# Parallel signal processing and system simulation using aCe

**John E. Dorband**
*NASA Goddard Space Flight Center, Greenbelt, MD 20771*
*John.E.Dorband@.nasa.gov*

**Maurice F. Aburdene**
*Department of Electrical Engineering, Bucknell University, Lewisburg, Pa 17837*
*aburdene@bucknell.edu*

**Abstract:** Recently, networked and cluster computation have become very popular for both signal processing and system simulation. A new language is ideally suited for parallel signal processing applications and system simulation since it allows the programmer to explicitly express the computations that can be performed concurrently. In addition, the new C based parallel language (aCe C) for architecture-adaptive programming allows programmers to implement algorithms and system simulation applications on parallel architectures by providing them with the assurance that future parallel architectures will be able to run their applications with a minimum of modification. In this paper, we will focus on some fundamental features of aCe C and present a signal processing application (FFT).

## 1. Introduction

Parallel and networked computer programming techniques have become popular and important computational tools for system simulation and signal processing[1]. The essential elements of parallel programming are concurrent computation (threads) and communications (messages). aCe is based on the concept of structured parallel execution [2-3].. First the programmer designs a virtual architecture that reflects the spatial organization of an algorithm. An example is shown in Fig. 1. A virtual architecture may consist of groups or bundles of threads of execution. Second the application program reflects the temporal organization of the algorithm. The code defines what each thread performs, which together with the virtual architecture, defines the algorithm's execution. aCe allows the programmer to both envision the most logical architecture for the application and then implement the algorithm using that architecture.. The aCe language allows the programmer to explicitly express that which can be performed concurrently. The purpose of aCe is to facilitate the development of parallel programs by allowing programmers to explicitly describe the parallelism of an algorithm.

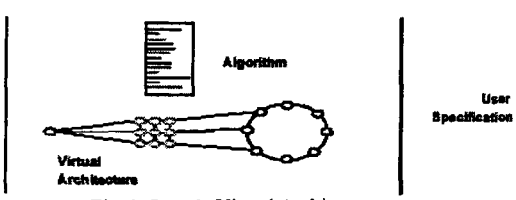


Fig. 1. Sample Virtual Architecuture

### 1.1 aCe micro-threads

One of the essential elements of concurrent computations is micro-threads. The following aCe program shows an example of micro-threads.

```
threads A[100];

threads B[10];

A int av;

B.{ ...

        A.{ int bv,cv;

                if (av==bv) cv=0 ;

        }

}
```

Any C code is valid, except 'goto's.

The difference between aCe and C is that while C has only one thread of execution, aCe, may have many threads of execution. Each thread may be referenced by name and index. Parallelism in aCe is expressed by first defining a set of concurrently executable threads. A group of parallel threads can be viewed as a *bundle* of executing threads, a *bundle* of processes, or an array of processors. These three views will be treated synonymously. In aCe, a bundle of threads is defined with the 'threads' statement. The statement "threads A[100]" only declares the intent of the programmer to use 100 concurrent threads of executions named 'A' at some later point in the

code. The statement A int av allocates an integer 'av' for each of the 100 threads of bundle 'A'.

### 1.2 aCe micro-messages

The second essential element of concurrent programming is micro-messages. Here is an example program showing communication.

```
threads A[100];

threads B[100];

A.{ int av,cv; B int bv;
        av = A[$$i+1].cv ; /* get operation */
        B[$$i/10].bv += av ; /* put operation */
}
/* Note: $$i is a threads unique ID */
```

### 1.3 Virtual Architecture

One of the major advantages of aCe for signal processing is that it allows the programmer to create a virtual architecture for the algorithm. Fig 2-6 show various examples of virtual architectures and communications along with the aCe code.

