Analysis of Sorting as a Streaming Application

Greg Galloway, *ggalloway@wustl.edu* (A class project report written under the guidance of Prof. Raj Jain)

Download 

Abstract Expressing concurrency of applications has always been a challenging and error-prone task, yet effective use of multi-core processors and hardware implementations requires that the concurrency of applications be well represented and understood. One approach to the expression of concurrency is streaming. In this paper I express the classic problem of sorting in the streaming paradigm, and explore various algorithmic and architectural design parameters.

Keywords: Sorting, Auto-Pipe, Sorting, X-Language, Performance Modeling

Table of Contents:

1 Introduction

With multi-core processors becoming the standard for general-purpose computing, there has been an increase of interest in parallel processing topics, specifically parallel algorithms. A relatively new approach is the stream programming paradigm. Expanding upon the traditional shared-memory programming paradigm and the message-passing programming paradigm, stream computing has been introduced as a more data-centric approach to authoring parallel applications.

There are many languages that support stream computing. It has been argued that languages expressing streams represent a better mechanism for reasoning about concurrency than thread-based approaches. The X Language [Franklin06, Tyson06] is a stream-based coordination language for hybrid systems (systems with architecturally diverse components, such as FPGAs, GPUs, etc.).

This paper describes the classic sorting problem in terms of a streaming computation. Variations examined include the degree of pipelining vs. data parallelism, the performance of the sorting application when

deployed on multi-processors, and shared vs. common memory systems. The Auto-pipe design environment was used to generate the data, which is also briefly described.

---

2 Auto-Pipe Streaming Application Development Environment

Auto pipe is a performance-oriented development environment for hybrid systems. It concentrates on applications that are represented primarily as dataflow graphs and is especially useful in dealing with streaming applications placed on pipelined architectures. In Auto-Pipe, applications are expressed in the X-language [Tyson06] as acyclic dataflow graphs. These graphs contain individual tasks called "blocks" and are connected with interconnections called "edges". Blocks represent the computational tasks, and edges indicate the type and flow of data between the blocks.
The implementations of the blocks are written in various languages for any subset of the available platforms. Currently Auto-Pipe supports C for general-purpose processors, HDL (Hardware Description Language) for FPGAs (Field Programmable Gate Arrays)[Gayen07], and assembly for network processors and DSPs (Digital Signal Processors) [Chamberlain07]. Auto-Pipe further provides an extensible infrastructure for supporting a wider variety of interconnection and computational devices, simulators, and languages.

2.1 Analysis Techniques

Built in analysis is lacking in the toolset itself, so making use of a good reference on systems performance analysis [ex: Jain91] assist in making the best use of the results. Anyone who makes use of computer systems should be able to analyze the performance of the systems they design, and be able to state the requirements. There is a strong need to be able to consider the provided alternatives and to choose the solution that best meets their needs [Jain91]. More on this topic will be discussed in a later section.

---

3 Sorting Application

Frequently large data sets are split into smaller groups before being sorted, and then merged in a different step. Split blocks are used to quickly divide an incoming data stream, routing half of the incoming records to another block, usually another split or a sort. After each group of records is sorted, they are then routed to a merge block, which uses the simple merge sort to recombine the data. The particular sorting algorithm used by the sort blocks is not significant; however in the results presented, comb sort [Lacey91] was used. Comb sort was selected due to it being a reasonably efficient O(*nlogn*) in-place algorithm. The process of splitting the data stream up can be done in a few different ways, shown below. The first figure demonstrates using two sort blocks in combination with a single merge and split block. The second figure shows a topology that uses four sort blocks, and figure 3 shows a topology consisting of eight sorts.

