CSE 262
Spring 2007
Scott B. Baden

Lecture 4
Data parallel programming
Announcements

• Projects
• Project proposal - Weds 4/25 - extra class
Data Parallel Programming
Data parallel programming

• Similar to SIMD parallelism
• Virtual processors execute the same operations on multiple data
• Implement the abstraction on MIMD architectures using SPMD parallelism
The data parallel model

- A parallel data structure, e.g. an array, list, sequence
- Apply an operation uniformly over all processors in a single step
- Assign each array element to a virtual processor
- Implicit barrier synchronization between each step
- Program executes as if in a global space
Practical data parallel languages

• APL (1962)
• Matlab
• Fortran 90, 95
• HPF (High Performance Fortran) - 1994
How do we express parallelism?

• Operations on whole arrays
• Forall, a parallel for loop
• FORALL ( triplet, triplet,… ) assignment statement

  forall (i=0:n-1) x[i] = (i*2.0/n)-1.0

  forall (i=0:n-1, j = 0:m-1) X[ i , j ] = 1.0/(i+j)

• The head of the loop defines an index domain
• We think of each member of the index domain as defining a virtual processor
  – In the first example, we have an index domain of 0 to n-1, with n-1 virtual processors
  – In the second example we have an index domain of n × m processors
Other uses of forall loops

• Indirect indexing
  \[
  \text{forall } (i = 0:n-1) \ D[\text{ indx}[i]] = C[i]
  \]

• Optional mask or guard
  \[
  \text{forall } (i=1:n, j:1:n, i == j) \ X[i,j] = 0 \ // \ \text{Guard}
  \]

• A parallel loop nested inside a serial loop
  \[
  \text{for } k = 1:n \\
  \quad \text{forall } (i=1:n, j=1:n) \ C[i,j] = C[i,j] + A[i,k] * B[k,j]
  \]

• An illegal forall loop
  \[
  \text{forall } (i=1:n, j=1:n, k=1:n) \\
  \quad C[i,j] = C[i,j] + A[i,k] * B[k,j]
  \]
Forall loop evaluation

- Evaluate entire RHS for all index values (in any order) and assign to a temporary
- Perform all assignments (in any order) using the temporary
- No more than one value for each element on the left hand side
- The following are equivalent

```latex
forall ( i = 1:n)\ unew[i] = (u[i-1] + u[i+1] ) / 2.0

forall ( i = 1:n)\ tmp[i] = (u[i-1] + u[i+1] ) / 2.0
forall ( i = 1:n)\ unew[i] = tmp[i]
```
Forall loop implementation

• The compiler may be able to eliminate some barriers
• Exposes opportunities for traditional loop optimizations, e.g. loop fusion
• Each array element has a processor owner
• By convention, we use the owner computes rule

forall ( i = 1:n) A[i] = (u[i-1] + u[i+1] ) / 2.0
forall ( i = 1:n) X[i] = sin(A[i])

for ( i = my_start : my_end ) {
    A[i] = (u[i-1] + u[i+1] ) / 2.0
    X[i] = sin(A[i])
}

• The compiler may be able to eliminate some barriers
• Exposes opportunities for traditional loop optimizations, e.g. loop fusion
• Each array element has a processor owner
• By convention, we use the owner computes rule
Array Operations

Parallel Assignment, equivalent to a forall

```plaintext
double A[N,N,N], Z[N,N], D[N], C[N], T[5]
A = 0 // scalar extension, // all elements set to 0
forall(i=0:N-1,j=0:N-1,k=0:N-1) A[i,j,k] = 0
Z = 3.7 //forall(i=0:N-1,j=0:N-1) Z[i,j] = 3.7
D = C // array copy
T = [1 2 3 4 5] // An array literal
```

Binary array operators operate pointwise on conforming arrays

- same size and shape
- The arrays could be multidimensional
Extension to array operations

• Scalars can be combined with arrays

• There are also specialized intrinsics

\[
\begin{align*}
T &= [1 \ 4 \ 9 \ 16 \ 25] \\
U &= 3 + T & \text{// 4 \ 7 \ 12 \ 19 \ 28} \\
Z &= \sqrt{T} & \text{// Built in intrinsic extended to array} \\
& & \text{// 1 \ 2 \ 3 \ 4 \ 5} \\
Y &= \text{max}(T,10) & \text{// 10 \ 10 \ 10 \ 16 \ 25}
\end{align*}
\]
Array Sections

• Portion of an array defined by a triplet in each dimension
• May appear wherever an array is used

A[1:5] ! first five elements
A[1:10:2] ! odd elements
B[j,:] ! jth row
B[:, 1] ! first column
Data Motion

\begin{align*}
\text{CSHIFT( array, dim, shift) } & \quad \text{! cyclic shift in one dimension} \\
\text{TRANSPOSE( matrix )} & \quad \text{! matrix transpose}
\end{align*}

\begin{figure}
\centering
\begin{tabular}{cccc}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
\end{tabular}
\quad \rightarrow \\
\begin{tabular}{cccc}
4 & 1 & 2 & 3 \\
8 & 5 & 6 & 7 \\
\end{tabular}
\caption{Example of a cyclic shift and matrix transpose.}
\end{figure}
Parallel prefix (scan) operations

\[ X = [4 \ 5 \ 6 \ 7 \ 8 \ 9] \]

\[ \text{SUM_PREFIX}(X) \]

\[ \text{SUM_PREFIX}(X, \text{MASK}=[T \ T \ F \ T \ F \ T]) \]

\[ \text{SUM_PREFIX}(X, \text{SEGMENT}=[T \ T \ T \ F \ F \ T]) \]
Parallel prefix & pointer jumping implementation

