Sheffield Hallam University

Profile driven dataflow optimisation of mean shift visual tracking

BHOWMIK, Deepayan <http://orcid.org/0000-0003-1762-1578>, WALLACE, Andrew, STEWART, Robert, QIAN, Xinyuan and MICHAELSON, Greg

Available from Sheffield Hallam University Research Archive (SHURA) at:

https://shura.shu.ac.uk/13808/

This document is the Accepted Version [AM]

Citation:

BHOWMIK, Deepayan, WALLACE, Andrew, STEWART, Robert, QIAN, Xinyuan and MICHAELSON, Greg (2015). Profile driven dataflow optimisation of mean shift visual tracking. In: Signal and Information Processing (GlobalSIP), 2014 IEEE Global Conference on. IEEE, 1-5. [Book Section]

Copyright and re-use policy

See http://shura.shu.ac.uk/information.html

Sheffield Hallam University Research Archive
http://shura.shu.ac.uk

Profile Driven Dataflow Optimisation of Mean Shift Visual Tracking

Deepayan Bhowmik & Andrew Wallace
School of Engineering and Physical Sciences
Heriot-Watt University, Edinburgh, UK
{D.Bhowmik,A.M.Wallace}@hw.ac.uk

Robert Stewart, Xinyuan Qian & Greg Michaelson
School of Mathematical & Computer Sciences
Heriot-Watt University, Edinburgh, UK
{R.Stewart,xq32,G.Michaelson}@hw.ac.uk

***Abstract*—Profile guided optimisation is a common technique used by compilers and runtime systems to shorten execution runtimes and to optimise locality aware scheduling and memory access on heterogeneous hardware platforms. Some profiling tools trace the execution of low level code, whilst others are designed for abstract models of computation to provide rich domain-specific context in profiling reports. We have implemented mean shift, a computer vision tracking algorithm, in the RVC-CAL dataflow language and use both dynamic runtime and static dataflow profiling mechanisms to identify and eliminate bottlenecks in our naive initial version. We use these profiling reports to tune the CPU scheduler reducing runtime by 88%, and to optimise our dataflow implementation that reduces runtime by a further 43% – an overall runtime reduction of 93%. We also assess the portability of our mean shift optimisations by trading off CPU runtime against resource utilisation on FPGAs. Applying all dataflow optimisations reduces FPGA design space significantly, requiring fewer slice LUTs and less block memory.**

I. Introduction

Enormous growth in computer vision (CV) research has prompted increasing interest in embedded real-time systems application domains *e.g.*, smart camera architectures [1], mobile robotics [2] and automotive applications (*e.g.*, self drive cars). While traditional CV algorithms are often developed for execution on sequential desktop computers, real-time CV algorithms require hardware platforms with time and resource bound guarantees to process data-intensive video streams at high frame rates. Application-specific embedded hardware is often the preferred option despite relatively complicated and long development cycles compared to software implementations on general purpose processors.

Targeting embedded hardware for CV algorithm execution is a challenging and time consuming task, especially when implementing algorithms directly with low-level hardware description language (HDL), such as VHDL or Verilog. Dataflow languages are a higher level and modular abstraction that have been demonstrated to be a good fit for embedded systems programming. The dataflow model is used to implement digital signal processing (DSP) systems [3], and more recently have been adopted for stream based image processing [4] and reconfigurable video coding (RVC) [5] applications.

Although dataflow models are well suited for stream based application domains to the best of the authors knowledge mapping CV algorithms to the dataflow model to exploit parallel hardware is largely unexplored with a few notable exceptions [6]–[8]. One possible reason is that CV algorithms cannot trivially be parallelised. For example, global operators may be mapped over the entire image frame or feedback loops may be required to derive values from one frame as inputs to functions on subsequent frames — sequentialising opportunities for pipelined parallelism.

