Running Interactive Perception Applications on Open Cirrus

Qian Zhu
Accenture Technology Labs
qian.zhu@accenture.com

Nezih Yigitbasi
Delft University of Technology
M.N.Yigitbasi@tudelft.nl

Padmanabhan Pillai
Intel Labs Pittsburgh
padmanabhan.s.pillai@intel.com

***Abstract*—**
Interactive perception applications, such as gesture recognition and vision-based user interfaces, process high-data rate streams with compute intensive computer vision and machine learning algorithms. Yet, they require extremely low latencies to remain interactive and ensure timely results to users. Cluster computing resources, such as those provided by Open Cirrus deployments, can help address the computation requirements, but significant challenges exist in practice. This paper highlights our efforts to parallelize interactive perception applications, tune them for best fidelity and latency, and place, schedule, and execute them on a cluster platform. We also look at remaining open problems and potential solutions.

I. Introduction

Multimedia recording and playback capability has become commonplace with the availability of low-cost digital cameras and recording hardware, but until recently, media applications have largely been limited to recording, compression, streaming, and playback for human consumption. Now, applications that can directly make use of video streams to sense the environment, detect activities, or serve as a form of input from users, are active areas of research and development [1], [2], [3], [4]. In particular, a new class of *interactive perception applications* that uses video and other high-data rate sensing for interactive gaming, natural gesture-based interfaces, and augmented reality is becoming increasingly important.

Interactive perception applications pose some unique challenges to their effective implementation in real systems. First, the data rates associated with video streams are high, making it challenging to process, store, and transmit the data without loss. Second, the state-of-the-art computer vision and machine learning techniques employed are often compute intensive. For example, Scale Invariant Feature Transform (SIFT) feature extraction [5], commonly used to find distinguishing features of an image, can take over a second to run for each frame of a standard definition video on a modern processor, over 30× too slow to keep up with the video stream.

Fig. 1. Interactive perception applications: (clockwise from top left) gesture-based gaming system; natural pointing detection; object pose detection; gesture-based TV control.

Furthermore, the computation load of these algorithms is highly variable, and depends on scene content, background clutter, motion, etc. Finally, these applications have tight response time requirements. To provide a crisp, responsive user experience, interactive applications may need to ensure that latency, from when sensor data arrive to when outputs are generated, is limited to 100–200 ms for each video frame.

Two broad approaches are being explored to achieve the required speeds. The first attempts to parallelize the execution of these applications on clusters of machines, e.g., as provided by Open Cirrus, by transforming the applications into a data flow graph of connected processing stages [6], [7], [8]. The effectiveness of this approach depends greatly on the extent to which particular steps are parallelized. The second approach trades off result quality, or fidelity, with computation cost [9]. The algorithms used in interactive perception applications typically have many parameters that can have a significant effect on both fidelity and latency, e.g., tunable detection thresholds, maximum iterations,

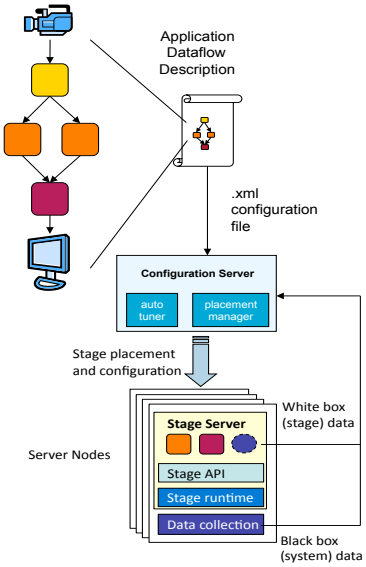

Fig. 2. Sprout architecture

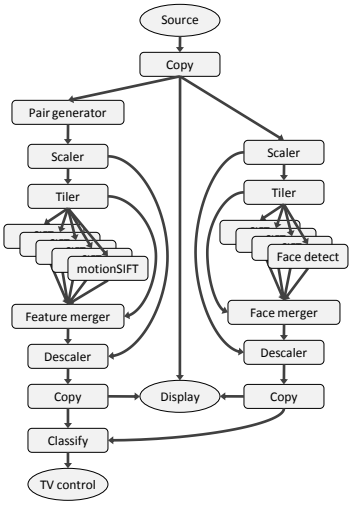

Fig. 3. Data flow for gesture-based TV control.

or even a switch to select alternative algorithms. The success of this approach depends on the parameters available in the application. If dynamically adjustable, both degree of parallelism and algorithmic parameters provide an opportunity to control latency.

