Oliver Hoffman and Faiyam Rahman
CS 5460: Parallel and Distributed
Lab 4, Report
April 28, 2015

Part 1.A

Assume a computation domain of size NxN, a stencil of size $(2K-1)x(2K-1)$ as well as a ghost boundary parallelization method. Assume we break the computation domain into subdomains of size XY, where $X|N$ and $Y|N$. We want to find the pair (X, Y) such that the ratio of computation to communication per 1 iteration is optimal. We assert that the ratio of computation to communication is:

$$\frac{\text{Computation}}{\text{Communication}} = \frac{N^2 * (2K-1)^2 * t_c}{\sum_{\mathbb{A}} (\alpha_x + \alpha_y + \alpha_x\alpha_y)t_s + (\alpha_y(K-1)Y + \alpha_x(K-1)X + \alpha_x\alpha_y(K-1)^2)t_u}$$

Let's look at the numerator first. The computation for one iteration is as follows

$$\begin{aligned}\text{Computation} &= \text{Number of Computational Units} * \text{Computation per unit} * \text{Time/Computation Unit} \\ &= N^2 * (2K-1)^2 * t_c\end{aligned}$$

Let's look at the denominator. Let's first establish what all the notation means.

$$\begin{aligned}\mathbb{A} &:= \{\alpha | \alpha \text{ is a computational subdomain of size } XY\} \\ \alpha_x &:= \text{the number of adjacent subdomains on the x axis for subdomain } \alpha \\ \alpha_y &:= \text{the number of adjacent subdomains on the y axis for subdomain } \alpha \\ t_s &:= \text{the latency time to send one message} \\ t_u &:= \text{the amount of time to send one computatational unit}\end{aligned}$$

Now let's breakdown the specifics of the denominator.

$$\text{Communication} = \sum_{\text{All Subdomains}} M_s * t_s + U_s * t_u$$

where M_s is the number of messages sent for the given subdomain and U_s is the number of computational units sent over all messages for the given subdmain. We have:

$$\begin{aligned}M_s = &\text{ Messages sent to adjacent subdomains purely on the x axis+} \\ &\text{ Messages sent to adjacent subdomains purely on the y axis+} \\ &\text{ Messages sent to adjacent diagonal subdomains} \\ = &\ \alpha_x + \alpha_y + \alpha_x\alpha_y\end{aligned}$$

Note that the number of messages sent to diagonally adjacent subdomains is $\alpha_x\alpha_y$ because its the product of the number of x axis and y axis adjacent subdomains.

$$\begin{aligned}U_s = &\text{ Number of units sent to adjacent subdomains purely on the x axis+} \\ &\text{ Number of units sent to adjacent subdomains purely on the y axis+} \\ &\text{ Number of units sent to adjacent diagonal subdomains}\end{aligned}$$

We have

$$\text{Number of units sent to adjacent subdomains purely on the x axis} = \alpha_x * (K-1) * Y$$

because each such subdomain needs $K-1$ columns from α and α's columns are of size Y. Similarly, we have

$$\text{Number of units sent to adjacent subdomains purely on the y axis} = \alpha_y * (K-1) * X$$

because each such subdomain needs $K-1$ rows from α and α's rows are of size X. Lastly, we have

$$\text{Number of units sent to adjacent diagonal subdomains} = \alpha_y \alpha_x * (K-1)^2$$

because the number of computational units α needs to send is determined by how much of α's computational domain is required for its adjacent subdomains corner computation. For that corner computation, the adjacent subdomain needs $(K-1)$ columns of height $(K-1)$, or equivalently, $(K-1)$ rows of width $(K-1)$ from α.

Now, let's consider how to compute this sum for a given N, X, Y. We have

$$\mathbb{A} = \bigcup_{x,y \in \{0,1,2\}} \mathbb{A}_{xy} \text{ where}$$
$$\mathbb{A}_{xy} = \{\alpha | \alpha_x = x \text{ and } \alpha_y = y\}$$

We now present without further justification formulas for each of the $\mathbb{A}_{xy}$ as they are pretty obvious once you look at an example or two.

