Towards an Efficient Functional Implementation of the NAS Benchmark FT

Clemens Grelck

Institute of Software Technology and Programming Languages

University of Lübeck, Germany

Sven-Bodo Scholz

Institute of Computer Science and Applied Mathematics

University of Kiel, Germany

NAS Benchmark FT

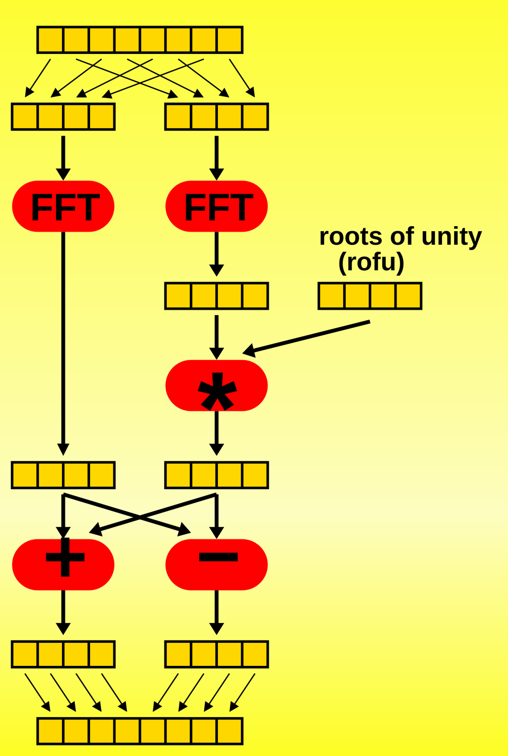
- Computational aerodynamic simulation kernel.
- Developed at NASA Ames Research Center.
- Solver for partial differential equations.
- 3-dimensional complex fast-Fourier transforms.

How would YOU prefer to implement NAS FT ?

SAC — Single Assignment C

Fortran

1-dimensional FFT



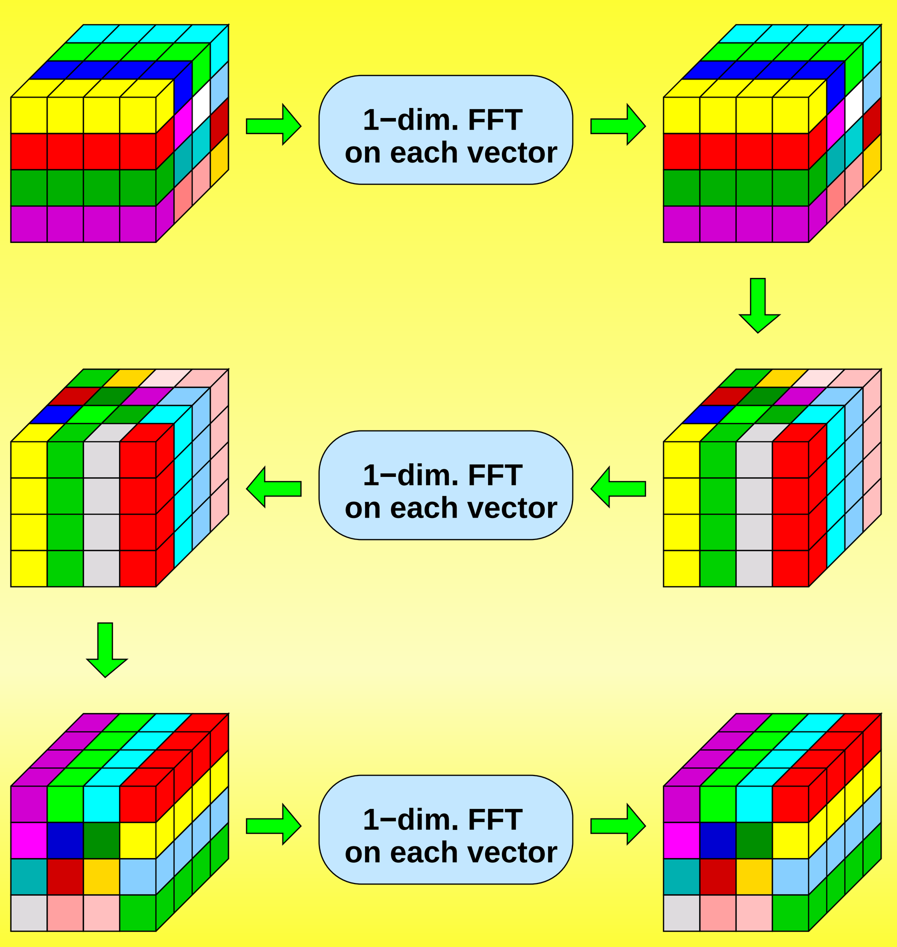

```
complex[.] FFT( complex[.] v, complex[.] rofu)
{
  even      = condense( 2, v);
  odd       = condense( 2, drop( [1], v));
  rofu_even = condense( 2, rofu);

  fft_even  = FFT( even, rofu_even);
  fft_odd   = FFT( odd, rofu_even);

  left      = fft_even + fft_odd * rofu;
  right     = fft_even - fft_odd * rofu;

  return( left ++ right);
}

complex[2] FFT( complex[2] v, complex[1] rofu)
{
  return( [ v[0] + v[1], v[0] - v[1] ]);
}
```

