Parallel Algorithm Design

Instructor: Tsung-Che Chiang
tcchiang@ieee.org
Department of Computer Science and Information Engineering
National Taiwan Normal University

Outline

- Task/Channel Model
- Foster’s Design Methodology
- Boundary Value Problem
- Finding the Maximum
- The n-body Problem
- Summary
Task/Channel Model

- It represents a parallel computation as a set of tasks that may interact with each other by sending messages through channels.
- A task is a program, its local memory, and a collection of I/O ports.
- A channel is a message queue that connects one task’s output port with another task’s input port.

Figure 3.1 The task/channel programming model. (a) A task consists of a program, local memory, and a collection of I/O ports. (b) A parallel computation can be viewed as a directed graph in which vertices represent tasks and directed edges represent communication channels.
Task/Channel Model

- Send vs. receive
  - Receiving is synchronous, while sending is asynchronous.

- Local memory vs. non-local memory
  - It is easy to distinguish between private data and non-local data over channels, which is good since we should think of local accesses as being much faster.

- Execution time of a parallel algorithm
  - the period of time during which any task is active.

Foster’s Design Methodology

Figure 3.2 Foster’s parallel algorithm design methodology.
To discover as much parallelism as possible, partitioning is the process of dividing the computation and the data into pieces.

- Decompositions
  - domain decomposition (data parallelism)
    - Remember to focus on the largest and/or most frequently accessed data.
  - functional decomposition
    - It often yields collections of tasks that achieve concurrency through pipelining.

- Whichever decomposition we choose, we call each of these pieces a primitive task.
- The number of primitive tasks is an upper bound on the parallelism we can exploit.
### Partitioning

**Foster’s Design Methodology**

- **Three domain decompositions of a 3-D matrix**

![Diagram](image)

By courtesy of M.J. Quinn, *Parallel Programming in C with MPI and OpenMP*, 2003

---

**Concurrency:**

While one task is converting an image from physical coordinates, a second task can be displaying the previous image, and a third task can be tracking instrument positions for the next image.

- Acquire patient images
- Register images
- Determine image locations
- Track position of instruments
- Display image

Functional decomposition of a system supporting interactive image-guided surgery
Partitioning

- Checklist to evaluate the quality of a partitioning:
  - There are at least an order of magnitude more primitive tasks than processors. (potential)
  - Redundant computations and redundant data structure storage are minimized. (scalability)
  - Primitive tasks are roughly the same size. (workload balance)
  - The number of tasks is an increasing function of the problem size. (future potential)

Communication

- Parallel algorithms have two kinds of communication patterns:
  - Local – a task needs values from a small number of other tasks
  - Global – a significant number of tasks must contribute data to perform the computation
    - e.g. computing the sum of values held by the primitive processes
    - It is generally not helpful to draw channels for global communications at this stage.
Communication

- Communication is the overhead of a parallel algorithm, and we want to minimize it.
- Check list:
  - The communication operations are balanced among the tasks.
  - Each task communicates with only a small number of neighbors.
  - Tasks can perform their communications concurrently.
  - Tasks can perform their computations concurrently.

Foster’s Design Methodology

Agglomeration

- In the last two steps of the design process, we have a target architecture in mind, and consider how to combine primitive tasks into larger tasks and map them onto physical processors to reduce the amount of parallel overhead.
- Goal of agglomeration
  - to lower communication overhead
  - to maintain the scalability
  - to reduce software engineering costs
Agglomeration

To lower communication overhead

Combining tasks that are connected by a channel eliminates that communication.

Combining sending and receiving tasks reduces the number of message transmissions.

To maintain the scalability

Suppose we want to develop a parallel program that manipulates a 3D matrix of size $8 \times 128 \times 256$.

If we agglomerate the second and third dimensions, we will not be able to port the program to a parallel computer with more than 8 CPUs.
Agglomeration

- To reduce software engineering costs
  - If we are parallelizing a sequential program, one agglomeration may allow us to make greater use of the existing sequential code, reducing the time and expense of developing the parallel program.

