Models of Parallel Computation
and Design of Parallel Programs

Jim Browne
9/19/2011
Outline for Lecture

• Problem Statement
• Definitions
• Goals
• Development processes
• Parallel Structure Derivation Processes
• Block Triangular Solve Example
• Basic Models of Parallel Computation
• Two-D FFT Example
What is Different about Parallel Programming?

- Sequential programming is a special case of parallel programming.
- A correct parallel program will execute correctly with a single thread of execution.
- Sequencing of computations is formulated in terms of dependence relations, not control flow.
Simplifying Principle

• Separation of Concerns
  – Identify units of computation
  – Derive parallel structure
  – Implement units of computation in parallel structure

• Why
  – KISS – Keep it Simple Stupid
Definition: Unit of Computation

- **Unit of computation**
  - A program which maps one set of inputs into a set of outputs
  - An initial condition on its inputs and states which enables the program to execute
  - A terminal action which distributes the outputs of the program
Definition: Dependence Relation or Component Interaction

Specification of the sources of inputs for the initial condition and the targets for the outputs of the computation.
Definitions

• Sequential – Single thread of execution on one processor
• Concurrent – interleaved execution of different units of computation on a single processor.
  – Example – Interleaved execution of two threads on a single processor
  – Far-fetched Example – interleaved execution of random units of computation from a single sequential program
• Parallel – “simultaneous” execution of different units of computation on multiple processors.
• Dependence Relation – Reliance of one unit of computation on some set of other computations for satisfaction of its initial condition.
Goals

• Design parallel programs in terms of models of computation
  – Domain confined to “macro” or task level parallelism.

• Show how the models of computation aid in optimization and scheduling.
Development Processes

• Sequential to parallel
  – Formulate the program as a sequential program – map to parallel structure by adding MPI and/or OpenMP structuring to the sequential program

• Parallel by design
  – Create parallel structure at “unit of computation” level and then complete program by develop of serial or parallel implementations of the units of computation.

• Model instantiation
  – Instantiate computation in some model of parallel computation such as bulk synchronous processes.
Example – Solution of Lower Triangular Matrix.

Algorithm

- $Ax = b$
- $A$ is lower triangular
- $X[1] = b[1]/A[1,1]$
- $\ldots$
- $X[m] = \left(b[m] - \sum A[m,i]*x[i]\right)/A[m,m]; \quad l = 1,m-1$

Reasonable units of computation will operate on blocks of the matrix
Blocked Algorithm

1. Partition the matrix into blocks of size k resulting in a matrix of n/k blocks.
2. Solve the top left hand triangular block using Gaussian elimination giving values to x[1] to x[k].
3. Compute, in parallel, the partial sums of the vector-vector multiply-sum for evaluating x[k+1] to x[2k] for the leftmost column of square blocks.
4. Solve the second triangular block from the left for x[k+1] through x[2k].
5. Compute, in parallel, the partial sums of the vector-vector multiply-sum for evaluating x[2k+1] to x[3k] for the second from left column of square blocks.
Separation of Concerns

• Specify units of computation
  – Partition matrix into blocks
  – Gaussian elimination solve for triangular blocks
    \((TS[i])\)
  – Vector-Vector multiply with sum for square blocks
    \((V-V)[i,j]\))

• Specify interactions among units of computation
Separation of Concerns

• Specify units of computation
  – Partition matrix into k blocks
  – Gaussian elimination solve for triangular blocks: TS[i]
  – Vector-Vector multiply with sum for square blocks: (V-V)[i,j]

• Specify dependences among units of computation
  – V-V[i,j] depends on TS[i] for j = i,k-1
  – TS[i] depends on V-V[i,i] and V-V[j,i] for j = i to k-1
Block Triangular Solver

Units of Computation
1. Sequential solve of triangular block – TS[k]
2. Vector- Vector multiply and sum – V-V[i,j]

Both of these operations are available in linear algebra libraries.

Data Dependences
1. V-V[1,j] on TS[1] ; j = 1, k-1
2. TS[2]2 on V-V[1,2] and V-V[2,2]
4. ETC.
Component Model: BTS - TS1

S1 – Initial condition
1. Dimension of matrix and size of blocks must be available
2. First triangular segment of A and corresponding elements of b must be available.

S1 – Program
1. Sequential solve of triangular linear system.

S1 – Terminal action
1. X[1] .... X[k] made available for other units of computation

Diagram:
- **A** -> Initial condition
- **b** -> Program
- **n,k** -> Terminal action
- **X[1] .... X[k]**
Component Model: BTS TS[m]

**Sm – Initial condition**
1. Dimension of matrix and size of blocks must be input.
2. mth triangular segment of A and corresponding elements of b must be input.
3. Result of vector-matrix multiple for [m-1,m-1] and [j,m-1] for j = m+1,k-1 blocks must be input.