- threads X[1] ;
- threads { Z[2][2] } Y[3][3] ;

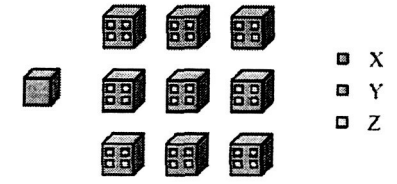


Fig. 2. Virtual Architecture

```
threads A[2][2] ;
A.{ int a=$Si; int b=3; int c;
   c = a*b ;
}
```

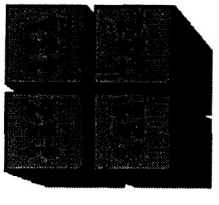

Fig. 3. No Communications

```
threads A[512][512] ;
a = .A[0][-1].c ; /* North */
a = .A[0][1].c ; /* South */
a &= .A[-1][0].c ; /* West */
a -= .A[1][0].c ; /* East */
```

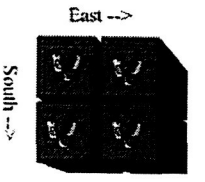


Fig. 4. 2-D Relative Communications

```
threads A[512][512] ;
a = A[0][$$ix[0]-1].c ; /* North */
a = A[0][$$ix[0] +1].c ; /* South */
a += A[$$ix[1]-1][0].c ; /* West */
a >?= A[$$ix[1]+1][0].c ; /* East */
```

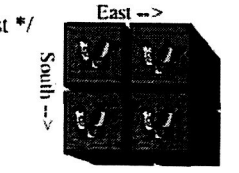


Fig. 5. Absolute Communications

- Aggregate Operations ( +, -, &, |, ^, >?, <? )
- threads X[1] ; threads { Z[2][2] } Y[3][3] ;
- Y.{ int a; X int b; X.b += a; }

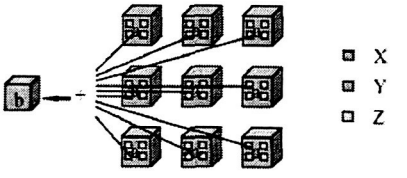


Fig. 6. Global Aggregate

- threads NODE[1000] ;
- threads { Corner[3] } CELL[2000] ;

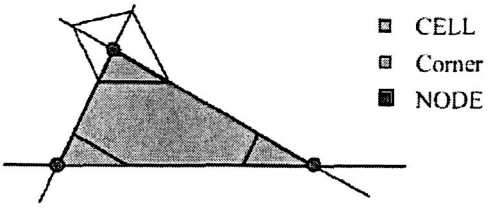


Fig. 7. Finite Element Example

## 2. FFT Example

The fast Fourier transform, FFT, is widely used in science and engineering and is easily implemented on parallel architectures [4-5]. Gupta and Kumar [6] presented the scalability analysis of an FFT algorithm for both mesh and hypercube architectures.

The aCe program shown in Appendix A performs a NxN 2-D FFT and inverse FFT on test data. This program performs N 1-D FFTs in parallel (partialFFT) in one dimension, transposes (Transpose) the transformed data, performs a second set of 1-D FFTs, and performs another transposes of the resulting transformed data. The 1-D FFT results in shuffled data and is unshuffled (UnShuffle) immediately after the 1-D FFT is performed.

The data was moved from the NxN 2-D bundle of threads to a N*N/2 1-D bundle of threads, thus the 2-D FFT was actually performed on a 1-D architecture.

## 3. Summary

This paper presented the fundamental concepts of a new C based parallel programming paradigm, aCe C. The intent of aCe is to allow a programmer to easily express the parallelism of an algorithm, allowing the compiler to more easily map the parallelism of that algorithm on to various parallel architectures. The ease of mapping facilitates the porting of programs and development /runtime environments to different architectures, promoting the development of new architectures. In addition, a parallel implementation of a 2-D FFT was presented.

## 4. References

[1]G.R. Andrews, *Foundations of multithreaded, parallel, and distributed programming*, Addison-Wesley, 2000.
[2] J.E. Dorband and M.F. Aburdene, "Architecture-adaptive computing environment: a tool for teaching parallel programming", Proceedings-Frontiers in Education Conference, Vol. 3, pp. S2F 7-S2F/12, 2002.
[3] J.E. Dorband, "aCe C Reference", NASA Goddard Space Flight Center, Greenbelt, MD.
[4] Michael J. Quinn, *Parallel Computing: Theory and Practice*, McGraw-Hill, Inc., 1994.
[5] Harold S. *Stone, High-Performance Computer Architecture*, Addison-Wesley, Mass., 1987.
[6] A. Gupta and V. .Kumar, "The scalability of FFT on parallel computers", IEEE Transactions. on Parallel and Distributed Systems, V. 4, no. 8, August 1993, pp. 922-932.

