Brooktran

Massimiliano Fatica, Alan Wray

Merrimac meeting
April 8 2003

Outline

- Motivation
- Brooktran syntax and examples
- Implementation in the Open64 compiler
- Future work

Motivation

Fortran

- Why do we need Fortran support?
  - Most scientific and high performance codes are in Fortran
  - National labs, NASA, aerospace companies have a huge investment in Fortran codes
  - The codes have been thoroughly tested and validated
  - They can be HUGE
  - Even if rewriting a code in a different language is not be a big deal, the validation process is
  - A code not fully validated can be acceptable in academia but not for real missions.

Porting codes to Merrimac

The path from C to Brook is much easier than the one from Fortran to Brook:

- C to Brook: similar to OpenMP parallelization in extent of changes
  - Start from original code and, one by one, "streamify" functions.
  - You can start working on the time consuming part of the code
  - Very easy to check the results since all the I/O & utility functions are working
- Fortran to Brook: more extensive changes than even MPI parallelization
  - The code has to be rewritten from scratch
  - A lot of time must be spent rewriting I/O and utility functions
  - Checking the results and debugging is very time consuming

Possible paths from Fortran to Brook (1)

- Mixed language programming: Fortran+Brook
  - Use the original structure of the Fortran code.
  - The subroutines that are floating-point intensive are replaced by Brook kernels
  - Streams are a view of memory, so we just need to pass the proper memory information to Brook
  - It requires some "glue" code and a standard Fortran compiler

Example of mixed language programming

```
// FORTRAN main
program sample
real, allocatable, dimension(:):: a,b,c
integer:: n
n=1000
allocate(a(n),b(n),c(n))
call brook_sum(a,b,c,n)
end program sample

// Brook function
void brook_sum(a,b,c,n)
float a[],b[],c[];
int *n;
{
 floats stream_a, stream_b, stream_c;
 streamLoad(stream_a,a,n);
 streamLoad(stream_b,b,n);
 add_array(stream_a,stream_b, stream_c);
streamStore(stream_c,c,n);
}

// Brook kernel
kernel add_array( floats a, floats b, floats)
{ c=a+b; }
```

Possible paths from Fortran to Brook (2)

- Brooktran: streaming language that uses Fortran syntax
- The setup of streams is done through library calls
- The kernels are written using a Fortran syntax and have the same constraints as Brook kernels

Example of Brooktran

```
// FORTRAN main
program sample
real, allocatable, dimension(:):: a,b,c
stream, real, dimension(:) :: stream_a, stream_b, stream_c
integer:: n
n=1000
allocate(a(n),b(n),c(n))
call streamLoad(stream_a, a, n)
call streamLoad(stream_b, b, n)
call add_array(stream_a, stream_b, stream_c)
call StreamStore(stream_c,c,n)
end program sample

// Brooktran kernel
kernel subroutine add_array(a, b, c)
stream, real, intent(in):: a, b
stream, real, intent(out):: c
c=a+b
end subroutine add_array
```

Brooktran

- A Fortran syntax of Brook will help porting legacy codes to Merrimac
- Open64 already has a Fortran95 front-end
- Fortran 9x array syntax makes stream code very compact

Plan

- Define Brooktran specifications
- Modify the Open64 Fortran parser to accept streaming syntax (e.g., stream, kernel) and do syntax checking
- Generate WHIRL and symbol table consistent with Brook
- Use the compiler infrastructure of Brook to produce SVM code

Brooktran syntax

New keywords

- We need to add the following keywords to Fortran:
- stream: used to define a stream; it is a native compound object much like an array
- kernel: used to specify a function or subroutine that can be executed by the streaming processor unit
- reduce: used for reduction arguments in kernels