Agglomeration

- Check list
  - The agglomeration has increased the locality of the parallel algorithm.
    - Replicated computations take less time than the communications they replace.
  - The amount of replicated data is small enough to allow the algorithm to scale.
  - Agglomerated tasks have similar computational and communications costs. (workload balance)
Agglomeration

- Check list (continued)
  - The number of tasks is (future potential)
    - an increasing function of the problem size.
    - as small as possible, yet at least as great as the number of processors in the likely target computers.
  - The trade-off between the chosen agglomeration and the cost of modifications to existing sequential code is reasonable. (software engineering)

Mapping

- Mapping is the process of assigning tasks to processors.
- Its goal is to
  - maximize processor utilization and
  - minimize inter-processor communication.

  - Processor utilization is maximized when the computation is balanced evenly.
  - Inter-processor communication increases when two tasks connected by a channel are mapped to different processors.
Mapping

- Mapping 8 tasks to 3 processors
  - The middle processor deals with twice as many tasks and inter-processor communications as the other processors.

Maximizing processor utilization and minimizing inter-processor communication are often conflicting goals.

e.g. Suppose there are $p$ processors. Mapping all tasks to one processor reduces all inter-processor communications, but also reduces the utilization to $1/p$.

There are no known polynomial-time algorithms to map tasks to processors to minimize the execution time. Hence, we must rely on heuristics.
Foster’s Design Methodology

Mapping

- Check list:
  - Designs based on one task per processor and multiple tasks per processor have been considered.
  - Both static and dynamic allocation of tasks to processors have been evaluated.
  - If a dynamic allocation has been chosen, the task manager is not a bottleneck to performance.
  - If a static allocation has been chosen, the ratio of tasks to processors is at least 10:1.

- Static number of tasks
  - Constant comp. time per task
  - Agglomerate tasks to minimize communication. Create one task per processor.

- Dynamic number of tasks
  - Varied comp. time per task
  - Cyclically map tasks to processors to balance computational load.

- Static load balancing algorithm

- Dynamic load balancing algorithm

- Use a run-time task-scheduling algorithm

- No intertask communications.

- Centralized distributed

Parallel Algorithms, Fall, 2008
Boundary Value Problem

- Introduction

\[ T_0 = 100\sin(\pi x) \]

\[ 0^\circ \text{C} \quad \text{and} \quad 0^\circ \text{C} \]

---

Figure 3.9 The rod cools as time progresses. The finite difference method finds the temperature at a fixed number of points in the rod at certain time intervals. Decreasing the size of the steps in space and time can lead to more accurate solutions.
Boundary Value Problem

- Introduction

\[ u_{i,j+1} = ru_{i,j} + (1 - 2r)u_{i,j} + ru_{i+1,j} \]

\[ r = \frac{k}{h^2} \]

Figure 3.10 Data structure used in a finite difference approximation to the rod-cooling problem presented in Figure 3.8. Every point \( u_{i,j} \) represents a matrix element containing the temperature at position \( i \) on the rod at time \( j \). At each end of the rod the temperature is always 0. At time 0, the temperature at point \( x \) is 100 sin(\( \pi x \)).

- Partition

  - There is one data item per grid point. Let’s associate one primitive task with each grid point.
  - This yields a 2-D domain decomposition.
Boundary Value Problem

- Communication

\[ u_{i,j+1} = ru_{i,j} + (1 - 2r)u_{i,j} + ru_{i+1,j} \]

Boundary Value Problem

- Agglomeration
  - Tasks computing rod temperature later in time depend upon the results produced by tasks computing rod temperatures earlier in time.
Boundary Value Problem

- Static number of tasks
  - Structured comm. pattern
  - Constant comp. time per task
  - Agglomerate tasks to minimize communication. Create one task per processor.
- Dynamic number of tasks
  - Unstructured comm. pattern
  - Varied comp. time per task
  - Cyclically map tasks to processors to balance computational load.
- Static load balancing algorithm.
- Use a static load balancing algorithm.
- Use a dynamic load balancing algorithm.
- Use a run-time task-scheduling algorithm.

