

Pipelined Execution of a Parallel Particle Filter for Real-time Feature Selection and Classification in Data Streams

OLE-CHRISTOFFER GRANMO
Department of ICT
Agder University College
Grooseveien 36, NO-4876 Grimstad
NORWAY

ole.granmo@hia.no <http://home.hia.no/~oleg>

Abstract: - Feature extraction, followed by pattern classification, is a promising approach to automatic recognition of events in data streams such as video. However, feature extraction/classification must often be *pipelined* and *parallelized* to meet recognition rate and response time requirements in *real-time* applications. On the other hand, single-CPU Bayesian network systems for *selective* extraction of features have also achieved real-time performance in some cases — the information value and cost of features are estimated on-line, and only efficient features are extracted. In this paper, pipelined/parallel processing is combined with selective extraction of features. We use dynamic Bayesian networks to specify event recognition tasks and the particle filter for approximate inference. To control recognition rate/response time, we propose a pipelined particle filter architecture that supports parallel extraction of features, at the same time as previously extracted features are classified in parallel, and at the same time as features are selected in parallel for future extraction. In other words, CPUs can be dedicated to feature extraction independently of dedication of CPUs to feature selection/classification. This flexibility allows focusing of resources on the processing bottleneck at hand. In object tracking experiments, we are able to increase classification rate from 3 (1 CPU) to 20 classifications/second (10 CPUs). Still, no loss of classification accuracy is measured. Furthermore, only extracting the e.g. 10% most efficient features, reduces classification accuracy by just 2%. In short, the empirical results indicate that we have successfully combined the benefits of selective feature extraction with the benefits of pipelined/parallel feature extraction and classification.

Key-Words: - Data streams, pattern classification, feature selection, parallel processing, pipelined execution, particle filtering, dynamic Bayesian networks

1 Introduction

Transmitting volumes of streamed data, such as video, across high-speed communication networks, is becoming increasingly common. To utilize a growing number of data streaming information sources, both the ease of use and the computational flexibility of methods for content based access must be addressed.

In order to make data streams more accessible, pattern classification systems that are capable of recognizing high-level spatio-temporal concepts have been taken into use [2,3,6]. For instance, in video streaming, pattern clas-

sification systems can be used for recognizing concepts such as “object moving to the left” and “object moving to the right”, based on low-level features describing e.g. color, texture, and edges in each video frame.

Pattern classification systems for *on-line* processing of data streams are typically required to respond to real world events within a certain time limit and achieve a certain throughput (classification rate). We refer to such constraints as *timeliness* constraints. Under timeliness constraints, attention must be paid to the processing environment used as well as to the selection and deployment of feature extraction- and pattern classification algorithms.

To elaborate, there are two main approaches to meeting the above indicated class of response time/throughput requirements. The first approach is pipelining/parallelizing feature extraction and classification in order to take advantage of multiple CPUs. The response time/throughput of an application can then be improved by making more CPUs available for processing. For instance, in [1] an extensible and modular software architecture for parallel processing of data streams is proposed. Furthermore, in [2] a compile-time scheduler is proposed that schedules the tasks of video stream processing applications on available CPUs. The scheduler exploits spatial (parallelism) and temporal (pipelining) concurrency to make the best use of available machine resources. Similarly, [3, 4] describe a framework for coarse-grained multi-level parallelization and distribution of video stream filtering/transformation, feature extraction, and classification.

The second approach to meeting processing rate/response time requirements is based on the assumption that the actual feature extraction can be significantly more processing intensive than classification. Feature extraction may involve costly on-line signal processing while a pattern classifier is normally trained/specified off-line, so that it performs efficiently on-line. Thus, the second approach, referred to as hypothesis driven feature extraction, is based on only extracting selected features. The features are selected on-line so that the feature extraction can be adapted to the current real-world situation. The goal is to minimize feature extraction processing cost, while still maintaining acceptable classification accuracy. E.g., in the dynamic Bayesian network based video frame classification system from [5], the features of each video frame are extracted sequentially in decreasing order of efficiency (value of information relative to cost of processing). When the efficiency falls below a threshold, the feature extraction stops. Likewise, [6] is a recent approach from the field of physical sensor management which supports activation of the most “informative” sensor at each time step of a tracking task.

