SHARCNET™

Frontiers of HPC: Unified Parallel C

David McCaughan, *HPC Analyst*
SHARCNET, University of Guelph
dbm@sharcnet.ca

Some of this material is derived from notes written by Behzad Salami (M.Sc. U. of Guelph)

SHARCNET™

Overview

- Parallel programming models are not in short supply
  - methodology dictated by underlying hardware organization
    - shared memory systems (SMP)
    - distributed memory systems (cluster)
  - trade-offs in ease of use; complexity
- Unified Parallel C:
  - an open standard for a uniform programming model
  - ***distributed shared memory***
  - fundamentals
  - a new set of trade-offs; is it worth it?

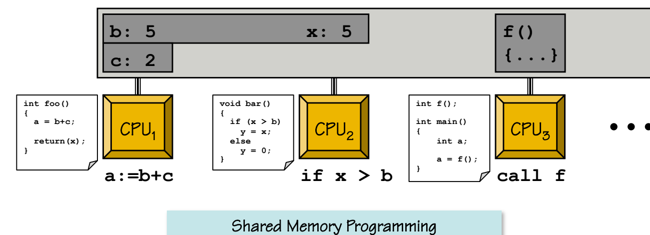
SHARCNET™

Thanks for the memory!

- Traditional parallel programming abstractly takes one of two forms, depending on how memory is able to be referenced by the program
- MIMD Models: Multiple Instruction Multiple Data
  - Shared memory
    - processors share a single physical memory
    - programs can share blocks of memory between them
    - issues: exclusive access, race conditions, synchronization, scalability
  - Distributed memory
    - unique memory associated with each processor
    - issues: communication is explicit, communication overhead

SHARCNET™

Shared Memory Model



SHARCNET™

Shared Memory Programming

- Shared memory programming benefits from the ability to handle communication implicitly
  - using the shared memory space
  - fundamentals of programming in SMP environments is relatively straightforward
    - issues typically revolve around exclusive access and race conditions
- Common SMP programming paradigms:
  - POSIX threads (pthreads)
  - OpenMP

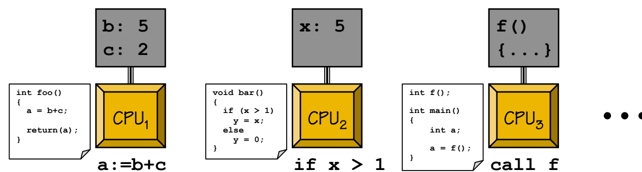
SHARCNET™

Hello, world! (pthreads)

```
#include <stdio.h>
#include "pthread.h"

void output (int *);

int main(int argc, char *argv[])
{
    int id;
    pthread_t thread[atoi(argv[1])];

    for (id = 0; id < atoi(argv[1]); id++)
        pthread_create(&thread[id], NULL, (void *)output, (void *)&id);

    return(0);
}

void output(int *thread_num)
{
    printf("Hello, world! from thread %d\n", *thread_num);
}
```

SHARCNET™

Distributed Memory Model



Distributed Memory Programming

HPC Resources

SHARCNET™

Distributed Memory Programming

- Communication is handled explicitly
  - processes send data to one another over an interconnection network
  - communication overhead limits granularity of parallelism
  - conforms to the strengths of traditional computing hardware so scalability can be excellent
- The past is littered with the corpses of distributed programming models
  - the modern standard: MPI

HPC Resources

Hello, World! (MPI)

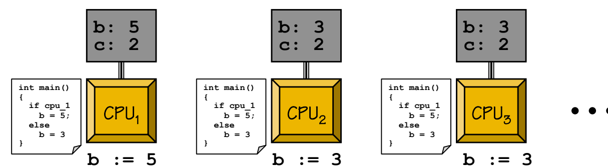
```
#include <stdio.h>
#include "mpi.h"

int main(int argc, char *argv[])
{
    int rank, size;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    printf("Process %d of %d\n", rank, size);

    MPI_Finalize();

    return(0);
}
```

SPMD

- *Single Program Multiple Data* (SPMD)
  - special case of MIMD model
  - many processors executing the *same* program
    - conditional branches used where specific behaviour required on the processors
  - shared/distributed memory organization
- MPI and UPC explicitly use this model

HPC Resources

SPMD Model



SPMD illustrating conditional branching to control divergent behaviour

HPC Resources

What is UPC?

