Stream Programming on the Blackfin Architecture

Michael G. Benjamin and David Kaeli
Northeastern University Computer Architecture Research Group
Department of Electrical and Computer Engineering
Northeastern University, Boston, MA 02115
{mbenjami,kaeli}@ece.neu.edu

***Abstract*— Streaming applications can be characterized by high data parallelism, low data reuse and a high ratio between computations to memory accesses. Memory accesses can be accelerated if the are serviced by low-latency memory and also by delaying writes to main memory until all processing kernels have been executed. In this paper, we apply this approach to the embedded Blackfin BF561 dual-core processor with configurable L1 and L2 SRAM. Using a set of two common image processing filters, we achieve a speedup of 2X-4X by making use of SRAM and the two cores. We also describe an application of these filters to accelerate a video object tracker.**

I. Introduction

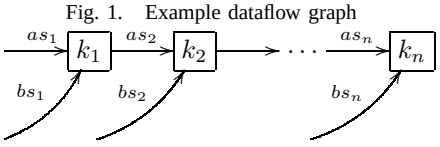
Stream applications and architectures continue to grow in popularity and demand. In this paper we apply stream programming to target the embedded Blackfin BF561 processor. The BF561 is a general-purpose RISC/DSP convergent, Harvard-style architecture supporting two cores and a memory hierarchy including configurable low-latency SRAM, scratch pad memory (SPM), and cache.

The stream model describes an application as streams of data flowing to and from computational kernels in which streams are processed. [1] By mapping kernels to cores, intra-kernel data to L1 SRAM, and inter-kernel communication to L2 SRAM, we find a performance increase in two common image processing filters and describe potential use in a video object tracker.

II. Stream Programming

Stream programs decouple memory accesses and computations into streams and kernels, respectively. Streams describe a sequence of data records which are to be processed; kernels describe a set of instructions which process the data. Figure 1 shows an example of the dataflow of a generalized stream program. For example, many image processing routines can be described using a stream of pixel blocks and a set of filter kernels.

Fig. 1. Example dataflow graph



Such a graph expresses parallelism and locality. Each data record in the stream can be processed simultaneously - providing extensive data parallelism. Inter-kernel dependencies are explicitly visible in the graph. Within kernels, independent instructions can execute in parallel; and kernels themselves can operate in parallel. During the execution of a kernel, values can be kept in local, low-latency memories; once a kernel has processed a record, intermediate results can be passed to the next appropriately sequenced kernel, again using local memory.

The stream model has recently impacted a number of new computer architecture designs and has also been shown to yield promising results in conventional general-purpose processors. [2]

III. Implementation and Results

In order for applications to make the most of the BF561, the configurable memory hierarchy and two cores must be utilized effectively. We highlight a simple edge detector using Gaussian blur and Sobel gradient convolution filters.

Ideally, each filter would only access data existing in low-latency, local memory - lessening the Memory Wall effect. But simply relying on cache is not sufficient because of the overhead and cache pollution often associated with media processing.

The dual-core Blackfin BF561 has the following memory available:

- 64 MB SDRAM main memory in 4 banks of 16MB (4x16MB), operating at 120 MHz (system clock)
- 128 KB on-chip L2 (8x16KB), at 300 MHz ($\frac{1}{2}$ core clock)
- 100 KB in-core L1, at 600 MHz (core clock), split into:
  - 32 KB instruction (16 KB SRAM; 16KB cache/SRAM)
  - 64 KB data (32 KB SRAM; 32 KB cache/SRAM)
  - 4 KB scratch-pad memories.

L3 SDRAM and configurable L1 and L2 SRAM are used as local store memories for data used within and between kernels. Table I shows the effect of various levels for kernel-local memory for a high-definition image (1920x1080 pixels) of a crowded market.

TABLE I
Use of memory hierarchy (seconds of execution time).

	L3	L2	L1
Gauss	1.89	0.58	0.33
Sobel	2.10	0.75	0.50
Gauss-Sobel	3.99	1.80	1.26

To utilize both cores effectively, we examine two approaches. The first is to map the set of all kernels to each core's instruction memory, processing a partition of data (e.g., half of the image). The second is to map some subset of kernels to cores such that the output of one core is input to the next, with inter-core communication stored in L2 SRAM. Table II compares these two approaches.

TABLE II
GAUSSIAN-SOBEL EDGE DETECTOR

Utilization Method	Execution time (sec)
Single-core L3	3.99
Two-core L3	3.14
Single-core L2	1.80
Two-core L2	0.93
Streamed	0.80

One major conclusion obtained from our results is that storing intra-kernel data in L1 and using the two-core streamed approach yields better performance.

IV. Potential Use

In order to track objects in a video sequence, we use a model image of an object of interest and compare it to each frame. The tracker used is published online from the Computer Vision group at Cornell University [3]. Using rigid models (such as a boat), the comparison between model and frame is done using the Hausdorff distance measure for point sets.

The code makes use of Gaussian smoothing and Canny edge detection to reduce a given image to a set of edges (similar to the filter kernels used above). Edges of the model are compared to frame edges and the Hausdorff distance calculated given a set of parameters: by how many pixels the image can differ from the model (and visa versa), how many pixels must match, etc. If the model matches an object in the frame above a threshold, the model image is drawn onto the frame, as in Figure 2.

Fig. 2. Source video frame

Fig. 3. Result of Video Object Tracker

This tracker stands to benefit by using the stream model and by applying the approach described above.

V. Conclusions

We have shown that for two common image processing filters, a stream program making use of the configurable memory hierachy and dual-core capabilities of the BF561 can achieve high performance. We are continue down this path to better quantify the benefits this approach versus using a similar, though more complex, video processing application.

Acknowledgment

The authors would like to thank Mimi Pichey, Giuseppe Olivadotti, Richard Gentile, and Ken Butler from Analog Devices for their generous donation of Blackfin EZ-KIT Lite evaluation boards, software, and extender boards, and for their support of this project.

References

[1] Rixner, S., Dally, W. J., Kapasi, U. J., Khailany, B., Lpez-Lagunas, A., Mattson, P. R., and Owens, J. D. 1998. *A bandwidth-efficient architecture for media processing*. In Proceedings of the 31st Annual ACM/IEEE international Symposium on Microarchitecture (Dallas, Texas, United States). International Symposium on Microarchitecture. IEEE Computer Society Press, Los Alamitos, CA, 3-13.

[2] J. Gummaraju and M. Rosenblum, *Stream Programming on General-Purpose Processors*, in Proc. MICRO 38, pages 343-354, Barcelona, Spain, 2005.

[3] D. Huttenlocher and W. Rucklidge, *A Multi-Resolution Technique for Comparing Images Using the Hausdorff distance*, in Proc. CVPR, pages 705-706, New York, NY, USA, 1993.