The process of porting sequential algorithms to dataflow models is not trivial, and a three-step process is proposed in [9]: *a*) decouple the algorithm into independent processing blocks (*actors*) and data flow (*tokens*) between these blocks, *b*) design a resource efficient computational architecture to exploit parallelism & *c*) map the algorithm onto target hardware. The steps are iterated to produce an optimised design. Optimisations can be reached using profile driven approaches to make small incremental improvements to dataflow graph structures and actor implementation code (Section II).

In this paper we use the mean shift visual tracking algorithm for single subject tracking [10] as a CV case study to assess the suitability of dataflow as a model for parallel implementation and optimisation of CV algorithms. This is a conscious choice as it has a proven convergence criteria and has substantial execution costs in computing 2D matrix operations and exhibits dynamic behaviour when estimating subsequent object tracking positions using an iterative optimisation. The RVC-CAL language [11] is used to implement mean shift. The contributions of this paper are threefold:

- We express a well known CV tracking algorithm using the dataflow model and implement a naive version in the RVC-CAL dataflow language .
- We use runtime tracing and dataflow profiling techniques for identifying and eliminating bottlenecks and implement optimised variants. The optimised versions reduces runtime of 43% on a 4 core CPU.
- We trade off CPU runtime performance with FPGA resource utilisation for all dataflow optimisations of the naive mean shift version, and discuss the portability of dataflow optimisation strategies.

II. Profile driven data-flow optimisation

Profile guided optimisation is used to shorten execution runtimes and to improve hardware resource utilisation across different architectures. Profilers have been embedded into *compilers e.g.*, for automatic function inlining [12], into profile

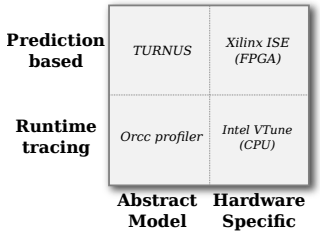

Fig. 1. Choices for dataflow profiling

Fig. 2. Example of single target mean shift visual tracking.

guided *JIT compilers* that compile bytecode of frequently used methods to native code [13], into *runtime systems* to reduce memory access latency on NUMA architectures using automatic page placement [14] and into *IDE tool support* for semi-automatic parallel refactorings *e.g.*, by introducing parallel algorithmic skeletons [15]. Profilers either *trace* a program's execution on a target hardware platform to identify runtime costs and resource utilisation, or they *predict* runtime costs or resource utilisation without executing machine code. Moreover, profilers can either be scoped to a specific model of computation such as the dataflow model, or can be hardware target specific, *e.g.*, profiling FPGAs or CPUs (Fig. 1).

Orcc [16] is primarily a compiler for the RVC-CAL dataflow language, though it also includes a CPU based runtime profiler for the C backend that traces action firings and workload on actors and connections. TURNUS [17] is a dataflow profiler that extends the Orcc profiling of RVC-CAL with behavioural predictions of dataflow graphs. In contrast, other profilers are not restricted to one abstract model of computation and instead profile domain agnostic source code at runtime. Intel VTune [18] traces software performance analysis on x86 CPUs, and assists in code profiling using stack sampling, thread profiling and hardware event sampling. Xilinx ISE [19] is a software tool for synthesis and analysis of HDL designs targeting FPGAs. It calculates perform timing analysis and statically predicts resource utilisation values unique to a particular FPGA models. Two of these profilers are used to identify bottlenecks in the naive mean shift implementation for optimisation opportunities in Section IV.

III. Dataflow Modelling of Meanshift Tracking

A. The algorithm

Mean shift [10] is a feature-space analysis technique for locating the maxima of a density function. An example of applying mean shift to image processing for visual tracking is shown in Fig. 2. The target is successfully tracked from the initial frame on the left, to the final frame on the right. The algorithm is a kernel based method normally applied using a symmetric Epanechnikov kernel within a pre-defined

