Stanford University

EE482C
Advanced Computer Architecture & Organization

Project Report

Multi-Node Programming

June 6, 2002

Henry FU

Harn Hua NG

Yeow Cheng ONG

1. Project goals

In this project, methods for mapping stream programs over multiple stream-processing nodes are developed and evaluated. Specifically, these methods are used to partition data and/or instructions across the nodes, communicate data/state information to coordinate the processors and perform load balancing.

1.1 Application

The example chosen for this project is that of IP Packet Routing – Longest Prefix Address Matching. It is a common application used in routers in the Internet to assign next-hop addresses in packets' paths to their final destination.

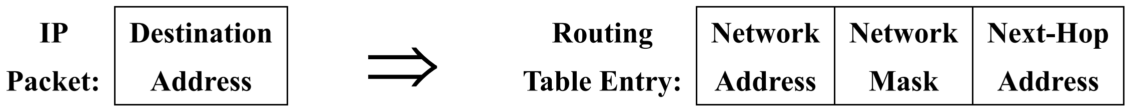

Figure 1a. Mask and Match to Find Corresponding Next-Hop Address

1.2 Findings

IP Address Longest Prefix Matching is a suitable applications for illustrating multi-node Stream processing as packet traffic lends itself to Data-Level, Thread-Level and Instruction-Level parallelism. For pipelined configurations, the speedup increases linearly with the number of processors, and for parallel configurations, there are synchronization and communication overheads which cause the rate of increase of speedup to decrease as the number of processors increases.

2. Implementation

2.1 Metric

The execution time of a single Stream Processor configuration is compared against that of a multi-node configuration. Speedup is plotted against the number of Imagines in the configuration to see the effects on execution time.

2.2 Setup

To implement the IP Routing application, the development environment provided by the Imagine Stream Processor Project at Stanford University is used. The code is written in StreamC and KernelC and simulated in the IDebug simulator. A performance measurement feature known as Profiling is used to record the execution times in cycle counts.

Consider the baseline case in *Figure 2a*, where only 1 host process and 1 Imagine are used.

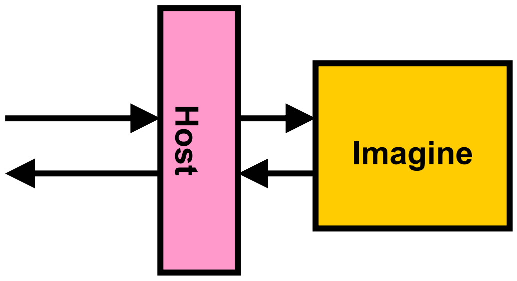

Figure 2a. Baseline configuration

In this configuration, there is only one parallel lane and one pipeline stage. All routing table entries and all data packets are given to a single Imagine processing node. All multi-node performance results are normalized to that of this baseline case.

Both *parallel* and *pipelined* configurations are tested for each subsequent configuration.

Other multi-node pipelined version includes two Imagines with two pipeline stages, as shown in *Figure 2b*, where the routing table is split into two, given to each Imagine.

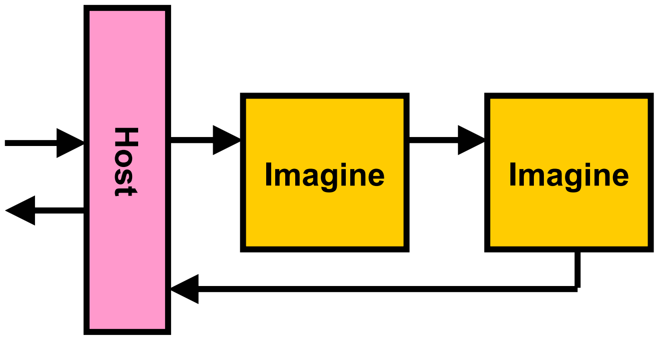

Figure 2b. 2 Imagines, 2 pipeline stages

Both Imagines process the destination addresses in the packet traffic before the correct next-hop address is computed.

The case with four Imagines and four pipeline stages is shown in *Figure 2c* below, where the routing table entries are split into four parts.