- C language extensions for HPC programming on large scale parallel systems
  - attempts to provide a single, uniform programming model for SMP and cluster-based machines
  - superset of C (any C program is automatically a valid UPC program)
- Explicitly parallel SPMD model
  - the same program runs on all processors
  - Global Address Space (GAS) language; an attempt to balance
    - convenience (threads)
    - performance (MPI, data layout)

HPC Resources

SHARCNET™

What is UPC? (cont.)

- Single shared address space
  - variables can be accessed by any process, however are physically associated with one
    - hugely convenient from the programmer's perspective
    - what implications does this have?
  - most of UPCs complexity comes from the way it handles pointers and shared memory allocations
- Front-end/back-end organization allows for great flexibility in implementation
  - high speed interconnect, SMP, MPI, etc.
- Well suited to parallelizing existing serial applications

SHARCNET™

Hello, World! (UPC)

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

int main()
{
    printf("Hello, world! from UPC thread %d of %d\n",
            MYTHREAD, THREADS);

    return(0);
}
```

- Note: even for the simplest of examples, the implicit availability of THREADS and MYTHREADS variable reduces code volume dramatically (over pthreads or MPI)

SHARCNET™

Major Features: Basics

- Multi-processing abstractly modeled as *threads*
  - pre-defined variables `THREADS`, `MYTHREAD` available at run-time
- new keyword: `shared`
  - defines variables available across all processes
  - ***affinity*** (physical location of data) can also be specified
    - scalars (affinity to process 0)
    - cyclic (per element)
    - block-cyclic (user-defined block sizes)
    - blocked (run-time contiguous affinity for "even" distribution)
  - can make affinity inquiries at run-time

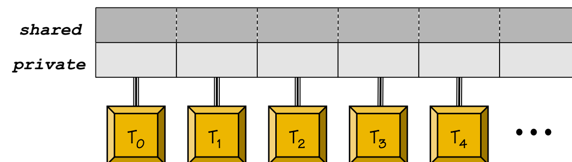
SHARCNET™

Major Features: Advanced

- Synchronization
  - locks (`upc_block_t`)
  - barrier (`upc_barrier, upc_notify, upc_wait`)
  - memory fence (`upc_fence`)
- User-controlled consistency models
  - Per-variable, per-statement
  - strict
    - all strict references within the same thread appear in program order *to all processes*
  - relaxed
    - all references within the same thread appear in program order *to the issuing process*

UPC and Memory

- Partitioned Global Address Space
  - private memory: local to a process
  - shared memory: partitioned over the address space of all processors
    - affinity refers to the *physical* location of data otherwise visible to all processes



HPC Resources

Variables

- Private
  - C-style declarations (i.e. the default)
  - e.g. `int foo;`
  - one instance per thread
- Shared
  - e.g. `shared int bar;`
  - one instance for all threads
  - shared array elements can be distributed across threads

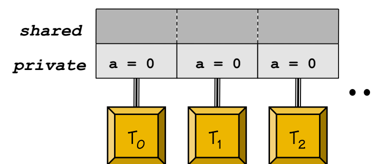
```
shared int foo[THREADS]
```
  - 1 element per thread

```
shared int bar[10][THREADS]
```
  - 10 elements per thread

HPC Resources

Example: private scalar

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

int main()
{
    int a;
    a = 0;

    a++;
    return(0);
}
```



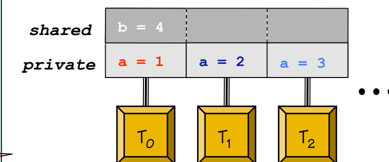
- NOTE: serial farming in a box; exactly like running multiple copies of the same program; no use of shared memory resources at all

HPC Resources

Example: shared scalar

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

int main()
{
    int a;
    static shared int b;

    a = b = 0;
    a++;
    b++;

    return(0);
}
```

caution: race conditions



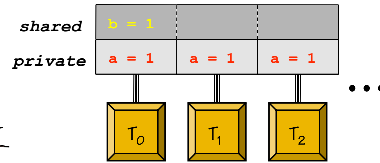
- NOTE: scalar affinity is to Thread_0; shared variables must be statically allocated (global data segment)

HPC Resources

SHARCNET™

Example: shared scalar