Input: Target position y_0 on 1^{st} frame;
Compute Epanechnikov kernel;
Calculate *target* color model $q_u(y_0)$
(*e.g.*, using RGB color histogram);
repeat
  Input: Receive next frame;
  Calculate *target candidate* color model: $p_u(y_0)$;
  Compute similarity function $\rho(y)$ between $q_u(y_0)$ & $p_u(y_0)$;
  repeat
    Derive the weights ω_i for each pixel
    in *target candidate* window;
    Compute new target displacement y_1;
    Compute new candidate colour model $q_u(y_1)$;
    Evaluate similarity function $\rho(y)$ between $q_u(y_0)$ & $p_u(y_1)$;
    while $\rho(y_1) < \rho(y_0)$ **do**
      Do $y_1 \leftarrow 0.5(y_0 + y_1)$;
      Evaluate $\rho(y)$ between $q_u(y_0)$ & $p_u(y_1)$;
    end
  until $|y_1 - y_0| < \epsilon$ *(near zero displacement)*;
  Output: y_1 *(Target position for current frame)*;
  Set $y_0 \leftarrow y_1$ for next frame;
until *end of sequence*;

Algorithm 1: Summary of Mean-shift tracking

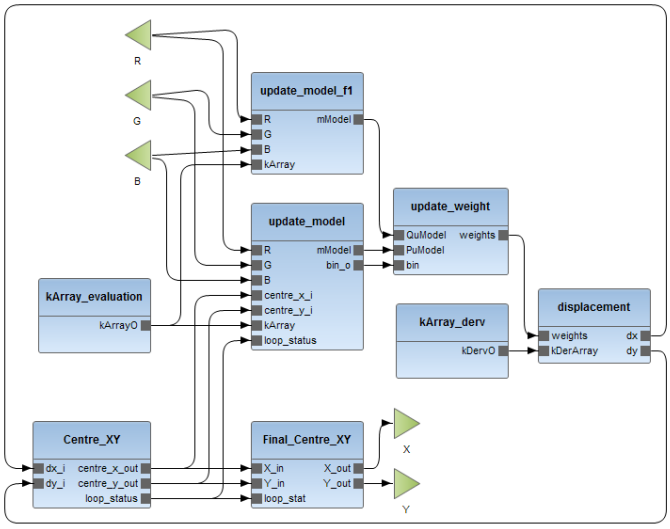

Fig. 3. Mean-shift tracking algorithm actor network.

elliptical or rectangular window. The target region of an initial image is modelled with a probability density function (a colour histogram) identifies a candidate position in the next image by finding the minimum distance between models using an iterative procedure. A summary is given in Algorithm 1.

B. Functional decomposition with dataflow actor network

Our dataflow versions of mean shift have been implemented in the RVC-CAL dataflow language. It is a port of an existing sequential implementation in C++ [20]. Coarse grained functional components were de-coupled and mapped into separate actors that communicate computation results using token passing, shown in Fig. 3.

The Epanechnikov kernel and its derivatives are calculated in *Actor kArray_evaluation* and *kArray_derv*, respectively. Their constant values are computed once because they depend only on the size of the target window. These values are passed as streaming tokens cyclically to the *update_model_f1*, *update_model* and *displacement* actors. The *update_model_f1*

and *update_model* actors calculate the colour models $q_u(y_0)$ & $p_u(y)$. The *update_model* actor calculates the histogram as a collection of bins. The histogram function is used to assign a particular RGB value to a bin in the feature space using the 3 values as an index into a 3D space modelled using a 1D array. Each bin u in the model is the normalised sum of all kernel values for the pixels in that bin.

Once the model q_u is calculated for the initial centre position on the first frame, subsequent frames are used to calculate the displacement y_1 on each frame using a feedback loop representing steps in Algorithm 1. The *update_weight* actor derives the weights w_i for each pixel in the target candidate window, while the *displacement* actor computes the displacement y_1 by Eq. (1).

$$y_1 = \frac{\sum_{i=1}^{N} x_i w_i g()}{\sum_{i=1}^{N} w_i g()}, \tag{1}$$

where N is the number of pixels in the target window, x is each pixel's relative position, its weight w_i and $g()$ is the kernel derivative function. This is iterated by actors *Centre_XY* and *Final_Centre_XY* until the convergence criteria ($|y_1 - y_0| < \epsilon$) is met and uses a feedback loop controlled by boolean tokens passed to the *loop_status* port in the *update_model* actor. To read and write video streams (Fig. 3), two smaller actor networks have been developed (not shown) to *a*) convert raw YUV video streams into RGB channels, and *b*) emit raw YUV video streams with a red rectangle highlighting the tracking window on every frame.

