Markov Prefetching for Multi-Block Particle Tracing on Parallel Post-Processors

M. Wolter*, A. Gerndt, T. Kuhlen and C. Bischof

Virtual Reality Group, RWTH Aachen University
Seffenterweg 23, 52064 Aachen, Germany
E-mail: wolter@rz.rwth-aachen.de - Web page: http://www.rz.rwth-aachen.de/vr

Key Words: Parallel post-processing, particle tracing

EXTENDED ABSTRACT

1 INTRODUCTION

Visualization of flow phenomena in large time-varying data sets demands extensive graphical, computational and memory resources. Recent research in the field of parallel post-processing algorithms and out-of-core data management shows that a good utilization of given resources together with a considerable speed-up may be achieved. Due to the growing complexity and size of today뭘 flow data sets, one of the main problems for parallel applications is the I/O bottleneck. I/O waiting time may cause severe load imbalance which reduces the speed-up. These problems are commonly alleviated by data caching as well as prefetching techniques overlapping I/O with computation. Common prefetching methods suffice for predictable data queries but fail for more unpredictable requests, e.g. in particle tracing.

While some prefetching approaches in CFD post-processing are user-based [1], none exploits information available in the grid topology or the flow field itself. Therefore, we introduce a Markov prefetcher integrated in a data management middleware for predicting data requests in multi-block data sets. Our prefetcher uses former request patterns as well as various information about the flow field and the data set뭘 topology graph to provide required data almost without delay. We achieve a reduction of data access and time lags when applying Markov prefetching in parallel computation of particle traces for different data sets.

2 MULTI-BLOCK PARTICLE TRACING

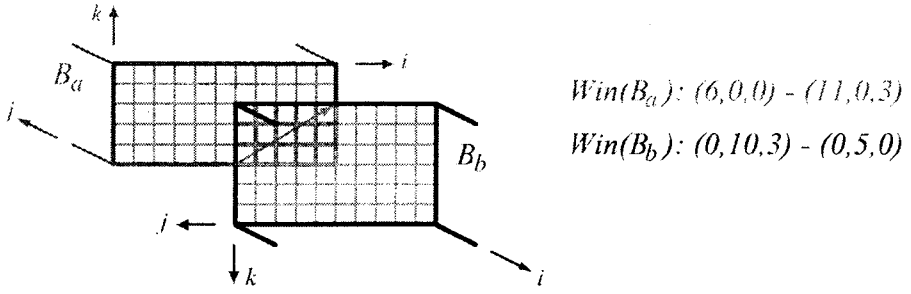

Figure 1: Connection window between blocks B_a and B_b in a multi-block topology.

For out-of-core access to structured grids, we utilize the multi-block topology structure described in [2]. This structure stores meta-information about block connectivities for cell search in the overall data set. Instead of memorizing each cell link between blocks, inter-block connectivity is stored by means of connection windows. Such a window is defined by *i-j-k-*

indices of two grid nodes lying diagonally at the corners, which is restricted to one side of a hexahedral block (see figure 1).

The complex computation of a single pathline is hardly parallelizable. Therefore, in our parallel post-processing framework, primarily data parallelization is applied, i.e., only the seed points are distributed to several processes. If a particle leaves the current cell but stays within the current block, the internal topology of structured blocks allows direct selection of neighboring cells. However, if a particle leaves a block, the newly entered block and cell are located facilitated by the connection windows. This results in a loading operation for the newly entered block, which may be anticipated by an adequate prefetching mechanism.

3 MARKOV PREFETCHING IN A PARALLEL POST-PROCESSOR

As the Markov prefetcher is a general approach, it is embedded into an existent data management system. The Viracocha software [3] provides parallel post-processing for connected visualization applications in a client-server setup, with Viracocha as the server component. Processes of the parallelization framework communicate via MPI, while requests and data are sent via TCP/IP between visualization client and Viracocha. A fundamental part of the software is the Viracocha Data Management System (VDMS), which provides processes with needed data. Therefore different loading strategies (interprocess communication, parallel I/O etc.), caching mechanisms and prefetching strategies are implemented. As Viracocha serves as a middleware, these techniques are independent of specific data formats.