By integrating the pipelining/parallelization approach and the hypothesis driven feature extraction approach, it should be possible to achieve further processing rate/response time improvements. However, several difficulties are introduced. Selected features should be extracted in parallel (rather than sequentially as in [5, 6]) to

reduce the response time and increase the throughput of feature extraction. Also, the actual feature selection, as well as the classification, should be parallelized to support complex, accurate, and/or timely inference. Finally, pipelining feature extraction, feature selection, and classification, can further increase throughput. However, the feature extraction stage requires input from the feature selection stage, and the feature selection stage depends on input from the feature extraction stage. This input/output cycle makes pipelining problematic.

In this paper we target the above described difficulties by proposing a pipelined parallel architecture for feature extraction, feature selection, and classification of streamed data. Specification of data stream event recognition tasks is based on dynamic Bayesian networks [7] and the particle filter [8, 9] is used for inference. As a basis, we overview dynamic Bayesian networks and the particle filter in Section 2. In Section 3, we show how the so-called *pooled* classifiers architecture can be applied to execute multiple particle filters in parallel in a Local Area Network (LAN). We also propose a pipelined three-stage data stream processing procedure, consisting of parallel feature extraction, parallel feature selection/classification, followed by coordination. The resulting architecture is evaluated empirically in Section 4. We conclude in Section 5.

2 Dynamic Bayesian Networks and Particle Filtering

We use DBNs to specify data stream event recognition tasks, and we shall here give a short introduction. For a more thorough treatment, readers are referred to e.g. [7]. We also review the principles behind particle filter based inference.

2.1 Dynamic Bayesian Networks (DBNs)

A DBN consists of a qualitative and a quantitative part. The qualitative part is a directed acyclic graph (DAG). The *nodes* in the DAG are variables with states. Each variable represent an entity of interest, and the states of a variable represent the states the corresponding entity can be in. For instance, consider a video frame divided by an 8×8 grid. A motion feature is extracted from each grid position, and the goal is to determine the coordinates of

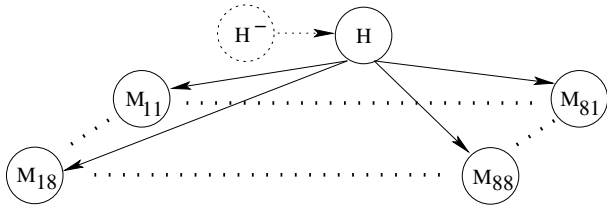


Figure 1: A DBN modeling object tracking.

a tracked object from these motion features. Then, one DBN variable could represent the position of the object. This variable would have 64 states, one for each grid position. Furthermore, we could assign a motion feature variable to each grid position with states “low”, “medium”, and “high”. These variables would represent the output of the motion feature extractors.

The *directed links* in the DAG represent causal impact between the variables. E.g., a moving object influences the features output by the motion feature extractors, and therefore we add a link from the object position variable to each motion feature variable. So far, the above description corresponds to an ordinary Bayesian network (BN). The difference between a BN and a DBN is that, in a DBN, the DAG is sectioned into a sequence of identical *time slices*. Each time slice represents a particular interval in time. This means that links between variables in two consecutive time slices indicate a causal impact from time interval to time interval. In a DBN, we could for instance let each time slice correspond to a video frame and then relate the possible positions of a moving object from time slice to time slice.

