Parallelization Techniques for Spatial-Temporal Occupancy Maps from Multiple Video Streams

Nathan DeBardeleben, Adam Hoover, William Jones and Walter Ligon

Parallel Architecture Research Laboratory
Clemson University
{ndebard, ahoover, wjones, walt}@parl.clemson.edu

1 Introduction

We describe and analyze several techniques to parallelize a novel algorithm that fuses intensity data from multiple video cameras to create a spatial-temporal occupancy map. Instead of tracking objects, the algorithm operates by recognizing freespace. The brevity of operations in the algorithm allows a dense spatial occupancy map to be temporally computed at real-time video rates. Since each input image pixel is processed independently, we demonstrate parallel implementations that achieve nearly ideal speedup on a four processor shared memory architecture. Potential applications include surveillance, robotics, virtual reality, and manufacturing environments.

2 Distributed Sensing

For this work, a network of video cameras resembling a security video network is assumed. The cameras are all connected to a single computer that processes the video feeds to produce a spatial-temporal occupancy map [1]. The occupancy map is a two-dimensional raster image, uniformly distributed in the floorplane. Each map pixel contains a binary value, signifying whether the designated floorspace is empty or occupied. Figure 1 shows an example occupancy map where grey cells indicate the space is occupied and white cells indicate the space is empty. A spatial frame of the occupancy map is computed from a set of intensity images, one per camera, captured simultaneously. Temporally, a new map frame can be computed on each new video frame sync signal. Thus in effect, the map is itself a video signal, where the pixel values denote spatial-temporal occupancy. A previous implementation of such a network has shown that a frame rate of 5 Hz is feasible [1]. Our goal is to improve the temporal resolution by providing a frame rate approaching 30 Hz through the use of a parallelized implementation of the algorithm.

3 Algorithms

All the calculations necessary to create the mapping from the *camera space* to the *occupancy map space* are independent of image content. Therefore it can be

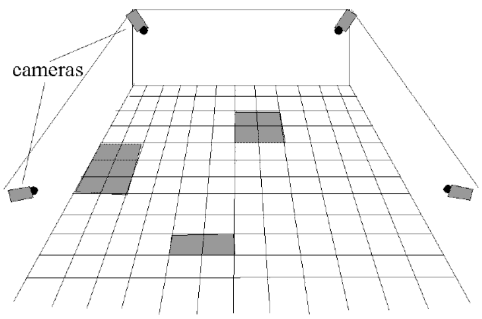

Fig. 1. A spatial occupancy map.

computed off-line and stored as a look-up table. The mapping provides a two-way relation, so that it may be applied in two different manners. The look-up table $L_1[n, c, r]$ relates each image pixel for each camera to a unique occupancy map cell. The look-up table $L_2[x, y]$ relates each occupancy map cell to a set of image pixels, where each set may include any number of pixels (including zero) from each camera. The use of $L_1[n, c, r]$ and $L_2[x, y]$ lead to different algorithms, which we will refer to as image-based and map-based.

Both the image-based and map-based algorithms show great potential for parallelism on a multiprocessor architecture. We describe three different divisions of the processing workload, and the corresponding parallel algorithms. We measure the performance of all the algorithms in Section 4, in terms of speed of execution.

In the following descriptions we maintain the following notation: $O[x, y]$ is the occupancy map, $I[n, c, r]$ is a set of live images from N cameras, and $B[n, c, r]$ is a set of background images acquired during system initialization. The indices x and y refer to map coordinates, c and r refer to image coordinates, and n refers to camera number. $L_1[n, c, r]$ and $L_2[x, y]$ refer to look-up tables storing the mappings described by $\mathcal{F}$ (Equation 1). The threshold T controls the sensitivity of the algorithm, i.e. as the threshold decreases, the system becomes more sensitive to denoting space as occupied. This is demonstrated and discussed further in Section 4.

$$\mathcal{F} : I[n, c, r] \leftrightarrow O[x, y] \tag{1}$$

The arrays $O[x, y]$, $I[n, c, r]$, $B[n, c, r]$, $L_1[n, c, r]$ and $L_2[x, y]$ are multi-dimensional, yet they can be accessed in one-dimensional order because they have discrete boundaries. For the sake of clarity, in the following algorithm descriptions we maintain the multi-dimensional notation. However, loops on (x, y), on (c, r), and on (n, c, r), can be written using a single-index loop. This reduction in loop overhead yields faster executions.