$$|\mathbb{A}_{00}| = \begin{cases} 1, & \text{if } N = X, N = Y \\ 0, & \text{otherwise} \end{cases}$$

$$|\mathbb{A}_{01}| = \begin{cases} 1, & \text{if } \frac{N}{Y} = 2, N = X \\ 0, & \text{otherwise} \end{cases}$$

$$|\mathbb{A}_{02}| = \begin{cases} 0, & \text{if } N > X, \frac{N}{Y} < 3 \\ \frac{N^2}{XY} - \frac{2N}{X}, & \text{otherwise} \end{cases}$$

$$|\mathbb{A}_{10}| = \begin{cases} 1, & \text{if } \frac{N}{X} = 2, N = Y \\ `0, & \text{otherwise} \end{cases}$$

$$|\mathbb{A}_{20}| = \begin{cases} 0, & \text{if } N > Y, \frac{N}{X} < 3 \\ \frac{N^2}{XY} - \frac{2N}{Y}, & \text{otherwise} \end{cases}$$

$$|\mathbb{A}_{11}| = \begin{cases} 0, & \text{if } \frac{N^2}{XY} < 4 \\ 4, & \text{otherwise} \end{cases}$$

$$|\mathbb{A}_{12}| = \begin{cases} 0, & \text{if } \frac{N}{X} < 2, \text{ or } \frac{N}{Y} < 3 \\ \frac{2N}{Y} - 4, & \text{otherwise} \end{cases}$$

$$|\mathbb{A}_{21}| = \begin{cases} 0, & \text{if } \frac{N}{Y} < 2, \text{ or } \frac{N}{X} < 3 \\ \frac{2N}{X} - 4, & \text{otherwise} \end{cases}$$

$$|\mathbb{A}_{22}| = \begin{cases} 0, & \text{if } \frac{N}{X} < 3 \text{ or } \frac{N}{Y} < 3 \\ \frac{N^2}{XY} - \frac{2N}{X} - \frac{2N}{Y} + 4, & \text{otherwise} \end{cases}$$

Part 1.B

We came up with a model for communication cost in a message-passing machine based on latency and throughput. Develop a model for the communication cost in a shared memory machine, and explain how your model can be used to estimate performance.

For the case of **message-passing**, we saw that the communication cost to send a message could be summarize as:

$$\text{comm} = \text{t}_s + m * \text{t}_w \tag{1}$$

where:

- t_s is the time required to prepare a message being sent both at the sending and receiving processors. This includes the time to prepare the message (adding header and error correction information), the time to execute the routing algorithm based on the type of network, and time to establish an interface for communication.
- t_w is the time to transfer a word
- m is the message size

Developing a communication cost model for **shared memory** machines is much more complicated due to a variety of reasons:

- Memory layout differs across systems: giving minimal control over where specific data items are located. In case of distributed systems, this is particularly important because it is difficult to know if the data item is a local or remote shared address space (and the time to access each varies).
- Cache also raises an issue, since depending on the program, a data item stored in a processor's cache may be needed by another processor. This adds an extra overhead when accessing such data item.
- Different data items will have different access costs: based on what was explained above, depending on the origin and use of a data item, its location may affect the communication cost. Moreover, since cache lines are generally longer than 1 word, different words might have different latencies associated with them based on whether they are in the same cache line or not.

Having said this, we base our model on the following:

- As an initial idea, it is easy to see that accessing a remote word results in a cache line being fetched into local cache. The time associated with this includes coherence overheads, network overheads and memory overheads. While all these overheads will vary much depending on what data item is accessed, we assume it is constant and call it t_s. Based on what we have read on literature, it is safe to assume that this constant cost of t_s is associated with gaining access to a contiguous chunk of m words of shared data, even if m is greater than the cache line size.

- Moreover, we also assume that accessing shared data is costlier than accessing local data, and so we assign a per-word access cost of t_w to shared data.

Based on this, our model is equivalent to the message passing one:

$$\text{comm} = t_s + m * t_w \tag{2}$$

to show the communication cost of sharing a single chunk of m words between a pair of processors.

- The difference in this case is that t_s is likely to be much smaller than in the message passing case. Note that this model is based on read-only access and between 2 processors. If multiple processes access the same data, then, just like in the message passing paradigm (where one process sends many others the message), the cost is multiplied by the number of processes. If the access is read write, then the cost will be incurred again by subsequent access from more processors (just like in the case of message passing, where a message can be sent back and forth).

- Having this generalized model based on p being the number of processors can be very helpful when analyzing costs as it is generalizable to both shared memory and message passing.