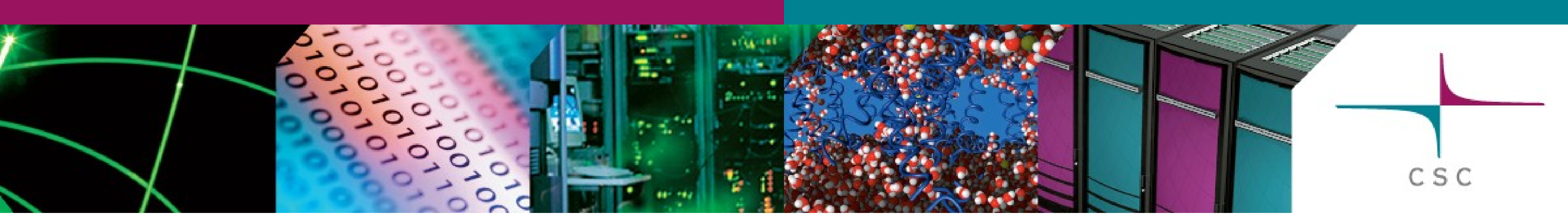


PGAS languages
& other parallel programming models

Sebastian von Alfthan

CSC – Tieteen tietotekniikan keskus Oy
CSC – IT Center for Science Ltd.

Introduction

CSC

- Parallel programming is necessary “evil” for utilizing HPC resources
- Large scale simulations on Top-10 supercomputers
  - Now 10-100 thousand cores
  - In the next few years 0.1-1 million cores
- Will the old programming models be enough?

Programming model

CSC

- Interface between the machine and the application
- Desirable features – the 5 P's
  - **P**roductivity – efficiency in writing good code
  - **P**erformance – fast and well scaling code
  - **P**arallelism – ability to express parallelism
  - **P**ortability – works on a range of machines
  - **P**rice – expensive licenses a hinder

Today…

CSC

- Parallel programming using sequential languages, e.g. C or Fortran, combined with directives or libraries for adding parallelism
- **Message passing** libraries for distributed memory
  - **MPI**, SHMEM, …
- **Threading** libraries & directives for shared memory
  - **OpenMP**, Pthreads, …
- **Accelerated regions** for accelerators (GPGPU's)
  - **Cuda**, OpenCL, HMPP, PGI

Parallel Languages

CSC

- PGAS (Partitioned Global Address Space)
  - Unified Parallel C (UPC)
  - Co-array Fortran (CaF), included in Fortran 2008
- HPCS (High Productivity Computing Systems DARPA program)
  - Chapel (Cray)
  - X10 (IBM)
  - Fortress (Sun)
- And more
  - Charm++ (Used in NAMD), Cilk, Haskel, etc…

PGAS

- Basic idea is to present the programmer with a shared memory space
- Data is associated with a certain processor but can be accessed by other processors
  - Local and remote variables
  - Arrays can be distributed or local
  - One-sided communication vs. two-sided in MPI
- Explicit synchronization needed

PGAS

- Works on distributed memory machines
- Hybrid PGAS+MPI possible!
  - Can utilize the strong features in both
  - One can introduce PGAS in a piece-wise fashion into an MPI program
- Main PGAS languages
  - Extended C: UPC
  - Extended Fortran: CAF

Support & performance

- Closed- & open-source implementations
  - CAF: Cray, G95 , CAF2
  - UPC: Cray, IBM XL, GCC UPC, HP, Berkley
- Performance varies...
  - Most use MPI to implement the parallelization
  - Some platforms have HW support - enables good performance!

http://www.g95.org
http://caf.rice.edu

http://upc.lbl.gov
http://www.gccupc.org

http://h30097.www3.hp.com/upc
http://www.alphaworks.ibm.com/tech/upccompiler

Support & performance on Cray

- Full support for UPC 1.2 & Fortran 2008 in Cray Compiler
- Cray XT: Implemented using MPI
  - Functional and with decent speed
- Cray XE: HW support
  - Gemini NIC can load and store remotely
  - Should be much faster, but performance still under NDA…

How to compiler & run on Cray XT (Louhi)

```
>module swap PrgEnv-pgi PrgEnv-cray
>module swap cce cce/7.2.7          # swap to latest version

>cc –h upc  [files]                 #to compile UPC programs
>ftn –h caf [files]                 #to compiler CAF programs

>aprun –n 16 ./prog                 #programs can be run in the
                                    #same way as MPI programs
```