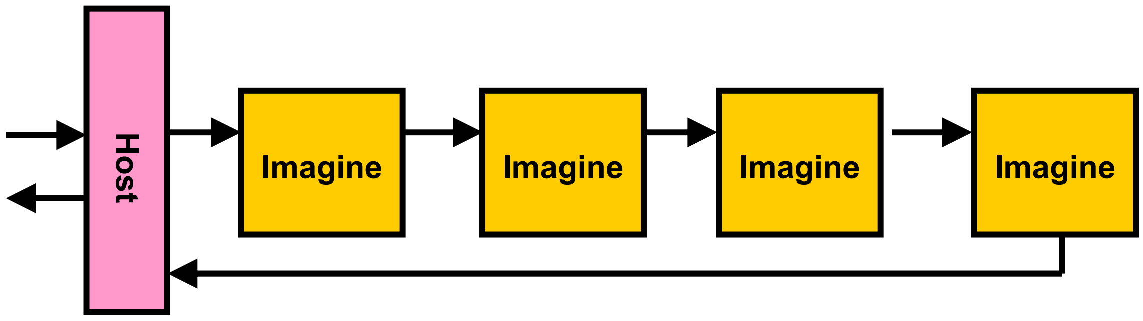

Figure 2c. 4 Imagines, 4 pipeline stages

The parallel versions of a multi-node configuration includes two Imagines with two parallel lanes (*Figure 2d*), and four Imagines with four parallel lanes as in *Figure 2e*, where data traffic is split into two and four parts respectively. In the parallel version, output data from all the Imagines are sent to the last Imagine to be combined before producing the final output.

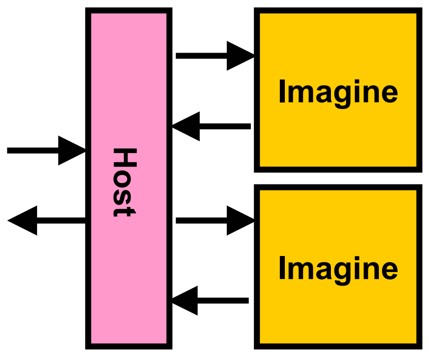

Figure 2d. 2 Imagines, 2 parallel lanes

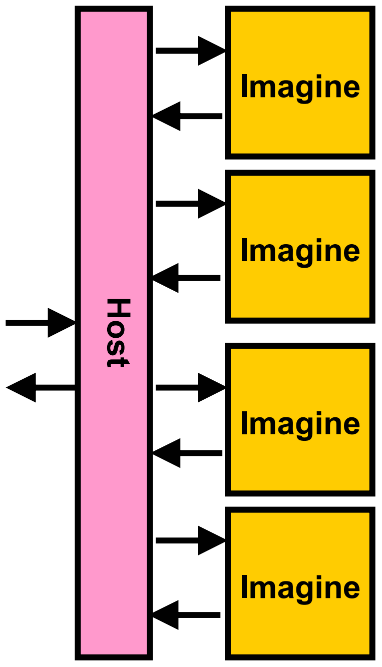

Figure 2e. 4 Imagines, 4 parallel lanes

Our code was divided into the following modules:

- (StreamC) Main function in Host Processor
- (KernelC) Address Matching kernel
- (KernelC) Kernel to combine outputs from other Imagines in a parallel configuration

2.2 Test Data

Routing table entries are captured from a major router in an ISP. The targeted ISP is known as *ner-routes.bbnplanet.net* and 119,967 routing table entries are obtained. A subset of these entries is randomly selected as test data. The corresponding result files are generated using a program written in C for easy comparison with the output produced by the Imagine application.

2.3 Longest Matching Prefix Algorithm

In the kernel, a batch of eight data packets is given to the eight clusters present in one Imagine processor, i.e. one packet per cluster.

A software kernel loops through all the routing table entries, and logically left-shifts the mask portion until it becomes zero, to find the maximum mask length. After that, routing table entries in batches of eight are communicated to all eight clusters, along with their respective mask lengths. Each cluster has all the eight routing table entries now. The long prefix match is then found by using the following logic:

Match = (pkt destination addr AND mask) XOR each routing table entry's dest. addr

Figure 2f. Mask and Match Logic

If the resultant match is zero, then the mask length is compared to the latest longest match mask length for that particular destination address, and if this new mask length is greater, its corresponding routing table entry's next-hop replaces that of the previous next-hop address.

