Lecture 8 - GP-GPU and High Performances Computing

Design of Parallel Program

Previously

- Organization of a computation with respect to data and architecture
- Computations should carefully adapt to the architecture and maximize the usage of the resources
- Matrix multiply example
- Reduce pattern

Foster Methodology

Choice Criterion

Three parameters may influence the choice of a kind of parallelism:

- **Flexibility**: support of different programming constraints. Should adapt to different architectures
- **Efficiency**: better scalability
- **Simplicity**: allows to solve complex problem but with a low maintenance cost

For each model of parallelism, we will expose the strengths and weaknesses for each element

Dependency Design Space

The analysis is made on three elements:

Grouping the Data

- Time dependency
- Collection of data
- Independence

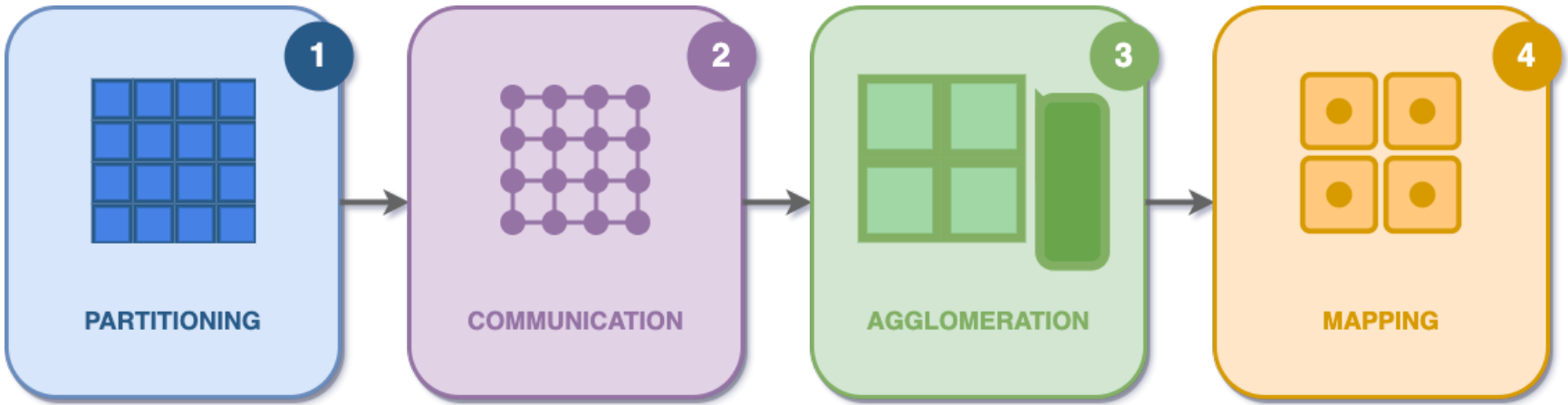
Scheduling

- Identify which data are required for executing a specific task
- Identify the tasks creating the different data

Sharing the Data

- Identify the data shared between tasks
- Manage access to data

Foster Design



Decomposition

Decomposition: Idea

- Identify the elements that allow parallel processing and determine the granularity of the decomposition
- Break up computation into tasks to be divided among processes
  - Tasks may become available dynamically
  - Number of tasks may vary with time
- Enough tasks to keep processors busy: the number of tasks available at a given time is an upper bound on achievable speedup

Decomposition: Focus

How to decompose the code in order to achieve maximum parallelism?

- **Focus on data: Domain Decomposition**
  - Partition data first into elementary blocks of independent data then associate computation tasks with data
- **Focus on computation: Functional Decomposition**
  - Partition computation first then associate data to tasks
- **Often, we use a combination of these decompositions**

Domain Decomposition

- Divide data into pieces of approximately equal size: data granularity
- Partition computation by associating each operation with the data on which it operates
- Set of tasks = (data, operations)

Use case: Problems with large central data structures

Example: Manipulation of 3D data on a grid

Functional Decomposition

- Determine set of disjoint tasks
- Determine data requirements of each task
- If requirements overlap, communication is required

Use case: Problems without central data structures or to different parts of a problem

Decomposition: Checklist

1. The granularity is controlled by the number of available processing units. The more tasks the better
   - ➜ Improves flexibility in the design
2. Limit the number of redundancy in data and computations
   - ➜ Improves scalability for large problems
3. Tasks should be of similar sizes
   - ➜ Improves load balancing
4. Number of tasks should depend on the size of the problem
   - ➜ Improves efficiency
5. All decompositions should be considered
   - ➜ Check for flexibility

Communication

Communication

Describe the flow of information between the tasks:

- **Structure**: relation between producers and consumers
- **Content**: volume of data to exchange

We should:

- Limit the number of communication operations
- Distribute communications among tasks
- Organize communication in such a way that they are concurrent to operations

Conceptual structure of a parallel program

Communication: Structure

Strong impact of decomposition on communication requirements:

- **Functional decomposition**: data flow between the tasks
- **Domain decomposition**: volume of data to perform a computation can be challenging or requires input from several other tasks

Communication: Types

- **Local or global**: small set of tasks or all?
- **Structured or unstructured**: grid or graphs?
- **Static vs dynamic**: known at the start of the program?
- **Synchronous or asynchronous**: cooperative tasks?

Communication: Checklist

1. Load balancing of the communication operations
   - ➜ Improves scalability
2. Small communication pattern
   - ➜ Improves scalability
3. Communications are concurrent to computations
   - ➜ Improves scalability
4. Computations are concurrent to communications in different tasks
   - ➜ Improves scalability

Agglomeration

Agglomeration

After partitioning and communication steps, we have a large number of tasks and a large amount of communication. Need to combine into large blocks:

- **Increase granularity**: reduce communication costs
- **Maintain flexibility**: improve scalability
- **Reduce engineering costs**: increase development overhead

Agglomeration: Checklist

1. Communication costs are reduced
2. Replication of data preserved scalability
3. Replication of computation preserved performances
4. Tasks are load balanced in terms of computation and communication
5. Scalability is preserved

Affectation

Mapping

Where to execute each task?

- Tasks that execute concurrently are placed on different processing units
  - → Increase concurrency
- Tasks that communicate frequently are placed on the same processing units
  - → Increase locality

⚠ **Mapping is NP-complete**

Mapping: Strategies

- **Static mapping**: equal-sized tasks, structured communication
- **Load balancing**: variable amount of work per task or unstructured communication
- **Dynamic load balancing**: variable number of computation and communication per task
- **Task scheduling**: short tasks

Mapping: Checklist

1. Single Program Multiple Data algorithm: consider dynamic task creation
   - ➜ Simpler algorithm
2. Dynamic task creation: consider SPMD algorithm
   - ➜ Greater control over scheduling of computation and communication
3. Centralized load-balancing: verify manager does not become bottleneck
4. Dynamic load-balancing: consider probabilistic/cyclic mappings
5. Probabilistic/cyclic methods: verify that number of tasks is large enough

Examples

1. **Heat equation**: Solve the evolution of the temperature in a rod using a 1D approach. The evolution in its discrete form is given by:

$$u_i^n = ru_{i-1}^{n-1} + (1-2r)u_i^{n-1} + ru_{i+1}^{n-1}$$

2. **Find the maximum** in an array

Conclusion

Conclusion

Design of parallel software:

- **Decomposition**
- **Communication**
- **Agglomeration**
- **Affectation**

Thank You!