In addition to the proper configuration of the application, parallel runtime overheads due to data transfer, resource contention, and coordination can significantly affect latency. Thus, successful execution of an interactive perception application hinges on careful allocation and scheduling of processing stages on different processors such that the latency for the distributed data flow to process each frame, including processing and data transfer time, i.e., the makespan, is minimized. However, given the variability in perception workloads, it is difficult to determine a good placement of the processing stages *a priori*, so an adaptive, incremental placement solution is paramount.

In the rest of this paper, we will highlight some of our efforts to address these concerns and execute interactive perception applications on Open Cirrus. We also look at remaining unresolved issues and some potential approaches to address them.

II. Work to Date

A. Programming framework for distributed interactive applications

To run interactive perception applications in parallel on Open Cirrus, we have developed a programming framework and runtime system called Sprout [8] (Figure 2). In the Sprout programming model, perception applications are structured as data flow graphs (see example in Figure 3). The vertices of the graph are coarse-grained sequential processing steps called *stages*, and the edges are *connectors* which reflect data dependencies between stages. The goal of this model is to exploit structural and data parallelism inherent in the applications, without requiring fine-grained parallelization of algorithm implementations. Stages interact only through connectors, and share no state otherwise. Source stages provide the input data to the application, for example as a stream of video from a camera. This data flows through and is transformed by multiple processing stages, which, for example, may implement a computer vision algorithm to detect when the user performs a particular gesture. These stages may be replicated and run in parallel on subsets of data to improve latency or throughput. Finally, the processed data is consumed by sink stages, which then control some actuator or display information to the user.

In the data flow model, concurrency is explicit – stages within an application can execute in parallel, constrained only by data dependencies and available processors. The Sprout runtime system distributes and executes application stages in parallel on a set of server nodes. The system provides mechanisms to migrate running stages, and to export and dynamically set tunable parameters, including both algorithmic parameters and controls for degree of parallelism (i.e., number of data-parallel stages). Our system also monitors application performance, and provides interfaces for extracting latency data at the stage level.

We have demonstrated several applications running on the Sprout framework. These include a gesture-based gaming system, a natural pointing interface [3],

a gesture-based tv control application [10], object pose detection [2], and video action recognition [1] (see Figure 1). Key takeaways from this work are that it is indeed possible to run latency-sensitive interactive perception applications on a cluster of machines, and that there is sufficient coarse-grained parallelism in these applications to make good use of 10s to 100s of processor cores.

B. Automatic modeling and tuning

A critical factor affecting latency of interactive perception applications is the appropriate setting of algorithmic parameters and controls for degree of parallelism. Initially, such parameters were adjusted manually in Sprout. To make a more effective system, we proposed a machine learning approach that automates parameter adaptation to yield the best combination of latency and fidelity, given an application and a set of computing resources. We first demonstrated how to model the effects of application tuning parameters on performance, and how to predict the latency of workloads using state-of-the-art techniques for online convex programming [11]. We then built a system that learns to predict the latency of workloads and uses this to optimize the fidelity subject to a latency bound [12]. The challenge in solving our problem online is that neither the latency function nor the fidelity function are typically known in advance. To make the problem more tractable, we assumed the fidelity function was known and focused on learning the latency function. This problem was formulated as an online constrained optimization problem [13] with unknown latency constraints. We used greedy strategies to explore the space of latency constraints, and showed that an appropriate mixture of exploration and exploitation leads to a practical system for solving our problem.