At the end of the routing table stream, a stream of length eight, containing longest mask length and next hop addresses for the eight data packets is output. This stream is then passed to the next node in line as input. The very last Imagine in the line therefore contains the correct next hop addresses for the eight data packets.

3.1 Results & Findings

Pipelined Algorithm								
Execution Time								
# Packets	# Entries	# Imagines	Imagine 0	Imagine 1	Imagine 2	Imagine 3	Avg/Img	Speed Up
1024	1024	1	6697600				6697600	1
		2	3281408	3420288			3350848	1.99877762
		4	1569280	1716224	1716224	1708160	1677472	3.99267469
32	1024	1	209300				209300	1
		2	102544	106884			104714	1.99877762
		4	49040	53632	53632	53380	52421	3.99267469
8	1024	1	52325				52325	1
		2	25636	26721			26178.5	1.99877762
		4	12260	13408	13408	13345	13105.25	3.99267469
8	512	1	25636				25636	1
		2	12260	13408			12834	1.99750662
		4	5950	6342	6720	6720	6433	3.98507695
8	64	1	3040				3040	1
		2	1592	1480			1536	1.97916667
		4	812	812	770	742	784	3.87755102
8	32	1	1669				1669	1
		2	833	868			850.5	1.96237507
		4	415	450	450	450	441.25	3.78243626

Table 3a. Cycle counts from Pipelined implementation

The above table (*Table 3a*) shows that improvements in performance are reaped from using multiple nodes in a pipelined configuration. The speedup is almost proportional to the number of Imagines utilized.

Note that when four Imagines are used, the execution time for each Imagines differs. This is because the cycle count is actually data dependent. When finding the mask length, the mask needs to be logically left-shifted by one bit per cycle until it becomes zero. Hence, the amount of time taken to find the mask length varies.

When the number of data packets is held constant, the speedup is not as good as when a small routing table (< 1024 entries) is used (*see figure 3b*). This demonstrates the short stream effects. The communication overhead is significant when such a small data set is used.

However, with the number of routing table entries held constant, the speedup is still the same with a varying number of data packets (*see Figure 3a*). This is due to the fact that data packets are always processed in batches of eight. Hence, with sixteen packets the kernel needs to be called twice for each Imagine. The amount of overhead is proportional to the eight packets. The speedup ratio is hence the same.

