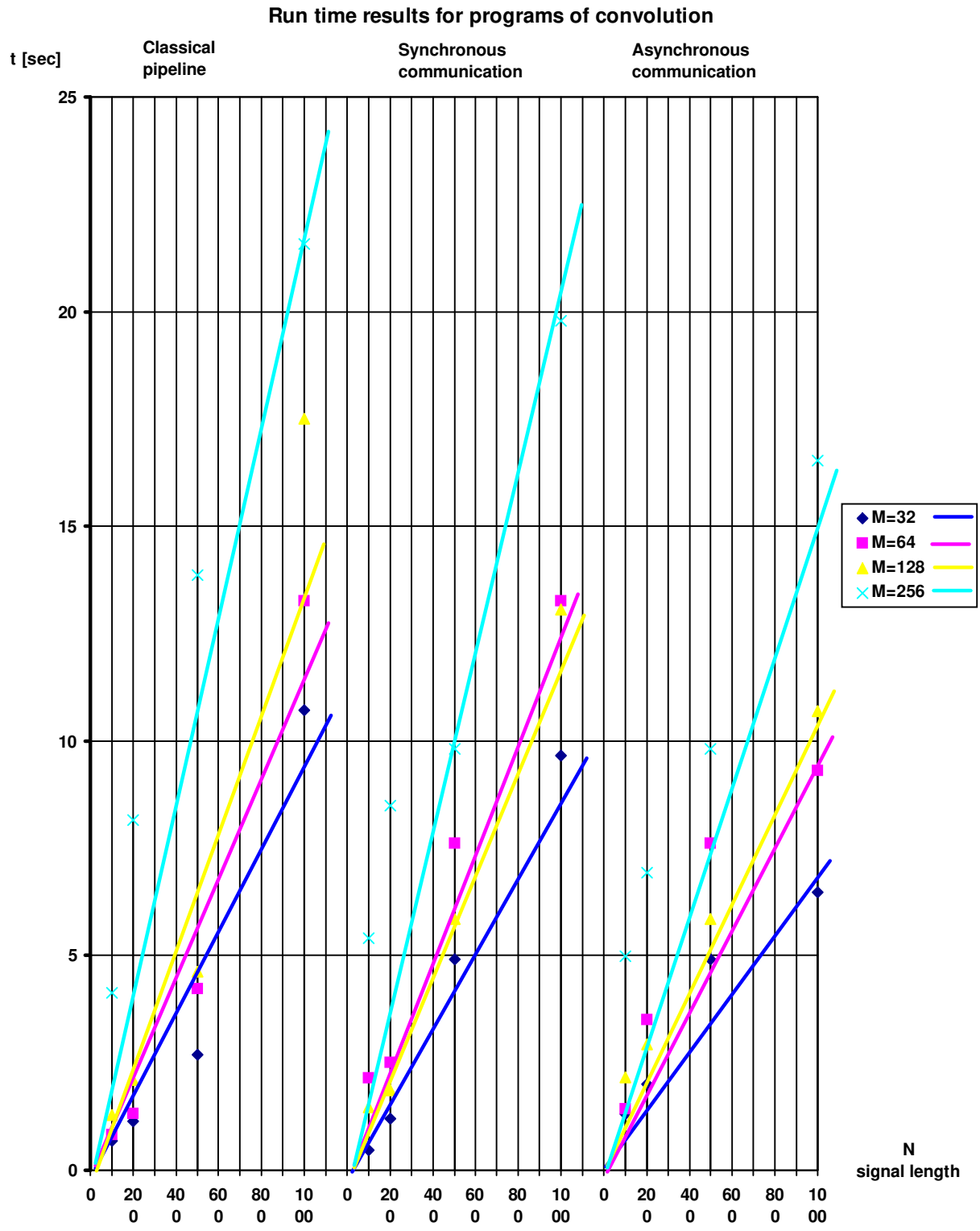
```
 /* convolution */
  switch(pipeData->status){
          case PIPE_NONE:
              break;

        case PIPE_HEAD:
                for (i = 0; i < N+pipe_len-1; ++i){
                    if (i > N-1)
                        y_send = 0;
                else
                        y_send = w[pipe_len-1-id] * u[i];
                    Send(topId, FORWARD, &y_send, sizeof(int));
            }
                 break;

        case PIPE_IN:
                for (i = 0; i < N+pipe_len-1; ++i){
                    if (i > N-1)
                        y_send = y_recv + 0;
                    else
                        y_send = y_recv + w[pipe_len-1-id] * u[i];
                    Send(topId, FORWARD, &y_send, sizeof(int));
                Recv(topId, BACKWARD, &y_recv,sizeof(int));
            }
                 break;

           case PIPE_TAIL:
                first = TimeNowHigh();          /*start timer*/
                for (i = 0; i < N+pipe_len-1; ++i){
                    if (i > N-1)
                        y_send = y_recv + 0;
                    else
                        y_send = y_recv + w[pipe_len-1-id] * u[i];
                    Recv(topId, BACKWARD, &y_recv, sizeof(int));
                    printf("%6d\n", y_send);     /*print result*/
                }
                 second = TimeNowHigh();        /*stop timer*/
             work_time = (double) (second-first)/CLK_TCK_HIGH;
                 printf("\nWork time: %6.24f mikrosec\n", work_time);
                 break;
   }
FreePipe(topId);
return 0;
}
```