The qualitative DBN part suggested above represents an object tracking task and is illustrated in Fig. 1. This DBN consists of 8×8 motion feature variables M_{ij} , organized spatially as a grid, and a hypothesis variable H representing the grid position of a tracked object. The inter time slice link (H^-, H) reflects that the position of the object depends on its position in the previous time slice. Similarly, the link from H to a motion feature M_{ij} reflects the causal impact of each object position on the motion feature at grid position i, j . Note that in a stationary model, the causal impact between time slices does not vary. Thus, only the causal impact between two arbitrary consecutive time slices needs to be specified, as illustrated in the figure.

We now turn to the quantitative part of a DBN — the

strength of the directed links are represented as conditional probabilities. For each variable A with parents $pa(A)$, we have to specify the conditional probabilities $P(A | pa(A))$. If $pa(A) = \emptyset$ we specify the prior probabilities $P(A)$. So, for the DAG in Fig. 1 we have to specify $P(H^-)$, $P(H | H^-)$, $P(M_{11} | H) \cdots P(M_{88} | H)$. E.g., we could set $P(M_{11} = high | H = [6, 6]) = 0.05$ in order to indicate that significant motion in block $[1, 1]$ is unlikely when the object is located at grid position $[6, 6]$.

The above DBN model is used as a basis for the object tracking application described in [3]. Other kinds of event recognition tasks can be modeled in a similar manner. To conclude, we mainly use DBNs to calculate *posterior probabilities*, i.e., to calculate the probability of certain variables being in certain states, given the observed states of other variables. E.g., assume that we consider a single video frame and that the states of the motion feature variables have been observed for that frame: $M_{11} = m_{11}, \dots, M_{88} = m_{88}$. We could then calculate $P(H = [i, j] | M_{11} = m_{11}, \dots, M_{88} = m_{88})$ for each coordinate $[i, j]$ and identify the a posteriori most probable object position. Note that we are free to select which features to observe, and the selection determines how accurately the position of the object can be decided.

2.2 A DBN Based Particle Filter (PF)

There exist many algorithms for calculation of posterior probabilities in DBNs [7]. The PF [8, 9] is a *real-time* approximate technique. We here describe the PF in two stages. I.e., we first describe the state of a PF at time slice t (including $t = 0$). From this state, posterior probabilities can be calculated given features observed up to and including time slice t . We then describe a procedure for advancing to time slice $t + 1$.

Let \mathcal{X}^t denote the set of DBN variables in time slice t . Furthermore, let \mathcal{H}^t denote the so-called hypothesis variables of time slice t , that is, the variables $\mathcal{H}^t \subseteq \mathcal{X}^t$ which cannot be observed directly. Finally, let \mathcal{F}^t denote the so-called feature variables in time slice t , i.e., the variables $\mathcal{F}^t \subseteq \mathcal{X}^t$ whose states can be observed.

For the current time slice (t) our PF maintains a set S of *particles*. The particles can be seen as weighted samples. A single particle s consists of two parts. The first part of s is simply an assignment of a state x^t to each hypothesis variable $X^t \in \mathcal{H}^t$: $s.X^t := x^t$. Similarly, each feature

```

1: % Extend particles to cover time slice  $t + 1$  from  $t$ 
2:  $S' := \text{copy}(S)$ ;
3: FOR EACH  $s \in S$  DO
4:   % Replace particle  $s$  based on sampling from
5:   % distribution defined by particle weights
6:    $s := \text{SAMPLE } P(S')$ ;
7: % Assign state to DBN variables in new time slice
8: FOR EACH  $X^{t+1} \in \mathcal{X}^{t+1}$  DO
9:   IF  $X^{t+1} \in \mathcal{H}^{t+1}$  THEN
10:     $s.X^{t+1} := \text{SAMPLE } P(X^{t+1} \mid pa(X^{t+1}) = s.pa(X^{t+1}))$ ;
11:   ELSE
12:     $s.X^{t+1} := \text{OBSERVE } X^{t+1}$ ;
13:    $s.w \times = P(X^{t+1} = s.X^{t+1} \mid pa(X^{t+1}) = s.pa(X^{t+1}))$ ;

```

Figure 2: The updating step of the PF algorithm.

variable $X^t \in \mathcal{F}^t$ is assigned its observed state. In our example, a particle assigns an object position to the current video frame as well as a degree of motion to each grid position.