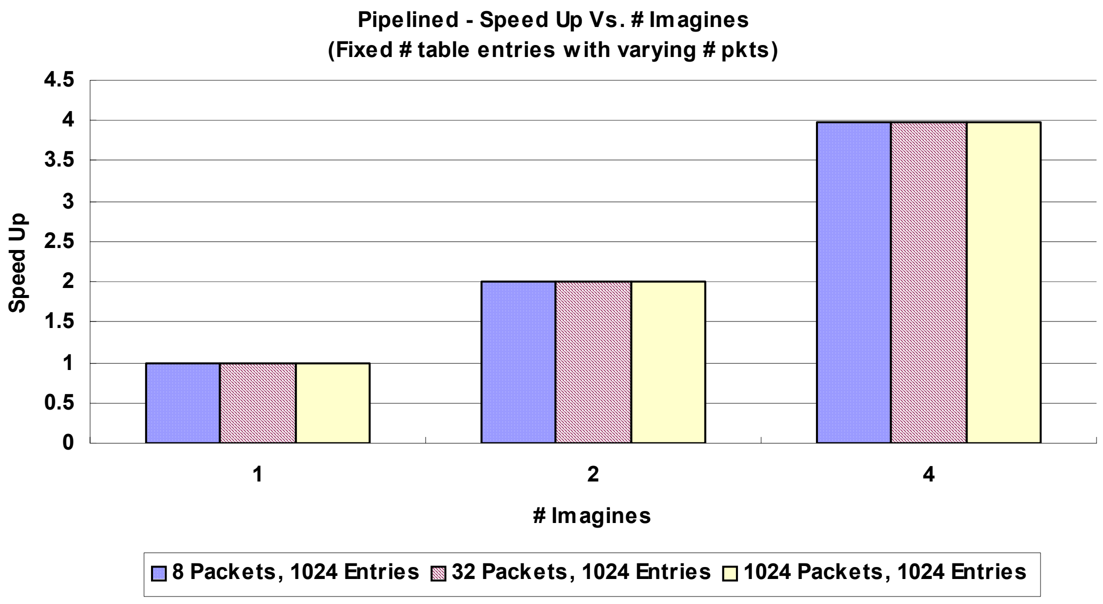

Figure 3a. Pipelined - Speedup vs. # Imagines

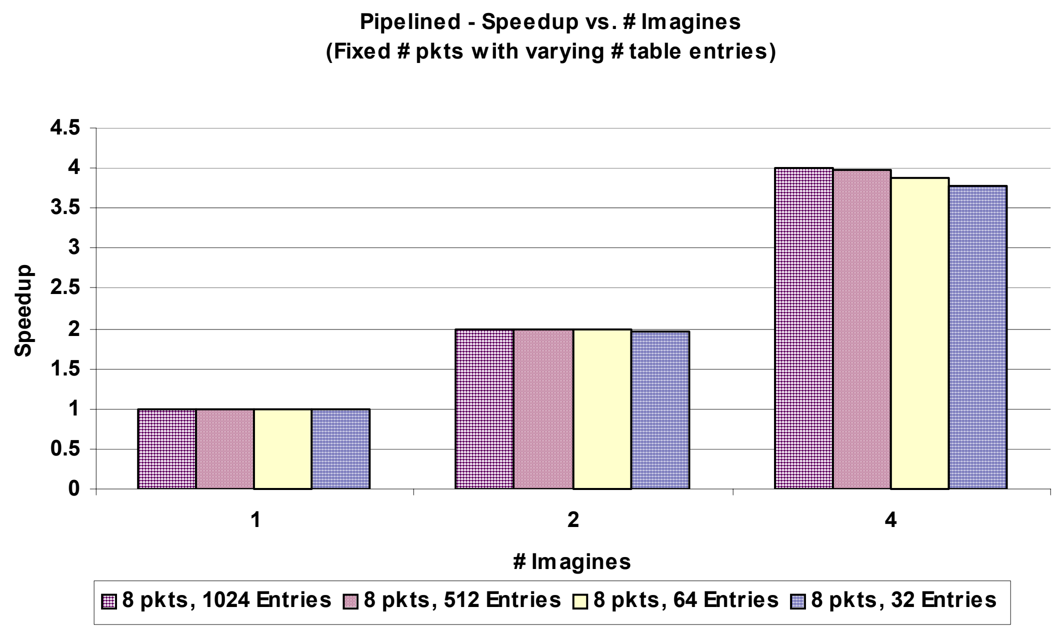

Figure 3b. Pipelined – Speedup vs. # Imagines

Pipelined Algorithm (Large Data Set)								
Execution Time								
# Packets	**# Entries**	**# Imagines**	**Imagine 0**	**Imagine 1**	**Imagine 2**	**Imagine 3**	**Avg/Img**	**Speed Up**
8	8192	1	415765				415765	1
		2	203037	212760			207898.5	1.99984608
		4	96596	106473	106872	105920	103965.25	3.99907661
32	8192	1	1663060				1663060	1
		2	812148	851040			831594	1.99984608
		4	386384	425892	427488	423680	415861	3.99907661
1024	8192	1	53217920				53217920	1
		2	25988736	27233280			26611008	1.99984608
		4	12364288	13628544	13679616	13557760	13307552	3.99907661
8192	8192	1	425743360				425743360	1
		2	207909888	217866240			212888064	1.99984608
		4	98914304	109028352	109436928	108462080	106460416	3.99907661

Table 3b. Cycle counts from Pipelined implementation (Large Data Set)

The above table (*Table 3b*) shows that improvements in performance are extracted from using multiple nodes in a pipelined configuration with large data set. The speedup is almost proportional to the number of Imagines utilized and is very close to the result extracted with regular data set. Therefore, once the data set is relatively large (> 1024 entries), it makes almost no difference to the speedup (*see Figure 3c*). The limiting factor is then due to the number of Imagines used rather than the size of the data set. If we use a large number of Imagines (> 4 Imagines), we should see noticeable drop off in speedup, because the communication cost associated with using a large number of Imagines should be quite significant.