For data parallel extractions of, for example, isosurfaces or cutplanes, the blocks required for computation and their order is known before the extraction algorithm starts. Simple techniques like one-block-lookahead (OBL) or strided prefetch provide easy and accurate predictions. In contrast, for particle tracing, the set of required blocks evolves during computation: the next position of a massless particle depends on the current position and the current flow direction. To provide useful prefetching for unpredictable access patterns, we introduce the Markov prefetcher for multi-block data sets.

The Markov prefetcher for computer systems is described in [4]. Similarly, the VDMS-Markov predictor works on the order of block requests by an algorithm. Using this request stream, the prefetcher builds a probability graph for the succession of blocks (see figure 2). To maintain a reasonable size for the graph, the predictor only keeps track of a window of the last n block requests instead of the whole request stream.

The Markov probability graph provides good results after a certain runtime, which is needed to adopt to the overall request behavior. However, the probability graph is empty after the system starts or when exploring a new data set, resulting in none or inaccurate prefetches. This drawback is prevented by initializing methods for Markov prefetchers.

3.1 INITIALIZING THE MARKOV GRAPH

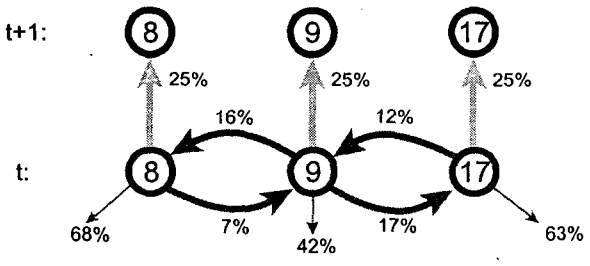

Figure 2: Example for a Markov graph containing three blocks numbered 8, 9 and 17 in two different timelevels.

We introduce two approaches to initialize the Markov prefetcher for multi-block CFD data: a connection window based heuristic and precomputed flow statistics. Both methods? goal is to generate a probability distribution that describes the flow field, i.e., for every pair of blocks A and B, the probability $P(B|A)$ that a particle which resides currently in A will eventually leave to B is computed.

The **connection window heuristics** uses the already existing topological information. As a particle can only move to neighboring blocks, we label the transition probability of two neighbors according to the size of their connection windows, i.e. the larger the window the higher the probability. This Markov graph is easy to compute but involves only the topological information of the grid. The behavior of the flow is completely disregarded, which leads to the second approach.

Precomputed flow statistics are generated in an offline step. By randomly inserting seed points in every block and tracing the according particles, a probability graph is build. This precomputation terminates if the probability distribution stays stable for several iteration steps, i.e. the distribution has reached some kind of balanced state. For a further speed-up, the pre-computation can be executed in parallel.

Both resulting probability distributions describe in a heuristic way the flow field in one time-step. The introduction of weighted timelevel transitions expands the Markov graph from one to multiple timelevels. That is, for unsteady particle tracing, predictions for changing the timestep are considered and the steady transition weights are accordingly adapted.

4 RESULTS

For the evaluation of the presented approaches, we currently use two different multi-block data sets. The nose data set represents the flow field inside an artificial human nasal cavity. The simulation has a perceivable main flow direction. The second utilized data set is the engine data set. It depicts the inflow and compression phase of a four-valve combustion engine, which is more turbulent than the nose flow. Both data sets are rather small (engine 360 MB, nose 2 GB) and would completely fit into main memory of most high performance computers. However, they are useful to verify the Markov prefetcher as they show different flow characteristics.

Three test series were performed. The first one labeled *Markov runtime evolved* starts with an empty Markov graph and evolves from consecutive particle computations with similar seed points. This behavior resembles an exploring user during runtime examining a special part of the flow field. The *Connection Windows* and *Flow Statistics* series present values of a single particle trace using appropriately initialized Markov graphs. Side-effects from previous particle traces are excluded, so that both series depict the pure benefit from an external initialization. The results are computed on a SunFire v880z with 4 GB main memory and four Ultra Sparc III Cu processors.

As shown in table 1, the prefetching efficiency (the percentage of useful prefetches) for both particles traces that base on initialized graphs are at most 60 %. However, up to 46 % of all blocking loading operations are reduced by the flow statistics method (measurement not depicted). This is a quite good result regarding the simple heuristic we used. Tracing in the more turbulent engine data set, the flow statistics approach beats the heuristical method. This is due to the larger portion of connection windows (more than 72 %) in the engine뭘topology that do not correspond to the main flow direction.

