INTEGRATING TASK AND DATA PARALLELISM: ISSUES

Alok N. Choudhary (Syracuse University)
Outline

- Task and Data Parallelism
- Implementation Issues
  - Language
  - Compiler
  - Runtime
- Example Applications.
- Case Studies
- Directions.
Task Parallelism

Task Parallelism: What is it?

- Task parallelism involves multiple (possible dynamic) threads of control.
- There can be explicit interaction between the threads during execution.
- Task Parallelism may be nested, may be integrated with other parallel paradigms, and is explicitly visible to the user at the algorithmic level.
- Task Parallelism covers from all static threads to completely dynamic threads.
Data Parallelism - Integration

Data Parallelism: What is it?

- There is only one thread of control.
- It involves parallel operations on large data structures.

Though the two paradigms are seemingly opposite, an integrated task and data parallel environment can be useful for large class of problems.

Who might use integrated systems?

- Multi-disciplinary Applications
- Image Processing and Computer Vision
- Scientific Simulations - Ocean Model, Radar Control....
- Transaction Systems.
- Command and Control.
- ...

Example Problem

A 2D convolution is a good example illustrating the need for an integrated task and data parallel system. The problem has:

- Two forward 2D ffts.
- A matrix multiplication and inverse 2D fft.

Each of this modules can be expressed as a data parallel task co-ordinated by a task parallel layer.

![Diagram showing the process of 2D convolution with FFT, MM, FFT^(-1), and convolution of Image 1 & Operator.]

FM process

FFT (HPF)

FM process

FFT (HPF)

FM process

MM, FFT^(-1) (HPF)

Convolution of Image 1 & Operator

FM channels
Implementation Issues

Some of the implementation issues involved in an integrated system are:

- **Language**: Integrated systems will require one common language to express both forms of parallelism. Extension to HPF maybe an answer to that.

- **Runtime Systems**: A runtime system which provides both communication and thread management is required. Nexus, developed at ANL and CalTech is one such system. HPF Runtime System at Syracuse uses the I/O primitives for communication.

- **Compilation**: An integrated compilation is required. The compilation requires thread management capabilities as well.

- **Heterogenous Systems**: The mixed mode of programming extends easily to heterogenous model as independant threads may execute on different systems. Processor and Memory allocation would become important issues in such an environment.

- **Performance Evaluation**: Performance evaluation would be an important aspect of mixed mode systems as the user would like to experiment with such systems to find a "good" set of machines to map the problem onto.
- Task parallelism in Fortran is important as there is a massive amount of legacy Fortran code already existing.

- Thread Creation: Dynamic thread creation can involve asynchronous spawns of threads.

- Resource management: Resources like memory, processor, I/O channels etc are allocated dynamically or statically. Static allocation may mean wastage of resources. Dynamic allocation brings in the question of deadlock etc.

- Scheduling multiple threads becomes an issue.

- Enforce the user to do the resource management?
• Thread interactions: Data vs Control interaction.

• Name Space: Global Name Space as interaction mechanism vs distributed name space. Hierarchichal name space may be a viable alternative.

• Message Passing as interaction mechanism. Should the links between the tasks be established statically or dynamically.

• Directives to control the task parallelism?

• Debugging of the integrated programs would be difficult. Need a very good debugging environment.
Example Applications: Ocean Simulation

- This model simulates the effect of land and ocean on the atmosphere.

- Ocean Simulation has two independent tasks simulating ocean and land respectively. Each of the ocean and land task are in themselves data-parallel in nature.
  - Task 1: Ocean - a data parallel task.
  - Task 2: Land - a data parallel task.
  - Task 1 and Task 2 interact at the boundaries after each time step. This interaction is co-ordinated using a task parallel layer.
Example Applications: Multibaseline Stereo

- Multibaseline Stereo: This an image processing application which uses two cameras to evaluate the depth features in a scene.
  - The input consists of a reference image and two match images.
  - The difference and the error images are used as intermediate images to calculate the depth of the pixels in the input image.
  - The output - a disparity image which gives the depth of each pixel.
Case Study: Fx Compiler at CMU

- The Fx compiler supports both task and data parallelism. The base language is Fortran 77 augmented with Fortran 90 array syntax and data layout statement based on HPF.
- Task parallelism is expressed in special code regions called parallel sections.
- Emphasis on obtaining a good mapping of the problem on the resources available on the system. They provide a framework for making processor and memory allocation decisions required in an integrated system.
Current Work : Fx ..contd

- Framework for obtaining a good mapping uses the following characteristics of parallel programs.