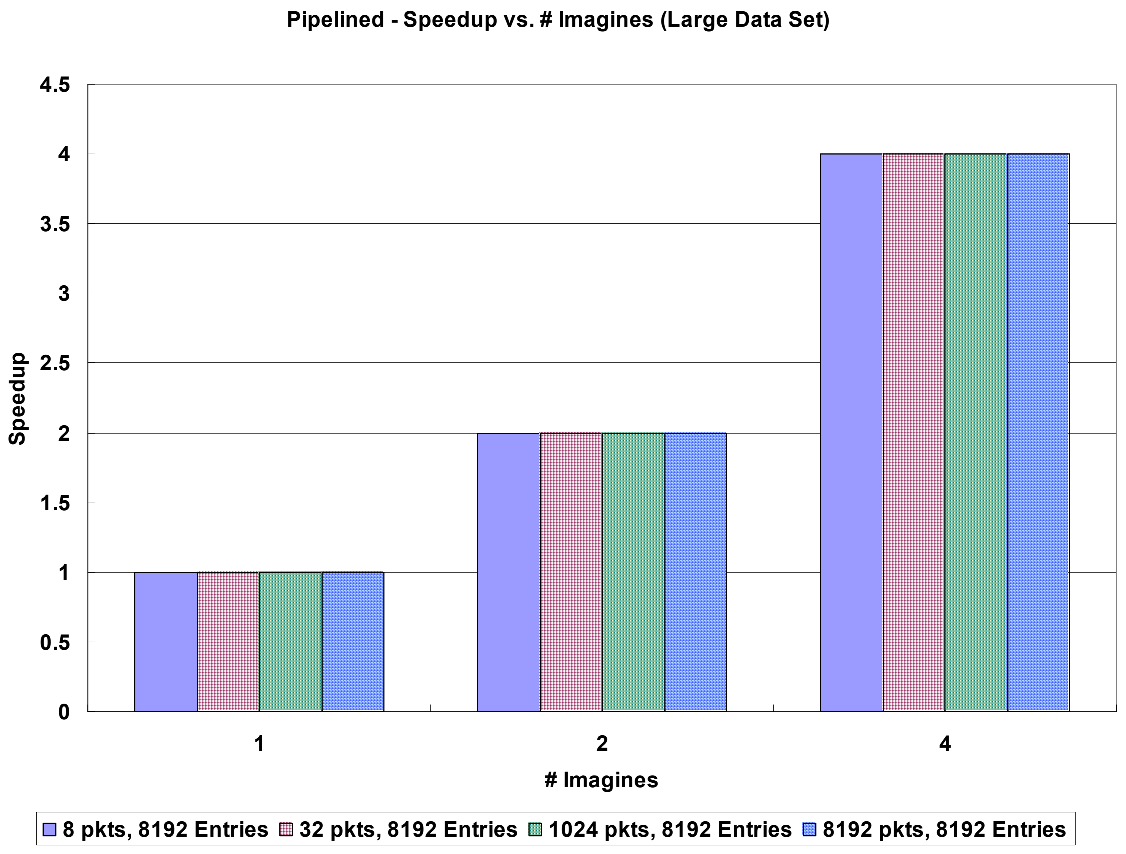

Figure 3c. Pipelined – Speedup vs. # Imagines (Large Data Set)

Parallel Algorithm								
Execution Time								
# Packets	**# Entries**	**# Imagines**	**Imagine 0**	**Imagine 1**	**Imagine 2**	**Imagine 3**	**Avg/Img**	**Speed Up**
32	1024	1	209311				209311	1
		2	104650	104661			104655.5	2
		4	52325	52325	52325	52336	52327.75	4
32	64	1	12171				12171	1
		2	6080	6091			6085.5	2
		4	3040	3040	3040	3057	3044.25	3.99802907
32	32	1	6687				6687	1
		2	3338	3349			3343.5	2
		4	1669	1669	1669	1686	1673.25	3.99641416
32	8	1	1811				1811	1
		2	900	911			905.5	2
		4	450	450	450	467	454.25	3.98679141
64	8	1	3611				3611	1
		2	1800	1817			1808.5	1.99668233
		4	900	900	900	929	907.25	3.98015982
1024	8	1	57701				57701	1
		2	28800	28997			28898.5	1.99667803
		4	14400	14400	14400	14789	14497.25	3.98013416

Table 3c. Cycle counts from Parallel implementation

The performance of multi-node using the parallel version is slightly better than the pipelined version on the average. However, the parallel version is not necessarily better as the performance also depends on how the data is distributed to the different Imagines. The load per Imagine is perhaps more balanced, when distributing the data rather than distributing the routing table. A large portion of this performance is data dependent, and a slight difference in performance does not tilt the balance in either configuration's favor.

