# Multithreaded Synchronous Data Flow Simulation

Johnson S. Kin, José Luis Pino – Agilent Technologies, Inc.

Email: johnson_kin, jpino@agilent.com

## Abstract

This paper introduces an efficient multithreaded synchronous dataflow (SDF) scheduler that can significantly reduce the running time of multi-rate SDF simulations on multiprocessor machines with only a slight increase of memory usage over standard cluster loop scheduler[1]. Experiments run on a dual processors machine achieves on average approximately 146% increase in performance with less than 2.4% increase in memory usage. There is an average of 2x speedup with a quad processors machine.

## 1. Introduction

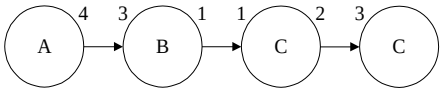
Synchronous dataflow (SDF) simulation[2, 3, 4] has been widely used as the simulation model for digital signal processing. Few have yet to target the performance issue in simulation environment. By changing the data structures, having cleaner models, the speedup is still limited and not scalable. For example, in a WCDMA 3GPP[5] design, examining the blocking characteristics of the base station receiver takes around 30 minutes (Intel PIII 450, 512M memory) for only 1 frame of data (10ms in real time).

In this paper, we develop a multithreaded SDF scheduler based on a hierarchical clustering algorithm[6,7]. With multiprocessors workstations becoming more accessible, simulation time using our proposed scheduler can be reduced dramatically and theoretically scalable with the number of processors used. We discuss how we can overcome the limitation of various overheads and exploit more parallelism using hierarchical clustering algorithm and other techniques. Finally we present some performance measurement on some practical DSP examples.

## 2. Multithreaded Scheduler

Given that SDF simulation is compute bound, finding fine grain parallelism such as pipelining commonly used in hardware architecture and software compilation[8,9] is not enough. Therefore we combine pipelining with hierarchical loop clustering to create highly concurrent parallel schedules. Additional techniques such as loop unrolling, deep level multithreading and clusters flattening are applied for more parallelism extraction.

Using Looped Scheduling, the schedule of Figure 1 becomes (9A)(12B)(12C)(8D. With the help of clustering, the schedule can be transformed into (3[(3A)(4B)]) (4[(3C)(2D)]), grouping A and B into cluster $\alpha$, C and D into cluster $\beta$. Now if we apply $\alpha$ and $\beta$ using pipelining on dual processors machine, two threads can run independently, one fires 21 components and the other 20 before they synchronize at the end of a time slot.




$G_f$

**Figure 1 A chain-structured SDF graph**

Synchronization overhead is one of the bottlenecks for multithreaded SDF simulation. It can be reduced by schedule unrolling to increase the workload before synchronization of each thread to generate higher throughput. One can unroll the schedule by a factor of n to reduce the overhead time by n. The side effect is that the prologue and epilogue are longer and the memory buffer size increases by a factor of n.

It is unlikely that 2 clusters are well balanced. Thus, the running time will be limited by the heavily loaded cluster. To further complicate the problem, each component may have different running times. In order to balance the loads on multiple threads, we first compute the firing statistics of each component at each level. Then we find the most balance level to allocate clusters.

Sometimes clustering takes away the available parallelism. In order to reduce the synchronization overhead, a process to reverse the clustering is introduced called cluster flattening. It extracts parallelism while making the tree more balanced. The disadvantages are high buffer increase and one extra pass of tree traversing. It cannot be applied to clusters with feedback loops.

Deep level multithreading finds multiple balance points at different levels and split clusters into multiple threads to gain speedup with the use of multiple processors. It can effectively compensate an unbalanced tree. However, one can easily overdo it where synchronization overhead becomes the bottleneck.

1530-1591/03 $17.00 © 2003 IEEE

## 3. Experiment and Results

We applied loop unrolling and cluster flattening to a set of wireless communication applications using Agilent ADS wireless design library. Several criteria are imposed. First, unrolling kicks in only at root level and total firings per schedule under root is less than 10000 to limit the explosion of buffer space. Flattening will only apply if a cluster does not consist of feedback loops and sibling branch has at least 20% of workload. Results are gathered using (Intel PIII 600, 256M memory). Finally, deep level multithreading is applied with a quad processor machine (Sun SparcV9 450, 1G memory). The result is compared with the same machine using uniprocessor scheduler.

| *Design #* | *Simulation Designs* |
|---|---|
| 1 | WCDMA3G-BS_TxACLR_SwitchingTransients |
| 2 | WCDMA3G-UE_Rx_RefLevel_PhyCHBER |
| 3 | WLAN80211a-TxSpectrum |
| 4 | WCDMA3G-3GPPFDD_UE_Tx_12_2 |
| 5 | WLAN80211a-RxSensitivity_54Mbps |
| 6 | WLAN80211a-TxEVM |
| 7 | WLAN80211a-TxEVM_Turbo |
| 8 | WLAN80211a-RxSensitivity_24Mbps |
| 9 | WCDMA3G-UE_Rx_RefLevel |
| 10 | WCDMA3G-UE_Rx_MaxLevel |
| 11 | WLAN80211a-RxAdjCh_36Mbps |
| 12 | WCDMA3G-UE_Rx_ACS |
| 13 | WLAN80211a-GI |
| 14 | WLAN80211a-RxAdjCh_18Mbps |
| 15 | WLAN80211a-ChannelCoding |
| 16 | WLAN80211a-RxNonAdjCh_48Mbps |
| 17 | WLAN80211a-RxAdjCh_9Mbps |
| 18 | WCDMA3G-UE_Rx_In_Band_Blocking |
| 19 | WLAN80211a-RxNonAdjCh_12Mbps |