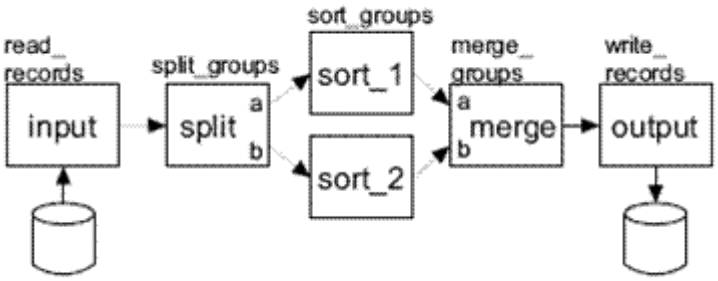

Figure 1: Example Two Sort Topology

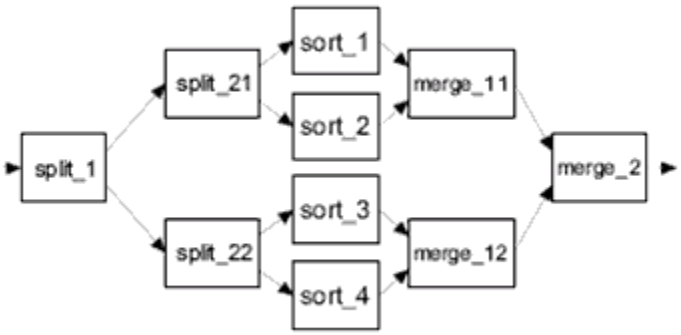

Figure 2: Example Four Sort Topology

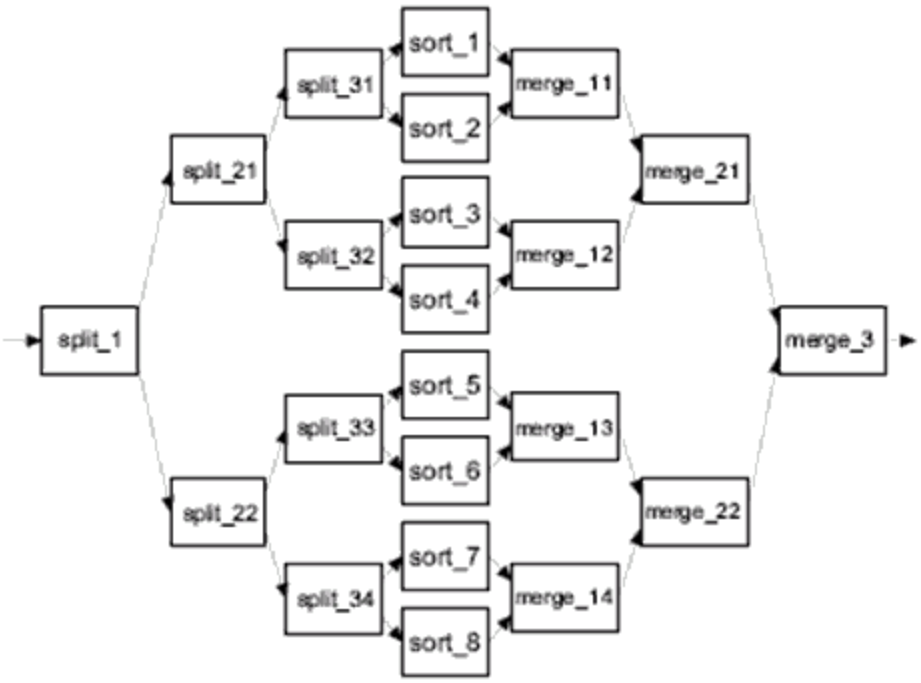

Figure 3: Example Eight Sort Topology

3.1 Benefits of Streaming Approach

The power of expressing the sorting application in these ways is that the computation supports a streaming data model, where pipelining is used to enable the sort blocks to work on one group of records while the merge blocks concurrently work on another group. In this model, pipeline-based parallelism and data parallelism are both explicitly represented[Franklin06].
There are a few clear benefits to authoring applications using this approach. First, it is possible to build a library of blocks that can be re-used [Tyson05]. This would enable application development primarily in the coordination language without requiring implementation of individual blocks. Second, the data movement between blocks is not something that needs to be explicitly managed by the application developer. The X-coordination language already determines where data is to be delivered [Tyson06]. Third, algorithm decomposition is known to the system, making it straightforward to adjust the mapping. This makes the process of distributing the blocks to various compute resources much easier. Finally, the streaming data paradigm is a natural approach to reasoning about the correctness of an application, which reduces the chances of making programming errors [Gayen07]. Also, having this framework in place avoids the complexity of correcting a synchronization error due to a missing lock in a shared-memory program.