IV. Optimisations

A. Abstract Dataflow Optimisations

The Orcc profiler (Section II) is used to identify mean shift bottlenecks in the context of high-level dataflow execution, by identifying actions on the critical path through the dataflow graph and finding where FIFOs are being starved of tokens.

1) FIFO Depth Reduction: A FIFO size of 32768 is needed to stream two consecutive YUV frames through the naive mean shift dataflow graph. Attempting to pass more frames through the graph deadlocked execution, suggesting that the naive version does not fully support the streaming model which is required for continuous tracking. For example, in order to pass 42 frames through the graph required a FIFO size of 1048576 and to pass 130 frames through required a FIFO size of 16777216. The Orcc profiler identified a starvation of tokens in the FIFO between the `R`, `G` and `B` ports and the *update_model* actor, because the tokens were not being consumed at the same rate by the *update_model* and *update_model_f1* actors. The *update_model_f1* was using RGB values to compute $q_u(y_0)$ only once on the first frame, whereas the *update_model* was computing $p_u(y)$ iteratively for consecutive frames. The remedy was to fuse both actors, merging their finite state machines (FSM) into *update_model*. The FSM was modified so that the action to compute $q_u(y_0)$ is fired only once. The optimisation recovers the FIFO size back to 32768 to process any number of frames, and the algorithm now supports the streaming model for which results are presented in Section V.

2) Language Use Refinement: Tracing the naive mean shift version with the profiler shows intensive scheduling of actors that have only a small number of computationally inexpensive actions. For example, the workload of the *kArray_evaluation* actor was profiled at 12.2%, despite there being only two actions in the actor, one of which computed the Epanechnikov kernel with no token passing and another action called `sendData` transmitting the kernel values to colour model actors. The latter was initially implemented as a transmission action looping over the kernel size. The optimisation was to opt for a builtin RVC-CAL language construct `repeat` in the implementation of `sendData`. The profiler reports a workload reduction of 82% for the *kArray_evaluation* actor.

B. CPU Optimisations

The Intel VTune profiler (Section II) is used to trace CPU clock cycles for every line of C code for both actors and the runtime scheduler that the Orcc compiler generates, and identifies bottlenecks in the CPU scheduler and hotspots *within* action implementations.

1) CPU Runtime System IO: The naive mean shift was first profiled to identify the most severed bottlenecks. The mean runtime of computing mean shift over a sequence of 130 YUV frames of dimension 176×144 is 24.6s. The profile reported 41% of overall runtime was executing an actor responsible for writing YUV frames to file. We fixed the suspect IO operation by replacing costly `fseek` and `fwrite` calls with a `putc` call in the runtime system, and the Orcc compiler was patched with this fix. The mean runtimes using this fix is 2.97s, a speedup of 8.3 reducing runtime by 88%. All remaining optimisation runtimes are measured using this runtime system scheduler optimisation.

2) Actor Fusion: In Orcc's C backend, FIFOs are implemented as lock-free C `structs` shared between the source and sink actors that it connects. These become bottlenecks if token passing frequency between two actors is high either in the absence of a multicore CPU or because the computational granularity of the actors is too small. A solution is to fuse multiple actors to eliminate FIFO bottlenecks and the scheduling overheads of computationally inexpensive actors [21]. The primary cost of actor fusion is code modularity and reuse. Fusion optimisation is applied to the naive version, which uses three separate actors to *1)* draw a tracking rectangle, *2)* convert RGB values to the YUV colour space and *3)* merging individual YUV channels into a single stream. The profiler reports runtimes of 0.25s, 0.26s and 0.29s respectively — a total of 0.8s. The optimisation involved fusing the FSMs of the three actors into a new single actor, and profiling reports a runtime of 0.33 for this actor — a runtime reduction of 59%.