CoArray Fortran

- Extended Fortran adding support for PGAS constructs
- In Fortran 2008 Co-Array Fortran has been added! [1]
- A scaled back version, some things pushed back to a technical report [2]
- Criticism of standard: CAF 2.0 [3]

[1] ftp://ftp.nag.co.uk/sc22wg5/N1801-N1850/N1824.pdf
[2] http://j3-fortran.org/doc/meeting/192/10-166.pdf
[3] http://caf.rice.edu/

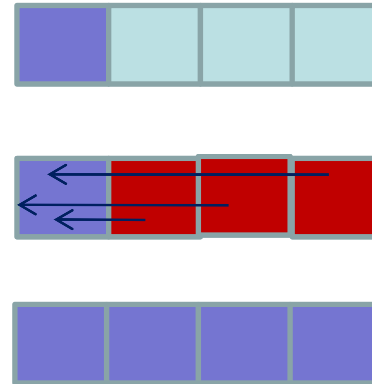
CAF example: images & coarrays

```
PROGRAM CAFtest
  IMPLICIT NONE
  INTEGER :: i  ! Local variable
  INTEGER :: j[*] ! scalar coarray
  INTEGER :: k(3)[*] ! vector coarray
  i=this_image()
  ! initialize on image 1 (images 1...N)
  IF (this_image() == 1) THEN
    j=456
    k(:)=(/ 7,8,9 /)
  END IF
  sync all ! Barrier for all
  !copy from image 1 (broadcast)
  j=j[1]
  !write out local i,j, remote k
  WRITE(*,*) i,j,k(:)[1]
END PROGRAM CAFtest
```

```
>ftn -h caf hello_caf.f90
>aprun -n 8 ./a.out  |sort -n
1,  456,  7,  8,  9
 2,  456,  7,  8,  9
 3,  456,  7,  8,  9
 4,  456,  7,  8,  9
 5,  456,  7,  8,  9
 6,  456,  7,  8,  9
 7,  456,  7,  8,  9
 8,  456,  7,  8,  9
```

CAF example: broadcast

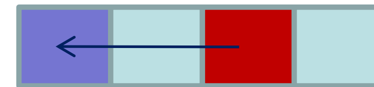
```
SUBROUTINE caf_bcast_simple(b)
  REAL,INTENT(inout)::b(:)[*]
  INTEGER ::size, caf_id
  b(:)=b(:)[1]
  sync all
 END SUBROUTINE caf_bcast_tree
```



- Simple “caf” way to do broadcast

CAF example: broadcast

```
SUBROUTINE caf_bcast_tree(b)
  REAL,INTENT(inout)::b(:)[*]
  INTEGER ::size, caf_id
  caf_id=this_image()
  size=num_images()/2
  DO WHILE (size>0)
    IF( MOD(caf_id-1,size*2) ==0 ) THEN
      sync images(caf_id+size)
    END IF
    IF( MOD(caf_id-1,size) ==0 .AND. \
        MOD(caf_id-1,2*size) .NE. 0) THEN
      sync images(caf_id-size)
      b(:)=b(:)[caf_id-size]
    END IF
    size=size/2
  END DO
  sync all
 END SUBROUTINE caf_bcast_tree
```



Size=2

Size=1

- Basic binomial tree algorithm
- Asumptions
  - Root image is 1
  - Number of images is a power of two

CAF example: broadcast

Performance of 512 KB broadcast on Cray XT

N-procs	MPI	b=b(:)[1]	Caf tree
256	4.9 ms	84 ms	5.4 ms
512	4.3 ms	146 ms	3.3 ms

- Easy one-line variant is slow
- More complex algorithm is faster, but is as complicated as a MPI version.

CAF benefits

- Potentially easier programmability
  - Easy syntax integrated in main language
  - One-sided communication supported
- Optimization potential for the compiler + runtime system
  - Optimizer can more easily analyze the parallel part
  - Can preload data from other node,arrange communication
- Coarrays of derived types allowed, these may contain pointers!

CAF benefits

CSC

- Communication easily visible through [] notation
  - Not true for OpenMP,UPC
  - Improved scalability compared to OpenMP due to better locality control
- Portable, both shared and distributed memory systems supported
- Can be combined with MPI!
  - Can be integrated into existing codes!

