Performance Modeling of Image Smoother

1 An iterative smoother

Image smoothing is a core operation in image processing. We’ll look at an iterative variant that progressively smoothes the image, replacing each pixel by the average of its nearest neighbors on the (4) Manhattan directions. We won’t smooth the boundary points, which are located along x-coordinates of 0 and \( N + 1 \) and along y-coordinates of 0 and \( N + 1 \), as shown in Figure 1.

![Figure 1: An image (left) and the accompanying stencil (right). The Grid \( U \) is defined over the 2-dimensional region with lower bound \((0, 0)\) and upper bound \((N+1, N+1)\). The image smoothing occurs on the inner portion of the mesh; the boundaries are preserved. This boundary region is one cell thick, as dictated by the stencil for the smoother, which in our case updates each cell as a function of its nearest neighbors along the Manhattan directions.](image)

Given an \((N + 2) \times (N + 2)\) image mesh \( U \), we compute a new mesh \( U^{\text{new}} \) such that each value in the solution is replaced by the average of the 4 nearest neighbors in the "old" mesh. We then set the old mesh equal to the new one, and repeat the calculation until we are satisfied with the result. The algorithm is quite simple:

\[
\text{repeat}
\begin{align*}
&\forall (1 \leq i, j \leq N) \\
&u_{i,j}^{(s)} = \frac{u_{i-1,j}^{(s-1)} + u_{i+1,j}^{(s-1)} + u_{i,j-1}^{(s-1)} + u_{i,j+1}^{(s-1)}}{4} \\
\end{align*}
\text{endfor}
\text{until done}
\]

where \( u_{i,j}^{(s+1)} \) is the value of the mesh at \((i, j)\) at iteration \( s + 1 \).
2 Factors Affecting Performance

The way we partition the data affects performance, along with the ratio of $N$ to $P$. We’ll derive a simple analytic performance model than can help us predict the effect of varying these two parameters, along with the partitioning strategy. This model ignores some important effects, however, like the memory hierarchy (e.g. cache) and contention on the interprocessor communication, so it gives us only a rough estimate.

We consider two types of partitioning, as shown in Fig. 2 and Fig. 3: “strips,” a one-dimensional scheme in which the cuts applied to the mesh extend completely across one extent of the mesh only, and “boxes,” in which the cuts extend completely across both extents and the geometry of the boxes is square (We leave as an exercise the general case when the boxes are not square.) In the case of strips, the partitions may be either vertical or horizontal.

In choosing $N$ and $P$ we’ll assume that work divides evenly among the processors, and that the extents of the local arrays are all greater than 2, and that there are no serial bottlenecks. For the two different types of partitionings we are considering, these assumptions imply that

- For strips: $P$ divides $N$ evenly, $\frac{N}{P} \geq 2$
- For boxes: $\sqrt{P}$ divides $N$ evenly, $\frac{N}{\sqrt{P}} \geq 2$

3 The Performance Model

The analytic model predicts two timings:

- $T(1, N)$, the running time of the best serial algorithm,
- $T(P, N)$, the running time of the computation on $P$ processors.

The first parameter used by our model is the grind time. The grind time is designated as $T_\gamma(P, N)$, and is the time taken to update a single point averaged over all updates. That is, we
Figure 3: (a) Box partitioning of the mesh showing (b) the location of the ghost cells.

divide the running time by \( N^2 \times N_{iter} \), where \( N_{iter} \) is the number of iterations:

\[
T_\gamma(P, N) = \frac{T(P, N)}{N^2 \times N_{iter}}
\]  

(1)

Generally the grind time (and running time) is sensitive to \( P \) and \( N \). In practice the compiler may render the grind time almost insensitive to these parameters, through a technique called blocking for cache. Let us designate \( \gamma(N) \) as the grind time on a single processor, that is \( T_\gamma(1, N) \). By definition,

\[
T(1, N) = \gamma(N)N^2N_{iter}
\]

(2)

To treat the parallel case we need a more general form of the grind time which makes sense for problems that aren’t square. This general form, \( \gamma(m, n) \), gives the grind time for a problem of size \( m \times n \). (It follows that \( \gamma(N) = \gamma(N, N) \).) In this case, the running time on a single processor is expressed as:

\[
T(1, (m, n)) = \gamma(m, n)(mn)N_{iter}
\]

(3)

With this definition in hand, we characterize \( T_P \) as follows:

\[
T(P, N) = T(1, (m, n)) + T_{local} + f \times (T_{global})
\]

(4)

where

- the \( N \times N \) problem is divided into partitions of size \( m \times n \);
- \( T_{local} \) is the time spent communicating boundary data (the boundary regions are shown in Fig. 2 and Fig. 3);
In general

\[ T(1,(m,n)) \leq \frac{T(1,N)}{P} \]  

(5)

with equality holding when the compiler is able to handle memory locality appropriately. The above inequality implies that in partitioning the problem we are is likely to improve memory locality. We can improve the running time on a single processor by simulating parallel execution, serially executing the smoother over the subsets of the problem corresponding to the partitions in the true parallel program.