```
#include <stdio.h>
#include <upc_relaxed.h>

int main()
{
    int a;
    static shared int b;

    a = b = 0;

    a++;
    b++;

    return(0);
}
```



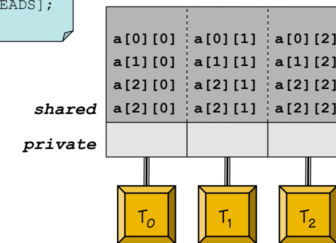
- NOTE: scalar affinity is to Thread_0; shared variables must be statically allocated (global data segment)

SHARCNET™

Example: shared arrays

```
…
    static shared int a[4][THREADS];
…
```

- assuming this is run with 3 threads, this memory will be organized as shown



SHARCNET™

Parallel Loop: `upc_forall`

- UPC defined loop (similar to the serial for loop)

```
upc_forall(init; condition; post; affinity)
```

  - affinity expression dictates which THREAD executes a given iteration of the loop
    - *pointer to shared type*: upc_threadof(affinity) == MYTHREAD
    - *integer*: affinity % THREADS == MYTHREAD
    - *`continue`*: MYTHREAD (all)

- `upc_forall` loops can be nested: careful

SHARCNET™

Exercise: To Affinity and Beyond

The purpose of this exercise is to allow you to explore your understanding of UPC shared declarations and memory affinity

Exercise

1) The `sharedarray.c` file in `~dbm/public/exercises/UPC` implements a working demonstration of a shared allocation of a 2-D array of integers and the output of its contents by processor affinity (using the `upc_forall` loop)

2) Ensure that you understand the issue of processor affinity by changing the initialization of the array so that it too occurs in parallel, in the thread with affinity to that section of the array

Pointers

- Declaration:

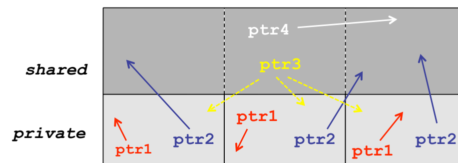
```
shared int *a;
```

  – a is a pointer to an integer that lives in the shared memory space

  – we refer to the type of **a** as a *pointer to shared*

A New Dimension for Pointers

```
int *ptr1;                    /* private pointer */
shared int *ptr2;             /* private pointer to shared */
int *shared ptr3              /* shared pointer to private */
shared int *shared ptr4;      /* shared pointer to shared */
```



Dynamic Memory Allocation

```
void *upc_all_alloc(int num, int size);
```

– allocates num * size bytes of data across all threads
– returns a pointer to the first element of the shared block of memory for all threads
– collective operations: *called by all threads*
– e.g. allocate 25 elements per thread

```
shared [5] int *ptr1;          /* note use of block size */
…
ptr1 = (shared int *)upc_all_alloc(25*THREADS, sizeof(int));
```

SHARCNET™

Dynamic Memory Allocation (cont.)

`void *upc_global_alloc(int num, int size);`

- allocates num * size bytes of data across all threads
- initializes shared pointers on all threads
- *only called by one thread*, but pointer will be initialized on all threads (i.e. allocation is run by one thread)
- e.g. allocate 25 elements per thread

```
shared [5] int *shared ptr2;  /* note use of block size */
…
ptr2 = (shared int *)upc_global_alloc(25*THREADS, sizeof(int));
```


SHARCNET™

Dynamic Memory Allocation (cont.)

`void *upc_alloc(int num, int size);`

- allocates num * size bytes of data in the shared memory space, only on the calling thread (i.e. affinity issue)
- returns a pointer to the first element of that shared block of memory for all threads
- e.g. allocate 25 elements on Thread_0, providing the reference to all threads

```
shared [5] int *shared ptr3;  /* note use of block size */
…
ptr3 = (shared int *)upc_alloc(25, sizeof(int));
```


SHARCNET™

Dynamic Memory Examples