(Excerpts from serial NAS reference implementation provided by NASA)

```
subroutine cfftz( is, m, n, x, y)
    integer is,m,n,i,j,l,mx
    double complex x, y
    dimension x(fftblockpad,n)
    dimension y(fftblockpad,n)
    do l = 1, m, 2
     call fftz2( is, l, m, n, fftblock,
                 fftblockpad, u, x, y)
     if (l .eq. m) goto 160
     call fftz2( is, l+1, m, n, fftblock,
                 fftblockpad, u, y, x)
    enddo
    goto 180
160 do j = 1, n
     do i = 1, fftblock
      x(i,j) = y(i,j)
      enddo
    enddo
180 continue
    return
    end
```

```
subroutine fftz2( is, l, m, n, ny,
                  ny1, u, x, y)
integer is,k,l,m,n,ny,ny1,n1,li,lj,lk
integer ku,i,j,i11,i12,i21,i22
double complex u,x,y,u1,x11,x21
dimension u(n), x(ny1,n), y(ny1,n)
n1 = n / 2
lk = 2 ** (l - 1)
li = 2 ** (m - l)
lj = 2 * lk
ku = li + 1
do i = 0, li - 1
 i11 = i * lk + 1
 i12 = i11 + n1
 i21 = i * lj + 1
 i22 = i21 + lk
 if (is .ge. 1) then
  u1 = u(ku+i)
 else
  u1 = dconjg(u(ku+i))
 endif
 do k = 0, lk - 1
  do j = 1, ny
   x11 = x(j,i11+k)
   x21 = x(j,i12+k)
   y(j,i21+k) = x11 + x21
   y(j,i22+k) = u1 * (x11 - x21)
  enddo
 enddo
enddo
return
end
```

3-dimensional FFT




```
complex[.,.,.] FFT( complex[.,.,.] a,
                    complex[.] rofu)
{
  a_t = transpose( ([2,1,0]), a);
  b   = { [x,y] -> FFT( a_t[[x,y]], rofu) };

  b_t = transpose( ([0,2,1]), b);
  c   = { [x,y] -> FFT( b_t[[x,y]], rofu) };

  c_t = transpose( ([1,2,0]), c);
  d   = { [x,y] -> FFT( c_t[[x,y]], rofu) };

  return( d);
}
```

```
subroutine fft( x1, x2)
double complex x1(ntotal), x2(ntotal)
double complex scratch( fftblockpad_default * maxdim * 2)
call cffts1( 1, dims(1,1), x1, x1, scratch)
call cffts2( 1, dims(1,2), x1, x1, scratch)
call cffts3( 1, dims(1,3), x1, x2, scratch)
return
end
```

```
subroutine
 cffts1( is, d, x, xout, y)
integer is, d(3), logd(3)
double complex
 x(d(1),d(2),d(3))
double complex
 xout(d(1),d(2),d(3))
double complex
 y(fftblockpad, d(1), 2)
integer i, j, k, jj
do i = 1, 3
 logd(i) = ilog2(d(i))
end do
do k = 1, d(3)
 do jj = 0,
    d(2) - fftblock,
    fftblock
  do j = 1, fftblock
   do i = 1, d(1)
    y(j,i,1) = x(i,j+jj,k)
   enddo
  enddo
  call cfftz( is, logd(1),
    d(1), y, y(1,1,2))
  do j = 1, fftblock
   do i = 1, d(1)
    xout(i,j+jj,k)
      = y(j,i,1)
   enddo
  enddo
 enddo
enddo
return
end
```

```
subroutine
 cffts2( is, d, x, xout, y)
integer is, d(3), logd(3)
double complex
 x(d(1),d(2),d(3))
double complex
 xout(d(1),d(2),d(3))
double complex
 y(fftblockpad, d(2), 2)
integer i, j, k, ii
do i = 1, 3
 logd(i) = ilog2(d(i))
end do
do k = 1, d(3)
 do ii = 0,
    d(1) - fftblock,
    fftblock
  do j = 1, d(2)
   do i = 1, fftblock
    y(i,j,1) = x(i+ii,j,k)
   enddo
  enddo
  call cfftz( is, logd(2),
    d(2), y, y( 1, 1, 2))
  do j = 1, d(2)
   do i = 1, fftblock
    xout(i+ii,j,k)
      = y(i,j,1)
   enddo
  enddo
 enddo
enddo
return
end
```

