ISSN:0976-0172

Journal of Bioscience And Technology
www.jbstonline.com

Application-Level Performance Optimization: A Computer Vision Case Study on STHORM

Mr. Merugu Anand Kumar[1], Lankala Mounika[2], Nagam Aanjaneyulu[3], Dr. Inaganti Shylaja[4]
Assistant Professor[1,2], Associate Professor[3], Professor[4]
meruguanand502@gmail.com[1], **lankala.mounikareddy@gmail.com[2],**
anji.amrexamcell@gmail.com[3],shyalajainaganti@gmail.com[4]

Department of CSE, A.M. Reddy Memorial College of Engineering and Technology, Petlurivaripalem, Narasaraopet, Andhra Pradesh

Article Info

Received: 29-06-2022 *Revised: 18-07-2022* *Accepted: 28-07-2022*

Abstract

In the realm of embedded many-core architectures, computer vision applications are a major motivator. A harmony between computation and communication is vital for such systems to reach their full potential, however many computer vision techniques provide a highly data dependent actions that make the work harder. The developer needs access to tools for rapid and accurate application-level performance analysis in order to optimize application performance. Here, we go through the steps required to migrate and fine-tune a face recognition program for the STHORM many-core accelerator by way of the STHORM OpenCL software development kit. We isolate the key bottlenecks and separate the effects of the application, the openly programming paradigm, and the STHORM Openly software development kit (SDK). Finally, we demonstrate how these problems might be fixed in the near future to provide programmers even more leeway to enhance the efficiency of their applications.

1 Introduction

For a long time now, breakthroughs in device technology, computer design, and programming language compilers have allowed us to meet the performance goals outlined by Moore's Law [11]. Architects have met the proverbial "power wall" [2] in their pursuit of increasing the performance of single-core processors. Profiting In order to maintain linear scaling of performance within a given power budget, the industry is moving away from ever-increasing transistor counts and toward more energy-efficient multi- and many-core architectures. ITRS forecasts a near-term rise in core count for multiprocessor systems of 1.4 xs per year [6], a trend that is also seen on embedded processors for battery-powered mobile devices, where low power consumption is essential.

New hardware is sometimes inspired by specific use cases; for example, computer vision applications have been cited as a driver for embedded many-core architectures [17]. Many-core architectures may take use of the inherent parallelism of embedded vision algorithms, which otherwise would need an enormous amount of processing power, to achieve excellent performance. To maximize CPU use, it's important to strike a good equilibrium between data transfer and processing time [4]. Even highly compute-bound algorithms may become memory-bound on many-core architectures due to the increased memory bandwidth required as the number of cores grows. The processing of video streams in real-time is a resource-intensive operation, and embedded vision algorithms are no exception. Load balancing is an additional difficulty in obtaining high parallel efficiency. On many-core machines, the performance of certain computer vision applications might suffer because of their extreme

Reliance on data. The author of [15] analyzes the efficacy of a facial expression. Detecting software for a GPGPU system developed in openly. He demonstrates how data-dependent behaviour affects parallel performance on the GPU and how certain categorization processes perform better when run on the host CPU. However, branch divergence penalties are incurred by data-dependent algorithms when executed on GPUs since

their computing units are SIMD (single instruction, multiple data). In order to better handle data-dependent algorithms[10], many-core designs like Karla's Multi-Purpose Processor Array[5] and STMicroelectronics's STHORM[12] are made up of clusters of MIMD (multiple instruction, multiple data) processors.

2 STHORM

STMicroelectronics' STHORM [12] many-core processor is designed for compute-intensive embedded applications. It was developed by STMicroelectronics and the French Atomic and Alternative Energy Commission (CEA) and was given the name Platform2012 [3]. Function both alone and as an accelerator in conjunction with a host CPU.

Architecture

STHORM's scalable architecture consists of processing element (PE) clusters that may be adjusted from 1 to 4 in size, with each cluster able to house up to 16 PEs. High-level block schematic of the STHORM architecture is shown in Figure 1. Connected by a network-on-a-chip (No), the clusters boast dynamically integrated

Figure 1: STHORM Architecture Block Diagram

Capacity for per-cluster frequency and voltage scaling (DFVS). The cluster uses dual-issue STxP70 processors, which are RISC processors with a 32-bit in-order data path and a floating point unit. One more STxP70 core is used exclusively as a dedicated controller for a group of things. Each cluster's internals include 256KB of shared memory, which may be accessed by any of the cluster's processors. The memory has been divided into 32 banks with address interleaving to lessen the likelihood of collisions. Concurrent single-cycle memory access is provided via the logarithmic interconnects mesh-of-trees (Moot) [14] architecture. When two or more processors attempt to read from the same bank at the same time, a conflict occurs and only one request is processed every cycle. Although processors wait in a blocking state until their request is fulfilled, a round-robin approach guarantees everyone gets a fair shot at the scarce resources.