Specifically, our system approaches the automatic tuning problem as follows. It first identifies a set of critical stages, based on their contribution to end-to-end latency. It then explores the parameter space and learns a predictor as a function of the relevant tuning parameters for each critical stage. We used SVM to learn a linear regressor for each of the critical stages. Non-critical stages are modeled with a moving average. The application end-to-end latency is predicted using the stage models by computing the critical path through the data flow graph. A solver is then employed to search for operating points that maximize fidelity for a given latency constraint. We have demonstrated that accurate performance models can be learned online, and that for fixed latency constraints, operating points within 90% of optimal fidelity can be achieved with our system.

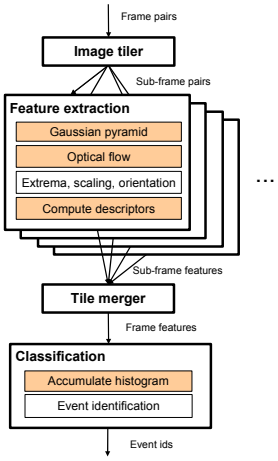

Fig. 4. Sprout application graph for video-based action recognition system. Images are partitioned into overlapping tiles, and features extracted in parallel in these tiles. In addition, the shaded components can use multiple threads on a multicore node.

C. Incremental Placement

In addition to the parallelization and tuning of application parameters, the placement and scheduling of the application components has a significant influence on the latency. In Sprout, we have devised a set of algorithms to *automatically* and *incrementally* place and schedule stages of an application on a set of processing nodes to minimize latency (makespan) [14]. The problem of initially placing an application graph onto a set of servers ultimately reduces to an NP-hard multiprocessor scheduling problem, requiring exponential time to solve optimally. We therefore employ a modified version of the Heterogeneous Earliest Finish Time (HEFT) [15] scheduling heuristic to quickly place application stages based on estimates of execution time and server performance.

Our system then continuously monitors performance of the running application stages, and, as conditions change, adjusts the placement by migrating stages between processing nodes. Due to the cost of moving running stages and the potential disruption to the application, we would like to change the existing placement as little as possible while improving application latency. To this end, we have developed four heuristics that perform incremental placement to minimize latency while bounding migration cost. Our heuristics all build on top of our modified HEFT scheduler, and target different points in the tradeoff between algorithm runtime and latency improvement. We have shown through simulation and experiments with real applications that our heuristics can improve median latency by up to 36% for interactive perception applications.

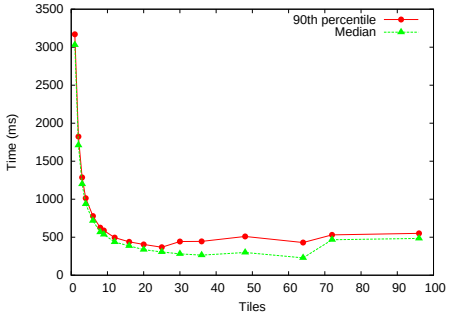

Fig. 5. Latency vs. number of tiles (cores) using non-threaded feature extraction.

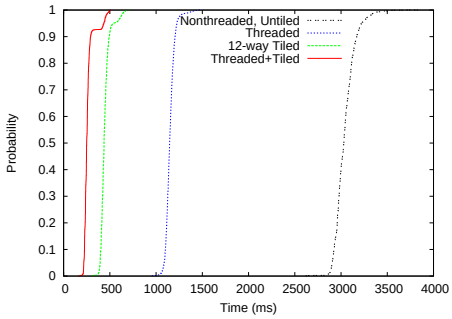

Fig. 6. Cumulative distribution of latency using threading and tiling. Threading decreases latency by 2.6×, tiling by 7×, and the combination by 12×.