3.1 Image-based

The image-based algorithm uses the look-up table $L_1[n, c, r]$, and is described by the following pseudo-code:

```
loop ... time ...
  loop x = 0 ... map columns
    loop y = 0 ... map rows
      O[x,y] = 1
    end loop
  end loop
  loop n = 0 ... number of cameras
    loop c = 0 ... image columns
      loop r = 0 ... image rows
        if (|I[n,c,r]-B[n,c,r]| < T)
          O[L1[n,c,r]] = 0
        end if
      end loop
    end loop
  end loop
end loop
```

The arrays $I[n, c, r]$, $B[n, c, r]$, and $L_1[n, c, r]$ are accessed in sequential order, which can be exploited by a cache memory. The array $O[x, y]$ is accessed in non-sequential order.

Entries in $L_1[n, c, r]$ that are unused (entries for image pixels which do not map to ground plane points) are given a sentinel value that points to a harmless memory location outside the occupancy map. For instance, the occupancy map array is allocated as $X \times Y + 1$ cells, and the address of the extra cell becomes the sentinel. An alternative is to add a second conditional statement testing a mask. For each camera, a mask is initially generated that distinguishes available floorspace from non-floorspace. In the code given above, the inner-most loop is modified as follows to test for occupation only if the mask states that this space is floor.

```
if (M[n,c,r] == 0)
  if (|I[n,c,r]-B[n,c,r]| < T)
    O[L1[n,c,r]] = 0
  end if
end if
```

In this case an extra conditional statement is executed for every pixel, whereas in the original code non-useful assignment statements may be executed for some pixels. The relative performance of these variations is described in Section 4.

3.2 Map-based

The map-based algorithm uses the look-up table $L_2[x, y]$. Entries in $L_2[x, y]$ are sets of image pixel identities. The size of each set varies depending on how many image pixels view the occupancy map cell. This detail can be simplified by placing a maximum on set size, so that $L_2[x, y]$ may be implemented as a three-dimensional array. The constant set size S is selected so that at least 95% of the mappings in Equation 1 may be found in $L_2[x, y, s]$. Once the pixel has been identified as unoccupied, the algorithm need not further traverse $L_2[x, y, s]$ in the s dimension. This is a form of short-circuit evaluation. The map-based algorithm is described by the following pseudo-code:

```
loop ... time ...
  loop x = 0 ... map columns
    loop y = 0 ... map rows
      O[x,y] = 1
      loop s = 0 ... S
        if (|I[L2[x,y,s]]-B[L2[x,y,s]]| < T)
          O[x,y] = 0
          exit loop s
        end if
      end loop
    end loop
  end loop
end loop
```

In the map-based algorithm, the arrays $L_2[x, y, s]$ and $O[x, y]$ are accessed in sequential order, while the arrays $I[n, c, r]$ and $B[n, c, r]$ are accessed in non-sequential order.

As with the image-based algorithm, unused entries in $L_2[x, y, s]$ may be handled using sentinel addressing or masking. The sentinel version of the code is shown above. In this case entries in $L_2[x, y, s]$ which do not map to image pixels are given a sentinel value that points to memory locations outside the image and background image spaces that cause the conditional statement to fail.

3.3 Image-level parallelism

The image-based algorithm can be split into equal numbers of iterations on the camera loop. In this case, given P processors and N cameras, each processor works on the images provided by $\frac{N}{P}$ cameras. Figure 2 illustrates the workload. In the pseudo-code for the image-based algorithm given above, the camera loop is modified as follows:

```
loop n = (N/P)p ... (N/P)(p+1)
```

where $0 \leq p < P$ identifies a particular processor. This algorithm provides contiguous blocks of memory for the live and background images to each processor, but requires $\frac{N}{P}$ to be an integral number in order to maintain a balanced workload. This algorithm also produces write hazards, because multiple processors may write to the same occupancy map cell at the same time.

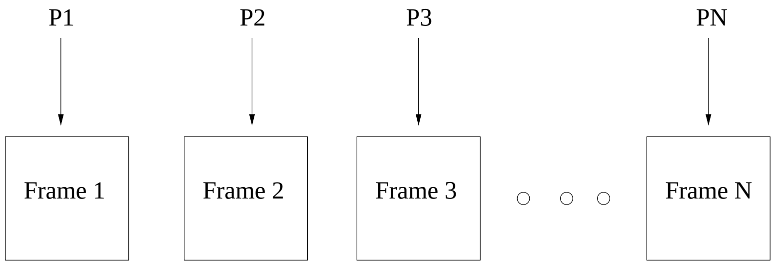

Fig. 2. The processor workload using image-level parallelism.