Note that the last Imagine always has a larger cycle count than the previous Imagines. This is because the last Imagine is responsible for combining all the processed data from the other processors. This is an important step as different Imagines may take varying lengths of time to complete the masking and address matching.

With a fixed number of packets, the speedup improves as the number of routing table entries increases (*Figure 3d*). With a smaller routing table, short stream effects occur whereby, first, each kernel incur startup costs such as variable initialization for each batch of data, and second, software-pipelined kernels also incur the cost of priming and draining their software-pipelined inner loops. With a larger routing table, short stream effects diminish.

With varying data packets but fixed routing table entries, the overhead cost increases as the data packets increases. More synchronizing is needed, i.e. combining the outputs of each Imagine takes a longer time with more data packets. Hence, speedup decreases as more data packets are used (*Figure 3e*).

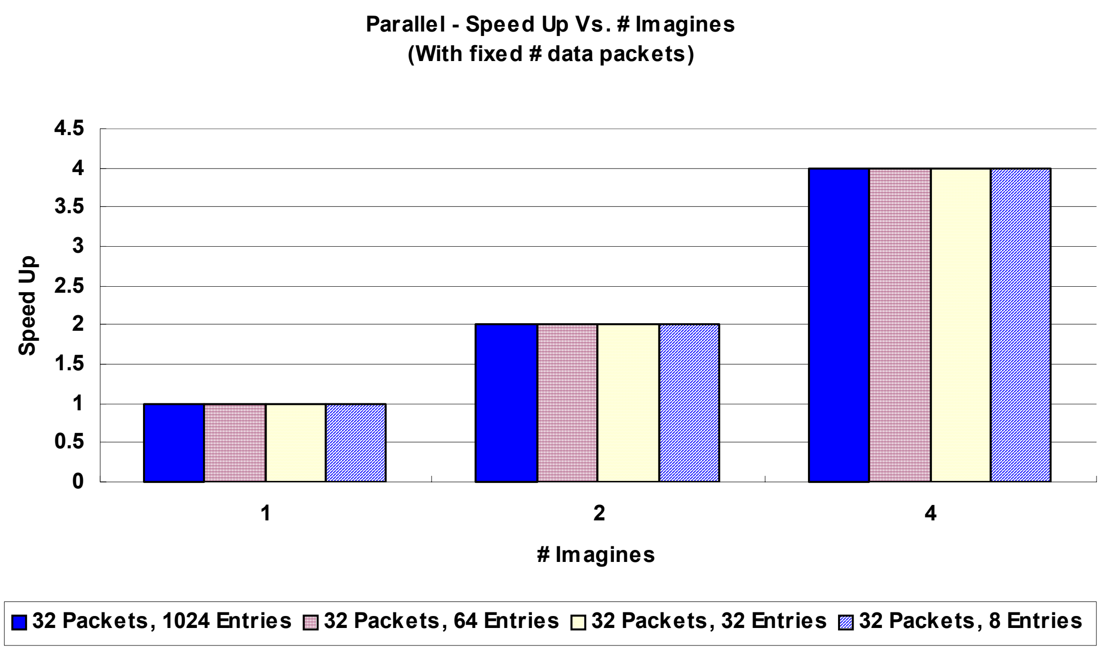

Figure 3d. Parallel – Speedup vs. # Imagines

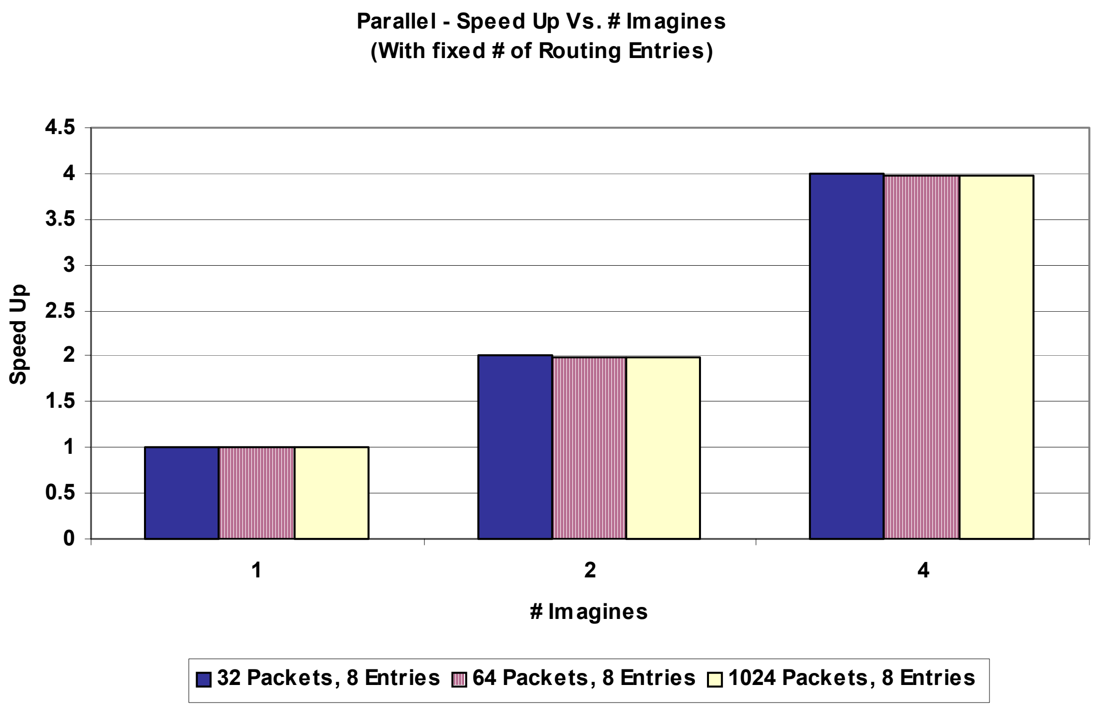

Figure 3e. Parallel – Speedup vs. # Imagines

Parallel Algorithm (Large Data Set)								
Execution Time								
# Packets	# Entries	# Imagines	Imagine 0	Imagine 1	Imagine 2	Imagine 3	Avg/Img	Speed Up
32	8192	1	1663071				1663071	1
		2	831530	831541			831535.5	2
		4	415765	415765	415765	415782	415769.25	3.99998557
1024	8192	1	53218021				53218021	1
		2	26608960	26609157			26609058.5	1.99999639
		4	13304480	13304480	13304480	13304869	13304577.3	3.99997835
8192	8192	1	425744133				425744133	1
		2	212871680	212873221			212872451	1.99999639
		4	106435840	106435840	106435840	106438917	106436609	3.99997835

Table 3d. Cycle counts from Parallel Implementation (Large Data Set)

The above table (*Table 3d*) shows that improvements in performance are extracted from using multiple nodes in a parallel configuration with large data set. The speedup is almost proportional to the number of Imagines utilized and is very close to the result extracted with regular data set. Similar to the pipelined configuration, once the data set is relatively large (> 1024 Entries), it makes almost no difference to the speedup (*see Figure 3f*). The limiting factor is the number of Imagines used rather than the size of the data set, because of the same reason as found from the simulation of the pipelined configuration with a large data set.

