Concepts of Parallel Programming Languages

Christoph Kessler, IDA

Parallel Language Concepts

- **Parallel control flow**
  - Fork-join style parallelism, SPMD style parallelism
  - Nested parallelism
  - Parallel loops, Sections
  - Parallel loop scheduling
  - Implicit parallelism
- **Synchronization & Consistency**
  - Futures
  - Supersteps and Barriers
  - Array assignments
  - Fence / Flush
  - Semaphores & Monitors
  - Atomic operations
  - Transactions

- **Address space**
  - Global Address Space, Sharing
  - Pointer models
  - Tuple space
- **Data locality & mapping control**
  - Co-Arrays
  - Virtual topologies
  - Alignment, distribution, mapping
  - Data distributions
  - Data redistribution
- **Communication**
  - Collective communication
  - One-sided communication
  - (see earlier lecture on MPI)

Some parallel programming languages (partly) covered here:

- Fork (see earlier lecture)
- Cilk (see earlier lecture)
- MPI (see earlier lecture)
- OpenMP
- HPF
- UPC / Titanium
- NestStep
- ZPL
- ...

Relationship between parallel and sequential programming languages

- Big issue: Legacy code in Fortran, C, (C++)
  - Practically successful parallel languages must be interoperable with, and, even better, syntactically similar to one of these
  - Compliance with sequential version is useful
    - e.g. C elision of a Cilk program is a valid C program doing the same computation
    - OpenMP
- Incremental parallelization supported by directive-based languages
  - e.g. OpenMP, HPF

Parallel Control Flow

\[
\begin{align*}
\text{sequential} & \quad !\text{omp parallel} \\
\text{parallel} & \quad !\text{omp end parallel} \\
\text{sequential}
\end{align*}
\]
Nested Parallelism

Parallel Loop Constructs

- Parallel Loop Scheduling (1)
  - Static scheduling
    - Chunk scheduling

- Parallel Loop Scheduling (2)
  - Dynamic Loop Scheduling
    - Chunk Scheduling

- Parallel Loop Scheduling (3)
  - Guided Self-Scheduling
    - Chunk scheduling
Parallel Loop Scheduling (4)

- Affinity-based Scheduling
  - Dynamic scheduling, but use locality of access together with load balancing as scheduling criterion
  - "cache affinity"

- Example: UPC forall loop
  - shared float x[100], y[100], z[100];
  - upc_forall (i=0; i<100; i++; &x[i])
    - x[i] = y[i] + z[i];
  - Iteration i with assignment x[i] = y[i] + z[i] will be performed by the thread storing x[i], typically (i % THREADS)

Futures

- A future cell by a thread T1 starts a new thread T2 to calculate one or more values and allocates a future cell for each of them.
- T1 is passed a read-reference to each future cell and continues immediately.
- T2 is passed a write-reference to each future cell
- Such references can be passed on to other threads
- As (T2) computes results, it writes them to their future cells.
- When any thread touches a future cell via a read-reference, the read stalls until the value has been written.
- A future cell is written only once but can be read many times.
- Used e.g. in Tera-C [Callahan/Smith’90], ML+futures [Bielloch/Reid-Miller’97], StackThreads/MP [Taura et al.’99]

Fence / Flush

- Producer thread
  - data = ...
  - flag = 1;
  - while (flag==0)...
  - data = ...

- Consumer thread
  - do
  - while (flag==0)...
  - data = ...

Supersteps

- BSP model: Program executed in series of supersteps
- Nestable supersteps
  - PUB library [Bonorden et al.’99], NestStep [K.’99]

Atomic Operations

- Atomic operations on a single memory word
  - SBPRAM / Fork mpadd etc.
  - OpenMP atomic directive for simple updates (x++, x--)
  - test&set, fetch&add, cmp&swap, atomicswap...
Atomic Transactions

- For atomic computations on multiple shared memory words
- Abstracts from locking and mutual exclusion
- coarse-grained locking does not scale
- declarative rather than hardcoded atomicity
- enables lock-free concurrent data structures
- Transaction either commits or fails
- Variant 1: atomic {...} marks transaction
- Variant 2: special transactional instructions
  e.g. LT, LTX, ST; COMMIT; ABORT
- Speculate on atomicity of non-atomic execution
- Software transactional memory
- Hardware TM, implemented e.g. as extension of cache coherence protocols [Herlihy,Moss’93]

Example: Thread-safe composite operation

- Move a value from one concurrent hash map to another
- Threads see each key occur in exactly one hash map at a time

void move (Object key) {
    synchronized (mutex) {
        map2.put (key, map1.remove(key));
    }
}

Any 2 threads can work in parallel as long as different hash table buckets are accessed.

Software Transactional Memory

User code:

```java
int foo (int arg) {
    ... atomic
    b = a + 5;
    ...}
```

Compiled code:

```java
int foo (int arg) {
    jmpbuf env;
    ... do
    if (setjmp(&env) == 0) {
        stmStart();
        temp = stmRead(&a);
        temp1 = temp + 5;
        stmWrite(&b, temp1);
        stmCommit();
        break;
    }
    while (1);
    ...
```

Instrumented with calls to STM library functions. In case of abort, control returns to checkpointed context by a longjmp().