3.2 Mappings

The Auto-pipe system supports mapping of application blocks to a variety of computational resources, such as processors, FPGAs, etc. Additionally it allows the mapping of the application edges to different interconnect resources. For this analysis, the mapping is constrained to cores within a chip multiprocessor and uses shared memory as the underlying interconnect resource. The blocks mapped to the processor are expressed in C/C++, using a model for an x86 processor core. Performance was predicted for two, four, and eight sort application topologies (i.e., those shown in the previous figures), executing on up to eight processor cores. Table 1 shows the mappings used. No assumptions are made that the mappings selected were optimal, but they are reasonable in that they evenly divide the sort blocks (the most computationally expensive) as evenly as possible across the processors.

Table 1. Mappings of blocks to processors.

No. of Sorts	No. of Processors	Processor	Blocks
2	1	1	all blocks
2	2	1	1 split, 1 sort
		2	1 sort, 1 merge
2	4	1	1 split
		2,3	1 sort each
		4	1 merge
4	1	1	all blocks
4	2	1	all splits, 2 sorts
		2	2 sorts, all merges
4	4	1	all splits, 1 sort
		2	1 sort
		3	1 sort, 1 merge
		4	1 sort, 2 merges
4	8	1	all splits
		2,3,4,5	1 sort each
		6,7,8	1 merge each
8	1	1	all blocks
8	2	1	all splits, 4 sorts
		2	4 sorts, all merges
8	4	1	5 splits, 2 sorts
		2	1 split, 2 sorts, 1 merge
		3	1 split, 2 sorts, 3 merges
		4	2 sorts, 3 merges
8	8	1	3 splits, 1 sort
		2	2 splits, 1 sort
		3,4	1 split, 1 sort
		5,6	1 sort, 1 merge
		7	1 sort, 2 merges
		8	1 sort, 3 merges

4 Performance Results

The experimental results are generated from simulations sorting 64-bit records (32 bits of key and 32 bits of tag). The input block reads one million records from a file, then sends them to the split block. All data is delivered via 256 record messages. One of the primary figures of merit is the latency to complete the sorting of these one million records (measured from the time the first element is provided to the initial split block to the time the last element is output from the final merge block). The results are from the performance evaluation subsystem within Auto-pipe, known as X-Sim. The use of X-Sim allows us to explore various hardware configurations that are not physically available, such as higher processor counts and FPGAs. X-Sim has been shown to be highly accurate in validation experiments [Gayen07].
Starting with the 2-sort topology of Figure 1, Figure 4 shows an event timeline of the sorting application mapped to four processors and assuming no delay in any of the communication links. The three event classifications are *avl* for available, *in* for input, and *out* for output. The avl event indicates the time when a data value is available at the input of a block. The in event indicates the time when the data is consumed by the block, and the out event indicates when data is produced at the output port of the block. Communication delays are modeled at a fixed delay, which is set to zero for the first simulation. This means that the out timestamps for one block match the in timestamps for the preceding block(s).

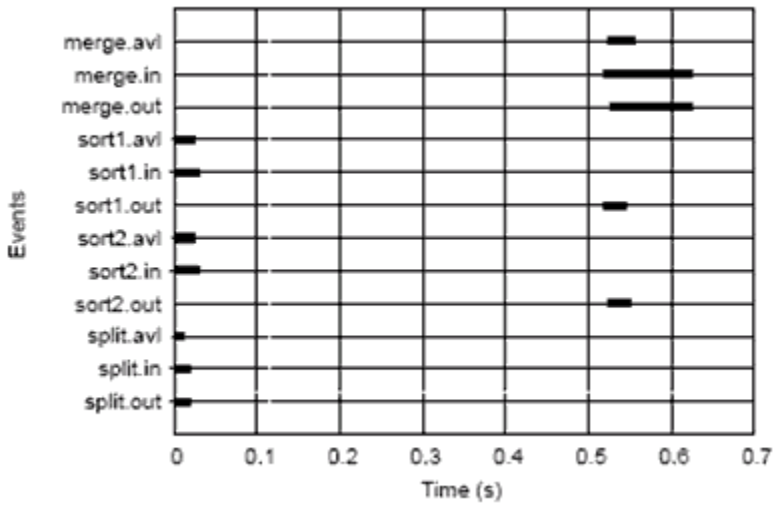

Figure 4: Timeline for 2-sort, 4-processor mapping with zero communication delay

In Figure 5, we see the timeline for a 4-sort, 4-processor topology. Although we can observe many traits, and a large amount of information is presented, these graphs do little to give precise insight into the effects of various parameters. Generalizations can be made, but without in depth analysis, this data is being used to generate pretty pictures.