C. FPGA Optimisations

1) Task Parallelism: Additional actors are introduced as a task parallelism optimisation for embedded dataflow hardware. By observing the dependencies between actors, and between

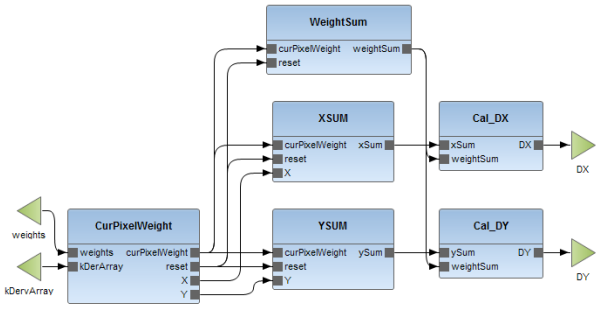

Fig. 4. Introduction of task parallel actors.

actions within actors, we identified the *displacement* actor as a candidate for a streaming optimisation. Its actions were unfolded into smaller independent actor due to the absence of data dependencies between these actions. This actor computes Eq. (1) which is decomposed into six actors, shown in Fig. 4. The *CurPixelWeight* actor maintains the interface with preceding actors using ports *weights* and *kDervArray*, and broadcasts the current pixel weight, and X and Y values to three new actors *WeightSum*, *XSUM* and *YSUM*. Actors *XSUM* and *YSUM* compute the numerator of Eq. (1) in the X and Y direction respectively. The *WeightSum* calculates the denominator of Eq. (1) used by actors *Cal_DX* and *Cal_DY* to compute the displacement y_1.

2) Pipeline Parallelism and Memory Tuning: A major challenge of porting sequential algorithms to the dataflow model is availability of limited memory. Stream processing on FPGAs can overcome memory bottlenecks by pipelining operations. Dataflow imposes a *share-nothing* memory architecture. Therefore image processing algorithms must be refactored to reshape large *N* dimensional data structures into pipelines of isolated memory regions in actors. The naive RVC-CAL implementation used memory inefficiently in its faithful port of the C++ version. We adapt this FPGA optimisation technique by modifying the way YUV frames are streamed into the network of actors that compute tracking. For example, an actor is responsible for drawing a rectangular window around the tracked target. In the naive version, this actor stored all R, G and B values for an entire frame before using the tracking location to determine which pixels must be highlighted. The optimisation uses tracking location to highlight pixels *on-the-fly* if their position is within the tracking window criteria.

V. Results and Discussion

The naive and optimised versions of meanshift algorithm are used to track a single target with a window size of 20×26 over 130 QCIF YUV444 frames from a standard tracking sequence from PETS dataset [22] (S2.L1). They have been run on an Intel Core 2 Quad CPU at 2.8GHz with 6Gb DDR3 memory, running the 64bit Linux 3.15 kernel, and the C was compiled with gcc 4.8.3. They are also synthesised for the Virtex 6 XC6VLX550T FPGA board. The high level synthesis (HLS) Orcc backend was used, and the VHDL was synthesised with Xilinx ISE 14.7.

The results are in Table I. It shows the CPU runtime and FPGA device utilisation for the naive mean shift version, each optimisation and then for all optimisations combined. CPU runtimes are reduced in four out of five cases. The FIFO optimisation yields a 31% shorter runtime compared the naive version using a FIFO size of 32768 instead of 16777216, a change enabled by this optimisation. Actor fusion reduced runtime by 10% and using the `repeat` RVC-CAL construct reduced runtime by 21%. Combining all optimisations gives a runtime of 1.70s using a FIFO size of 131072, a 43% runtime improvement over the naive version. The optimal mean shift version is available online [23]. The task parallel optimisation introduced six additional actors connected with 12 FIFOs, which is a potential CPU bottleneck and shared memory contention on RAM which is reflected in a 7% longer runtime.