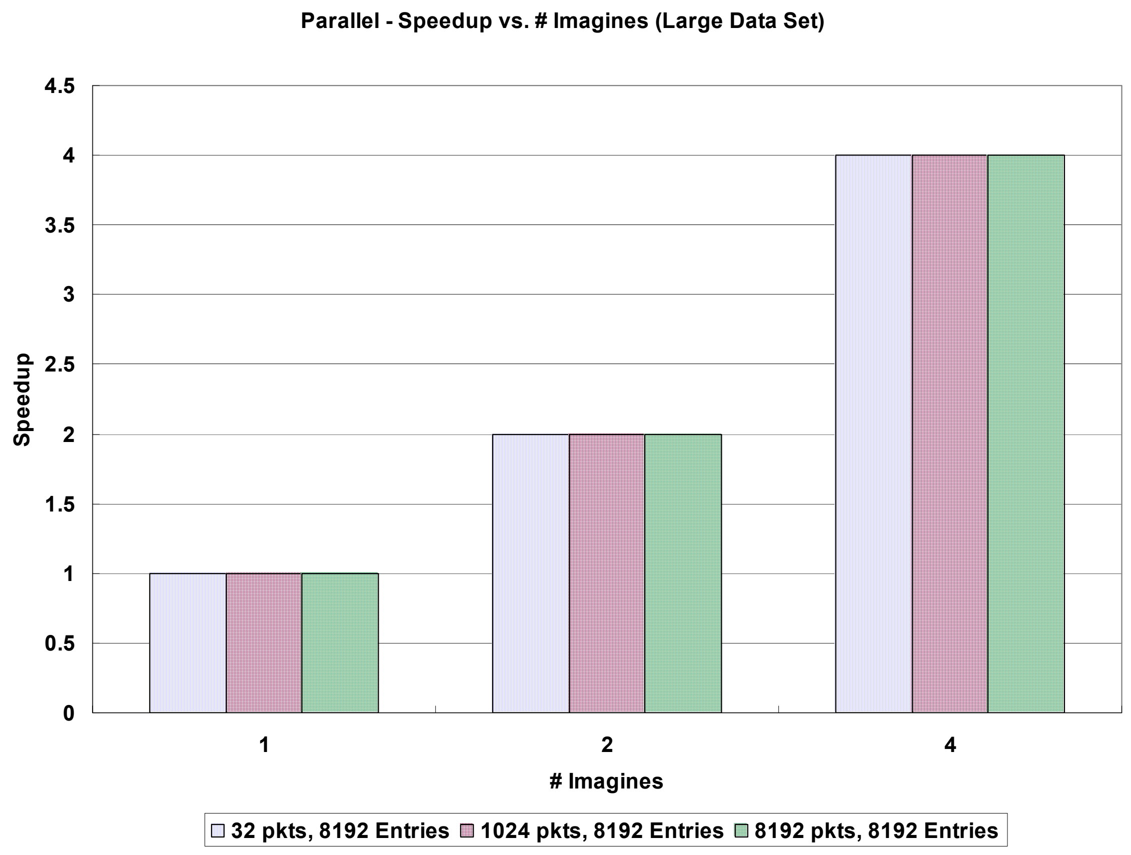

Figure 3f. Parallel – Speedup vs. # Imagines (Large Data Set)

4.1 Conclusion

We found that speedup increases with the number of processing nodes, as it increases parallelism. Communication and synchronization overheads are present, but the effect on performance is not that great with a small number of Imagines (4 or less). This is expected as the large part of the application execution time is spent on finding the longest prefix match, not on the synchronization and this is even more true with large data sets. With IP packet routing, we are able to exploit data and instruction parallalism and hence, comunication and synchronization overheads are very much minimized, resulting in almost perfect speedup with more imagines.

Multi-node programming is an effective way of exploiting the inherent data-level, instruction-level and thread-level parallelism present in stream processing applications. Depending on the nature and design of the application, it is possible to reduce the amount of state variable synchronization and inter-node communication such that the benefits of parallel processing are maximized.

4.2 Challenges

Due to time and resource constraints, we were unable to implement simulations involving > 4 Imagines in multi-node configurations. Some restrictions and implementation criteria dictated by the development tools made it more complicated to implement inter-Imagine communications and synchronization. For example, multi-host simulations gave rise to memory access violations, most likely due to the out-of-order execution of the hosts. As such, we are only able to get one host running at this point in time. One host can only support four Imagines, and hence this limits our testing configurations. Profiling imposed many restrictions on coding style and since profiling was needed for performance analysis, we had to modify our code several times in order for profiling to work.

4.3 Future Work

Multi-host simulation is definitely an avenue to explore in order to get more Imagines running. Specifically, ISim should be used to do multi-host and multi-imagine simulation so that cycle accurate results can be extracted for large data set running on as many as 16 Imagines processors and 4 hosts. Also, other applications such as signal processing and graphics rendering should be tested in multi-node environments to see how speedup varies with the number of Imagines or hosts used.

5. Acknowledgements

Finally, we would like to thank Professor Dally, our TAs Mattan Erez and Abhishek Das for giving us invaluable advice and help throughout the course of our project, especially when things were not going smoothly.