Motivating application: the N-body problem

• A classical problem
• Compute trajectories of a system of N bodies moving under mutual influence
• The bodies can be molecules, planets, stars, charged particles…
Trajectories

• Particles move continuously through space and time
• On a computer we represent continuous values using a discrete approximation
• Evaluate force field at discrete points in time, called timesteps $\Delta t$, $2\Delta t$, $3\Delta t$, …
  – $\Delta t$ is called the *time step* (a parameter)
• “Push” the bodies according to Newton’s third law
  \[ F = ma = m \frac{du}{dt} \]
Solving the N body problem

while (current time < end time)
    forall bodies i ∈ 1:N
        compute force $F_i$ induced by all bodies j ∈ 1:N
        $$F_i = \sum_i F_{ij}$$
        update position $x_i$ by $F_i \Delta t$
        current time += $\Delta t$
end
Computing the force

• The running time of the computation is dominated by the force computation, so we ignore the rest
• The simplest approach is to use the direct method, with a running time of $O(N^2)$
  
  \[
  \text{Force on particle } i = \sum_{j=0}^{N-1} F(x_i, x_j)
  \]

• $F(\ )$ is the force law
• One example is the gravitational force law

  \[
  G \frac{m_i m_j}{r_{ij}^2} \text{ where } r_{ij} = \text{distance}(x_i, x_j)
  \]

  $G$ is the gravitational constant
Data parallel coding

• The particles are described by
  • the position and mass array $xyzm[ ]$
  • shifted in copy $xyzmC$
  • Let the force law be a given function $F(xyzm,xyzmC)$

$$xyzmC = xyzm$$

do while (t < t_end)
  for i = 1 : n
    force = force + F(xyzm,xyzmC)
    $xyzmC = \text{CSHIFT}(xyzmC, 1)$ ....
  end for
end do
Reduction Operators

Reduce an array to a scalar under an associative binary operation

- sum, product
- minval, maxval

\[
\text{do while (maxdiff < epsilon)}
\]

\[
\text{unew}[1:N] = (\text{uold}[0:N-1] + \text{uold}[2:N+1]) / 2.0
\]

\[
diff = \text{unew} - \text{uold}
\]

\[
\text{absdiff} = \text{abs}(\text{diff})
\]

\[
\text{maxdiff} = \text{maxval}(\text{absdiff})
\]

\[
\text{enddo}
\]
Conditional Operations

- The following statement
  \[ \text{dist} = \max(\text{abs}(\text{fishp}), 0.01) \]
  Is equivalent to
  \[
  \text{where } (\text{abs}(\text{fishp}) \geq 0.01) \text{ forall } (i=0:n-1) \\
  \quad \text{dist} = \text{abs}(\text{fishp}) \quad \text{if}(\text{abs}(\text{fishp}[i]) \geq 0.0) \\
  \text{elsewhere} \quad \text{dist}[i] = \text{abs}(\text{fishp}[i]) \\
  \quad \text{dist} = 0.01 \quad \text{else} \\
  \text{end where} \quad \text{dist}[i] = 0.01
  \]

- Recall that in an SIMD architecture, processors can individually opt out of executing an operation

- An SIMD machine needs 2 steps to execute the statement

- What about an MIMD machine?
Data Distribution

• Three kinds of distributions
  – Block
  – Cyclic
  – Block_cyclic

• Can we use one of these to express the others?
Layouts on Processor Grids

- [Block, *]
- [*], Block]
- [Block, Block]
- [Cyclic, *]
- [Cyclic, Cyclic]
- [Cyclic, Block]
Block cyclic

[Cyclic, *]
[Cyclic, Cyclic]
[Cyclic, *]
[Cyclic(2), *]
[Cyclic(2), Cyclic(2)]
[Cyclic(2), Block]
Implicit Communication

Enforcement of owner computes rules may induce communication

\[ A(1:7) = B(2:8) \quad ! \text{A and B are distributed BLOCKwise} \]

\[
\text{forall (i=1:7) } A[i] = B[i+1]
\]
Subtleties of implicit communication

• Communication may occur if LHS and RHS have different layouts
  
  • Consider $A[1:7] = C[2:8]$ where $A$ is BLOCK distributed and $C$ is CYLIC

• Abstraction obscures performance penalty
Passing Arguments

• When the actual and formal parameter have different layouts

• HPF identified 3 cases
  http://hpff.rice.edu versions/hpf2/hpf-v20/node49.html

  • **Prescriptive:** compiler must “make it so”
  • **Descriptive:** a weak assertion that no mapping is needed, but if false, the compiler must deal with it
  • **Transcriptive:** no assertions available for the compiler: must be dealt with at run time
Global Communication

\[ X = X[n:1:-1] \quad ! \text{permutation (reverse)} \]
\[ B = A[\text{Indx}[:]] \quad ! \text{"gather"} \]
\[ C[\text{Indx}[:]] = B \quad ! \text{C = "scatter:"} \]
\[ \quad ! \text{no duplicates on left!} \]
Array Homes

• The Connection Machine system consists of the front end and the CM

• The home of an array is the machine it is located on

• Determined by usage
  • Front end: any array accessed via subscript only
  • CM: any array referenced at all using whole array

```plaintext
Integer A(4), B(4)
do i = 1, n
   sum = sum + A(i)* B(i)
end do
A = A + B
```