Plan for Today

Announcements

- Possible papers have been sent out on piazza. Reading abstracts for papers on list is reading assignment for Thursday.
- No quiz this week. Quiz next week can include questions about readings for this week.
- Reading for today was posted late.

Review Course Goals

Paper Reviews

- Over the semester you need to write 8 reviews.
- Reviews are due before class on the day the paper is being discussed.

Some slides courtesy of David Lowenthal from his Programming Models lecture.
Course Research Goals

Questions we will be exploring

– With the advent of multi-core and many-core, what key features should a parallel programming model include?
– How can we separate implementation concerns from the algorithm specification?
– How much should parallel programming models be specialized for particular domains?
– How can we compose programs from different domains?
– How does implementation of programming constructs impact design?
– How should programming models be evaluated?

Approach

– Read and critique influential papers
– Learn and present the key features in a variety of programming models.
– Research project where compare three programming models used to specify same computation
Selecting a Paper to Present

Email papers you are interested in reading for the class (optional)

Read through the potential list of papers
- Will provide and cover Tuesday January 26\textsuperscript{th} in class.
- Read the abstract, introduction, and conclusion for all papers you are seriously considering.
- Consider selecting a paper related to your course project.

Schedule around your deadlines in this course and other courses

Paper selection will be first-come-first-serve
- Paper selection can start January 28\textsuperscript{th}.
- I will start posting assignments as soon as requests are made.
- You must indicate which paper you would like to present by February 4 before class.
Next Time

Reading (posted on class schedule)
  – Introduction to Parallel Computing by Blaise Barney.
  – Abstracts from papers on the possible papers list.

Class
  – Approaches to evaluate parallel programming models.

Due
  – Paper suggestions (optional), can still suggest papers other than those on the provided list.
Example computations (Slide 1 of 2)

Sparse Matrix Vector Multiply

– Explicit representation of non-zeros and row and column indices
– Sparse data structure varies, some variants implicitly represent portions of the index space
– Similar kernels: matrix powers kernel and page rank have similar dependence patterns

Molecular dynamics simulations

– Implicit representation of non-zeros and explicit row and column indices
– List of interacting atoms for non-bonded computation
Stencil computations
- Implicit representation of non-zeros and row and column indices
- Rectangular grids
- Shallow Water Model (SWM)

Climate modeling
- Explicit representation of non-zeros and implicit row and column indices
- Unions of rectangular grids, unknowns and non-zeros are stored aligned with the grid
- In CGPOP miniapp

What are the implementation details for the touchstone kernels?
Matrix Powers Kernel (CSR version)

for (int k=1; k <= kparam; k++) {
    // $x^k = A x^{k-1}$
    for (int row=0; row<csr_mat->numrow; row++) {
        for (int p=csr_mat->IA[row]; p<csr_mat->IA[row+1]; p++) {
            int col = csr_mat->JA[p];
            x[k][row] += x[k-1][col]*csr_mat->A[p];
        }
    }
}
Parallelizing with OpenMP

• Add annotations to a sequential program, target is multicore machines
  – Language independent---implementations exist in C, C++, Fortran
  – Programmer does not add library calls (as in MPI)

• Programmer is responsible for finding parallelism
  – It’s just easier to use than pthreads
Parallelizing with OpenMP

- Most common: parallel directive
  - Can be a parallel for loop or task parallelism

  ```c
  int N = 100000; int i, a[N];
  #pragma omp parallel for shared(a) private(i)
      for (i = 0; i < N; i++)
          a[i] = 2 * i;
  ```

- Implicit barrier at end of the for loop, unless “nowait” specified
- Number of threads can be specified by programmer, or the default will be chosen
Matrix Powers Kernel (CSR version)

for (int k=1; k <= kparam; k++) {
  // x^{k} = A x^{k-1}
  for (int row=0; row<csr_mat->numrow; row++) {
    for (int p=csr_mat->IA[row]; p<csr_mat->IA[row+1]; p++) {
      int col = csr_mat->JA[p];
      x[k][row] += x[k-1][col]*csr_mat->A[p];
    }
  }
}

Which loop(s) are parallel?
Parallelizing Reductions with OpenMP

Reduction: efficiently handles finding the sum, product, max, or min of a shared variable when many threads are updating the shared variable

```c
#pragma omp parallel for reduction(+:sum)
for (i=0; i < n; i++)
    sum = sum + (a[i] * b[i]);
```
for (int k=1; k <= kparam; k++) {
    // x^k = A x^{k-1}
    for (int p=0; p<=copy_mat->nnz; p++) {
        int row = copy_mat->row[p];
        int col = copy_mat->col[p];
        x[k][row] += x[k-1][col]*copy_mat->val[p];
    }
}
Implementation Details