The second part of s is a weight $s.w$ which determines the probability of the feature observations up to and including time slice t , given the states s assigns to the hypothesis variables: $P(\mathcal{F}^1 = f^1, \dots, \mathcal{F}^t = f^t \mid \mathcal{H}^1 = s.\mathcal{H}^1, \dots, \mathcal{H}^t = s.\mathcal{H}^t)$. Accordingly, this part indicates how accurately the assignments of s reflect the “truth”. Note that each particle is initialized by sampling from a prior distribution $P(\mathcal{X}^0)$ specified as a part of DBNs, e.g., $P(H^-)$ in our example DBN. The particle weights are set to 1 — no observations done for $t = 0$.

By normalizing the weights of the particles, the particles can be seen as an approximation of the joint posterior probability distribution $P(\mathcal{H}^t \mid \mathcal{F}^1 = f^1, \dots, \mathcal{F}^t = f^t)$ [8]. From this joint probability distribution, the posterior distribution of individual hypothesis variables can be calculated.

The question then is how to update the particle set S so that it covers a new time slice $t + 1$. An algorithm which performs this updating step is found in Fig. 2 and detailed below. Each particle s is updated as follows. First, s is replaced by a particle drawn randomly from the particle set. The particle is drawn according to the probability distribution derived by normalizing the particle weights, hereafter denoted $P(S')$. Generally, the resulting replaced particle set will provide a better starting point for approximating $P(\mathcal{H}^{t+1} \mid \mathcal{F}^1 = f^1, \dots, \mathcal{F}^{t+1} = f^{t+1})$ compared to the original particle set. This is because particles representing less likely scenarios are equivalently less likely to be

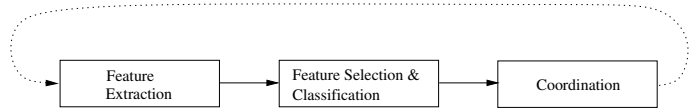


Figure 3: Pipelined feature extraction, feature selection/classification, and coordination.

included in the new particle set. Thus, the particles are concentrated on the likely scenarios and will not spread out across an exponentially large set of possible, but unlikely scenarios.

As a second step, the DBN variables in time slice $t + 1$ are ordered topologically with respect to the DBN DAG and assigned states by particle s in that order. Each hypothesis variables $X^{t+1} \in \mathcal{H}^{t+1}$ is assigned a state drawn from the conditional probability distribution $P(X^{t+1} \mid pa(X^{t+1}) = s.pa(X^{t+1}))$. Because of the ordering of the variables, the particle has already assigned states to the parents $pa(X^{t+1})$ of X^{t+1} . The feature variables are treated differently as the states of these are given. In short, the particle weight $s.w$ is updated to include each new observation ($s.X^{t+1} := \text{OBSERVE } X^{t+1}$): $s.w := s.w \times P(X^{t+1} = s.X^{t+1} \mid pa(X^{t+1}) = s.pa(X^{t+1}))$.

In essence, the above mechanisms evolve the particles to be a summarization of likely video frame interpretations.

3 Pipelined Parallel Feature Extraction, Feature Selection, and Classification

The limited processing resources available on a typical host restrict the complexity, accuracy, and timeliness of feature extraction and pattern classification in data streams. To reduce this problem, we here propose a *pipelined* and *parallel* architecture for real-time feature selection, feature extraction, and classification of streamed data.

3.1 Processing Architecture

First of all, our architecture *pipeline*s execution as illustrated in Fig. 3. As seen in the figure, processing is divided into a chain of stages, namely, feature extraction, feature selection/classification, and coordination, with each stage handling results obtained from the previous stage.