III. Evaluation

We have had reasonable success in improving throughput and latency of interactive perception applications using our Sprout framework on top of Open Cirrus. Here, we show some results (full details in [1]) using an activity recognition application (Figure 4) on 15 8-core nodes in the Intel Open Cirrus cluster. For this application, the feature extraction stage is the computational bottleneck, requiring on the order of 3 seconds to execute on a single core for each pair of consecutive video frames. We parallelize execution in two ways: a) by threading key components to make use of multiple processing cores; and b) by splitting the source frames into overlapping tiles, and processing the tiles in parallel. The parallelization methods and degree to which they are applied greatly affects the resulting performance.

Figure 5 shows how the tiling strategy (alone) scales as the number of tiles (i.e., cores dedicated to feature extraction) increases. We see an initial sharp drop in latency, but shows diminishing returns after 20 tiles. This is partly due to tile overlap (so the total number of pixels processed increases), and due to overheads of high fan-in and queuing delays at the downstream merging stage.

Figure 6 shows the effects of threading and tiling on latency. For the threaded configurations, we dedicate a whole 8-core node for each instance of the feature detector. Threading does improve latency, but because not all parts of the algorithm can make good use of all of the cores, only a factor of 2.6 improvement is achieved. 12-way tiling alone results in 450 ms latencies, while threading and tiling together can reduce latency to around 250 ms.

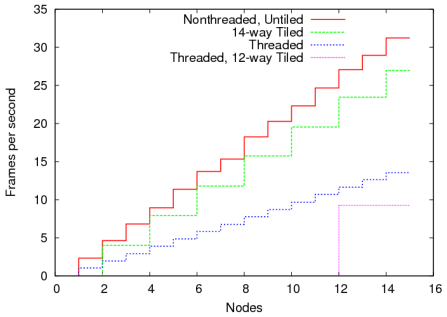

Fig. 7. Scaling up throughput by pipelining multiple instances of threaded and tiled configurations.

In addition to minimizing latency, we would also like to ensure sufficient throughput, i.e., number of frames per second. Figure 7 shows how throughput varies with the number of 8-core nodes used. We invoke multiple instances of the threaded or tiled feature detector, and pipeline execution by sending frames to the instances in round-robin order. Throughput is significantly better for the non-threaded configurations as they more efficiently use the available cores. So for this application and set of resources, the 14-way tiled configuration provides good throughput (>25 fps) and reasonable latency (<450 ms).

IV. Remaining Challenges

The work to date has been quite successful in running latency-sensitive interactive perception applications on server clusters provided by Open Cirrus. However, many challenges remain, some due to simplifying assumptions in the work so far, while others are due to the shared nature of Open Cirrus.

One issue is that to make the tuning and placement problems tractable, these challenging problems were considered separately. In particular, the auto tuner currently does not consider placement and scheduling issues, and cannot, for example, account for latency due to multiple stages sharing a processor core. Similarly, the placement system takes the application configuration as a given, and does not consider adjusting parameters or degree of parallelization in its scheduling decisions. Attempting to treat this as a joint placement and tuning problem will likely be intractable.

A potential solution we plan to explore is to keep the core tuning and placement algorithms separate, but modify the system so that they call on each other to better use the available information. The auto-tuning system currently estimates the latency of a particular application configuration by checking the longest path through the application graph. We can modify this to instead ask the placement manager to best fit the hypothesized configuration to the available resources and provide a more accurate latency estimate that considers scheduling and resource sharing. Thus, the autotuner will incorporate the more accurate latency information as it searches the configuration space.

A second issue is that in a shared environment like Open Cirrus, it would be very useful to adjust the resources consumed by long-lived perception applications as demand changes. When spare nodes are available, we would like our system to check whether it can make effective use of them (i.e., improve application fidelity or latency), and only request as many nodes as it can use well. Likewise, when the cluster is oversubscribed, given the scalable nature of the interactive perception applications, our system should be able to relinquish some nodes. Selecting which and how many nodes to minimize impact on the application remains an open research problem.