## 5. Acknowledgments

This research was funded in part by NASA grant NAG5-10821.

## Appendix A: Parallel FFT Program

```
# include <stdio.aHr>
# include <math.aHr>

# define real
      double
# define twoPI
      ((double)6.28318530717959)
# define PI
      (twoPI/2)
# define size 512
# define radius 3
# define nproc2 (size*size)
# define nproc  (nproc2>>1)

struct complex { real rr,ii ; } ;
typedef struct complex complex ;


generic void ComplexAdd (
   complex *C,
   complex *A,
   complex *B ) {
C->rr = A->rr + B->rr ;
C->ii = A->ii + B->ii ;
}

generic void ComplexSub (
   complex *C,
   complex *A,
   complex *B ) {
C->rr = A->rr - B->rr ;
C->ii = A->ii - B->ii ;
}

generic void ComplexMult (
   complex *C,
   complex *A,
   complex *B ) {
real D;
D     = A->rr*B->rr - A->ii*B->ii ;
C->ii = A->rr*B->ii + A->ii*B->rr ;
C->rr = D ;
}


generic complex Init_FFT
      ( int nxproc ) {
      real i; complex WS1 ;
```

```
/* initialize fft coefficients */

i = $$i%(nxproc>>1) ;
WS1.rr = cos((twoPI*i)/nxproc) ;
WS1.ii = sin((twoPI*i)/nxproc) ;

return WS1 ;
}


generic(WorkSpace[])void partialFFT
      ( complex A[2],
        complex *W1,
        int nxproc ) {
   complex A1,A2 ;
   complex Z1,Z2,C ;
   int i,iproc,n ;

   iproc = $$i ;
   n = nxproc>>2 ; /* nxproc/4 */
   A1 = A[0] ;
   A2 = A[1] ;

   ComplexAdd  (&C,&A1,&A2) ;
   ComplexSub  (&A2,&A1,&A2) ;
   ComplexMult (&A2,&A2,W1) ;
   A1 = C ;
   ComplexMult (W1,W1,W1) ;

   for ( i=n ; i>0 ; i>>=1 ) {

      if ( (iproc&i)!=0 ) {
      W1->rr = -W1->rr ;
      W1->ii = -W1->ii ; }

      Z1 = .WorkSpace[ i].A1 ;
      Z2 = .WorkSpace[-i].A2 ;
      if ( (iproc&i)==0 ) A2=Z1 ;
      else A1=Z2 ;

      ComplexAdd  (&C,&A1,&A2) ;
      ComplexSub  (&A2,&A1,&A2) ;
      ComplexMult (&A2,&A2,W1) ;
      A1 = C ;
      ComplexMult (W1,W1,W1) ;

      Z1 = .WorkSpace[ i].A1 ;
      Z2 = .WorkSpace[-i].A2 ;
      if ( (iproc&i)==0 ) A2=Z1 ;
      else A1=Z2 ;

   }

   A[0] = A1 ;
   A[1] = A2 ;
}
```

```
generic(WorkSpace[]) void UnShuffle
      ( complex A[2], int nxproc ) {
   complex A1,A2 ;
   complex Z1,Z2 ;
   int i,j,iproc,n ;

   iproc = $$i ;
   n = nxproc>>2 ; /* nxproc/4 */
   A1 = A[0] ;
   A2 = A[1] ;

   for( i=n,j=2; i>j; i>>=1,j<<=1 ){

      Z1 = .WorkSpace[ j].A1 ;
      Z2 = .WorkSpace[-j].A2 ;
      if ( (iproc&j)==0 ) A2=Z1 ;
      else A1=Z2 ;

      Z1 = .WorkSpace[ i].A1 ;
      Z2 = .WorkSpace[-i].A2 ;
      if ( (iproc&i)==0 ) A2=Z1 ;
      else A1=Z2 ;

      Z1 = .WorkSpace[ j].A1 ;
      Z2 = .WorkSpace[-j].A2 ;
      if ( (iproc&j)==0 ) A2=Z1 ;
      else A1=Z2 ;

   }

   Z1 = .WorkSpace[ 1].A1 ;
   Z2 = .WorkSpace[-1].A2 ;
   if ( (iproc&1)==0 ) A2=Z1 ;
   else A1=Z2 ;

   A[0] = A1 ;
   A[1] = A2 ;
}


generic(WorkSpace[]) void Transpose
   ( complex A[2], int nxproc ) {
   complex A1,A2 ;
   complex Z1,Z2 ;
   int i,j,iproc,n0,n1 ;

   iproc = $$i ;
   n0 = nxproc>>1 ; /* nxproc/2 */
   n1 = nxproc>>1 ; /* nxproc/2 */
   A1 = A[0] ;
   A2 = A[1] ;

   for(i=n1,j=1; j<n0; i<<=1,j<<=1){

      Z1 = .WorkSpace[ j].A1 ;
```

```
      Z2 = .WorkSpace[-j].A2 ;
      if ( (iproc&j)==0 ) A2=Z1 ;
      else A1=Z2 ;

      Z1 = .WorkSpace[ i].A1 ;
      Z2 = .WorkSpace[-i].A2 ;
      if ( (iproc&i)==0 ) A2=Z1 ;
      else A1=Z2 ;

      Z1 = .WorkSpace[ j].A1 ;
      Z2 = .WorkSpace[-j].A2 ;
      if ( (iproc&j)==0 ) A2=Z1 ;
      else A1=Z2 ;

   }

   Z1 = .WorkSpace[ i].A1 ;
   Z2 = .WorkSpace[-i].A2 ;
   if ( (iproc&i)==0 ) A2=Z1 ;
   else A1=Z2 ;
   A[0] = A1 ;
   A[1] = A2 ;