```
shared [5] int *ptr1, *ptr2, *ptr3;
shared [5] int *ptr4, *ptr5;
…
/* 25 elements per threads, allocation in each thread */
ptr1 = (shared [5] int *)upc_all_alloc(25, sizeof(int));

/* 25 elements per thread, run allocation on thread 5 */
if (MYTHREAD == 5)
    ptr2 = (shared [5] int *)upc_global_alloc(25*THREADS),sizeof(int));

/* 25 elements only on thread 3 */
if (MYTHREAD == 3)
    ptr3 = (shared [5] int *)upc_alloc(25),sizeof(int));

/* 25 elements per thread, run allocation on thread 4, visible to all */
if (MYTHREAD == 4)
    ptr4 = (shared [5] int *)upc_global_alloc(25*THREADS),sizeof(int));

/* 25 elements only on thread 2, visible to all */
if (MYTHREAD == 2)
    ptr5 = (shared [5] int *)upc_alloc(25),sizeof(int));
```

SHARCNET™

Synchronization

`upc_barrier;`

- all threads will wait until all threads execute the barrier
- very similar to MPI_Barrier call (for those familiar)
- e.g.

```
if (MYTHREAD == 0)
     do_initialization();

upc_barrier;
/* computation */
upc_barrier;

if (MYTHREAD == 0)
    report_results();
```

SHARCNET™

Example/Exercise

- Let's consider a simple problem:
  - matrix multiplication
- UPC refinement from serial code:
  - code it sequentially
  - data parallel: share/block data using `THREADS`
  - task parallel: organize parallel work by processor affinity using `upc_forall`, `THREADS` and `MYTHREAD`
  - optimizing it by playing with memory affinity options

SHARCNET™

Exercise: *The Matrix: Reparallelized*

The purpose of this exercise is to illustrate the parallelization of a simple problem using UPC

SHARCNET™

Exercise

1) The `serialmatrix.c` file in `~dbm/public/exercises/UPC` implements the classic matrix multiplication algorithm

2) Parallelize this code using UPC by following the refinement steps outlined previously:
   - redeclare the 2D array to distribute it among threads
   - parallelize the work using affinity of data and upc_forall
   - consider alternate memory affinity to optimize performance

SHARCNET™

Summary

- Pro…
  - no need for explicit communication (on one hand: simpler code)
  - unified SMP/cluster model
  - compiler "optimized" communication
    - ensure optimal use of transport layer
    - *relaxed* consistency allows source reordering not possible in MPI
  - promise of scalability across paradigms
  - amenable to parallelizing existing serial code (for some domains)
- Con...
  - no need for explicit communication (on the other: hidden complexity)
  - introduces <u>new</u> complexity
    - distributed memory allocation
    - UPC advantages subverted by other language features (upc_fence)
  - language translation occurs at compile time limiting flexibility
  - on some level: all the baggage of threads (race conditions, etc.) **and** MPI (collective communication, etc.)

SHARCNET™

Food for Thought

- Small study at U. of Guelph (B. Salami, 2005):
  - attempt to study "ease of programming" when using MPI/UPC, by those new to the programming model
  - not large enough to draw statistical conclusions, but some interesting observations
  - experienced undergraduate CS students:
    - MPI-trained group were challenged by synchronizing send/recv calls, and issues of task parallelism
    - UPC-trained group were challenged by the pointer models and distributed memory allocation process; a more fundamental issue
    - MPI-trained group got further with solutions
  - interesting implications for development in a research setting

SHARCNET™

Discussion

- We haven't even seriously addressed issues of efficiency here
  - what are your expectations?
- Distributed shared memory is a non-trivial problem
  - UPC is close to state of the art in terms of exposing a usable programming framework for DSM
    - significant complexity remains, some of it hidden from the programmer which can cause serious performance issues
- Trading complexity
  - simple examples look promising; real code requires significant effort to do right; trade-offs in what requires the programmer's attention
- Is UPC ready for prime-time?

HPC Resources

SHARCNET™

Further Reading

- Berkeley Unified Parallel C:

  `http://upc.lbl.gov/docs/`

  - examples
  - reference documentation
  - software (free)

HPC Resources