Transactional Memory

- Good introduction:

- More references:
  See course homepage – list of papers for presentation
Tuple space

- Linda [Carriero, Gelernter 1988]
- Tuple space
  - Associative memory storing data records
  - Physically distributed, logically shared
  - Atomic access to single entries: put, get, read, ...
  - Query entries by pattern matching
    get( "task", &task_id, args, &argc, &producer_id, 2 );
- Can be used to coordinate processes
  - E.g., task pool for dynamic scheduling
  - E.g., producer-consumer interaction

Co-Arrays

- Co-Array Fortran [Numrich / Raid '98]
- Co-Arrays
  - Distributed shared arrays with a co-array dimension
    spanning the processors in a SPMD environment
  - arr[j][k] — addresses processor k’s copy of arr[j]
  - x(i) = y(i)[q]

Virtual topologies

Example: arrange 12 processors in 3x4 grid:

```c
int dims[2], coo[2], period[2], src, dest;
period[0]=period[1]=0;  // 0=grid, !0=torus
reorder=0; // 0=use ranks in communicator,
            // !0=MP uses hardware topology
dims[0]= 3; // extents of a virtual
            // 3X4 processor grid
// create virtual 2D grid topology:
MPI_Cart_create( comm, 2, dims, period, reorder, &comm2 );
// get my coordinates in 2D grid:
MPI_Cart_coords( comm2, myrank, 2, coo );
// get rank of my grid neighbor in dim. 0
MPI_Cart_shift( comm2, 0, +1, &src, &dest );
```

Co-Array Fortran Example

```fortran
subroutine laplace ( nrow, ncol, u )
  integer, intent( in ) :: nrow, ncol
  real, intent( inout ) :: u(nrow) [*]
  real :: new_u(nrow)
  integer :: i, me, left, right
  new_u(1) = u(nrow) + u(2)
  new_u(nrow) = u(1) + u(nrow-1)
  new_u(2:nrow-1) = u(1:nrow-2) + u(3:nrow)
  me = this_image(u)   ! Returns the co-subscript within u
                           ! that refers to the current image
  left = me-1;        ! Returns the co-subscript within u
  right = me + i;     ! that refers to the current image
  if (me == i) left = ncol
  if (me == ncol) right = 1
  call sync_all( (/left,right/) )  ! Wait if left and right have not already reached here
  new_u(1:nrow) = new_u(1:nrow) + u(1:nrow) [left] + u(1:nrow) [right]
  call sync_all( (/left,right/) )
  u(1:nrow) = new_u(1:nrow) - 4.0 * u(1:nrow)
end subroutine laplace
```

Virtual topologies

Example: arrange 12 processors in 3x4 grid:

```c
int dims[2], coo[2], period[2], src, dest;
period[0]=period[1]=0;  // 0=grid, !0=torus
reorder=0; // 0=use ranks in communicator,
            // !0=MP uses hardware topology
 dims[0]= 3; // extents of a virtual
            // 3X4 processor grid
 // create virtual 2D grid topology:
 MPI_Cart_create( comm, 2, dims, period, reorder, &comm2 );
 // get my coordinates in 2D grid:
 MPI_Cart_coords( comm2, myrank, 2, coo );
 // get rank of my grid neighbor in dim. 0
 MPI_Cart_shift( comm2, 0, +1, &src, &dest );
```

HPF Mapping Control: Alignment, Distribution, Virtual Processor Topology

<table>
<thead>
<tr>
<th>Fortran arrays</th>
<th>Template / Arrays</th>
<th>Abstract HPF processors</th>
<th>Physical processors</th>
</tr>
</thead>
<tbody>
<tr>
<td>!HPF$ ALIGN</td>
<td>!HPF$ TEMPLATE</td>
<td>!HPF$ PROCESSORS</td>
<td></td>
</tr>
<tr>
<td>!HPF$ DISTRIBUTE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Data Distribution (1)**

1. **Processors**: P(4)
2. **Real Dimension**: 23 :: A
3. **Distribute**: (Block) onto P :: A
   - A
   - A
   - A
   - A

**Data Distribution (2)**

1. **Processors**: P(4)
2. **Real Dimension**: 8 x 8 :: A
3. **Distribute**: (Block) onto P :: A
   - A
   - A
   - A
   - A

**Data Distribution (3)**

1. **Processors**: P(2, 2)
2. **Real Dimension**: 8 x 8 :: A
3. **Distribute**: (Block, Block) onto P :: A
   - A
   - A

**Communication**

- Single cast (P0)
- Reduction (+)
- Prefix (+)
- Scatter (P0)
- Gather (P0)
- Cyclic shift (+)
- Multibroadcast

**Encapsulation of communication context**

- Example: MPI Communicator
- Needed for parallel components