}


generic(WorkSpace[]) void FFT (
      complex A[2],
      complex *WS,
      int nxproc ) {
   complex W ;

   /* fft coefficients */
   W = *WS ;

   /* X dimension fft */
   partialFFT( A, &W, nxproc) ;
   /* X dimension unshuffle */
   UnShuffle ( A, nxproc ) ;
   /* Transpose data */
   Transpose ( A, nxproc ) ;

   /* fft coefficients */
   W = *WS ;

   /* X dimension fft */
   partialFFT( A, &W, nxproc) ;
   /* X dimension unshuffle */
   UnShuffle ( A, nxproc ) ;
   /* Transpose data */
   Transpose ( A, nxproc ) ;
}


generic(WorkSpace[]) void invFFT (
      complex A[2],
      complex *WS,
      int nxproc ) {
   complex W ; int Size ;

   /* fft coefficients */
   W.rr =  WS->rr ;
   /* complex conjugate */
   W.ii = -WS->ii ;

   /* X dimension fft */
   partialFFT( A, &W, nxproc ) ;
   /* X dimension unshuffle */
   UnShuffle ( A, nxproc ) ;
   /* Transpose data */
   Transpose ( A, nxproc ) ;

   /* fft coefficients */
   W.rr =  WS->rr ;
   /* complex conjugate */
   W.ii = -WS->ii ;

   /* X dimension fft */
   partialFFT( A, &W, nxproc ) ;
   /* X dimension unshuffle */
   UnShuffle ( A, nxproc ) ;
   /* Transpose data */
   Transpose ( A, nxproc ) ;

   /* normalize result */
   Size = (nxproc*nxproc) ;
   A[0].rr /= (Size) ;
   A[0].ii /= (Size) ;
   A[1].rr /= (Size) ;
   A[1].ii /= (Size) ;
}


threads WorkSet [nproc] ;
threads Frame   [size][size] ;

int main ( int argc, char **argv ) {

 WorkSet.{
   complex A[2],WS ;

   /* initialize test data set */

   Frame.{
     int KK, LL;
     real RIMAGE ;

     KK = $$ix[0]-(size>>1) ;
     LL = $$ix[1]-(size>>1) ;

     RIMAGE = 0;
     if (
     KK*KK+LL*LL < radius*radius
     ) RIMAGE = 1 ;
```

```
    WorkSet.{
      int i=$$i;
      A[0].rr =
      Frame [ i/(size>>1) ]
        [ i%(size>>1) ].RIMAGE ;
      A[0].ii = 0.0 ;
      A[1].rr =
      Frame [ i/(size>>1) ]
        [ (i%(size>>1))+(size>>1) ]
        .RIMAGE ;
      A[1].ii = 0.0 ;
    }
  }

  Frame.{
    real RIMAGE ;

    WorkSet.{ int i=$$i;
    Frame[ i/(size>>1) ]
      [ i%(size>>1) ].RIMAGE
      = sqrt( A[0].rr*A[0].rr
        + A[0].ii*A[0].ii ) ;
    Frame [ i/(size>>1) ]
      [ (i%(size>>1))+(size>>1) ]
      .RIMAGE
        = sqrt( A[1].rr*A[1].rr +
      A[1].ii*A[1].ii ) ;
  }

  printf ( "%c%10g",
    ( ($$ix[0])?' ':'\n' ),
    RIMAGE ) ;
  if (!$$i) printf ( "\n");
}



/* initialize fft coefficients */

WS = Init_FFT ( Frame.($$Nx[0]) );

/* perform FFT */

FFT ( A, &WS, Frame.($$Nx[0]) ) ;

Frame.{
  real RIMAGE ;

  WorkSet.{ int i=$$i;
  Frame[ i/(size>>1) ]
    [ i%(size>>1) ].RIMAGE
    = sqrt( A[0].rr*A[0].rr
      + A[0].ii*A[0].ii ) ;
  Frame[ i/(size>>1) ]
    [ (i%(size>>1))+(size>>1) ]
    .RIMAGE
    = sqrt( A[1].rr*A[1].rr
    + A[1].ii*A[1].ii ) ;
  }

  printf ( "%c%10g",
    ( ($$ix[0])?' ':'\n' ),
    RIMAGE );
  if (!$$i) printf ( "\n") ;
}



/* perform inverse FFT */

invFFT ( A, &WS, Frame.($$Nx[0]) ) ;

/*  clean up result */

if ( A[0].rr < 1e-13 ) A[0].rr=0.0 ;
if ( A[0].ii < 1e-13 ) A[0].ii=0.0 ;
if ( A[1].rr < 1e-13 ) A[1].rr=0.0 ;
if ( A[1].ii < 1e-13 ) A[1].ii=0.0 ;

Frame.{
  real RIMAGE ;

  WorkSet.{ int i=$$i;
    Frame[ i/(size>>1) ]
      [ i%(size>>1) ].RIMAGE
      = sqrt( A[0].rr*A[0].rr
       + A[0].ii*A[0].ii ) ;
    Frame[ i/(size>>1) ]
      [ (i%(size>>1))+(size>>1) ]
      .RIMAGE
      = sqrt( A[1].rr*A[1].rr
      + A[1].ii*A[1].ii ) ;
  }

  printf ( "%c%10g",
    ( ($$ix[0])?' ':'\n' ),
    RIMAGE ) ;
  if (!$$i) printf ("\n") ;
}

}

return 0;

}
```

