Modeling Parallel Programs

Prof. Dr. Antônio Augusto Fröhlich
André Luís Gobbi Sanches
LISHA - UFSC

{guto|gobbi}@lisha.ufsc.br
http://www.lisha.ufsc.br/~{guto|gobbi}
November 2003
Introduction

- Modeling parallel programs
  - Parallel programming relies on creativity
  - A formal methodology eases the development of parallel programs
Chosen Methodology

- Proposed by Ian Foster
- Simplifies Parallel Analysis
- Freely Available Book
- Based on 4 steps
  - Partitioning
  - Communication
  - Agglomeration
  - Mapping
Example

```
G A T A C A
0 -2 -4 -6 -8 -10 -12
C -2 -1 -3 -5 -7 -7 -9
A -4 -3 0 -2 -4 -6 -6
T -6 -5 -2 +1 -1 -3 -5
A -8 -7 -4 -1 +2 0 -2
A -10 -9 -6 -3 0 +1 +1
```
Example
Step 1: Partitioning

- Division of the problem in tasks
- Fine-grained
- Aiming flexibility
- Will be revisited later
- Decomposition
  - Structural
  - Functional
- Define tasks that partition computation and data in disjoint sets
  - We can consider replication of data or computation
Structural Decomposition

- Decompose the data in small pieces of approximately equal size
- Partition the computation to be performed
- Blocks of data and associated computation define tasks
  - Computation between tasks may be necessary
- Different alternative partitions may be possible
Structural Decomposition

- Different decompositions are possible
Functional Decomposition

- Initial focus is on the computation rather than on the data
- Then we examine the data manipulated by the computation
  - If the data is disjoint, the partition is complete
  - Otherwise maybe structural decomposition should be considered
Example

- Network packet preparation

Diagram:

1. Packing
2. Compression
3. Encrypting
Example
Partition Design Checklist

1. Does your partitioning define at least one order of magnitude more tasks than there are processors in your target computer?
2. Does your partitioning avoid redundant computation and storage requirements?
3. Are tasks of comparable size?
4. Does the number of tasks scale with the problem size?
5. Have you identified several partitioning possibilities?
Step 2: Communication

- Tasks execute concurrently but, in most cases, not independently
- Data must be transferred between tasks
- We should be minimalistic with communications
- Easier if decomposition was functional than if it was domain
Classification of Communication

- Local x Global
- Structured x Unstructured
- Static x Dynamic
- Synchronous x Asynchronous
Example
Communication Design Checklist

1. Do all tasks perform the same number of communication operations?
2. Does each task communicate only with a small number of neighbors?
3. Are communication operations able to proceed concurrently?
4. Is the computation associated with different tasks able to proceed concurrently?
Step 3: Agglomeration

- We revisit design decisions made in the partition and in the communication steps
- We move from an abstract analysis to a concrete analysis
- We chose a class of parallel computer as target
- We consider:
  - Agglomeration of tasks
  - Replication of data or computation
Example

(a) 

(b)
Example
Example
Goals of Agglomeration

- Preserve flexibility
- Reduce software engineering costs
- Reduce computation
- Increase the grain of parallelism
Agglomeration Design Checklist

1. Has agglomeration reduced communication costs by increasing locality?
2. Has the replication benefits outweighed the costs?
3. Have you verified that the replication of data don't compromise scalability?
4. Has agglomeration yielded tasks with similar computation and communication costs?
Agglomeration Design Checklist

5. Does the number of tasks still scale with the problem size?
6. If agglomeration eliminated opportunities for concurrent execution, have you verified that there is sufficient concurrency for current and future scalability?
7. Can the number of tasks be reduced still further?
8. Have you considered the cost of modifications to the original code?
Step 4: Mapping

- We specify where each task execute
- Goal: minimize total execution time
- Two strategies:
  - We place tasks that can be executed concurrently on different processors, so as to enhance concurrency
    - We place tasks that communicate frequently on the same processor, so as to increase locality
  - The mapping problem is NP-Complete
Dynamic Load Balancing

- If the number of tasks or the amount of computation or communication changes dynamically, we may need a dynamic load-balancing strategy.
- Not necessary on our example.
- Out of context for this course.