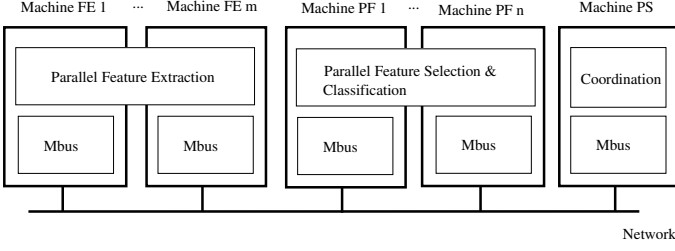


Figure 4: The feature extraction and feature selection/classification stages are independently parallelized.

Second, both the feature extraction and feature selection/classification stages may be processing intensive on their own. Therefore, we parallelize each of these stages as illustrated in Fig. 4. The parallelization of feature selection/classification is based on our parallel particle filter proposed in [4]. In short, multiple *particle filters* execute in parallel and a *pooling system* coordinates inference.

When pipelining and parallelizing processing as indicated above, it is important that messages can be exchanged efficiently. This is because communication overhead may reduce the benefits of pipelining and parallelizing execution. In our case, CPUs are connected in a LAN, and communication is based on transmitting messages across the LAN. Modern LANs typically allow messages to be transferred between machines in a few microseconds or so. To further support efficient coordination of concurrent components, we apply the Message Bus [10] (Mbus) which facilitates native multi-cast based group communication.

Let us now take a closer look at the above indicated processing stages and the resulting communication scheme.

3.2 Feature Extraction

The first processing stage of our architecture is *feature extraction*. In [3] we target the generic problem of scheduling extraction of m features on n CPUs under execution order constraints. Here, on the other hand, we assume that m features are to be extracted concurrently on m CPUs without any restrictions on execution order. Parallelizing feature extraction in this manner helps reducing both the response time as well as the throughput of the overall pattern classification system. E.g., applying m CPUs for feature extraction rather than only a single CPU, potentially cuts the response time of feature extraction by m and/or

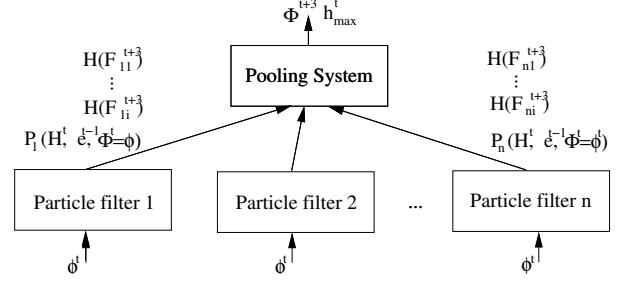


Figure 5: Pooled classifiers architecture applied to particle filtering.

allows m times higher throughput, as argued in [3].

3.3 Feature Selection and Classification

The second processing stage of our architecture is *feature selection* and *classification*. As seen in Fig. 5, we parallelize feature selection and classification by executing n independent particle filters in parallel and coordinating their output by means of a pooling system (coordinator). When processing a new time slice t , each particle filter takes the features produced by the preceding feature extraction stage (denoted by ϕ^t in the figure) as input. Based on this input, each particle filter updates its particles to cover the new time slice in the traditional manner (as described in Section 2).

The updated particles are then used for classification and feature selection as follows. Let e^t denote already extracted features (including ϕ^t). Each particle filter first approximates and outputs the unnormalized hypothesis probability distribution $P(\mathcal{H}^t, e^t)$ for the hypothesis variables in the new time slice. Note that $P(\mathcal{H}^t, e^t)$ can be used for determining the a posteriori probability distribution of the hypothesis variables \mathcal{H}^t given extracted features e^t :

$$P(\mathcal{H}^t | e^t) = \frac{P(\mathcal{H}^t, e^t)}{\sum_{h \in \mathcal{H}^t} P(\mathcal{H}^t = h, e^t)}.$$