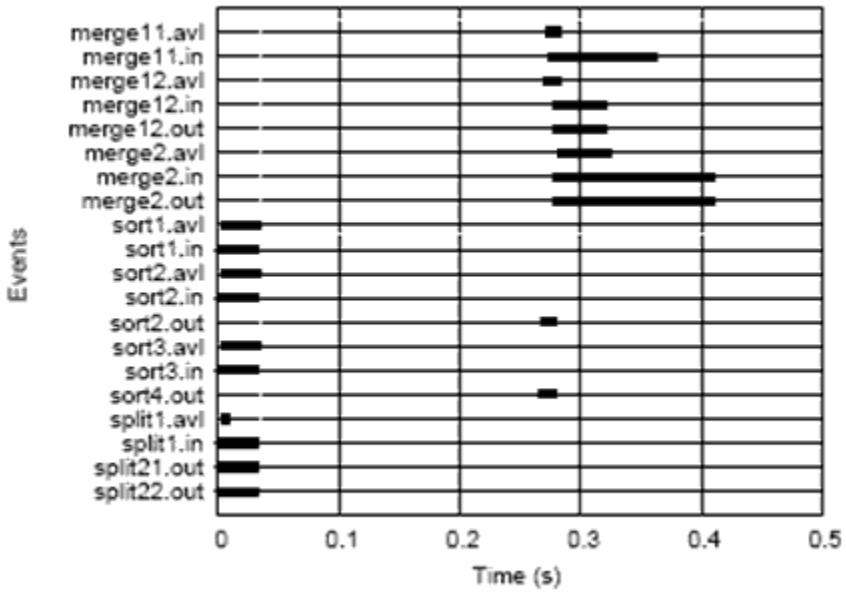

Figure 5: Timeline for 4-sort, 4-processor mapping with zero communication delay

4.1 Benefits and Drawbacks of Initial Presentation

There are many reasons why the timeline plots are useful. For one, they contain almost the full set of data. Although the events create solid regions, it comes as close as realistically possible to displaying all events on a single graph. The interactions between blocks can be clearly seen, allowing these graphs to be a good check for plausibility. The time required for processing the whole data set can be approximated and compared for different trials. So, although useful for many purposes, displaying the information this way is not ideal. Representing the data in this way gives access to volumes of information, but at what costs? There are not many extremely clear assumptions one can make that are specific enough to be of use. Exact numeric metrics cannot be drawn from the graphs. Also, the frequency of events prevents having both a timeline that contains all events and a granularity small enough to inspect specific situations. Conclusions drawn are either educated guesses or approximations. Also, the data is presented without any pre-analysis. It's simply printed out in its entirety. While this presents the whole picture, it leaves little opportunity, in and of itself, for further inspection.

4.2 Application of Performance Techniques

Many performance techniques can be applied at each step in the simulation process. Selecting metrics and evaluation techniques are key steps in any performance evaluation project [Jain91]. For the sorting application, there was little attention paid to the analytical modeling, as the simulation had been well-verified, and could be adjusted as the project progressed. Measurement should also be done to demonstrate that the simulations are both useful and accurate.
The workload was guessed at initially, with values ranging from a few records (10-100) to a few million. Through various trials it was determined that one million records gave consistent results without causing undesired side-effects. Considerations included disk access, reliability, repeatability, time-per-trial, and how the records would be processed. If the number of records exceeded a given section of memory, there was the possibility of invoking additional delays due to caching. If the number of records was too small, small delays due to seemingly random system processes would play a part in the results. Also, the number of records should be chosen to divide well into each topology, this was to allow each split block to evenly divide each group of records.