Some problems in Fortran 2008

- No support for image (=process) subsets
  - Teams not included (similar to MPI communicators)
  - All coarrays on all images…
- No collective operations
- No mechanism to tolerate latency when manipulating remote data
- Performance not great on current generation systems

Unified Parallel C (UPC)

- C extended with constructs for parallelism
  - Shared and local data & pointers
  - Synchronization – both blocking, nonblocking
  - Collective operations & iteration statements
- Versions
  - V 1.0 2001
  - V1.1.1 2003
  - V1.2 2005

UPC first example

```
#include <upc.h>
#include <stdio.h>
#include <stdlib.h>

void main(void){
printf("thread %d of %d threads\n",
     MYTHREAD,THREADS);

}
```

```
cc -h upc hello_upc.c
>aprun -n 8 ./a.out |sort –n
thread 0 of 8 threads
thread 1 of 8 threads
thread 2 of 8 threads
thread 3 of 8 threads
thread 4 of 8 threads
thread 5 of 8 threads
thread 6 of 8 threads
thread 7 of 8 threads
```

- Threads correspond to MPI processes, can reside on same node or another one

UPC example

```
#include <upc.h>
#include <stdio.h>
#include <stdlib.h>
#define SIZE 2*THREADS
void main(void){
    int i;
    static shared double a[SIZE],b[SIZE],c[SIZE];
    if(MYTHREAD==0)
        for(i=0;i<SIZE;i++)
            a[i]=b[i]=1.0;
    for(i=0;i<SIZE;i++)
        if(i%THREADS==MYTHREAD)
            c[i]=a[i]+b[i];
    if(MYTHREAD==0)
        for(i=0;i<SIZE;i++)
            printf("c[%d]=%g\n",i,c[i]);
}
```

```
>aprun -n 4 ./a.out
c[0]=2
c[1]=1
c[2]=0
c[3]=0
c[4]=2
c[5]=0
c[6]=0
c[7]=0
```

- Shared variables are cyclically distributed
- Have to be static

UPC example

```
#define SIZE 2*THREADS
void main(void){
    int i;
    static shared double a[SIZE],b[SIZE],c[SIZE];
    if(MYTHREAD==0)
        for(i=0;i<SIZE;i++)
            a[i]=b[i]=1.0;
    upc_barrier;
    for(i=0;i<SIZE;i++)
        if(i%THREADS==MYTHREAD)
            c[i]=a[i]+b[i];
    upc_barrier;
    if(MYTHREAD==0)
        for(i=0;i<SIZE;i++)
            printf("c[%d]=%g\n",i,c[i]);
}
```

```
>aprun -n 4 ./a.out
c[0]=2
c[1]=2
c[2]=2
c[3]=2
c[4]=2
c[5]=2
c[6]=2
c[7]=2
```

- Synchronization important…
- Also non-blocking split-phase barriers (notify & wait)

UPC data layout

CSC

- Cyclic data layout – default block size is 1
  - Block size can be changed, see below
  - Shared scalars are on thread 0
- When operating on data it is not immediately clear if data is local or remote, unlike CAF…

THREADS is 4

shared a[8]	0	1	2	3	0	1	2	3
shared [2] a[8]	0	0	1	1	2	2	3	3

UPC example

```
#define SIZE 2*THREADS
void main(void){
    int i;
    static shared double a[SIZE],b[SIZE],c[SIZE];
    if(MYTHREAD==0)
        for(i=0;i<SIZE;i++)
            a[i]=b[i]=1.0;
    upc_barrier;
    upc_forall(i=0;i<SIZE;i++;&a[i])
        c[i]=a[i]+b[i];
    upc_barrier;
    if(MYTHREAD==0)
        for(i=0;i<SIZE;i++)
            printf("c[%d]=%g\n",i,c[i]);
}
```

- Worksharing with **upc_forall(; ; ; affinity)**
- Pointer affinity: Loop body executed when pointer is on local thread
- Integer affinity: executed when MYTHREAD= affinity%THREADS

UPC pointers

- Shared pointer to shared data
  - int shared * shared a;
- ~~Shared pointer to private data~~
  - not allowed
- Private pointer to shared data
  - int shared * a;
- Private pointer to private data
  - int * a;

UPC further things

- Additional features not covered here
  - Collective operators
  - Shared data can of course also be multidimensional…
  - Dynamic allocation of shared data
  - And more…
- Not yet in standard
  - Parallel IO

Chapel

- Completely new programming language designed for parallel computing
- **Not for production use**, performance low & implementations not ready
- Supports
  - Task & Data parallelism
  - Object oriented & Generic programming
- Strong locality control through “locale” & “domain”

http://chapel.cray.com/papers.html

Conclusion

CSC

- PGAS languages intriguing but not quite here from a performance point of view
  - Situation might change soon – see Gemini
  - Can be used already in old programs in parallel with MPI!
- More advanced HPCS PGAS languages interesting but only as a research subject atm.