CS 267 Applications of Parallel Computers

Lecture 8: Programming Model Wrap-up
Work Assignment

2/14/97

David E. Culler

http://www.cs.berkeley.edu/cs267/
Outline

° Recap
° Complete our Fish Walkthrough
° Mechanisms for Assigning Work
° Locks
° Blocking and Higher Dimensions
Solaris Threads

° http:\\now.cs\~cs267\fish\threads.c
Split-C

° http:\\now.cs\~cs267\fish\split_c.sc
MPI

° http:\~now.cs\cs267\fish\mpi.c
Global / Local View

° Don’t casually flip between global and local view.
° Maintain a clear notion of global view and local view
° Switch between them at clear boundaries
° http:\now.cs\~cs267\fish\local_split_c.sc
Work Distribution

- The Split-C, MPI, and Threads version all use static assignment of the iteration space
  - without global data structures, must convert index
- The HPF versions “suggests” the same to the compiler
- Code must contain work assignment whether or not describes data assignment.

```c
for (l = MyMin; l <= MyMax; l++) {
    A[l] = f(l);
}
for (l = 0; l < MyCount; l++) {
    A[l] = f(l + MyMin);
}
```
What about cyclic assignment

° Assignment of work is easier in a global address space
° It is faster if it corresponds to the data placement!
° Hardware replication moves data to where it is accessed
Static Assignment on Irregular Data Structure
An Irregular Problem: EM3D

Maxwell's Equations on an Unstructured 3D Mesh

Irregular Bipartite Graph of varying degree (about 20) with weighted edges

Basic operation is to subtract weighted sum of neighboring values

for all E nodes
for all H nodes
typedef struct node_t {
    double value;
    int edge_count;
    double *coeffs;
    double **values;
    struct node_t *next;
} node_t;

void all_compute_E()
{
    node_t *n;
    int i;
    for (n = e_nodes; n; n = n->next) {
        for (i = 0; i < n->edge_count; i++)
            n->value = n->value - *(n->values[i]) * (n->coeffs[i]);
    }
}

---

How would you optimize this for a uniprocessor?
- minimize cache misses by organizing list such that neighboring nodes are visited in order
EM3D: Simple Parallel Version

Each processor has list of local nodes

typedef struct node_t {
    double value;
    int edge_count;
    double *coeffs;
    double *global (*values);
    struct node_t *next;
} node_t;

void all_compute_e()
{
    node_t *n;
    int i;
    for (n = e_nodes; n; n = n->next) {
        for (i = 0; i < n->edge_count; i++)
            n->value = n->value - *(n->values[i]) * (n->coeffs[i]);
    }
    barrier();
}

How do you optimize this?
  – Reorder the graph to minimize the number of remote global references
  – Balance the load across processors:
    \[ C(p) = a \cdot \text{Nodes} + b \cdot \text{Edges} + c \cdot \text{Remotes} \]

This is now an array of lists (per proc)
Optimizing within the Global/Local Space
void all_compute_e()
{
    ghost_node_t *g;
    node_t *n;
    int i;
    for (g = h_ghost_nodes; g; g = g->next)  g->value = *(g->rval);
    for (n = e_nodes; n; n = n->next) {
        for (i = 0; i < n->edge_count; i++)
            n->value = n->value - *(n->values[i]) * (n->coeffs[i]);
    }
    barrier();
}
void all_compute_e()
{
    ghost_node_t *g;
    node_t *n;
    int i;
    for (g = h_ghost_nodes; g; g = g->next) g->value := *(g->rval);
    sync();
    for (n = e_nodes; n; n = n->next) {
        for (i = 0; i < n->edge_count; i++)
            n->value = n->value - *(n->values[i]) * (n->coeffs[i]);
    }
    barrier();
}

