# CS4961 Parallel Programming

## Lecture 10: Data Locality, cont. Writing/Debugging Parallel Code

Mary Hall
September 23, 2010



## Observations from the Assignment

- Many of you are doing really well
- Some more are doing fine/well and don't realize it
- Some had some Linux/Solaris issues, which were unanticipated, but we were able to help
- There were problems with programming in C and debugging
  - Study how to use dbx on water, gdb or some other debugger
  - It will pay off in being a more efficient programmer
  - You MUST have this skill to be successful in a job that requires programming
- Solve the problem in steps (see next slide)
- Many started the assignment way too late
  - Some looked at the assignment for the first time on Tuesday or Wednesday
  - Some had not studied OpenMP or read the textbook



## Techniques for Writing/Debugging Parallel Code

- Start by writing nearby sequential code and debugging that
- Initially, use very small problem sizes so that you can compute the solution by hand and prove the computed solution is correct
- Once you believe your sequential code is correct, then you can add parallel constructs
- Debug parallel code that uses a single thread on a small problem size
- Once you believe your single-thread parallel code is working, try a small number of multiple threads on a small problem size



## OpenMP Issues (Review Lectures 5 & 6; Textbook pages 193-199)

- When to use shared/private?
  - E.g., Lecture 5, slide 33; Lecture 6, slide 8; Textbook page 195
- SPMD and OMP Parallel constructs
  - E.g., Lecture 5, slide 28
- OMP Parallel For
  - E.g., Lecture 5. slide 20
  - Textbook pages 195-196

## 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



## 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



## Can Use Reordering Transformations!

- Analyze reuse in computation
- Apply loop reordering transformations to improve locality based on reuse
- With any loop reordering transformation, always ask
  - **Safety?** (doesn't reverse dependences)
  - **Profitablity?** (improves locality)

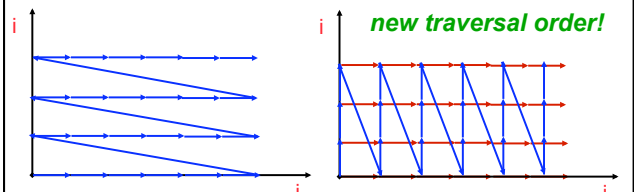


## Loop Permutation: An Example of a Reordering Transformation

Permute the order of the loops to modify the traversal order

```
for (i= 0; i<3; i++)
 for (j=0; j<6; j++)
   A[i][j+1]=A[i][j]+B[j];
```

```
for (j=0; j<6; j++)
 for (i= 0; i<3; i++)
   A[i][j+1]=A[i][j]+B[j];
```




**Which one is better for row-major storage?**

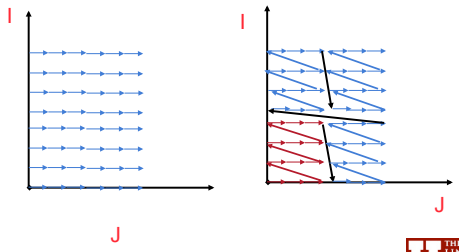


## Permutation has many goals

- Locality optimization
  - Particularly, for spatial locality (like in your SIMD assignment)
- Rearrange loop nest to move parallelism to appropriate level of granularity
  - Inward to exploit fine-grain parallelism (like in your SIMD assignment)
  - Outward to exploit coarse-grain parallelismx
- Also, to enable other optimizations

## Tiling (Blocking): Another Loop Reordering Transformation

- Blocking reorders loop iterations to bring iterations that reuse data closer in time
- Goal is to retain in cache between reuse






## Tiling is Fundamental!

- Tiling is very commonly used to manage limited storage
  - Registers
  - Caches
  - Software-managed buffers
  - Small main memory
- Can be applied hierarchically



## Tiling Example

```
for (j=1; j<M; j++)
   for (i=1; i<N; i++)
     D[i] = D[i] +B[j,i]
```

Strip mine

```
for (j=1; j<M; j++)
   for (i=1; i<N; i+=s)
      for (ii=i; ii<min(i+s-1,N); ii++)
         D[ii] = D[ii] +B[j,ii]
```

Permute

```
for (i=1; i<N; i+=s)
   for (j=1; j<M; j++)
      for (ii=i; ii<min(i+s-1,N); ii++)
         D[ii] = D[ii] +B[j,ii]
```



## How to Determine Safety and Profitability?

- Safety
  - A step back to Lecture 2
  - Notion of reordering transformations
  - Based on data dependences
- Profitability
  - Reuse analysis (and other cost models)
  - Also based on data dependences, but simpler

## Summary of Lecture

- This Time
  - 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



## 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