Second, each particle filter estimates the *Mutual Information Gain* [7] of one n th of the features for time slice $t+3$, so that the particle filters in combination estimate the Mutual Information Gain of all the features $F_j \in \mathcal{F}^{t+3}$ of time slice $t+3$:

$$Gain(F_j) = H(\mathcal{H}^{t+3}) - H(\mathcal{H}^{t+3} | F_j).$$

I.e., the entropy of the hypothesis distribution, $P(\mathcal{H}^{t+3})$, is used to measure the hypothesis uncertainty of time slice

$t + 3$:

$$H(\mathcal{H}^{t+3}) = - \sum_{h \in \mathcal{H}^{t+3}} P(h) \log[P(h)],$$

and the *expected entropy* of the hypothesis distribution is used to measure the hypothesis uncertainty to be expected after extracting a feature F :

$$H(\mathcal{H}^{t+3}|F) = - \sum_{f \in F} P(f) \sum_{h \in \mathcal{H}^{t+3}} P(h|f) \log[P(h|f)].$$

Note that the distributions $P(\mathcal{H}^{t+3} | F_j)$, $F_j \in \mathcal{F}^{t+3}$ and $P(\mathcal{H}^{t+3})$ are obtained by means of *forward sampling* [7].

3.4 Coordination

The third processing stage is *coordination*. Based on the approximations of $P(\mathcal{H}^t, e^t)$ obtained from each particle filter, the pooling system determines the most probable hypothesis state $\mathcal{H}^t = h_{\max}^t$. This determination is based on mimicking the inference of a non-parallel particle filter by summing the received approximate distributions:

$$h_{\max}^t = \operatorname{argmax}_{h \in \mathcal{H}^t} \sum_{i=1}^n \hat{P}_i(\mathcal{H}^t = h, e^t).$$

h_{\max}^t is then output from the pooling system.

Lastly, the pooling system completes feature selection as follows. First, features are sorted according to their efficiency:

$$Ef(F) = \frac{Gain(F)}{Cost(F)}.$$

Here, $Cost(F)$ denotes the extraction cost of feature F . The m most efficient features Φ^{t+3} are then given as output. This finalizes the processing of time slice t .

3.5 Pipelined Execution and Communication Scheme

Because of our pipelining of feature extraction, feature selection/classification, and coordination, the above assignment of time slices to hypothesis and feature variables require further explanation, in particular with respect to the feature selection. As illustrated in Fig. 6, while features are extracted for the current time slice t , the particle filters

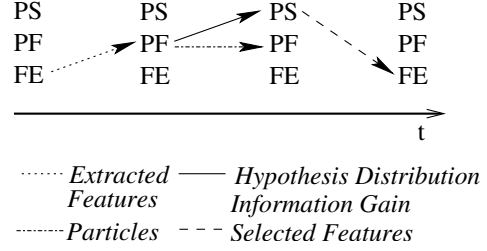


Figure 6: Pipelined execution of feature extractors (FEs), particle filters (PFs), and a pooling system (PS).

process the features extracted in the previous time slice $t - 1$. Likewise, the pooling system processes hypothesis distributions and Mutual Information Gains produced from time slice $t - 2$. To complicate, the feature extractors must know what features to extract in the next time slice ($t + 1$). As a result, the particle filters must predict the Mutual Information Gain of features three time slices ahead. E.g., when the particle filters process features extracted in time slice $t - 1$, they must estimate the Mutual Information Gain of the features at time slice $t + 2$.

Another complicating factor is the fact that each particle filter may lose track of the real-world situation one-by-one. This is because the overall pool of particles are fragmented into n sets, as discussed in [4]. This problem can be significantly reduced by letting the particle filters exchange particles using the scheme we propose in [4]. Fig. 6 shows how particles can be exchanged as part of our pipelined architecture. Essentially, each particle filter submits its most likely particle to the other particle filters after processing each time slice. Each particle filter then incorporates the received particles into its local pool of particles before processing the next time slice.