Mapping
- A good strategy is to associate a contiguous piece of the rod with each task, which preserves the simple nearest-neighbor communication between tasks and eliminates unnecessary communications for those data points within a single task.
Boundary Value Problem

- Analysis – sequential algorithm

\[ u_{i,j+1} = ru_{i,j} + (1 - 2r)u_{i,j} + ru_{i+1,j} \]

\( \chi \): time needed to compute

\[ m(n - 1)\chi \]

- Analysis – parallel algorithm

\[ m \left( \frac{(n - 1)}{p} \right) \chi + m(2\lambda) \]

\( p \): number of processors

\( \lambda \): communication time
Finding the Maximum

- **Reduction**
  - Given a set of \( n \) values, \( a_0, a_1, \ldots, a_{n-1} \) and an associative binary operator \( \otimes \), reduction is the process of computing \( a_0 \otimes a_1 \otimes \ldots \otimes a_{n-1} \).

  Example: addition, minimum, maximum

  - Since reduction requires exactly \( n - 1 \) operations, it has time complexity \( \Theta(n) \).

  How quickly can we perform a reduction on a parallel computer?

---

Finding the Maximum

- **Partitioning**
  - Since the list has \( n \) values, let’s divide it into \( n \) pieces, as finely as possible.
  - If we associate one task per piece, we have \( n \) tasks, each with one value.
  - Our goal is to find the maximum of all \( n \) values.
Finding the Maximum

- Communication
  - Since a task cannot directly access a value stored in the memory of another task, we must set up channels between the tasks.
  - In one communication step, each task may either send or receive one message.
  - At the end of the computation, we want one task (the root task) to have the global maximum.

\( n - 1 \) tasks

\( (n - 1)(\lambda + \chi) \)

\( \chi \) : time to perform max(.)

\( \lambda \) : communication time

\( n/2 - 1 \) tasks

\( (n/2 - 1)(\lambda + \chi) + (\lambda + \chi) \)

\((n/2)(\lambda + \chi)\)
Finding the Maximum

- Communication

- A single message-passing step is sufficient to combine two values into one. Two message-passing steps are sufficient to combine four values into one.

- In general, it is possible to perform a reduction of \( n \) values in \( \log n \) message passing steps.

\[
(n/4 - 1)(\lambda + \chi) + 2(\lambda + \chi) = (n/4 + 1)(\lambda + \chi)
\]
Finding the Maximum

- Communication – An example

<table>
<thead>
<tr>
<th></th>
<th>4</th>
<th>2</th>
<th>0</th>
<th>7</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-3</td>
<td>5</td>
<td>-6</td>
<td>-3</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td>4</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>-4</td>
<td>4</td>
<td>6</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Step 1.

Maximum = 8.

Finding the Maximum

- Communication – An example

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


Maximum = 8.
Finding the Maximum

Communication – What if \( n \) is not a power of 2?

- Suppose \( n = 2^k + r \). In the first step, \( r \) tasks send values to the other \( r \) tasks and then become inactive.

- If \( n \) is not a power of 2, \( \lceil \log n \rceil + 1 \) communications are required.
- In general, \( \lceil \log n \rceil \) communications are required.

Finding the Maximum

Agglomeration

- Static number of tasks
  - Structured comm. pattern
  - Constant comp. time per task
  - Agglomerate tasks to minimize communication. Create one task per processor.

- Dynamic number of tasks
  - Unstructured comm. pattern
  - Varied comp. time per task
  - Cyclically map tasks to processors to balance computational load.

- Static number of tasks
  - Use a static load balancing algorithm.

- Dynamic number of tasks
  - Use a dynamic load balancing algorithm.
  - No intertask communications.

- Use a run-time task-scheduling algorithm.
Finding the Maximum

- Agglomeration & mapping
  - We try to minimize communication by assigning \( n/p \) tasks to each processor.

\[ \text{max} \]

By courtesy of M.J. Quinn, *Parallel Programming in C with MPI and OpenMP*, 2003

Finding the Maximum

- Analysis
  - If \( n \) integers are divided evenly among the \( p \) processors, each processor has no more than \( \lceil n/p \rceil \) integers.
  - Since all processors perform concurrently, the time to compute maximum of their integers is \( (\lceil n/p \rceil - 1)\chi \).
  - Each reduction step requires \( \lambda + \gamma \) to send/receive a message and calculate the maximum.
  - Since there are \( \lceil \log p \rceil \) communications, the overall time of this parallel program is \( (\lceil n/p \rceil - 1)\chi + \lceil \log p \rceil (\lambda + \gamma) \).