  - Scalability: As the number of processors allocated to a task is increased, the average processor efficiency should also scale correspondingly.

  - Inter-task communication: Although the amount of data transferred between two task sub-routines is a feature of the program, the actual cost of the communication depends on the number of processors executing task subroutines and the mode of data distributions.

  - Memory Requirement: This determines the minimum number of processors needed to execute a task subroutine or a module containing a set of task subroutines.

- Framework for mapping programs: In deriving an efficient mapping for a parallel program expressed as data-parallel task sub-routines, following are the major steps involved:

  1. Partitioning task subroutine into modules.
  2. Allocate processors to modules.
  3. Replicate modules into module instances.
Case Study: FM-FD integration at Syracuse-ANL

- Language Integration (Example)
  - Each process/task written in data-parallel language (e.g. HPF)
  - Tasks connected using channels
  - Each task assigned to a FM submachine
  - Process initiated using the statement:
    HPF_CALL p_name (c1, c2) submachine(a:b)
    * Sets up the input and output channels c1 and c2.
    * Allocates a set of processors (submachine) for the process.
Case Study: 2D Convolution using FM/HPF

- Each process is Data-Parallel.
- Process Compiled using HPF Compiler.
- Processes communicate using the FM channels.
- The FM channels communicate data using parallel sends and parallel receives.
C HPF Code

program data_parallel
complex A(512,512), B(512,512), C(512,512)
!HPF$ processors p(32)
!HPF$ align B, C with A
!HPF$ distribute A(BLOCK,*)
do i = 1,nimages  ! Repeat for each image:
call read(A,B)
call fft(A)  ! FFT first input array
call fft(B)  ! FFT second input array
C = A*B
forall(i=1:512, j=1:512) C(i,j) = CONJG(C(i,j))
call fft(C)  ! Inverse FFT
call write(C)
enddo
Code : FM-FD Version

C     FM Code
program mixed_parallel
processors pr(24)
outport (complex x(512,64)) os1(8)
outport (complex x(512,64)) os2(8)
inport (complex x(512,64)) is1(8)
inport (complex x(512,64)) is2(8)
channel(in=is1(:), out=os1(:))
channel(in=is2(:), out=os2(:))
...
processes
  hpfcall fftW(img,os1) submachine(1:8)
  hpfcall fftW(img,os2) submachine(9:16)
  hpfcall iftW(is1,is2) submachine(17:24)
endprocesses
end
Channel Implementation : T-D Parallel System

- Channel are used as modes of communication between the data-parallel tasks.
- Channel Implementation is crucial for performance and the semantics of the model. Typically a channel may require all to all connection between the processors in two data parallel tasks.
- The channels implemented at Syracuse University have the following properties.
  - The channels are distribution independant. The channels do the required transformations to convert from the sending end’s distribution to the receiving end’s distribution.
  - Information Hiding : The tasks by themselves are independant of each other.
Parallel I/O to Integrate T-D Parallelism

- Each of the FM process is written in HPF.
- Processes communicate using the parallel sends and parallel receives.
- The channels are established using files where the data is stored in an intermediate format.
- There may be a single file or multiple files as intermediate storage.
- The files are not necessarily on disks. They are in the buffer space.
- The single file format has the bottleneck of multiple processors accessing the same file.
Parallel I/O to integrate T-D Parallelism

- Data on the file may stored in the striped format or contiguous blocks.
- Data in file 1 is contiguous.
- Data in file 2 is striped.
- MR is the mapping function which maps the T1 processors to the file.
- MW is the the mapping function to map the T2 processors to the file.
DIRECTIONS

- Eventual goal is of producing a single language/compiler for an integrated task/data parallel system.
- An integrated runtime system for both modes of parallelism.
- Language constructs for dynamic creation/deletion of processes.
- A performance model for evaluating a mixed mode system in a heterogeneous computing environment.
- Compilation and task allocation strategies for a heterogeneous computing environment.