Openly Programming Model

Openly 1.1[7] is one of the three parallel programming frameworks that STHORM supports at varying abstraction levels [13]. Although it was created for use with GP-GPUs in a heterogeneous setting, it is now also widely used to program embedded multi- and many-core computers. In addition. The idea behind it is that a host processor does the actual work of the program while a many-core compute device handles the heavy lifting of the calculation kernels. Multiple compute units, each with many processing components, make up the computer itself. The burden of the kernel is broken up into smaller chunks called "work-groups," each of which consists of many individual tasks. The OpenCL runtime organizes the tasks to be performed by the compute units and their individual processors. Barriers, locks, and atomic operations are just some of the synchronization techniques at your disposal. There are four possible locations for data storage: public, internal, local, and private. Global data buffers are allocated in the L3 host memory by the STHORM Openly runtime, whereas constant data are stored in the L2 memory. L1, the cluster's shared memory, is where data is stored that is only accessible by a select few. In order to move information across these buffers, the Openly API recommends using asynchronous work-group copy methods, which initiate a DMA transfer for the full work-group. In STHORM's openly implementation, the idea of a work-item copy is introduced, in which separate tasks initiate DMA transfers on their own.

3 Face Detection Performances

Optimization

From a sequential face detection application, we construct a parallel implementation in this case study. The STHORM architecture is then optimized for the application via experimentation.

Application Description

The study's foundational sequential face identification technique is based on the work of Viola and Jones [16], which employs a detector made up of a classifier cascade of Hear-like features that was trained using Gadabouts. The accuracy of the classifier is checked often using scanning scale insensitivity is achieved by applying the window approach on a pyramidal picture. The detector constructs an integral picture at each tier of the image pyramid to speed up the calculation of features. Classifier cascades are structured as a hierarchy of steps. At each tier, we calculate a response based on a set of characteristics and check it against lower and higher boundaries. Detection is continued and the window is accepted if the stage answer is within tolerances; otherwise, it is terminated. An acceptable window and a positive detection result are supplied if and only if all classifier cascade steps are successful. Detections in close proximity to one another are combined into a single detection and given a score depending on the number of original detections that were combined. The detections with the lowest scores are thrown out in the final filtering phase.

Methodology

The speedup of a parallel application is known to be constrained by the execution time of its sequential portion [1] according to Amdahl's law. In order to reduce the overall execution duration of the program, we plan to parallelize the most time-consuming parts of it. To begin, we profile the reference sequential program using a STxP70 cycle-approximate simulator and then sort the functions by their total execution time to determine which ones take the most time. Our first set of parallelization candidates are the highest-rated functions.

Clear inputs and outputs are defined for these functions when they are refectories into openly kernels. The next step is to optimize the load distribution by sizing the parallel task such that computation and communication are overlapping. Important choices for parallel implementation include: • the parallel granularity, for example, image frame, line, window, or pixel; • the workload distribution strategy, for example, static or dynamic workload distribution; • the working data placement, for example, globally, locally, privately, etc.; • the data transfer strategy, for example, individualistic or collaborative.

Hotspot Analysis

By using a cycle-approximate STxP70v4 simulator, we have analyzed the reference sequential implementation of the face recognition application to locate the bottlenecks. Here, we refer to a QVGA picture with 24 faces as the test image, which will serve as the basis for our study. Results from the test image profiling are shown in Table 1, organized by algorithmic step and rated according to their total execution times. The classifier cascade, integral picture creation, and the scalar were shown to be the three primary bottlenecks. These three steps account for almost majority of the runtime; hence they were chosen to be parallelized. As an example, Table 1 displays the profiling statistics for our sequential face identification program using the worst-case picture (with 24 faces) from our testing database. We acquire these outcomes using a cycle-accurate STxP70v4 emulator in a dual-issue arrangement with a clock frequency of 500 MHz

Application Phase	Cycles	Time (ms)	% of Total
Detection Cascade	61,879,829	123.8	56.8%
Integral Image	27,159,728	54.3	24.9%
Scaler	14,627,913	29.3	13.4%
Other	5,196,975	10.4	4.8%
Total	108,864,445	217.7	100.0%