Kernel

- Kernel functions or subroutines are declared by placing the "kernel" keyword before the function or subroutine name
- Arguments of the call have the same restrictions as in Brook.
- All arguments need to have explicit "intents"

```
kernel subroutine streamsum( a ,b, c, sum)
stream, real, intent (in):: a,b
stream, real, intent (out):: c
real, intent(reduce):: sum
    c=a+b
    sum=sum+c
end  subroutine streamsum
```

Stream

- Streams are a native compound object like arrays
- The shape is defined by the dimension given in the declaration

```
stream, type(real), dimension(:,:):: a
```

For streams that are generated from stencil of group operators, we can specify the "shape" of each element

```
stream, type(real), dimension(:,:):: b(3,3)
```

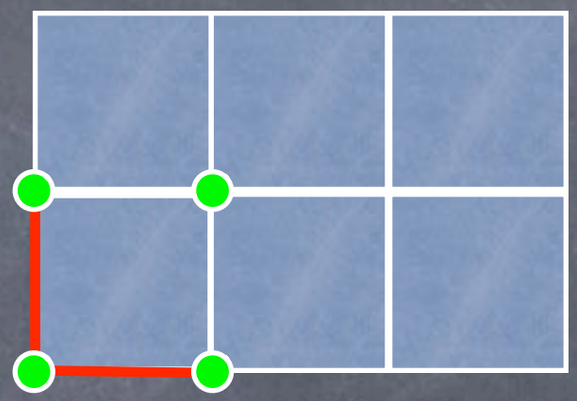
```
real, dimension(:,:) :: mesh
streamSource(a,array,2,100,100)
streamGroup(b,a,STREAM_STENCIL_HALO,2,-1,1,-1,1)
```

Example

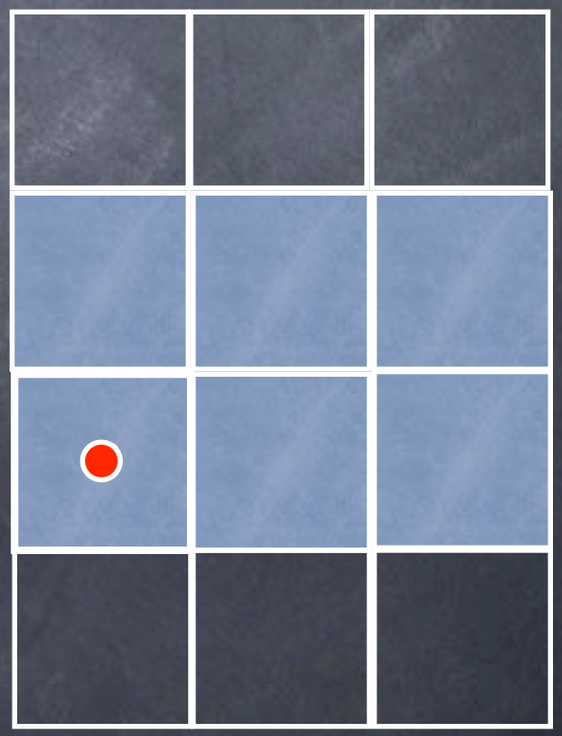
```
program compute_mesh
type gridcell
            real:: x
            real:: y
end type gridcell
type(gridcell), dimension(:,:),allocatable :: mesh
real, dimension(:,:),allocatable:: vol
stream, type(gridcell), dimension(:,:):: a, b(2,2)
stream, type(real), dimension(:,:):: c
nx=3; ny=2
allocate(mesh(nx+1,ny+1),vol(nx,0:ny+1))
.....
call streamSource(a,mesh,2,nx+1,ny+1)
call streamStencil(b,a,STREAM_STENCIL_HALO,2,0,1,0,1)
call ComputeMetric(b,c,volmin,volmax)
call streamSink(c,vol(1,1),nx,ny)

kernel subroutine ComputeMetric(grid,volume,volmin,volmax)
 stream, intent(in), type(gridcell):: b(2,2)
 stream, intent(out),type(real)::volume
real, intent(reduce):: volmin,volmax
 volume=.5*((grid(2,2).x-grid(1,1).x)*(grid(1,2).y-grid(2,1).y) &
          &   -(grid(2,2).y-grid(1,1).y)*(grid(1,2).x-grid(2,1).x));
volmin=min(volmin,volume)
vomax=max(volmax,volume)
end subroutine ComputeMetric
```