To conclude, the messages exchanged in all of the above discussed steps are illustrated in Fig. 7. As seen in the figure, m feature extractors transmit features to n particle filters by means of m multi-cast messages. The particle filters estimates the probability distribution of hypothesis variables and calculates the Mutual Information Gain of features. These quantities are multi-casted to the pooling system by means of n messages. In addition, as discussed above, each particle filter multi-casts its most likely particle to the other particle filters. Finally, the pooling system estimates the most likely hypothesis states as well as selecting which features are to be extracted next (1 message). In total, $m + 2n + 1$ messages are exchanged

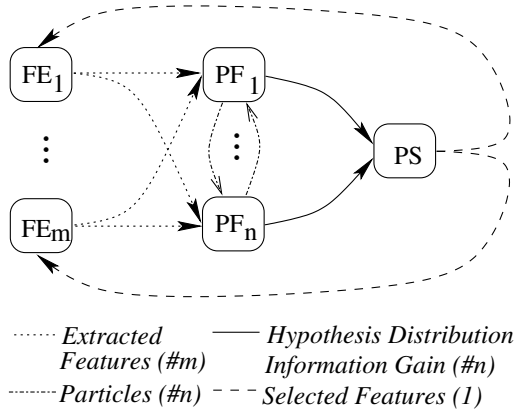


Figure 7: Messages exchange between feature extractors (FEs) particle filters (PFs) and a pooling system (PS).

0.02	0.10	0.02
0.10	0.52	0.10
0.02	0.10	0.02

Figure 8: The modeled probability of each possible object move centered on the current object position.

when processing a time slice. Note that we in [3] are able to exchange 9000 messages per second across our message exchange service, which forms a current upper bound on the number of parallel feature extractors and the number of parallel particle filters that are practical to execute in our architecture.

4 Empirical Results

In this section we evaluate our proposed architecture empirically by means of an object tracking case. Imagine a 20×16 grid of motion sensors. The motion sensor at a given grid position detects whether an object is located in that position. However, the motion sensors are noisy, having a certain probability of falsely detecting an object or missing an object. The data stream recognition task is tracking of an object that moves according to the movement model shown in Fig. 8, by controlling the 320 motion sensors. As seen in the figure, an object moves to an adjacent grid position with a certain probability.

For the above case, we have measured the tracking error and the throughput (time slices processed per second) when different numbers of CPUs are applied for fea-

ture selection and classification, under different degrees of sensor noise. We also vary the number of sensors (features) that can be activated in each time slice. The results are shown in Table 1. As seen in the table, we are able to increase classification rate from 3 (1 CPU) to 20 classifications/second (10 CPUs) when 320 sensors are activated in each time slice. Still, no loss of classification accuracy can be measured. Note that an average of respectively 1.6, 3.2, and 6.4 sensors provide the wrong result in each time slice when the probability of false/missing detections are 0.005, 0.01, and 0.02.

Also note that the throughput of our parallel particle

Table 1: Tracking error and throughput for different degrees of parallelization, noise, and features allowed extracted within a time slice.

#CPUs	#Features	Noise	Tracking Error	Throughput
1	320	0.005	0.02 ± 0.004	3
1	320	0.01	0.042 ± 0.006	3
1	320	0.02	0.068 ± 0.008	3
1	32	0.005	0.041 ± 0.006	4
1	32	0.01	0.061 ± 0.008	4
1	32	0.02	0.129 ± 0.011	4
1	16	0.005	0.054 ± 0.007	4
1	16	0.01	0.098 ± 0.009	4
1	16	0.02	0.144 ± 0.011	4
5	320	0.005	0.021 ± 0.004	11
5	320	0.01	0.033 ± 0.006	11
5	320	0.02	0.072 ± 0.008	11
5	32	0.005	0.029 ± 0.005	14
5	32	0.01	0.043 ± 0.006	14
5	32	0.02	0.11 ± 0.01	14
5	16	0.005	0.046 ± 0.007	14
5	16	0.01	0.068 ± 0.008	14
5	16	0.02	0.154 ± 0.011	14
10	320	0.005	0.024 ± 0.005	20
10	320	0.01	0.031 ± 0.005	20
10	320	0.02	0.071 ± 0.008	20
10	32	0.005	0.027 ± 0.005	25
10	32	0.01	0.048 ± 0.007	25
10	32	0.02	0.1 ± 0.009	25
10	16	0.005	0.061 ± 0.008	25
10	16	0.01	0.077 ± 0.008	25
10	16	0.02	0.143 ± 0.011	25