NASA's Goddard Space Flight Center

# Route Sheet

| Code | Name | Pur-pose* | Initials** | Date** In | Date** Out | – |
|---|---|---|---|---|---|---|
| 935 | John Dorband | | | | | |
| | | | | | | |
| 935 | William Campbell | | | | | |
| | | | | | | |
| 930 | Richard Rood | | | | | |
| | | | | | | |
| 230 | Thomas A. Weisz | | | | | |
| | | | | | | |
| 935 | Nancy Dixon | | | | | |
| | | | | | | |
| 234.0 | Terri Randall | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

| ORIGINATOR | CODE | PHONE NUMBER |
|---|---|---|
| John Dorband | 935.0 | 6-9419 |
| **TYPED BY** | **CODE** | **PHONE NUMBER** |
| Nancy Dixon | 935.0 | 6-3898 |

QUALITY CHECK (secretary; please review and initial)

BR: X DIV: _____ DIR: _____

MAIL LOG

BR Log # _____ DIV Log # _____ DIR Log # _____

SUBJECT

Request for approval for publication release: Parallel signal processing and system simulation using aCe

DUE DATE

02/19/03

SPECIAL INSTRUCTIONS

* PURPOSE LEGEND
1. For information
2. For review/approval/concurrence
3. For necessary action
4. For signature

**INITIALS/DATE: Ensure that all initials and in/out dates are completed for all lines

GSFC 11-20 (9/99)