mesh



vol



Stream manipulation

- Stream load/store, domain, etc is done with function calls:

```
call streamSource(Y,X,100*200)
```

or

```
Y=streamSource(X,100*200)
```

- In Brooktran, streams have an associated shape. We should modify the load operator

```
call streamSource(Y,X,2,100,200)
```

- We can use the Fortran 9x array syntax:

```
call streamSource(Y,X(51:100,101:200),50*100)
```

Implementation in the Open64 compiler

Open64

- We are using ORC2.0:
  - The compiler has been installed
  - Generates IA64 binary
  - Executables can be run on IA32 using NUE

Whirl example

C

```
int main()
{ float a=1.f,b=2.f,c;
  c=a+b;
}
```

```
[brooktran@ctr-sgi12 code]$ ir_b2a test_c.B
 LOC 1 1 int main()
 LOC 1 2 { float a=1.f,b=2.f,c;
FUNC_ENTRY <1,20,main>
BODY
 BLOCK
 END_BLOCK
 BLOCK
 END_BLOCK
 BLOCK
 PRAGMA 0 120 <null-st> 0 (0x0) # PREAMBLE_END
  F4CONST <1,21,____1.000000>
 F4STID 0 <2,1,a> T<10,.predef_F4,4>
  F4CONST <1,22,____2.000000>
 F4STID 0 <2,2,b> T<10,.predef_F4,4>
 LOC 1 3    c=a+b;
   F4F4LDID 0 <2,1,a> T<10,.predef_F4,4>
   F4F4LDID 0 <2,2,b> T<10,.predef_F4,4>
  F4ADD
 F4STID 0 <2,3,c> T<10,.predef_F4,4>
 RETURN
 END_BLOCK
```

Fortran

```
program sum
real::a=1.,b=2.,c
c=a+b
end program sum
```

```
[brooktran@ctr-sgi12 code]$ ir_b2a test_add_f.B
 LOC 1 1 program sum
FUNC_ENTRY <1,20,MAIN__>
BODY
 BLOCK
 END_BLOCK
 BLOCK
 END_BLOCK
 BLOCK
 PRAGMA 0 120 <null-st> 0 (0x0) # PREAMBLE_END
 LOC 1 2 real::a=1.,b=2.,c
 LOC 1 3 c=a+b
   F4F4LDID 0 <2,1,A> T<10,.predef_F4,4>
   F4F4LDID 0 <2,2,B> T<10,.predef_F4,4>
  F4ADD
 F4STID 0 <2,3,C> T<10,.predef_F4,4>
 LOC 1 4 end program sum
 VCALL 2174 <1,22,_END> # flags 0x87e
 RETURN
 RETURN
 END_BLOCK
```

Status of the front-end

- A new source tree and output "phase" have been added to give a completely separate front-end for modification into Brooktran
- A new file extension, ".brt", and executable, orbrt, have been added, analogous to .f90 and orf90, or .c and orcc

Status of the front-end

- The "kernel" keyword has been implemented to
  1. lexically parse,
  2. generate symbol table entries reflecting the kernel nature of the entry point, and
  3. output error messages to impose the semantics;
  - 41 files changed, 120+ lines changed/added.
- “stream” is almost done, “reduce” will follow

