Introduction of Parallel Programming

www.see-grid.eu

SEE-GRID-SCI Training Event,
Yerevan, Armenia, 24-25 July 2008

Mikayel Gyurjyan
Institute for Informatics and Automation Problems
National Academy of Sciences of the Republic of Armenia
Mikayel_Gyurjyan@ipia.sci.am

The SEE-GRID-SCI initiative is co-funded by the European Commission under the FP7 Research Infrastructures contract no. 211338

Outline

SEE-GRID
South Eastern European GRid-enabled eInfrastructure Development

- Problems in parallel computing
- Fosters Four step Process for Designing Parallel Algorithms
- Partitioning methods
- Communication
- Agglomeration
- Mapping
- Summary of Software

Solving Problems in Parallel(1)

SEE-GRID
South Eastern European GRid-enabled eInfrastructure Development

- It is true that the hardware defines the parallel computer. However, it is the software that makes it usable

- Parallel programmers have the same concern as any other programmer:

  - Algorithm design
  - Efficiency
  - Debugging ease
  - Code reuse
  - Lifecycle.

Solving Problems in Parallel(2)

SEE-GRID
South Eastern European GRid-enabled eInfrastructure Development

- However, they are also concerned with:
  - Concurrency and communication
  - Need for speed (need high performance)
  - Plethora and diversity of architecture

Choose Wisely

- How do I select the right parallel computing model/language/libraries to use when I am writing a program?
- How do I make it efficient?
- How do I save time and reuse existing code?

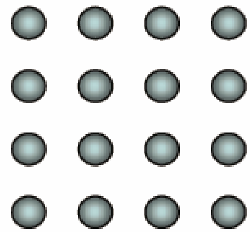
Fosters Four step Process for Designing Parallel Algorithms

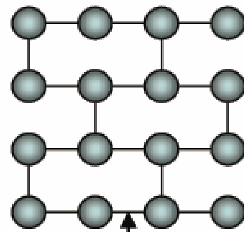
SEE-GRID
South Eastern European GRid-enabled eInfrastructure Development

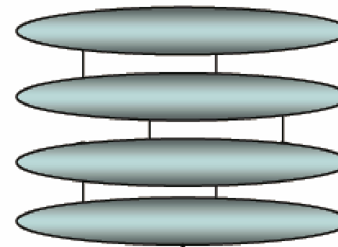
1. Partitioning – process of dividing the computation and the data into many small pieces – decomposition
2. Communication – local and global (called overhead) minimizing parallel overhead is an important goal and the following check list should help the communication structure of the algorithm
3. Agglomeration is the process of grouping tasks into larger tasks in order to improve the performance or simplify programming. Often in using MPI this is one task per processor.
4. Mapping is the process of assigning tasks to processors with the goal to maximize processor utilization

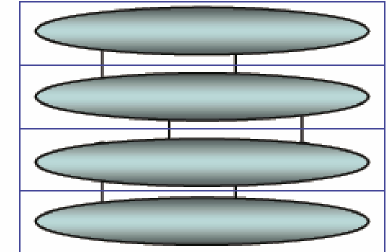
Abstract Diagram

Partitioning → Communication → Agglomeration → Mapping









Determine Tasks

Derive Communication Channels (timelines not shown here)

Tasks are Grouped

Processors Assigned to Tasks

Partitioning methods

SEE-GRID
South Eastern European GRid-enabled eInfrastructure Development

- Perfectly parallel
- Domain
- Control
- Object-oriented
- Hybrid/layered (multiple uses of the above)

Perfectly parallel

SEE-GRID
South Eastern European GRid-enabled eInfrastructure Development

- Applications that require little or no inter-processor communication when running in parallel
- Easiest type of problem to decompose
- Results in nearly perfect speed-up
- The pi example is almost perfectly parallel
  - The only communication occurs at the beginning of the problem when the number of divisions needs to be broadcast and at the end where the partial sums need to be added together
  - The calculation of the area of each slice proceeds independently

Domain decomposition(1)