**Table 1 Simulation Designs used**

Designs which run slower than the original is mainly due to smaller physical memory of the machine. As mentioned before, loop unrolling and cluster flattening can be used to increase the parallelism with a price of increase in buffer size. Moreover, different components have different running times; the differences can have a major effect in determining the balance point. We expect a significant improvement be achieved once the time profiling of components is added to determine the balance point. Table 3 also shows the effect of 4 processors machine which on average gives 2X speed improvement. It shows the effectiveness of deep level multithreading and loop unrolling with larger physical memory.

## 4. Conclusion

We demonstrated the potential parallelism that exist in a SDF graph and transform an efficient uniprocessor hierarchical cluster looped schedule into multithread parallel schedule. By using our multithreaded synchronous dataflow (SDF) scheduler on some real life communication applications, we showed that our scheduler significantly reduces the running time of multi-rate SDF simulations on multiprocessor machines.

| *Design #* | *Orig PC (sec)* | *MT Dual PC (sec)* | *Speedup dual* | *Orig Sun (sec)* | *MT Quad Sun (sec)* | *Speedup Quad* |
|---|---|---|---|---|---|---|
| 1 | 137 | 65 | 2.11 | 280 | 112 | 2.50 |
| 2 | 223 | 164 | 1.36 | 470 | 206 | 2.28 |
| 3 | 300 | 223 | 1.35 | 687 | 309 | 2.22 |
| 4 | 320 | 334 | 0.96 | 343 | 303 | 1.13 |
| 5 | 399 | 257 | 1.55 | 810 | 372 | 2.18 |
| 6 | 576 | 435 | 1.32 | 1357 | 535 | 2.54 |
| 7 | 596 | 406 | 1.47 | 1425 | 563 | 2.53 |
| 8 | 618 | 384 | 1.61 | 1180 | 563 | 2.10 |
| 9 | 723 | 543 | 1.33 | 1810 | 801 | 2.26 |
| 10 | 717 | 534 | 1.40 | 1831 | 791 | 2.31 |
| 11 | 922 | 451 | 2.04 | 1480 | 693 | 2.14 |
| 12 | 1154 | 845 | 1.37 | 2696 | 1147 | 2.35 |
| 13 | 1175 | 1158 | 1.02 | 2214 | 1914 | 1.16 |
| 14 | 1429 | 760 | 1.88 | 2315 | 1036 | 2.23 |
| 15 | 1690 | 2483 | 0.68 | 3283 | 2377 | 1.38 |
| 16 | 900 | 2099 | 2.33 | 1431 | 2951 | 2.06 |
| 17 | 2952 | 2542 | 0.86 | 2711 | 4017 | 1.48 |
| 18 | 2372 | 2595 | 1.09 | 3275 | 5679 | 1.73 |
| 19 | 5404 | 4808 | 0.89 | 4476 | 6922 | 1.55 |
| *Mean* | | | *1.40* | | | *2.007* |

**Table 2 Speedup Improvements**

## 5. References


[1] P. K. Murthy, S. S. Bhattacharyya, and E.A. Lee, *Joint Minimization of Code and Data for Synchronous Dataflow Programs.* Journal of Formal Methods in System Design, 1997. **11**(1): p. 41-70.

[2] E.A. Lee and D.G. Messerschmitt. *Synchronous data flow*. in *Proceedings of the IEEE*. 1987.

[3] J. T. Buck, et al. *Multirate Signal Processing In Ptolemy*. in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*. 1991. Toronto.

[4] Bhattacharyya, S.S., *Compiling Dataflow Graphs for Signal Processing*. 1994, Ph.D. thesis, Memorandum No. UCB/ERL M94/52, Electronics Research Laboratory, University of California at Berkeley.

[5] *3GPP Technical Specification TS 25.104 V3.2.0, "UTRA(BS) FDD: Radio transmission and Reception"*. March 2000.

[6] S. S. Bhattacharyya, P. K. Murthy, and E.A. Lee, *Software Synthesis from Dataflow Graphs*. 1996, Norwell, MA: Kluwer Academic Press.

[7] J. Buck, et al., *Ptolemy: A framework for simulating and prototyping heterogeneous systems.* International Journal of Computer Simulation, special issue on Simulation Software Development, 1994. **4**.

[8] J. L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach.* 1993: Morgan Kaufman, San Francisco, CA.

[9] Lam, M. *Software Pipelining: An Effective Scheduling Technique for VLIW Machines*. in *Proceedings of ACM SIGPLAN '88 Conference on Programming Language Design and Implementation*. 1988.