Version	CPU		FPGA		
	Runtime	FPS	Slice LUT (/343680)	Slice registers (/687360)	BRAM/FIFO (/632)
Naive	2.97s	43.8	58973	114404	109
FIFO reduction	2.06s	63.1	82608	201510	94
Task parallelism	3.17s	41.0	169165	421464	110
Pipeline parallelism	2.96	43.9	132572	316115	53
Actor fusion	2.67s	48.7	94017	234423	85
Language refinement	2.34s	55.6	112157	260002	94
Combined	**1.70s**	**76.5**	**52884**	**127609**	**34**

TABLE I
Mean shift optimisation CPU speedup and FPGA Utilisation

The CPU optimisations to the dataflow graph result in big differences in FPGA device utilisation. There are nearly three times as many slice LUTs in the task parallelism optimisation, due to the expansion of the *displacement* actor into multiple actors. This change introduces an addition of six actors, 12 connections and 19 ports. The pipeline optimisation reduced block RAM use by 51% and there was no change in CPU runtime. The combination of all optimisations is a dataflow graph that synthesises to use 10% fewer slice LUTs, 12% more slice registers and 69% fewer block RAMs or FIFOs.

VI. Conclusions

This paper uses mean shift tracking to investigate profile driven dataflow graph transformation trading off CPU runtime performance with FPGA design space. Tuning the runtime system yielded 88% shorter runtimes and optimising the dataflow graph for CPU execution reduced runtimes by a further 43% — an overall improvement of 93%. Combining all optimisations reduced FPGA memory utilisation by 69%. We will next investigate formal methods for semi-automatic dataflow graph transformation, using predictive and runtime dataflow profiling to select optimal rewrite compositions. We will also investigate hardware constrained dataflow transformation in the context of an embedded image processing processor called IPPro we are developing in the Rathlin project.

Acknowledgement

We acknowledge the support of the Engineering and Physical Research Council, grant references EP/K009931/1 (Programmable embedded platforms for remote and compute intensive image processing applications) and the Scottish Informatics and Computer Science Alliance sponsored internship through Scottish Resource Centre for Women.

REFERENCES

[1] C. Bourrasset, L. Maggianiy, J. Serot, F. Berry, and P. Pagano, "Distributed FPGA-based Smart Camera Architecture for Computer Vision Applications," in *Proc. Int'l Conf. on Distributed Smart Cameras (ICDSC)*, Oct 2013, pp. 1–2.

[2] F. Nava, D. Sciuto, M. D. Santambrogio, S. Herbrechtsmeier, M. Porrmann, U. Witkowski, and U. Rueckert, "Applying Dynamic Reconfiguration in the Mobile Robotics Domain: A Case Study on Computer Vision Algorithms," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 4, no. 3, pp. 29:1–29:22, Aug. 2011.

[3] S. Sriram and S. S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronizations*. CRC press, 2012.

[4] J. Keinert and J. Teich, *Design of Image Processing Embedded Systems Using Multidimensional Data flow*. Springer, 2010.

[5] R. Gu, J. Janneck, S. Bhattacharyya, M. Raulet, M. Wipliez, and W. Plishker, "Exploring the Concurrency of an MPEG RVC Decoder Based on Dataflow Program Analysis," vol. 19, no. 11, pp. 1646–1657, Nov 2009.

[6] D. Stichling and B. Kleinjohann, "CV-SDF - A Model For Real-Time Computer Vision Applications," in *Proc. IEEE Workshop on Applications of Computer Vision (WACV 2002).*, 2002, pp. 325–329.

[7] M. Sen, I. Corretjer, F. Haim, S. Saha, J. Schlessman, T. Lv, S. Bhattacharyya, and W. Wolf, "Dataflow-Based Mapping of Computer Vision Algorithms onto FPGAs," *EURASIP Journal on Embedded Systems*, vol. 2007, no. 1, 2007.

[8] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "NeuFlow: A Runtime Reconfigurable Dataflow Processor for Vision," in *Proc. IEEE Computer Vision and Pattern Recognition Workshops (CVPRW)*, June 2011, pp. 109–116.