- In simulation and modeling this is the most common solution
  - The solution space (which often corresponds to the real space) is divided up among the processors. Each processor solves its own little piece
  - Finite-difference methods and finite-element methods lend themselves well to this approach
  - The method of solution often leads naturally to a set of simultaneous equations that can be solved by parallel matrix solvers
  - Sometimes the solution involves some kind of transformation of variables (i.e. Fourier Transform). Here the domain is some kind of phase space. The solution and the various transformations involved can be parallized

Domain decomposition(2)

SEE-GRID
South Eastern European GRid-enabled eInfrastructure Development

- Solution of a PDE (Laplace's Equation)
  - A finite-difference approximation
    - Domain is divided into discrete finite differences
    - Solution is approximated throughout
    - In this case, an iterative approach can be used to obtain a steady-state solution
    - Only nearest neighbour cells are considered in forming the finite difference

- Gravitational N-body, structural mechanics, weather and climate models are other examples

Control decomposition(1)

SEE-GRID
South Eastern European GRid-enabled eInfrastructure Development

- If you cannot find a good domain to decompose, your problem might lend itself to control decomposition
  - Good for:
    - Unpredictable workloads
    - Problems with no convenient static structures
  - One set of control decomposition is functional decomposition
    - Problem is viewed as a set of operations. It is among operations where parallelization is done
    - Many examples in industrial engineering (i.e. modelling an assembly line, a chemical plant, etc.)
    - Many examples in data processing where a series of operations is performed on a continuous stream of data

Control decomposition(2)

SEE-GRID
South Eastern European GRid-enabled eInfrastructure Development

- Control is distributed, usually with some distribution of data structures
- Some processes may be dedicated to achieve better load balance
- Examples
  - Image processing: given a series of raw images, perform a series of transformation that yield a final enhanced image. Solve this in a functional decomposition (each process represents a different function in the problem) using data pipelining
  - Game playing: games feature an irregular search space. One possible move may lead to a rich set of possible subsequent moves to search.
    - Need an approach where work can be dynamically assigned to improve load balancing
    - May need to assign multiple processes to work on a particularly promising lead

Control decomposition(3)

- Any problem that involve search (or computations) whose scope cannot be determined a priori, are candiates for control decomposition
  - Calculations involving multiple levels of recursion (i.e. genetic algorithms, simulated annealing, artificial intelligence)
  - Discrete phenomena in an otherwise regular medium (i.e. modelling localized storms within a weather model)
  - Design-rule checking in micro-electronic circuits
  - Simulation of complex systems
  - Game playing, music composing, etc..

Object-oriented decomposition(1)

- Object-oriented decomposition is really a combination of functional and domain decomposition
  - Rather than thinking about a dividing data or functionality, we look at the objects in the problem
  - The object can be decomposed as a set of data structures plus the procedures that act on those data structures
  - The goal of object-oriented parallel programming is distributed objects
- Although conceptually clear, in practice it can be difficult to achieve good load balancing among the objects without a great deal of fine tuning
  - Works best for fine-grained problems and in environments where having functionally ready at-the-call is more important than worrying about under-worked processors (i.e. battlefield simulation)
  - Message passing is still explicit (no standard C++ compiler automatically parallelizes over objects).

Object-oriented decomposition(2)

SEE-GRID
South Eastern European GRid-enabled eInfrastructure Development

- Example: the client-server model
  - The server is an object that has data associated with it (i.e. a database) and a set of procedures that it performs (i.e. searches for requested data within the database)
  - The client is an object that has data associated with it (i.e. a subset of data that it has requested from the database) and a set of procedures it performs (i.e. some application that massages the data).
  - The server and client can run concurrently on different processors: an object-oriented decomposition of a parallel application
  - In the real-world, this can be large scale when many clients (workstations running applications) access a large central data base – kind of like a distributed supercomputer

Decomposition summary

SEE-GRID
South Eastern European GRid-enabled eInfrastructure Development

- A good decomposition strategy is
  - Key to potential application performance
  - Key to programmability of the solution
- There are many different ways of thinking about decomposition
  - Decomposition models (domain, control, object-oriented, etc.) provide standard templates for thinking about the decomposition of a problem
  - Decomposition should be natural to the problem rather than natural to the computer architecture
  - Communication does no useful work; keep it to a minimum
  - Always wise to see if a library solution already exists for your problem
  - Don't be afraid to use multiple decompositions in a problem if it seems to fit

Communication

- Local Communication
  - constructed from tasks which need data from “small” set of other tasks.
- Global Communication
  - Significant number of tasks must contribute data to perform computation.
- Quality Evaluation
  1. The communication operations are balanced among tasks
  2. Each task communicates with only a small number of neighbours
  3. Tasks can perform their communications concurrently
  4. Tasks can perform their computations concurrently

Agglomeration(1)

SEE-GRID
South Eastern European GRid-enabled eInfrastructure Development

- Group tasks into larger tasks to improve performance or simplify programming.