4.3 Improvements to Analysis

Some performance techniques were simplified or omitted entirely. Workload selection was not approached artistically, but rather as a guess-and-check sort of problem. Another glaring issue was the specific selection of metrics. All trials should have been approached in a way that would generate specific numeric results. Storing all the intermediate data is unrealistic, and simply keeping a graph for each is insufficient. If metrics were determined, they would accurately represent the most relevant information provided by each trial.
The processing of the data is another focus that could have been more thoroughly explored. Summarizing the trials by single numbers would allow application of probability and statistical concepts. This would further allow the comparison of the systems with a good approximation of how representative it is of the population. Comparison of multiple alternatives would draw more concise conclusions, and help determine appropriate sample sizes. Finally, factors were not appropriately considered and their influence extracted from the simulations. This causes the results to be too generalized to be significant, and prevents some of the deeper insights from being clear.
Applying regression models would allow prediction of results that were not run due to time constraints. Using these regression models, visual verification helps lead to appropriate conclusions, and helps avoid the problem of having a high coefficient of determination with a poor model. Using the techniques of experimental design, the initial trials could be run with more specific goals in mind. Rather than days or weeks of attempting to run every simulation, a subset of simulations could be run and still yield the results being sought. Should this reduced set of trials be insufficient, the results will help determine which trials to do next. Saving time would allow the more important trials to be replicated, giving more statistically sound results.
Finally, there were many considerations to be made when inspecting the simulation. The transients were clipped from the data, so the more pertinent results could be considered. The data set used for each trial was exactly the same, so no randomness considerations played a part in the trials. The simulation language had been thoroughly utilized and its soundness proven.

---

5 Applied Performance Techniques

Given such a variety of factors and values, it can be difficult and time consuming to run every possible trial. The factors are listed in table 2 below. Due to the fact that the data set was held constant, and is common to all trials, it does not need to be considered as a factor. For this analysis, a communication delay of zero will be considered, the optimal mapping will be used, and processor counts will be constrained to one, two and four. Sort counts of two, four and eight will be used.

Table 2: Factors and Their Values

FACTOR	POSSIBLE VALUES
Number of Sort Blocks	1, 2, 4, 8
Processor Count	1, 2, 4, 8
Mappings	A, B, C
Communication Delay	0us, 5us, 10us, 15us, 20us, 25us, 30us

5.1 Result Analysis

The table below displays the overall time required to sort the one million record set with zero communication delay.

Table 3: Overall Sort Processing Times (in seconds)

Sort Blocks	Processor Count			Row Calculations		
	One	Two	Four	Sum	Mean	Effect
Two	1.153731718	0.636823483	0.622245268	2.4128	0.804267	-0.00136
Four	1.192156942	0.706137322	0.408802067	2.307096	0.769032	-0.0366
Eight	1.258003719	0.778717964	0.494057253	2.530779	0.843593	0.037962
Column Sum	3.603892379	2.121678769	1.525104588	7.250676		
Column Mean	1.20129746	0.707226256	0.508368196		0.805631	
Column Effect	0.395666822	-0.098404381	-0.297262441			

As expected, if only two sort blocks are used, increasing the processor count to four serves little purpose.