Program for realization of convolution of two signal sequences with asynchronous communication between the processor elements

```
#include <stdio.h>
#include <virt_top.h>
#include <sys/comm.h>
#include <sys/time.h>

#define      BACKWARD 0      /*virtual link from previous processor*/
#define      FORWARD  1      /*virtual link to next processor*/
#define      N        1000   /*number of input signal*/
#define      M        256    /*number of processor element*/

int main (void)
{
 int         first, second;
 double      work_time;

 int         i;
 int         u[N];
 int         w[M];
 int         y_send = 0;
 int         y_recv = 0;
 int         id;

 int         topId;
 int         reqId = 0;
 PipeData_t    *pipeData = NULL;

 int         pipe_len = M;
 int         error;

 topId = MakePipe(reqId, pipe_len, MINSLICE , MAXSLICE,
                  MINSLICE, MAXSLICE, MINSLICE, MAXSLICE);
 AInit(topId, -1, -1);
 pipeData = GetPipe_Data(topId);

 id = pipeData->id;

 /* assignment for signal sequences u[i] and w[i] */
 for (i = 0; i < N; ++i)
     u[i] = i;
 for (i = 0; i < pipe_len; ++i)
     w[i] = i;
```

```
 /* convolution */

    switch(pipeData->status){
          case PIPE_NONE:
              break;

         case PIPE_HEAD:
                for (i = 0; i < N+pipe_len-1; ++i){
                    if (i > N-1)
                        y_send = 0;
                else
                        y_send = w[pipe_len-1-id] * u[i];
                    ASend(topId, FORWARD, &y_send, sizeof(int), &error);
                }
                break;

         case PIPE_IN:
                for (i = 0; i < N+pipe_len-1; ++i){
                    if (i > N-1)
                        y_send = y_recv + 0;
                    else
                        y_send = y_recv + w[pipe_len-1-id] * u[i];
                    ASend(topId, FORWARD, &y_send, sizeof(int), &error);
               ARecv(topId, BACKWARD, &y_recv, sizeof(int), &error);
                    ASync(topId, BACKWARD);
                }
                break;

           case PIPE_TAIL:
                first = TimeNowHigh();
                for (i = 0; i < N+pipe_len-1; ++i){
                    if (i > N-1)
                        y_send = y_recv + 0;
                    else
                        y_send = y_recv + w[pipe_len-1-id] * u[i];
                    ARecv(topId, BACKWARD, &y_recv, sizeof(int), &error);
                    ASync(topId, BACKWARD);
                    printf("%8d\n", y_send);
                }
                second = TimeNowHigh();
                work_time = (double) (second-first)/ (double) CLK_TCK_HIGH;
              printf("\nWork time: %6.24f sec\n", work_time);
              break;
     }
FreePipe(topId);
return 0;
}
```

Run time results for programs of convolution t [sec]				
Signal length	Pipe length			
N	M=32	M=64	M=128	M=256
Classical pipeline				
100	0,687	0,828	1,277	4,125
200	1,142	1,329	2,095	8,146
500	2,692	4,235	4,624	13,856
1000	10,724	13,254	17,516	21,578
Synchronous communication				
100	0,462	2,152	1,464	5,406
200	1,198	2,503	1,855	8,494
500	4,359	6,146	4,843	14,223
1000	9,654	13,263	13,051	19,798
Asynchronous communication				
100	1,301	1,432	2,154	4,990
200	2,001	3,508	2,917	6,924
500	4,903	7,616	5,858	9,807
1000	6,466	9,324	10,691	16,522


Run time results for programs of convolution
t [sec]
Classical pipeline
Synchronous communication
Asynchronous communication
25
20
15
10
5
0
0
200
400
600
800
1000
M=32
M=64
M=128
M=256
N
signal length

MA (FIR) filter

The algorithm for the MA filter is given in formula 3.2.