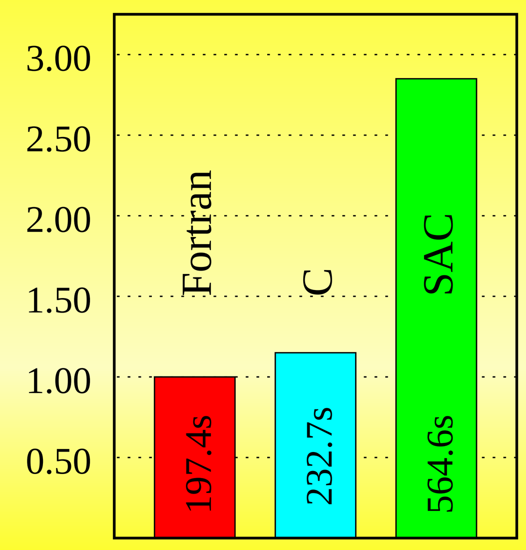
```
subroutine
 cffts3(is, d, x, xout, y)
integer is, d(3), logd(3)
double complex
 x(d(1),d(2),d(3))
double complex
 xout(d(1),d(2),d(3))
double complex
 y(fftblockpad, d(3), 2)
integer i, j, k, ii
do i = 1, 3
 logd(i) = ilog2(d(i))
end do
do j = 1, d(2)
 do ii = 0,
    d(1) - fftblock,
    fftblock
  do k = 1, d(3)
   do i = 1, fftblock
    y(i,k,1) = x(i+ii,j,k)
   enddo
  enddo
  call cfftz( is, logd(3),
    d(3), y, y( 1, 1, 2))
  do k = 1, d(3)
   do i = 1, fftblock
    xout(i+ii,j,k)
      = y(i,k,1)
   enddo
  enddo
 enddo
enddo
return
end
```

Facts about SAC

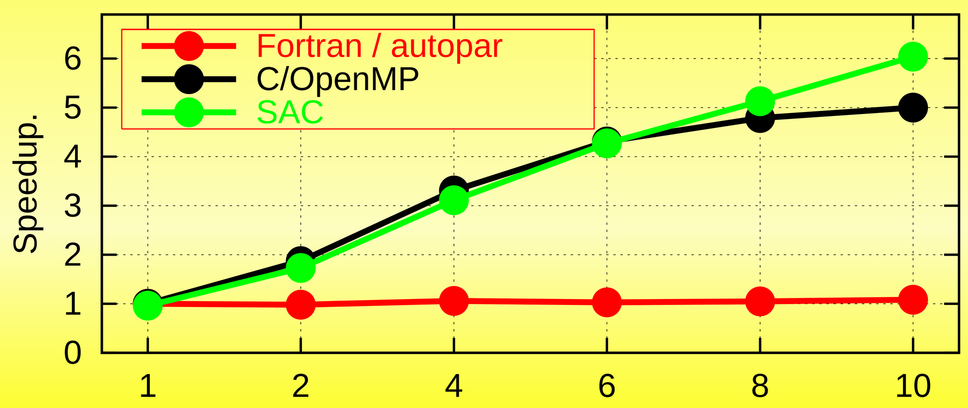
- Functional array programming language.
- Execution based on context-free substitution.
- Implicit memory management for arrays.
- High-level APL-like specifications with a C-like syntax.
- Highly optimizing compiler.
- Reasonable runtime performance.
- Implicit parallelization for shared memory multiprocessors.
- Find out more about SAC at
  `http://www.sac-home.org/`

Runtime Performance

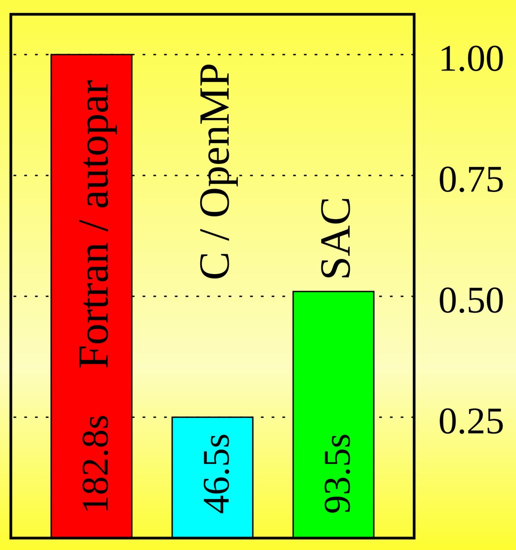
- 12-processor SUN Ultra Enterprise 4000
- C / OpenMP solution by RWCP, Japan
- SUN Workshop compilers



1 processor



Number of processors involved.



10 processors

Take Home Message

- **Debugging time-consuming?**
- **Development cycles too long?**
- **Code maintenance a nightmare?**
- **Programming gurus not available?**
- **SAC combines high-level array programming with reasonable runtime performance and implicit parallelization at your finger tips.**
- **Visit** SAC -home.org