Examples

```
program test
    call sub()
end program test

kernel subroutine sub()
    print *, 'Hello brt!'
end subroutine sub
```

```
LOC 1 1
 program test
FUNC_ENTRY <1,20,MAIN__>
BODY
 BLOCK
 END_BLOCK
 BLOCK
 END_BLOCK
 BLOCK
PRAGMA 0 120 <null-st> 0 (0x0) # PREAMBLE_END
LOC 1 2 call sub()
VCALL 2174 <1,22,sub_> # flags 0x87e
LOC 1 3 end program test
VCALL 2174 <1,23,_END> # flags 0x87e
RETURN
RETURN
END_BLOCK
```

```
LOC 1 4
LOC 1 5  kernel subroutine sub()
FUNC_ENTRY <1,22,sub_>
BODY
 BLOCK
 END_BLOCK
 BLOCK
 BLOCK
PRAGMA 0 120 <null-st> 0 (0x0) # PREAMBLE_END
LOC 1 6     print *,'Hello brt!'
COMMENT <2,1,print*,'Hello brt!'> #
COMMENT <2,2,START_IO> #
PRAGMA 0 173 <null-st> 0 (0x0) # START_STMT_CLUMP
IO_ITEM <1,NONE>
  IO_ITEM <10,NONE>
   I4INTCONST 3 (0x3)
  IO_ITEM <73,FIRST_LAST_FLAG>
   I4INTCONST 0 (0x0)
  IO_ITEM <70,END_EOR_EOF_FLAG>
   I4INTCONST 0 (0x0)
  IO_ITEM <76,ENCODE_DECODE_FLAG>
   U8LDA 0 <1,25,(10_bytes)_"Hello_brt!"> T<32,anon_ptr.,8>
   I8INTCONST 140763258159104 (0x800600000000)
   U4INTCONST 10 (0xa)
 IO_ITEM <96,CHAR> T<31,.ch_str.,1>
IO <22,FORMATTED_WRITE,cray> 2
PRAGMA 0 174 <null-st> 0 (0x0) # END_STMT_CLUMP
COMMENT <2,3,END_IO> #
LOC 1 7 end subroutine sub
RETURN
END_BLOCK
```

Error messages (1)

```
program test
    call sub()
end program test

 kernel subroutine sub()
   integer :: i,j
   common /a/ i
   i  =  37
   j  =  -14
   print *,'Hello brt!'
end subroutine sub
```

```
$ orbrt katest.brt
  i = 37
  ^
cbrt-1672 orbrt: ERROR SUB, File = katest.brt, Line = 8, Column = 4
  Assignment to variables in COMMON, such as "I", cannot occur inside KERNEL routines.

sgif90: SGI Pro64 Fortran 90 Version 2.0.0 (f14) Mon Apr  7, 2003  16:08:51
sgif90: 12 source lines
sgif90: 1 Error(s), 0 Warning(s), 0 Other message(s), 0 ANSI(s)
cbrt: "explain cbrt-message number" gives more information about each message
```

Error messages (2)

```
program test
call sub()
end program test

kernel subroutine sub()
  integer :: i
  save :: i
  print *,'Hello brt!'
end subroutine sub
```

```
$ orbrt kstest.brt

  save :: i
  ^
cbrt-1671 orbrt: ERROR SUB, File = kstest.brt, Line = 7, Column = 4
  The SAVE attribute cannot be given for data inside a KERNEL subprogram since KERNEL routines cannot retain static
data.

sgif90: SGI Pro64 Fortran 90 Version 2.0.0 (f14) Mon Apr  7, 2003  16:10:24
sgif90: 10 source lines
sgif90: 1 Error(s), 0 Warning(s), 0 Other message(s), 0 ANSI(s)
cbrt: "explain cbrt-message number" gives more information about each message
```

Future work

Memory model for multinode machine

- Brook is adopting the memory model of UPC
- Brooktran will adopt the memory model of Co-array Fortran (CAF) , the Fortran equivalent of UPC.
- There is already a CAF implementation in Open64 (John Mellor-Crummey, Rice University, http://www.pmodels.org)

- Refine the Brooktran syntax:
  - we need the latest Brook specs……
- Complete the parser:
  - we need to agree on a common WHIRL and symbol table
  - Fold in the CAF