In point of view of the realization it is nothing else as replacing

w(i) to b_k

and

u(i) to x(i).

$$y(i) = \sum_{k=0}^{q} b_k x(i-k), \qquad (3.2)$$

Program for realization of FIR (MA) filter with asynchronous communication between the processor elements

```
#include <stdio.h>
#include <virt_top.h>
#include <sys/comm.h>
#include <sys/time.h>

#define     BACKWARD 0      /*virtual link from previous processor*/
#define     FORWARD  1      /*virtual link to next processor*/
#define     N       1000    /*number of input signal*/
#define     Q       256     /*number of processor element*/

int main (void)
{
 int        first, second;
 double       work_time;

 int        k;
 int        x[N];
 int        b[Q];
 int        y_send = 0;
 int        y_recv = 0;
 int        id;

 int        topId;
 int        reqId = 0;
 PipeData_t    *pipeData = NULL;

 int        pipe_len = Q;
 int        error;

 topId = MakePipe(reqId, pipe_len, MINSLICE , MAXSLICE,
                  MINSLICE, MAXSLICE, MINSLICE, MAXSLICE);
 AInit(topId, -1, -1);
 pipeData = GetPipe_Data(topId);

 id = pipeData->id;

 /* assignment for signal sequences x[i] and b[k] */
 for (k = 0; k < N; ++k)
     x[k] = k;
 for (k = 0; k < pipe_len; ++k)
     b[k] = k;
```

```
 /* filter function */

    switch(pipeData->status){
          case PIPE_NONE:
              break;

        case PIPE_HEAD:
                for (k = 0; k < N+pipe_len-1; ++k){
                    if (k > N-1)
                        y_send = 0;
                else
                        y_send = b[pipe_len-1-id] * x[k];
                    ASend(topId, FORWARD, &y_send, sizeof(int), &error);
                }
                break;

        case PIPE_IN:
                for (k = 0; k < N+pipe_len-1; ++k){
                    if (k > N-1)
                        y_send = y_recv + 0;
                    else
                        y_send = y_recv + b[pipe_len-1-id] * x[k];
                    ASend(topId, FORWARD, &y_send, sizeof(int), &error);
               ARecv(topId, BACKWARD, &y_recv, sizeof(int), &error);
                    ASync(topId, BACKWARD);
                }
                break;

          case PIPE_TAIL:
                first = TimeNowHigh();
                for (k = 0; k < N+pipe_len-1; ++k){
                    if (k > N-1)
                        y_send = y_recv + 0;
                    else
                        y_send = y_recv + b[pipe_len-1-id] * x[k];
                    ARecv(topId, BACKWARD, &y_recv, sizeof(int), &error);
                    ASync(topId, BACKWARD);
                    printf("%8d\n", y_send);
                }
                second = TimeNowHigh();
                work_time = (double) (second-first)/ (double) CLK_TCK_HIGH;
              printf("\nWork time: %6.24f sec\n", work_time);
              break;
     }
FreePipe(topId);
return 0;
}
```

3.3. AR (IIR) filter

The algorithm for the AR filter is given in formula 3.3.

$$y(i) = \sum_{k=1}^{p} a_k \, y(i-k) + x(i) \tag{3.3}$$

Realization of AR filter with synchronous communication between the processors

```
#include <virt_top.h>
#include <sys/time.h>
#include <vt_lib.h>
#include <sys/root.h>
#include <sys/link.h>
#include <stdlib.h>
#include <sys/sys_rpc.h>
#include <unistd.h>

#define  N  6
#define  FORWARD  1
#define  BACKWARD 0

int topid;
int reqid = 0;

void
main(void)
{
  int pipe_len = nProcs, temp;
  PipeData_t *pipeData = NULL;
  unsigned int first, second;
  double   work_time;
  int       i, k, error, pid;
  int       x[N+1], y[N+1], *a;

  topid = MakePipe(reqid, pipe_len, 0, DimX-1, 0, DimY-1, 0, MAXSLICE);
  pipeData = GetPipe_Data(topid);

  if(pipeData->status != PIPE_NONE){

    a = (int *) malloc((pipe_len+1) * sizeof(int));
    for(i=1; i < pipe_len+1; i++) a[i] = i;

    if (pipeData->status == PIPE_TAIL){
      first = TimeNowHigh();
      for(i=0; i < N+1; ++i) x[i] = i;
      y[0] = x[0];
    }
/*************** END OF INITIALIZATION STEP ********************/
```

```
    k = 1;
    temp = 0;
    pid=pipeData->id;
    while(k <= N){
     switch(pipeData->status){
       case PIPE_HEAD:
         Recv(topid, FORWARD, &y[k-1], sizeof(int));
         Send(topid, FORWARD, &temp, sizeof(int));
         temp = y[k-1] * a[pipe_len-pid];
         break;
       case PIPE_IN:
         Recv(topid, FORWARD, &y[k-1], sizeof(int));
            Send(topid, BACKWARD, &y[k-1], sizeof(int));
            Send(topid, FORWARD, &temp, sizeof(int));
         Recv(topid, BACKWARD, &temp, sizeof(int));
            temp += y[k-1] * a[pipe_len-pid];
            break;
          case PIPE_TAIL:
            Send(topid, BACKWARD, &y[k-1], sizeof(int));
            Recv(topid, BACKWARD, &temp, sizeof(int));
         y[k] = temp + a[1] * y[k-1] + x[k];
            break;
      }
      k++;
    }
  }
  if (pipeData->status == PIPE_TAIL){
    for(i=0; i < N+1; i++) printe("%d\n", y[i]);
    second = TimeNowHigh();
    work_time = (double) (second-first)/ (double) CLK_TCK_HIGH;
    printf("\nWork time: %6.2f sec\n", work_time);
    fflush(stdout);
  }
  FreePipe(topid);
  return;
}
/*********   END OF PROGRAM ******************?
```