**Sm – Program**
1. Sequential solve of triangular linear system.

**Sm – Terminal action**
1. X[m*k+1] ..., X[m*k+k] made available for V-V multiplies unless m = k and of course, to output.
Component Model: BTS V-V[m,m]

**V-V[m,m] – Initial condition**
1. Dimension of matrix and size of blocks must be input
2. Block of A must be input
3. $X[m*k+1] \ldots X[m*k+k]$ must be input

**V-V[] – Program**
1. Vector-Vector multiplies with sum

**V-V[m,m] – Terminal action**
\[ \sum M[m,i]x[i] \]
for $j = m*k +1$ to $m*k +k$ made available for other units of computation
Data Flow Graph Formulation

A graph model of a computation where:

• The nodes represent units of computation
• The arcs represent the data the data required to satisfy the initial condition for execution of the sink node from the source node to the sink node

Nodes may be hierarchically parallel
Nodes are units of computation
Arcs carry data
Note that some nodes have multiple inputs
Properties of Data Flow Graph Model

• Independent of implementation and execution environment
• Parallel structure is resolved to individual components
• Resolution to units of computation assist in identifying components which can be found in libraries.
• Makes migration across system architectures simple.
Implementation of Data Flow Graph

• Distributed Memory
  – Arcs carry messages, eg. MPI send/receive

• Shared Memory
  – Arcs are implemented as writes and reads to the shared memory
  – Requires attention to synchronization between writes and reads
State Machine Formulation

• Parallel programs as interacting state machines
• Mealy state machine
  – Nodes are units of computation – implement state transitions.
  – Arcs carry events
• Data flow graphs and state machines are different but equivalent models of execution.
• State machine model is normally used only for formulations intended for distributed memory.
State Machine Formulation

• Initial condition is a predicate on receiving sets of events
• State transition is the unit of computation
• Terminal action is sending sets of events to other nodes in the state
• Nodes are not aware of which other nodes generate or receive events.
Example – Two Dimensional FFT

\[ F(k, l) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i, j) e^{-i2\pi \left( \frac{ki}{N} + \frac{lj}{N} \right)} \]

\[ f(a, b) = \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} F(k, l) e^{i2\pi \left( \frac{ka}{N} + \frac{lb}{N} \right)} \]
Example – Two Dimensional FFT

• FFTs are important components in many formulations of scientific and engineering computing problems.
• Multi-dimensional FFTs occur quite commonly.
• Computation is determination of two-D matrix of coefficients
• Standard formulation of multidimensional FFTs are difficult to parallelize as macro-level tasks.
• Swwarztrauber's algorithm decomposes a multidimensional FFT into many one-D FFTs
Swarztrauber's Multiprocessor FFT Algorithm

1. Partitioning the matrix of coefficients row wise (horizontally) into P submatrices, one for each processor.
2. Sending these submatrices to each of the P processors for computation. The size of each the submatrix is N/P x M.
3. Each processor performs a 1D FFT on every row of the submatrix that it received.
4. Collecting these 1D FFT’s and then transposing the N x M matrix. The resulting matrix is of size M x N.
5. Splitting the M x N matrix row wise into P submatrices. The size of each of the submatrix is M/P x N.
6. Sending these submatrices to the each of the P processors for computation.
7. Again each processor performs a 1-D FFT on every row of the submatrix that it received.
8. Collecting all the submatrices from the P processors and transposing the M x N matrix to get an N x M matrix. The resulting N x M matrix is the 2D FFT of the original matrix.
Separation of Concerns: 
Units of Computation

• An initialization and partitioning component: Initialize
• A one-dimensional FFT component: fft_row
• A component which partitions and distributes matrices: distribute
• A component which merges rows or columns to recover a matrix and which may optionally transpose the recovered matrix: gather_transpose
• An output component: print
Separation of Concerns
Dependences/Interactions

1. Distribute on initialize
2. Each fft_row on distribute
3. GatherTranspose on each fft_row
4. Distribute on gatherTranspose
5. Each fft_row on distribute
6. GatherTranspose on each fft_row
7. Print on gatherTranspose
2D FFT Example (Cont’d)

Fig. 1. Data Flow Graph of 2D FFT Computation
Scheduling with DFGs

• Measure or estimate compute times and communication times.
• Theoretical problem is mapping of DFG graph, to resources such that its execution time is minimized.
• Find the longest path and be sure it has resources to finish as quickly as possible.
• Use compute times of nodes to schedule for load balance.
Summary

- Formulation of parallel programs is simplified by formulating them in a structured model of parallel computation.
- Separation of concerns makes design of parallel structures a set of local designs.
- DFG graph program representations have many advantages with respect to implementation.