Computations Covered So Far

– Matrix Powers Kernel, main kernel is sparse matrix vector multiply

Implementation Details

– Loop over rows is parallel.
– Loop within a row is a reduction.
– Sparse matrices (and graphs) can be stored in MANY different formats.
Jacobi in SWM code (stencil computation)

do ksdm=1,nsdm
    do j=npad+1,ny-npad
        do i=npad+1,nx-npad
            work(i,j,:) = &
                    rw7(i, j, ksdm) * ( + rhs(i, j, :, ksdm) &
                        - l_weights( 1, i, j, ksdm) * xout(i-1, j , :, ksdm) &
                        - l_weights( 2, i, j, ksdm) * xout(i-1, j-1, :, ksdm) &
                        - l_weights( 3, i, j, ksdm) * xout(i , j-1, :, ksdm) &
                        - l_weights( 4, i, j, ksdm) * xout(i+1, j , :, ksdm) &
                        - l_weights( 5, i, j, ksdm) * xout(i+1, j+1, :, ksdm) &
                        - l_weights( 6, i, j, ksdm) * xout(i , j+1, :, ksdm))
        enddo
    enddo
endo
Parallelizing with Data Parallel Languages

• HPF: High Performance Fortran
• Idea: data parallel language, plus annotations that suggest data distributions
• If the compiler knows the data distribution, it is straightforward to distribute the work and determine where to insert communication
  – If the rule is “owner computes”
• Problem:
  – Figuring out the data distribution isn’t easy for the programmer
Example HPF code fragment

```c
double A[N,N], B[N,N] {BLOCK, BLOCK}

...

for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        A[i][j] = 0.25 * (B[i][j+1] + B[i][j-1] + B[i+1][j] + B[i-1][j]);
    }
    ...
}
```
HPF Compiler Analysis

• From number of processors and distribution annotations, determines what each processor *owns*
• Determines what can legally execute in parallel (user could annotate here, conceptually)
• Divides up iterations between processors based on data distribution annotations
• Determines necessary communication by looking at array references along with data owned; then, inserts that communication
Implementation Details

Example Computations Covered So Far
- Matrix Powers Kernel, main kernel is sparse matrix vector multiply
- Stencil computation

Implementation Details
- Loop over rows is parallel.
- Loop within a row is a reduction.
- Sparse matrices (and graphs) can be stored in MANY different formats.
- Data and computation distribution
Parallelizing with Cilk

- Supports efficiently recursive parallelism
- Classic recursive example is computing Fibonacci numbers (an idiotic program but a good example)

```c
int fib(int n) {
    if (n < 2) return 1;
    else {
        x = fib(n-1);
        y = fib(n-2);
        return x + y;
    }
}
```
Parallelizing with Cilk

```cilk
int fib(int n) {
    if (n < 2) return 1;
    else {
        x = spawn fib(n-1);
        y = spawn fib(n-2);
        sync;
        return x + y;
    }
}
```
Parallelizing with Cilk

• Extremely simple model
  – Can quickly adapt sequential recursive programs
  – But, can only adapt sequential recursive programs

• What does Cilk runtime system have to do?
  – Quite a bit, actually
  – Must deal with:
    • Not creating too much parallelism
    • Efficiently allowing work stealing to happen between processors to avoid load balancing
Implementation Details

Example Computations Covered So Far
- Matrix Powers Kernel, main kernel is sparse matrix vector multiply
- Stencil computation
- Fibonacci (not real), mergesort

Implementation Details
- Loop over rows is parallel.
- Loop within a row is a reduction.
- Sparse matrices (and graphs) can be stored in MANY different formats.
- Data and computation distribution
- When should recursion stop?
Barnes-Hutt

PseudoCode

Set bodies = /* read input */;
for (int step = 0; step < maxTimestep; step++) {
    Octree octree = new Octree();
    foreach (Body b : bodies)
        octree.Insert(b);
    octree.SummarizeSubtrees();
    foreach (Body b : bodies)
        b.ComputeForce(octree);
    foreach (Body b : bodies)
        b.Advance();
}
for (tstep=0; tstep<=n_tstep-1; tstep++) {
    ...
    for (i=0; i<=n_moles-1; i++) {
        x(i) = x(i) + vhx(i) + fx(i);
        ...
        if ( x(i) < 0.0 ) x(i) = x(i) + side ; ...
        if ( x(i) > side ) x(i) = x(i) - side ; ...
    
    vhx(i) = vhx(i) + fx(i); ...
    fx(i) = 0.0; ...
    }
    for (ii=0; ii<=n_inter-1; ii++) {
        i = inter1(ii); j = inter2(ii);
        fx(i) += ... x(i) ... x(j)...
        fx(j) += ... x(i) ... x(j)...
    }
    for (i=0; i<=n_moles-1; i++) {
        ...
        vhx(i) = ... fx(i) ...; ...
    }
    }
}
Important Tangent: Parallelizing Histogram Computations

Questions we will be exploring

– With the advent of multi-core and many-core, what key features should a parallel programming model include?

Source: http://www.futurechips.org/tips-for-power-coders/writing-optimizing-parallel-programs-complete.html