3.4. ARMA filter

The algorithm for the ARMA filter is given in formula 3.4.

$$y(i) = \sum_{k=0}^{q} b_k x(i-k) + \sum_{k=1}^{p} a_k y(i-k) \qquad (3.4)$$

The realization of the ARMA filter combine

the topology of AR filter

with algorithm of MA filter.

Realization of ARMA filter with synchronous communication between the processors

```
#include <virt_top.h>
#include <sys/time.h>
#include <vt_lib.h>
#include <sys/root.h>
#include <sys/link.h>
#include <stdlib.h>
#include <sys/sys_rpc.h>
#include <unistd.h>

#define  N  6
#define  FORWARD  1
#define  BACKWARD 0

int topid;
int reqid = 0;

void
main(void)
{
  int pipe_len = nProcs, temp;
  PipeData_t *pipeData = NULL;
  unsigned int first, second;
  double   work_time;
  int       i, k, error, pid;
  int       x[N+1], y[N+1], *a, *b;

  topid = MakePipe(reqid, pipe_len, 0, DimX-1, 0, DimY-1, 0, MAXSLICE);
  pipeData = GetPipe_Data(topid);

  if(pipeData->status != PIPE_NONE){
    pid=pipeData->id;
    a = (int *) malloc((pipe_len+1) * sizeof(int));
    b = (int *) malloc(pipe_len * sizeof(int));
    for(i=1; i < pipe_len+1; i++) a[i] = 1;
    for(i=0; i < pipe_len; i++) b[i] = 1;
    for(i=0; i < N+1; ++i) x[i] = 1;

    if (pipeData->status == PIPE_TAIL){
      first = TimeNowHigh();
      y[0] = b[0]*x[0];
    }
    else {
      temp = b[pipe_len-pid-1]*x[0];
    }
/*************** END OF INITIALIZATION STEP ********************/
```

```
    k = 1;
    while(k <= N){
     switch(pipeData->status){
       case PIPE_HEAD:
         Recv(topid, FORWARD, &y[k-1], sizeof(int));
         Send(topid, FORWARD, &temp, sizeof(int));
         temp = b[pipe_len-1-pid] * x[k] + y[k-1] * a[pipe_len-pid];
         break;
       case PIPE_IN:
         Recv(topid, FORWARD, &y[k-1], sizeof(int));
            Send(topid, BACKWARD, &y[k-1], sizeof(int));
            Send(topid, FORWARD, &temp, sizeof(int));
         Recv(topid, BACKWARD, &temp, sizeof(int));
            temp += b[pipe_len-1-pid] * x[k] + y[k-1] * a[pipe_len-pid];
            break;
          case PIPE_TAIL:
            Send(topid, BACKWARD, &y[k-1], sizeof(int));
            Recv(topid, BACKWARD, &temp, sizeof(int));
         y[k] = temp + b[pipe_len-1-pid] * x[k] + a[1] * y[k-1];
            break;
      }
      k++;
    }
  }
  if (pipeData->status == PIPE_TAIL){
   for(i=0; i < N+1; i++) printe("%d\n", y[i]);
   second = TimeNowHigh();
   work_time = (double) (second-first)/ (double) CLK_TCK_HIGH;
   printf("\nWork time: %6.2f sec\n", work_time);
   fflush(stdout);
  }
  FreePipe(topid);
  return;
}
```