filter increases when the number of features extracted per time slice is reduced. This is because features that are not extracted need not be processed by the particle filter. This effect is in addition to the processing benefits obtained by only extracting a fraction of the features in each time slice. E.g., only extracting the e.g. 10% most efficient features, reduces classification accuracy by just 2%.

5 Conclusion

The limited processing resources available on a typical host restrict the complexity, accuracy, and timeliness of feature extraction and pattern classification in data streams. In this paper, we have integrated the pipelined/parallel and hypothesis driven feature extraction and classification approaches. The particle filter has been modified for real-time hypothesis driven feature extraction based on dynamic Bayesian networks, and selected features are extracted in parallel to support computationally expensive feature extraction. We use a pooling system to coordinate output from multiple particle filters. The empirical results indicate that multiple CPUs can be taken advantage of to significantly increase throughput, without any measurable loss in inference accuracy, indicating a feasible solution to the targeted processing bottleneck problem. Finally, by only extracting the most efficient features, the feature extraction resource usage can in many cases be reduced, while the classification accuracy is maintained.

Acknowledgement We thank the people involved in the Distributed Media Journaling project, and in particular Viktor Sigurd Wold Eide, for helpful discussions.

References

- [1] Francois A. and Medioni G. A Modular Software Architecture for Real-Time Video Processing. In *Proceedings of the Second International Workshop on Computer Vision Systems (ICVS 2001)*, vol. 2095 of *LNCS*. Springer, 2001.
- [2] Yang M.T., Kasturi R., and Sivasubramaniam A. A Pipeline-Based Approach for Scheduling Video Processing Algorithms on NOW. *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, February 2003, pp. 119–129.
- [3] Eide V.S.W., Eliassen F., Granmo O.C., and Lysne O. Supporting Timeliness and Accuracy in Distributed Real-time Content-based Video Analysis. In *Proceedings of the 11th Annual ACM International Conference on Multimedia (MM'03)*. ACM Press, 2003.
- [4] Granmo O.C., Eliassen F., Lysne O., and Eide V.S.W. Techniques for Parallel Execution of the Particle Filter. In *Proceedings of the 13th Scandinavian Conference on Image Analysis (SCIA2003)*, vol. 2749 of *LNCS*. Springer, 2003.
- [5] Granmo O.C., Eliassen F., and Lysne O. Dynamic Object-oriented Bayesian Networks for Flexible Resource-aware Content-based Indexing of Media Streams. In *Proceedings of the Scandinavian Conference on Image Analysis, SCIA'2001*, 2001.
- [6] Doucet A., Ba-Ngu V., Andrieu C., and Davy M. Particle filtering for multi-target tracking and sensor management. In *Proceedings of the Fifth International Conference on Information Fusion*, vol. 1, July 2002.
- [7] Jensen F.V. *Bayesian Networks and Decision Graphs*. Series for Statistics for Engineering and Information Science. Springer Verlag, 2001.
- [8] Liu J. *et al.* Sequential Monte Carlo methods for dynamic systems. *Journal of the American Statistical Association*, vol. 93(443), 1998, pp. 1032–1044.
- [9] Isard M. and Blake A. Condensation – conditional density propagation for visual tracking. *International Journal of Computer Vision*, vol. 29(1), 1998, pp. 5–28.
- [10] Ott J., Perkins C., and Kutscher D. A Message Bus for Local Coordination. *Internet Draft, draft-ietf-mmusic-mbus-04.txt*, 2001.