Performance Measurements

The worst-case QVGA picture (24 faces) in our testing database is shown in Table 2 along with the measured kernel times for both the group and individual efforts. The prototype board's findings are contrasted with those obtained from the simulator. Effectiveness is reflected in the length of time it takes to process the kernel. Time spent in runtime includes asynchronous data transfer time, as well as time spent waiting for events and on barriers, and time spent in kernel prolong and epilogue accounts for overheads in launching and terminating kernels. These findings demonstrate that synchronization hurdles, which account for around a third of total kernel time, have a detrimental effect on the collaborative version. In particular, the lower synchronization overhead afforded by the individualistic variant improves performance on both the simulator and the board. The findings of the simulator indicate that the kernels prolong and epilogue is the primary causes of inefficiency.

The simulator tests show that the collaborative method requires less time for the kernel to process data (6.9 ms) (7.8 ms). The prototype findings, however, reveal an inversion: although the individualistic technique has a lower processing time (44.9 ms), the collaborative approach has a greater one (12.4 ms). Waiting for events accounts for less than one percent of simulator time but as much as half of kernel time on the prototype board. This means that despite the fact that the STHORM simulator used is cycle-approximate, a significant discrepancy exists between the simulator's output and that of the prototype board.

Table 2: Execution time for the face detection application on STHORM simulator and prototype, for 4 clusters of 16 processing elements at 500 MHz % are relative to the total time.

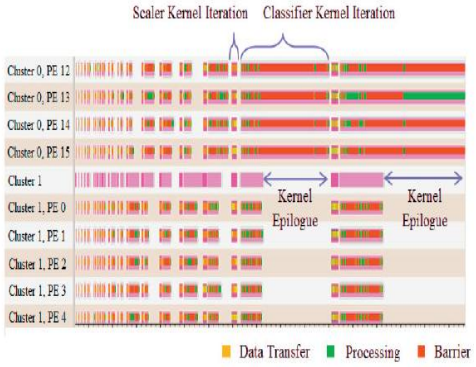
	Simulator				Prototype			
Data Transfer Strategy	Collaborative		Individualistic		Collaborative		Individualistic	
Kernel Processing Time	6.9 ms	21.5%	7.8 ms	34.7%	44.9 ms	26.3%	12.4 ms	13.9%
Kernel Prolog & Epilog	13.6 ms	42.2%	14.1 ms	62.4%	47.9 ms	28.0%	32.4 ms	36.1%
Time Spent in Runtime	11.7 ms	36.3%	0.7 ms	2.9%	78.0 ms	45.6%	44.8 ms	50.0%
- Asynchronous Copies	0.3 ms	0.9%	0.3 ms	1.2%	0.0 ms	0.0%	1.6 ms	1.8%
- Waiting for Events	0.4 ms	1.1%	0.3 ms	1.2%	30.2 ms	17.7%	43.2 ms	48.2%
- Waiting on Barriers	11.1 ms	34.3%	0.1 ms	0.5%	48.3 ms	28.3%	0.0 ms	0.0%
Total Time in Kernels	32.2 ms	100.0%	22.7 ms	100.0%	170.9 ms	100.0%	89.7 ms	100.0%

4. Detailed Analysis

According to Table 2, the simulator's kernel prolongs and epilogue account for the majority of the overall runtime. For example, in Figure 2, we can see that the kernel prolong and epilogue are executed throughout the collaborative approach's execution. Takes into account not only the time required to start and stop kernels, but also any downtime the cluster may experience between work-group executions. These often result from the interaction with the host processor or from an imbalance in the workload between different task groups as a result of data-dependent behaviour.

We see longer kernel processing times on the prototype board compared to the simulator. The kernel processing time is what really matters, and the STHORM simulator doesn't do a good job of modelling memory access timings, which are included in the kernel processing time (save for DMA transfers). Memory inconsistencies are not simulated in the simulator. This, in addition to the prototype board's increased latency and lower bandwidth to the global memory, results in a significant discrepancy between the two. In addition, because the critical path processors will take longer to reach the barriers due to the higher processing time, more time will be spent waiting on the barriers. This is because synchronization barriers need all processors to reach the barrier call before proceeding. The biggest discrepancy between the simulator and the actual product is the amount of time spent waiting for events. The runtime will return an event handle once you initiate a DMA transfer using a non-blocking asynchronous copy. It's hardly the only task processors can handle. Asynchronously, and then wait for the transfer to finish using a wait call on the event handle. Thus, the elapsed time between events in our studies is equivalent to the elapsed time between non-blocking DMA transfers. Due to the large discrepancy, it may be concluded that the simulator does not faithfully replicate the DMA transfer times seen in the prototype.