Should the compiler figure it out?
void all_compute_e()
{
    node_t *n;
    int i;
    for (i = 0; i < boundary_nodes; i++) *gptrs[i] :- *vals[i];
    all_store_sync();
    for (n = e_nodes; n; n = n->next) {
        for (i = 0; i < edge_count; i++)
            n->value = n->value - *(n->values[i]) * (n->coeffs[i]);
    }
    barrier();
}

Generalize Dataparallel (bulk synchronous) operation
void all_compute_e()
{
    node_t *n;
    int i;
    for (i = 0; i < boundary_nodes; i++) *gptrs[i] :- *vals[i];
    store_sync(rcv_count);
    for (n = e_nodes; n; n = n->next) {
        for (i = 0; i < edge_count; i++)
            n->value = n->value - *(n->values[i]) * (n->coeffs[i]);
    }
    barrier();
}

Further Optimization: Use bulk_store() to send all ghost values to a processor at once.

Now you can see how to write the MPI code
Dynamic Work Assignment

° What if the time per fish was highly non-uniform?
self scheduling

while (fetch&inc (l) < n) {
    A[l] = f(l);
}

- Impact on load balancing?
- Impact on data access?
Global View is Critical

```c
void all_init_fish(int num_fish, fish_t fishes[])
{
    int i, n;
    double total_fish = NFISH;
    fish_t *fish;
    while fetch_and_inc (i) < NFISH {
        fish = &fishes[i];
        fish->x_pos = i*2.0/total_fish - 1.0;
        fish->y_pos = 0.0;
        fish->x_vel = 0.0;
        fish->y_vel = fish->x_pos;
        fish->mass = 1.0 + i/total_fish;
    }
}
```
Guided Self-scheduling

- **Grab decreasing size chunks**
  - reduce load on critical section
  - reduce parallelism overhead
  - get everyone to finish at the same time

```plaintext
chunk = initial_chunk(n)
while (fetch&add (ii, chunk) < n) {
    for (i=ii; i<min(ii+chunk,n); i++) {
        A[i] = f(i);
    }
}
```
Generalization: Task Queues

° **Grab arbitrary descriptors of work**
  - ex: segments a line in which to search for zero crossings or eigenvalues

```
while (task = dequeue_work(shared_queue)) {
    f(task) ;
}
barrier();
```

° **More general form: task generate more tasks**

```
while (task = dequeue_work(shared_queue)) {
    work on the task;
    enqueue(shared_queue, new_task);
}
... termination detation;
```
Flags, Spin Locks, etc.

° Why is this example a bad idea?

```c
if ( myID == thread_ptr[0].tid ) {
    g_dmax.accum = 0.;
    g_dmax.zeroed = 1;
}
while ( !g_dmax.zeroed ) ;

lock(&mul_lock);

lock(&mul_lock);
g_dmax.accum = MAX(g_dmax.accum, dmax);
unlock(&mul_lock);
```
Spining on Test

```c
void spinlock (int *x)
{
    repeat {
        compare_and_swap(0, x, MY_ID);
        until (*x == MY_ID);
    }
    void unlock (int *x)
    {
        *x = 0;
    }
```
Test and Test and set

° Spin in your cache and then go for it

void spinlock (int *x)
{
    repeat {
        while (*x) {};
        compare_and_swap(0, x, MY_ID);
        until (*x == MY_ID);
    }
    void unlock (int *x)
    {
        *x = 0;
    }

What about the crowd when the gates open?
Load_locked and store_conditional

repeat {
  x = load_locked(g_dmax.accum);
  y = max(x, dmax);
}
until (store_conditional(g_dmax.accu, y);

° Lock location (or more) upon load

° Store is aborted if location (or region) modified since load_locked

° Returns a status
Highly Contended Locks over Small Critical Section

Locks with null body (0.027 usec) on

![Graph showing time vs. processor for various lock implementations](image)
Larger Critical Section

Locks with 3.495usec Critical Section on
Barriers

Barrier performance on Cha

Time (usec)

Central
Tree
Tournament
ArrivalTree
Dissemination
SGI

Processors

1 2 3 4 5 6 7 8