	Engine		Nose	
	Streamlines	Pathlines	Streamlines	Pathlines
Markov runtime evolved	100 %	74 %	90 %	87 %
Connection windows	40 %	38 %	55 %	35 %
Flow statistics	60 %	47 %	53 %	24 %

Table 1: Prefetching efficiency.

The total savings in loading time for particle tracing are depicted in figure 3 (left). Regarding waiting time for I/O, both initializing methods reduce loading time, in particular when applied to streamline calculation. The inferior I/O time reduction with pathlines in contrast to streamlines may be ascribed to the time-dependent nature of pathlines, which is not fully characterized by the timelevel transitions. The benefit for parallel particle tracing is shown in figure 3 (right): 64 pathlines are computed in the engine data set with up to 8 processes. The amount of reduced loading time with markov prefetching grows with the number of processes.

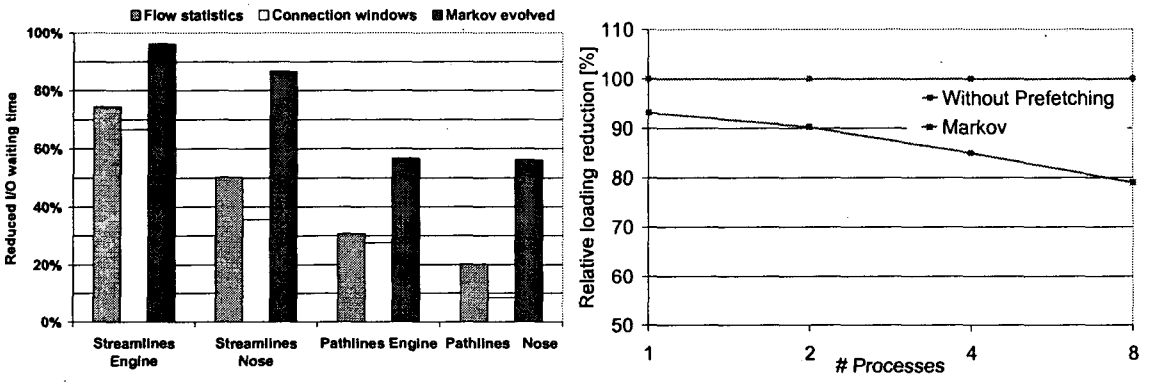

Figure 3: *Left:* Reduced I/O waiting time for the three test series. *Right*: Relatively reduced loading time for tracing 64 pathlines in parallel.

5 CONCLUSION AND FUTURE WORK

We presented a Markov prefetcher as an optional component of the parallel post-processor Viracocha. The prefetcher is used for predicting data requests in parallel multi-block particle tracing. To be efficient even when the post-processing framework is started or new data sets are selected, different approaches of Markov graph initializations are integrated. This yields a substantial improvement in comparison to uninitialized Markov prefetching or sequential prefetching strategies.

One shortcoming of the applied multi-block meta-data is that it merely considers topology information on each time-level separately. This occasionally results in inadequate Markov initializations for time-variant algorithms. In the future, we will also work on the improvement of the presented heuristics, e.g., by more reasonable seeding strategies or by considering the main flow direction when using connection windows.

REFERENCES

[1] P. R. Doshi, G. E. Rosario, E. A. Rundensteiner, M. O. Ward (2003). A Strategy Selection Framework for Adaptive Prefetching in Data Visualization. In *Proceedings of the 15th International Conference on Scientific and Statistical Database Management*, pages 107 116, Cambridge, Massachusetts.

[2] A. Gerndt, B. Hentschel, M. Wolter, T. Kuhlen, C. Bischof (2004). Viracocha: An Efficient Parallelization Framework for Large-Scale CFD Post-Processing in Virtual Environments. In *Proceedings of the IEEE SuperComputing (SC2004).*

[3] A. Gerndt, M. Schirski, T. Kuhlen, C. Bischof (2005). Parallel Calculation of Accurate Path-lines in Virtual Environments through Exploitation of Multi-Block CFD Data Set Topology. *Journal of Mathematical Modelling and Algorithms* (JMMA 2005), 1(4):35? 2.

[4] D. Joseph, D. Grunwald (1999). Prefetching using Markov Predictors. *IEEE Transactions on Computers*, 48(2):121? 33.