By courtesy of M.J. Quinn, *Parallel Programming in C with MPI and OpenMP*, 2003
The n-body Problem

- We are simulating the motion of \( n \) particles of varying mass in two dimensions.
  During each iteration, we need to compute the new position and velocity vector of each particle, given the positions of all the other particles.

![Diagram showing the influence of gravitational forces on a particle](image)

The future position of the pink particle is influenced by the gravitational forces exerted by the other two particles.

The n-body Problem

- Partitioning
  - Let's assume we have one task per particle.
  - In order to compute the new location, it must know the locations of all the other particles.
The n-body Problem

- Communication
  - A gather operation is a global communication that takes a dataset distributed among a group of tasks and collects the items on a single task.

After $n-1$ communications, each task has the position of all the other particles.

Is there a quicker way?

Figure 3.19 One way to make all data values available to all tasks is to set up a channel between every pair of tasks.
The n-body Problem

- Communication – a faster way

A logarithmic number of exchange steps are necessary and sufficient to allow every processor to acquire all the values.

In the $i^{th}$ exchange step, the messages have length $2^{i-1}$.
The n-body Problem

**Agglomeration & Mapping**
- Assume that $n$ is a multiple of $p$.
- We associate one task per processor and agglomerate $n/p$ particles into each task.
- Now the all-gather communication requires $\log p$ communication steps.
- In the $i^{th}$ exchange step, the messages have length $2^{i-1}(n/p)$.

**Analysis**
- In the previous examples, we assumed that it took $\lambda$ units of time to send a message since in those examples the messages always have length 1.
- From now on, time for communication is determined by
  - $\lambda$ (latency): the time needed to initiate a message
  - $\beta$ (bandwidth): the number of data items that can be sent down a channel in one unit of time.

  $\Rightarrow$ Sending a message containing $n$ data items requires time $\lambda + n/\beta$. 
The n-body Problem

- **Analysis**
  - The communication time for each iteration of simulation is
    \[
    \sum_{i=1}^{\log p} \left( \lambda + \frac{2^{i-1}n}{\beta p} \right) = \lambda \log p + \frac{n(p-1)}{\beta p}
    \]
  - Hence, the total time is
    \[
    \lambda \log p + \frac{n(p-1)}{\beta p} + \chi(n/p)
    \]

- **How about adding data I/O?**
  - Let’s assign I/O duties to processor 0.
  - We need to input/output 2-D positions \(p_x\) and \(p_y\) and velocities \(v_x\) and \(v_y\) of \(n\) particles.
  - It takes \(\lambda_{io} + 4n/\beta_{io}\) to send all the values.
The n-body Problem

- After the data are read, we need to assign suitable subsections containing $4n/p$ values to each processor.
- A straightforward way takes $(p-1)(\lambda + 4n/(p \beta)) = (p - 1)\lambda + (p - 1)4n/(p \beta)$ units of time.

```
\begin{align*}
\sum_{i=1}^{\log_p p} (\lambda + 4n \cdot 2^{i-1}) &= \lambda \log p + 4n(p - 1) \\
&= \lambda \log p + 4n(p - 1)/\beta p
\end{align*}
```
The n-body Problem

- Scatter communication time
  - Straightforward way: \((p-1)\lambda + (p-1)4n/(p\beta)\)
  - Smart way: \(\lambda \log p + (p-1)4n/(p\beta)\)

- Note that the data transmission time (the term with \(\beta\)) is identical for the previous two ways.
  - We assume that our task/channel model supports the concurrent transmission of messages from multiple tasks.
  - It is reasonable for commercial systems, but perhaps unreasonable for a network of workstations connected by a hub or any shared medium that supports only a single message at a time.

The n-body Problem

- The expected overall execution time of the parallel computation (suppose \(m\) iterations) is composed of
  - data input and output
  - scattering at the beginning and gather at the end
  - \(m\) (all-gathering + computation)

\[
2(\lambda_{io} + 4n/\beta_{io}) + 2(\lambda \log p + 4n(p-1)/(\beta p)) + m(\lambda \log p + 2n(p-1)/(\beta p) + \chi(n/p)(n-1))
\]
Summary

- Task/channel model for parallel computation
- Foster’s design methodology
  - partitioning, communication, agglomeration, mapping
  - Goal
    - Maximize processor utilization
    - Minimize inter-processor communications
- Common operations
  - reduction
  - all-gather
  - gather/scatter