[9] D. Bailey and C. Johnston, "Algorithm Transformation for FPGA Implementation," in *Electronic Design, Test and Application, 2010. DELTA '10. Fifth IEEE International Symposium on*, Jan 2010, pp. 77–81.

[10] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-Based Object Tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 5, pp. 564–577, May 2003.

[11] J. W. Janneck, M. Mattavelli, M. Raulet, and M. Wipliez, "Reconfigurable Video Coding: A Stream Programming Approach to the Specification of New Video Coding Standards," in *Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems, MMSys 2010, Phoenix, Arizona, USA, February 22-23, 2010*, W. chi Feng and K. Mayer-Patel, Eds. ACM, 2010, pp. 223–234.

[12] P. P. Chang, S. A. Mahlke, W. Y. Chen, and W. mei W. Hwu, "Profile-Guided Automatic Inline Expansion for C Programs," *Software, Practice & Experience*, vol. 22, no. 5, pp. 349–369, 1992.

[13] A.-R. Adl-Tabatabai, M. Cierniak, G.-Y. Lueh, V. M. Parikh, and J. M. Stichnoth, "Fast, Effective Code Generation in a Just-In-Time Java Compiler," in *Conference on Programming Language Design and Implementation, Montreal, Canada, June 17-19, 1998*, J. W. Davidson, K. D. Cooper, and A. M. Berman, Eds. ACM, 1998, pp. 280–290.

[14] J. Marathe and F. Mueller, "Hardware Profile-Guided Automatic Page Placement for ccNUMA Systems," in *Symposium on Principles and Practice of Parallel Programming, New York, USA, March 29-31, 2006*, J. Torrellas and S. Chatterjee, Eds. ACM, 2006, pp. 90–99.

[15] C. Brown, M. Danelutto, K. Hammond, P. Kilpatrick, and A. Elliott, "Cost-Directed Refactoring for Parallel Erlang Programs," *International Journal of Parallel Programming*, vol. 42, no. 4, pp. 564–582, 2014.

[16] H. Yviquel, A. Lorence, K. Jerbi, G. Cocherel, A. Sanchez, and M. Raulet, "Orcc: Multimedia Development Made Easy," in *Multimedia Conference, Barcelona, Spain, October 21-25, 2013*, A. Jaimes, N. Sebe, N. Boujemaa, D. Gatica-Perez, D. A. Shamma, M. Worring, and R. Zimmermann, Eds. ACM, 2013, pp. 863–866.

[17] S. C. Brunet, C. Alberti, M. Mattavelli, and J. W. Janneck, "Turnus: A Unified Dataflow Design Space Exploration Framework for Heterogeneous Parallel Systems," in *2013 Conference on Design and Architectures for Signal and Image Processing, Cagliari, Italy, October 8-10, 2013*. IEEE, 2013, pp. 47–54.

[18] Intel, "Intel VTune Performance Analyzer," https://software.intel.com/en-us/intel-vtune-amplifier-xe.

[19] Xilinx, "ISE Design Suite," http://www.xilinx.com/products/design-tools/ise-design-suite.

[20] A. Bonenfant, Z. Chen, K. Hammond, G. Michaelson, A. Wallace, and I. Wallace, "Towards Resource-Certified Software: A Formal Cost Model for Time and Its Application to an Image-Processing Example," in *Proc. ACM Symposium on Applied Computing*, 2007, pp. 1307–1314.

[21] J. Boutellier, A. Ghazi, O. Silven, and J. Ersfolk, "High-Performance Programs by Source-Level Merging of RVC-CAL Dataflow Actors," in *Proc. IEEE Workshop on Signal Processing Systems (SiPS)*, 2013, pp. 360–365.

[22] "Performance evaluation of tracking and surveillance (PETS 2009) data set," 2009. [Online]. Available: http://www.cvg.rdg.ac.uk/PETS2009/

[23] D. Bhowmik and R. Stewart, "Mean shift RVC-CAL source code," August 2014, https://github.com/robstewart57/meanshift-rvccal.