- Have an architecture in mind (Shared/Distributed memory)
- Combine Primitive Tasks into Large Tasks (usually 1task/proc)
- Map onto Architecture

Agglomeration(2)

SEE-GRID
South Eastern European GRid-enabled eInfrastructure Development

- Agglomeration focuses on:
  - Lower communication costs:
    - Combine (agglomerate) tasks that communicate with each other to increase local communication.
      - try to combine later tasks that depend on data from earlier tasks
      - combine groups of tasks that send, and groups of tasks that receive
  - Retain Scalability
    - Choose combination "direction" that allows scaling (e.g. for 12x128x128 select 128 dimension for parallelization.)
  - Reuses Software
    - Try to reuse complicated serial tasks that are alreadyavailable.
    - Parallel Libraries are available for basic operations.

Agglomeration(3)

SEE-GRID
South Eastern European GRid-enabled eInfrastructure Development

Quality Evaluation

- Agglomeration increases communication locality
- Replicated computation is less than communication cost
- Amount of replicated data allows program to scale. (e.g. for matrix-matrix multiply you cannot have the full matrices on each processors.)
- Tasks should have similar computation and communications costs
- Number of tasks is small as possible.
- Tradeoff between agglomeration and code rewrite is reasonable.

Mapping(1)

SEE-GRID
South Eastern European GRid-enabled eInfrastructure Development

- Assign Tasks to Processors
- Maximize load balance
- Increasing processor utilization and minimizing Inter-Processor Communication (IPC) are often conflicting goals)
- When communication channels are combined onto one processor IPC decreased.
- When communication channels are mapped onto two different processors, IPC increases.

Mapping (2)

- Domain decomposition usually produces balanced tasks
- Often the number and costs of tasks are known, and a Static (predetermined) Mapping before the run can be use to balance the loads.
  - If some tasks costs are similar and others higher (or costs gradually increase), balance via interleaving.
- Use Dynamic Mapping when tasks are create at run time and vary widely in cost.
  - Centralized Task Scheduling (management performed by one task; work performed by pool of workers)
    - For distributed memory (use course grain mechanism)
      - Allow single worker to pre-fetch several tasks
      - Use push/pull mechanism to redistribute work
    - For shared memory (can use fine grain work distribution)

Mapping(3)

SEE-GRID
South Eastern European GRid-enabled eInfrastructure Development

Quality Evaluation

- Selection of static and dynamic task allocation is optimal.
- Dynamic allocation manager is not a bottleneck for work assignment.
- Static allocations that require balancing should have a 10:1 ratio of tasks to processors.

For the program

SEE-GRID
South Eastern European GRid-enabled eInfrastructure Development

- Choose a decomposition
  - Perfectly parallel, domain, control etc.
- Map the decomposition to the processors
  - Ignore topology of the system interconnect
  - Use natural topology of the problem
- Define the inter-process communication protocol
  - Specify the different types of messages which need to be sent
  - See if standard libraries efficiently support the proposed message patterns

Summary of Software

SEE-GRID
South Eastern European GRid-enabled eInfrastructure Development

Software	Notes
Compilers	Moderate *O(4-10)* parallelism Not concerned with portability Platform has a parallelizing compiler
OpenMP	Moderate *O(10)* parallelism Good quality implementation exists on the platform Not scalable
MPI	Scalability and portability are important Needs some type of message passing platform A substantive coding effort is required
PVM	All MPI conditions plus fault tolerance Still provides better functionality in some settings
High Performance Fortran (HPF)	Like OpenMP but new language constructs provide a data-parallel implicit programming model
P-Threads	Not recommended Difficult to correct and maintain programs Not scalable to large number of processors
High level libraries	POOMA and HPC++ Library is available and it addresses a specific problem