4 Communication Costs

Let the message passing time be given by the relation

\[ T_{\text{message}}(m) = \alpha + m\beta \]  

(6)

where \( \alpha \) is the message startup time, \( \beta \) is the inverse peak message bandwidth, and \( m \) is measured in words, and each pixel is a single word (double precision in our case). We may measure \( \alpha \) and \( \beta \) using the Ring program, which was discussed in class.\(^1\)

We’ll use the \( \alpha-\beta \) formula to model performance of the communication required to fill the ghost cells prior to the start of each iteration.

4.1 Strip Decomposition

For the strip decomposition of Fig. 2, each processor computes a chunk of \( \frac{N}{P} \) complete rows of data. (We assumed that the mesh will be split evenly.) Thus, each processor sends and receives 2 sets of ghost cell data, coming from the two nearest neighboring processors. Each set of data contains \( N \) words. Thus, the local cost of communication for strips is:

\[ T_{\text{comm}}^{\text{local}}(\text{STRIPS}) = 2(\alpha + \beta N) \]  

(7)

4.2 Box Decomposition

For the box decomposition of Fig. 3, each processor computes a square chunk of data \( \frac{N}{\sqrt{P}} \) on a side (assuming that \( P \) is a perfect square, and \( \sqrt{P} \) divides \( N \) evenly.) Each processor sends and receives four sets of ghost cell data, coming from the four nearest neighboring processors on the Manhattan directions (why aren’t the corners needed?). Thus, the local cost of communication for boxes is:

\[ T_{\text{comm}}^{\text{local}}(\text{BOXES}) = 4(\alpha + \beta \frac{N}{\sqrt{P}}) \]  

(8)

\(^1\)As we’ll see later on, the Ring benchmark will predict an optimistic value for \( \beta \). In practice, the actual peak bandwidth may be much lower. We will return to this later on.
5 Putting It All Together

To estimate the running time of our program, we determine $\alpha$ and $\beta$, and then measure $\gamma(m, n)$. To measure $\gamma(m, n)$ we must first determine the values of $m$ and $n$. For strip partitioning we have two possibilities: $(m, n) = (1, N)$ or $(m, n) = (N, 1)$ depending on whether we have horizontal or vertical strips. For box partitioning $m = n = \sqrt{P}$. We then run the best serial program on a problem of size $m \times n$ and use this timing to arrive at the appropriate grind time.

From our analysis we observe that communication with strips is $O(N)$ while communication with boxes is just $O(\frac{N}{\sqrt{P}})$. However, asymptotic analysis doesn’t tell us the full picture since it ignores the the startup time $\alpha$, which significantly affects performance. Let’s ask the following question:

Under what conditions will the strip decomposition lead to a shorter running time than the box decomposition?

To answer this question we need consider the local communication time only (Eq. 7 and Eq. 8). A strip decomposition will lead to a faster running program when:

$$2(\alpha + \beta N) < 4(\alpha + \beta \frac{N}{\sqrt{P}})$$  \hspace{1cm} (9)

or (simplifying, assuming $P \geq 2$):

$$N < \frac{\sqrt{P}}{\sqrt{P} - 2\beta} \frac{\alpha}{\sqrt{\beta}}$$  \hspace{1cm} (10)

For example, on $P = 16$ processors of the T3E, we require

$$N < \frac{2\alpha}{\beta}$$  \hspace{1cm} (11)

But $\frac{\alpha}{\beta} = \frac{20\text{us}}{0.004\text{ms}} = 5000$ and thus strips are faster when $N < 10000$. In practice it is unlikely that $N \geq 10000$ (we would want to use a more efficient solver like multigrid), and so we generally won’t employ box decomposition contrary to what was predicted by asymptotic analysis.

6 Refinements to the Model

6.1 Memory strides

Earlier we mentioned that the value of $\beta$ reported by the Ring program may be inaccurate. We now look at refinements to the communication performance model that remove some of the inaccuracy.

The value reported by Ring for peak inverse bandwidth $\beta$ may actually be overstated by the Ring program in some cases. This can happen when the cost of sending a message also includes the cost of copying the bytes from the user’s data structure into the network interface’s memory buffers, or even into an intermediate user data structure. The cost of this copying depends on the stride of the memory copy, where the stride is the distance between adjacent elements that are transferred.
In the ring program, the stride is 1. However, in our parallel programs we may pass columns of a 2D array between processors, and these accesses have a stride of $N + 2$ (remembering that for a problem of size $N$ we need an $(N + 2) \times (N + 2)$ mesh), the linear dimension of the mesh. If $N$ is sufficiently large, that is a row of the mesh is larger than the cache line size, then successive memory accesses along a column will not be able to exploit spatial locality in the cache. As a result, the bandwidth to memory will be reduced. Moreover, if we have to first pack the data into an intermediate user buffer, because we use a message passing primitive which assumes that data lies contiguously in memory, then the message passing costs increase still further.

In effect we need come up with three different values of $\beta$: $\beta_x$, $\beta_y$, and $\beta_z$, depending on whether the data to be transmitted come from a cut made along the $x$, $y$, or $z$ axis, respectively. We may also need to add a term that is proportional to the number of cache lines actually touched.