Unfortunately, the STHORM SDK does not include any adjustable options to address this issue. For this reason, the presented data does not allow for an exact estimation of the total time wasted as a result of load imbalance. However, as shown in Figure 2, a significant portion of the kernel prolong and epilogue is related to inter-work-group imbalance, and the amount of time lost due to inter- and intra-work-group load imbalance could amount to as much as 70% of the total kernel time for the collaborative approach on the simulator.



Piece of a trace for the face detection collaborative technique on STHORM (Figure 2). Displays the kernel execution traces of many picture pyramid iterations for a single image frame. Maximizing Efficiency in an Application From left to right: Schwambach, Cleyet-Merle, Issard, and Mancini A large degree of inter-work-group imbalance persists, despite the fact that intra-work-group imbalance is almost eliminated using the individualistic strategy. The collaborative technique is less efficient because of the greater memory conflict penalties and lower load balancing it introduces, but the individualistic approach on the board still produces superior performance despite the longer delays in data transfers.

5. Conclusion

Current architectures are being pushed to their limits by the rising popularity of computer vision applications. However, vision algorithms frequently have very variable data-dependent execution durations, which make it difficult to parallelize them. Unequal distribution of effort. The latter is a major cause of system inefficiency and has a detrimental effect on parallel efficiency. It is up to the programmer to do some juggling in order to restructure the algorithm and reduce the imbalance, since current data-parallel programming models such as Openly are unable to effectively schedule work to fill-in the gaps caused by this imbalance. In order to better allocate tasks on the fly, a more dynamic programming paradigm is required. Finding the right equilibrium between processing and communication is crucial for getting the most out of a parallel platform. Internal memory conflicts, external communication latencies and throughputs, and interactions with the host processor are not modelled adequately by the current generation of STHORM many-core simulation tools. It is consequently impossible to utilize the simulators for performance estimate or adjustment at the application level. Because of the greater setup work and the limits of the prototype itself, exploring the design space is constrained when only physical prototypes are available. In addition, a completely functional prototype is frequently not available until after crucial architectural choices have been made throughout the design process of a new embedded system. Therefore, it is crucial to have simulation platforms that can offer accurate timing data for the host, accelerator, and memory subsystems in order to conduct accurate performance evaluation and optimization of the application at an early stage in the development cycle.

References

[1] G. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. Proceedings of the April 18-20, 1967, spring joint computer conference, pages 1–4, and 1967.
[2] K. Casanova, J. Wawrzynek, D. Wessel, K. Yelick, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, and K. Sen. A view of the parallel computing landscape. Communications of the ACM, 52(10):56, Oct. 2009.
[3] L. Benini, E. Flamand, D. Fuin, and D. Melpignano. P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator. In Proceedings of the Conference on Design, Automation and Test in Europe, pages 983–987. EDA Consortium, 2012.
[4] K. Czechowski, C. Battaglino, C. McClanahan, A. Chandramowlishwaran, and R. Vuduc. Balance principles for algorithm-architecture co-design. USENIX Wkshp. Hot Topics in Parallelism (HotPar), pages 1–5, 2011.
[5] B. D. de Dinechin, P. G. de Massas, G. Lager, C. L´eger, B. Orgogozo, J. Reybert, and T. Strudel. A Distributed Run-Time Environment for the Kalray MPPA-256 Integrated Manycore Processor. 2013 International Conference on Computational Science. Procedia Computer Science., 18(0):1654–1663, 2013.
[6] ITRS. International Technology Roadmap for Semiconductors. http://www.itrs.net, 2011.
[7] Khronos OpenCL Working Group. The OpenCL Specification. Version 1.1, Document Revision: 44, 2010.
[8] Khronos OpenCL Working Group. The OpenCL Specification. Version 2.0, Document Revision: 19, 2013.
[9] B. Kisacanin. Integral Image Optimizations for Embedded Vision Applications. Image Analysis and Interpretation, 2008. SSIAI 2008. IEEE Southwest Symposium on, (1):181–184, 2008.
[10] D. Melpignano, L. Benini, E. Flamand, B. Jego, T. Lepley, G. Haugou, F. Clermidy, and D. Dutoit. Platform 2012, a many-core computing accelerator for embedded SoCs: performance evaluation of visual analytics applications. In Proceedings of the 49th Annual Design Automation Conference,pages 1137–1142. ACM, 2012.