3.4 Pixel-level parallelism

The image-based algorithm can be split into equal numbers of iterations on the image pixels. In this case, given P processors and N cameras producing $R \times C$ size images, each processor works on $\frac{RC}{P}$ pixels of each image. Figure 3 illustrates the workload. In the pseudo-code for the image-based algorithm given above, the

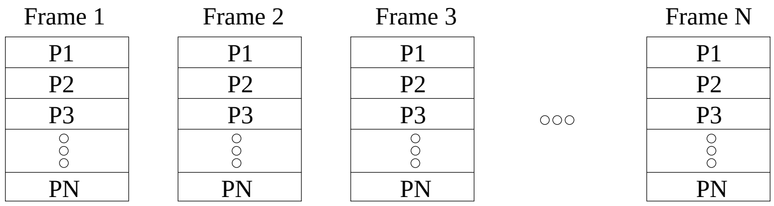

Fig. 3. The processor workload using pixel-level parallelism.

image rows loop is modified as follows:

```
loop r = (R/P)p ... (R/P)(p+1)
```

where $0 \leq p < P$ identifies a particular processor. This algorithm does not provide contiguous blocks of memory for the live and background images to each processor, but maintains a more balanced workload in the case $\frac{N}{P}$ is not an integral number. This algorithm also produces write hazards, because multiple processors may write to the same occupancy map cell at the same time.

3.5 Map-level parallelism

The map-based algorithm can be split into equal numbers of iterations on the map cells. In this case, given P processors and an $X \times Y$ size occupancy map, each processor works on all the image data for $\frac{XY}{P}$ cells. Figure 4 illustrates the workload. In the pseudo-code for the map-based algorithm given above, the map rows loop is modified as follows:

```
loop y = (Y/P)p ... (Y/P)(p+1)
```

where $0 \leq p < P$ identifies a particular processor. This algorithm has no write hazards, because only one processor may write to each map cell. However, the workload balance is directly related to the uniformity of distribution of mappings in $L2[x, y, s]$. If some areas of the map are scarcely covered by image data while other areas are densely covered, then the workload will be correspondingly unbalanced.

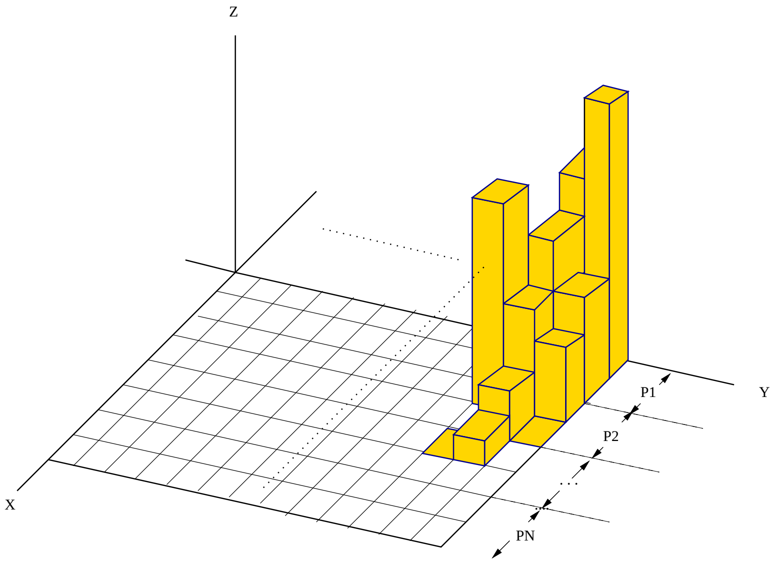

Fig. 4. The processor workload using map-level parallelism.

4 Results

The frame rate of our system depends on the number of cameras, the size of the camera images, the size of the occupancy map, and the algorithm and computer architecture. The frame rate is also upper-bounded by the frame rate of the cameras. In our case, we are using NTSC cameras (video signals), which fixes the camera image size to 640×480 and upper-bounds the frame rate at 30 Hz. We are using an NTSC signal to output the map, fixing the map size to 640×480. The remaining variables are the number of cameras, and the algorithm and computer architecture.

Fixing the number of cameras at four, we examined the performance of the sequential and parallel algorithms on a multi-processor architecture. Simulations were conducted on a Sun HPC 450 with four UltraSparc II processors operating at 300 MHz. A set of real look-up tables used in the sequential prototype were re-used for these experiments. Live images were simulated using a set of randomly valued arrays. The images were replaced on each iteration of the time-loop, to simulate real system operation, so that the 1 MB cache on each processor would have to re-load. Figure 5 plots the frame rates of each algorithm as a function of the threshold T, which is varied across the reasonable range of operation.