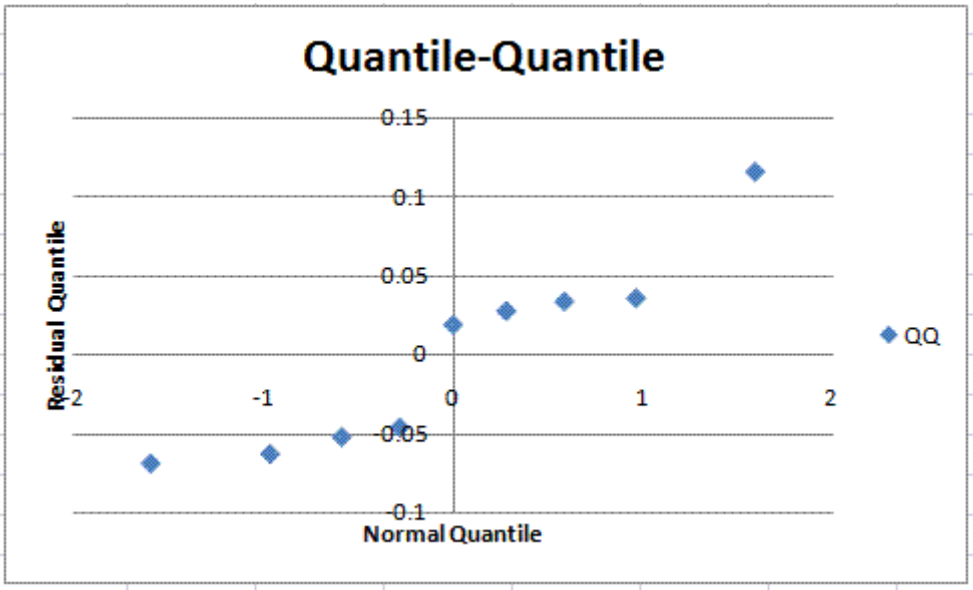

Figure 6: Quantile-Quantile plot

6 Summary

Based on the QQ plot, it can be seen that this model is perhaps not ideal. Not having a linear trend in the data means that the model used was not sufficient, and that the results drawn may be inconclusive. This could be due to not running enough trials, not considering the right design, or simply choosing the wrong factors to analyze.

X-Sim is a simulator that has built to be used with the Auto-Pipe system. It allows users to simulate their applications on a given set of heterogeneous resources before deployment, allowing both correctness testing and performance analysis. The comprehensive gathering of trace data provides opportunities for the user to analyze and better understand their application's behavior [Gayen07]. However, without proper analysis skills, the results may be misinterpreted or misrepresented.

References:

[Franklin06] M. A. Franklin, E. J. Tyson, J. Buckley, P. Crowley, and J. Maschmeyer.
"Auto-pipe and the X language: A pipeline design tool and description language"
In Proc. of Intl. Parallel and Distributed Processing Symp., Apr. 2006.
http://www.arl.wustl.edu/~pcrowley/ftbcm06.pdf

[Gayen06] Gayen et. al., *"X-Sim: A Federated Heterogeneous Simulation Environment"*,
In Proceedings of 10th High Performance Embedded Computing (HPEC) Workshop, September 2006, pp. 75-76
Available at: http://sbs.cse.wustl.edu/pubs/gtfcc06.pdf

[Jain91] R. Jain,
"The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling",
Wiley- Interscience, New York, NY, April 1991.

[Tyson06] Tyson, *"Auto-Pipe and the X Language: A Toolset and Language for the Simulation, Analysis, and Synthesis of Heterogeneous Pipelined Architectures"*,
Master's Thesis, Washington University Dept. of Computer Science and Engineering, August 2006.

[Chamberlain07] R. D. Chamberlain, E. J. Tyson, S. Gayen, M. A. Franklin, J. Buhler, P. Crowley, and J. Buckley.
"Application Development on Hybrid Systems,"In *Proc. of Supercomputing (SC07)*, Nov. 2007.
http://sc07.supercomputing.org/schedule/pdf/pap442.pdf

[Lacey91] S. Lacey and R. Box. A fast, easy sort. Byte, 16(4):315.ff., Apr. 1991. http://www.math.utah.edu /pub/tex/bib/toc/byte1990.html#16(4):April:1991

[Tyson05] Tyson, *"X Language Specification 1.0"*,
Washington University Dept. of Computer Science and Engineering Technical Report WUCSE-2005-47

[Gayen07] S. Gayen, E. J. Tyson, M. A. Franklin, and R. D. Chamberlain. "A federated simulation environment for hybrid systems"
In *Proc. of 21st Int.l Workshop on Principles of Advanced and Distributed Simulation*, pages 198.207, June 2007. http://portal.acm.org/citation.cfm?id=1249025

Acronyms:

COV Coefficient of Variation
DSP Digital Signal Processor
FPGA Field Programmable Gate Array
GNU GNU's Not Unix
GPP General Purpose Processor
GPU Graphical Processor Units
HDL Hardware Description Language
Q-Q Quantile-Quantile

Last modified on November 24, 2008
This and other papers on latest advances in performance analysis are available on line at
http://www.cse.wustl.edu/~jain/cse567-08/index.html


Back to Raj Jain's Home Page