Finally, many shared cluster infrastructures like Open Cirrus use virtual machines to dynamically allocate custom environments for different users or applications. This makes it easy to deploy our Sprout system on many nodes quickly, but introduces its own set of issues. One problem with virtualization is that it incurs some amount of processing overhead. Given that our processing model is quite scalable, these overheads can be offset with a larger number of nodes. More significant, however, is that the variation in performance increases significantly under virtualization. For parallel fork-join structures in the application graph, the execution time is gated by the slowest instance, so an increase in execution time variance greatly limits performance improvements due to parallelization. One possible way around this is to avoid virtualization and allocate physical machines directly. There have been some efforts to provide this service in some Open Cirrus deployments.

V. Conclusion

We have demonstrated that it is possible to parallelize interactive perception applications and execute them on a server cluster. We have also made great headway in solving the challenges that such applications pose, in particular on dynamically tuning application configuration and incrementally placing and scheduling application graphs to ensure low latency. In addition, we have identified several remaining open issues. Despite these remaining challenges, we believe that the class of interactive perception applications is growing in importance, and should be a research priority to the Open Cirrus community.

References

[1] M.-y. Chen, L. Mummert, P. Pillai, A. Hauptmann, and R. Sukthankar, "Exploiting multi-level parallelism for low-latency activity recognition in streaming video," in *ACM Multimedia Systems Conference*, 2010.

[2] A. Collet, D. Berenson, S. Srinivasa, and D. Ferguson, "Object Recognition and Full Pose Registration from a Single Image for Robotic Manipulation," in *IEEE International Conference on Robotics and Automation*, pp. 3534–3541, 2009.

[3] P. Matikainen, P. Pillai, L. Mummert, R. Sukthankar, and M. Hebert, "Prop-Free Pointing Detection in Dynamic Cluttered Environments," in *IEEE Conference on Automatic Face and Gesture Recognition*, 2011.

[4] A. Sridhar and A. Sowmya, "Multiple camera, multiple person tracking with pointing gesture recognition in immersive environments," in *Advances in Visual Computing*, vol. 5358 of *Lecture Notes in Computer Science*, pp. 508–519, 2008.

[5] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[6] U. Ramachandran, R. Nikhil, J. M. Rehg, Y. Angelov, A. Paul, S. Adhikari, K. Mackenzie, N. Harel, and K. Knobe, "Stampede: a cluster programming middleware for interactive stream-oriented applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 11, pp. 1140 – 1154, 2003.

[7] J. Allard, V. Gouranton, L. Lecointre, S. Limet, E. Melin, B. Raffin, and S. Robert, "FlowVR: a Middleware for Large Scale Virtual Reality Applications," in *Proceedings of Euro-Par*, 2004.

[8] P. Pillai, L. Mummert, S. Schlosser, R. Sukthankar, and C. Helfrich, "SLIPStream: Scalable Low-latency Interactive Perception on Streaming Data," in *ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pp. 43–48, 2009.

[9] M. Satyanarayanan and D. Narayanan, "Multi-fidelity algorithms for interactive mobile applications," *Wireless Networks*, vol. 7, no. 6, pp. 601–607, 2001.

[10] M.-Y. Chen, L. Mummert, P. Pillai, A. Hauptmann, and R. Sukthankar, "Controlling Your TV with Gestures," in *ACM International Conference on Multimedia Information Retrieval*, pp. 405–408, 2010.

[11] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *Proceedings of the 20th International Conference on Machine Learning*, 2003.

[12] Q. Zhu, B. Kveton, L. Mummert, and P. Pillai, "Automatic tuning of interactive perception applications," in *Conference on Uncertainty in Artificial Intelligence*, pp. 743–751, 2010.

[13] S. Mannor and J. Tsitsiklis, "Online learning with constraints," in *Proceedings of 19th Annual Conference on Learning Theory*, pp. 529–543, 2006.

[14] N. Yigitbasi, L. Mummert, P. Pillai, and D. Epema, "Incremental placement of interactive perception applications," in *International Symposium on High Performance Distributed Computing*, 2011.

[15] H. Topcuouglu, S. Hariri, and M.-y. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.