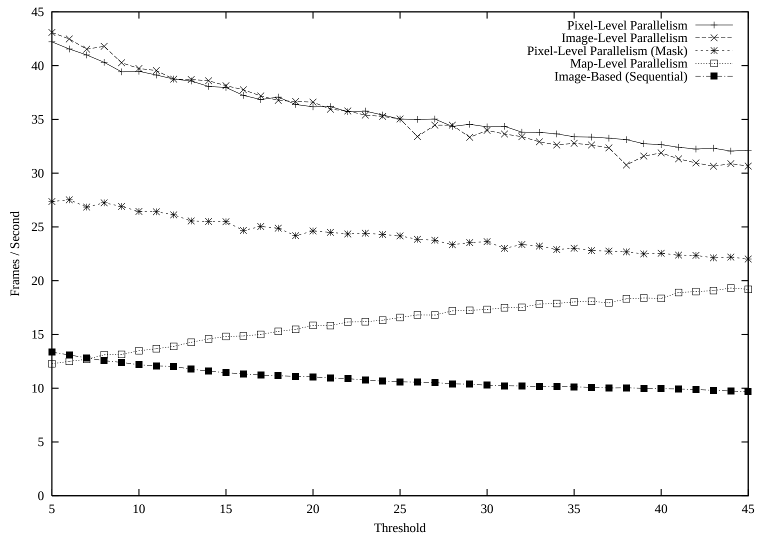

Fig. 5. System throughput of algorithms on multiprocessor architecture.

Based on Figure 5, we observe six results:

1. Both the map-based and image-based parallel algorithms achieved almost linear speedup in the number of processors compared to the sequential algorithms. For instance, between thresholds of 5 and 35, the pixel-level parallel algorithm showed the best average speedup of 3.3 over the image-based sequential algorithm (the theoretical maximum is 4.0, the number of processors).
2. As in our prototype system, the simulations showed a greater performance for the image-based algorithms compared to the map-based algorithms (we show only the fastest map-based algorithm in Figure 5). We suppose this is due to the fact that three out of the four arrays are accessed in sequential order in the image-based algorithms (see Section 3.1), while only two out of four arrays are accessed in sequential order in the map-based algorithms (see Section 3.2). The benefit provided by the increased hit rate in the cache memory (in the image-based parallel algorithms) outweighs the benefit provided by the avoidance of write hazards (in the map-based parallel algorithm).

3. Both the image-based parallel algorithms (pixel-level and image-level) performed equally. This suggests that the small penalty incurred by having a few (in our case four) non contiguous blocks of memory for each processor is relatively insignificant (see Section 3.4). Therefore the pixel-level algorithm is to be preferred, specifically in cases where the number of cameras is not an integral multiple of the number of processors.
4. Using an image mask decreased performance, as compared to sentinel (out-of-map or out-of-images) addressing for unused lookup table entries. The execution of an extra conditional statement for every pixel, along with the cost of loading an additional large array into memory, was more costly than executing the relatively small number of superfluous assignment statements.
5. The performance of each of the algorithms appears to degrade as the threshold increases, with the exception of the map-level algorithm. The map-level algorithm provides a short-circuit mechanism in the inner-most loop as discussed in Section 3.2 while the image-based algorithms do not.
6. It should be noted that, while using simulated I/O, frame rates exceeding the NTSC upper-bound of 30 Hz are indicative of being able to process incoming data at a rate faster than it becomes available. In a physical implementation, this would translate into one or more of the processors being idle waiting for the next frame to arrive from the video capture device.

The sequential prototype described above was constructed in 1997. The multiprocessor hardware described above was constructed in 1998. In 1999, we are constructing a second prototype using a Dell workstation with two Intel processors operating at 450 MHz. Based on the above experiments, we expect this system to operate at approximately 20 Hz. Based on projections of computer architecture performance [2], we expect that an average computer will be able to operate our system at 30 Hz for twenty cameras in the year 2004.

5 Conclusion

We describe and analyze several techniques to parallelize a novel algorithm that fuses intensity data from multiple video cameras to create a spatial-temporal occupancy map. This work provides a foundation to explore distributed sensing on a much larger scale. Future work will include increasing both the number of input data streams as well as the size of the output occupancy map to provide enhanced spatial resolution and coverage.

References

1. A. Hoover and B. Olsen, "A Real-Time Occupancy Map from Multiple Video Streams", in *IEEE ICRA*, 1999, pp. 2261-2266.
2. D. Patterson and J. Hennessy, Computer Architecture: A Quantitative Approach, second edition, Morgan Kaufmann, 1996.