CS4961 Parallel Programming

Lecture 9: Red/Blue and Introduction to Data Locality

Mary Hall
September 21, 2010

Programming Assignment 1 Due Wednesday, Sept. 21 at 11:59PM

- Logistics:
  - You'll use water.eng.utah.edu (a Sun Ultrasparc T2), for which all of you have accounts that match the userid and password of your CADE Linux account.
  - Compile using "cc –O3 –xopenmp p01.c"
- Write the prefix sum computation from HW1 in OpenMP using the test harness found on the website.
  - What is the parallel speedup of your code as reported by the test harness?
  - If your code does not speed up, you will need to adjust the parallelism granularity, the amount of work each processor does between synchronization points. You can adjust this by changing numbers of threads, and frequency of synchronization. You may also want to think about reducing the parallelism overhead, as the solutions we have discussed introduce a lot of overhead.
  - What happens when you try different numbers of threads or different schedules?

Programming Assignment 1, cont.

- What to turn in:
  - Your source code so we can see your solution
  - A README file that describes at least three variations on the implementation or parameters and the performance impact of those variations.
  - handin "cs4961 p1 <gzipped tarfile>"
- Lab hours:
  - Thursday afternoon and Tuesday afternoon

Red/Blue Computation from text

#10 in text on p. 111:

The Red/Blue computation simulates two interactive flows. An n x n board is initialized so cells have one of three colors: red, white, and blue, where white is empty, red moves right, and blue moves down. Colors wrap around on the opposite side when reaching the edge.

In the first half step of an iteration, any red color can move right one cell if the cell to the right is unoccupied (white). On the second half step, any blue color can move down one cell if the cell below it is unoccupied. The case where red vacates a cell (first half) and blue moves into it (second half) is okay.

Viewing the board as overlaid with t x t tiles (where t divides n evenly), the computation terminates if any tile's colored squares are more than c% one color. Use Peril-L to write a solution to the Red/Blue computation.

Steps for Solution

Step 1. Partition global grid for n/t x n/t processors

Step 2. Initialize half iteration (red) data structure

Step 3. Iterate within each t x t tile until convergence (guaranteed?)

Step 4. Initialize data structure -- Compute new positions of red elts & associated white elts

Step 5. Communicate red boundary values

Step 6. Compute new positions of blue elts & associated white elts

Step 7. Communicate blue boundary values

Step 8. Check locally if DONE

Issues to consider

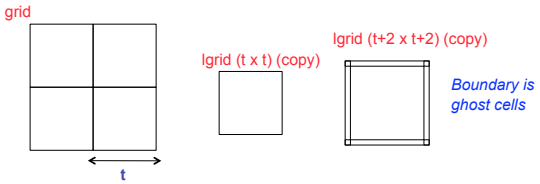
- Termination test
- In-place update? Or update a copy?
- What if communicating boundary values was REALLY expensive (high value of λ)?
  - Does that change how you solve the problem?

Common solution in distributed memory: Ghost cells

Local partition corresponds to just the tile

But, we also care about red elts to the left and blue elts above

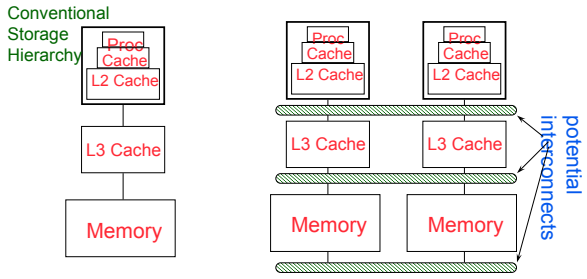
Add a boundary, a ghost cell



Locality – What does it mean?

- We talk about locality a lot
- What are the ways to improve memory behavior in a parallel application?
- What are the key considerations?
- Today's lecture
  - Mostly about managing caches (and registers)
  - Targets of optimizations
  - Abstractions to reason about locality
  - Data dependences, reordering transformations

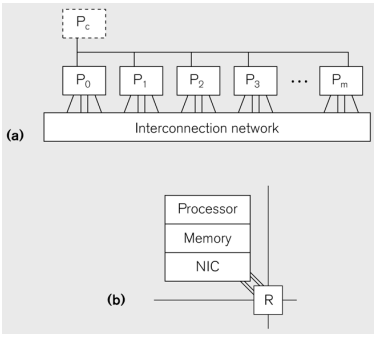
Locality and Parallelism (from Lecture 1)



- Large memories are slow, fast memories are small
- Cache hierarchies are intended to provide illusion of large, fast memory
- Program should do most work on local data!

Lecture 3: Candidate Type Architecture (CTA Model)

- A model with *P* standard processors, *d* degree, *λ* latency
- Node == processor + memory + NIC
- Key Property: Local memory ref is 1, global memory is *λ*



Managing Locality

- Mostly, we have focused on accessing data used by a processor from local memory
  - We call this data partitioning or data placement
  - Let's take a look at this Red/Blue example
- But we can also manage locality within a processor in its cache and registers
  - We'll look at this too!
  - Not really a parallel programming problem, but if you do not think about locality, you may give up a lot of performance.

Targets of Memory Hierarchy Optimizations

- Reduce ***memory latency***
  - The latency of a memory access is the time (usually in cycles) between a memory request and its completion
- Maximize ***memory bandwidth***
  - Bandwidth is the amount of useful data that can be retrieved over a time interval
- Manage overhead
  - Cost of performing optimization (e.g., copying) should be less than anticipated gain

Reuse and Locality

- Consider how data is accessed
  - *Data reuse:*
    - Same or nearby data used multiple times
    - Intrinsic in computation
  - *Data locality:*
    - Data is reused and is present in "fast memory"
    - Same data or same data transfer
- If a computation has reuse, what can we do to get locality?
  - Appropriate data placement and layout
  - Code reordering transformations

Cache basics: a quiz

- **Cache hit:**
  - -in-cache memory access—cheap
- **Cache miss:**
  - -non-cached memory access—expensive
  - -need to access next, slower level of hierarchy

Parameters to optimization

- **Cache line size:**
  - -# of bytes loaded together in one entry
  - -typically a few machine words per entry
- **Capacity:**
  - -amount of data that can be simultaneously in cache
- **Associativity**
  - -direct-mapped: only 1 address (line) in a given range in cache
  - -*n*-way: $n \geq 2$ lines w/ different addresses can be stored

Temporal Reuse in Sequential Code

- Same data used in distinct iterations I and I'

```
for (i=1; i<N; i++)
  for (j=1; j<N; j++)
    A[j]= A[j+1]+A[j-1]
```

- `A[j]` has self-temporal reuse in loop `i`

Spatial Reuse

- Same data transfer (usually cache line) used in distinct iterations I and I'

```
for (i=1; i<N; i++)
  for (j=1; j<N; j++)
    A[j]= A[j+1]+A[j-1];
```

- `A[j]` has self-spatial reuse in loop `j`
- **Multi-dimensional array note:**
  - C arrays are stored in row-major order
  - Fortran arrays are stored in column-major order

Summary of Lecture

- This Time
  - Discussion of programming assignment
  - Red/Blue example
  - Introduction to Reuse and Locality
- Next Time
  - How to think about data reuse to:
    - Partition data across processors
    - Reorder computation on a processor to "create" locality from data reuse