STACK: Sparse Timing of Algorithms Using Computational Knowledge

Vasanth Iyer[1], S. Sitharama Iyengar[2], Garmiela Rama Murthy[1], Kannan Srinathan[1], Mandalika B. Srinivas[3], and Regeti Govindarajulu[1]

[1] International Institute of Information Technology, Hyderabad, India-500 032
vasanth@research.iiit.ac.in, rammurthy@iiit.ac.in, srinathan@iiit.ac.in, gregeti@iiit.ac.in
[2] Louisiana State University, Baton Rouge, LA 70803, USA
iyengar@csc.lsu.edu
[3] Birla Institute of Technology & Science, Hyderabad Campus, Hyderabad-500078, India
srinivas@bits-hyderabad.ac.in

Abstract. Research in computational aspects and algorithm optimizations help design tools to acceleration the execution of algorithms. Cost and availability of FPGA design boards have driven number of computations per second close to the general-purpose model of CPUs. In this chapter, we study the effects of algorithms with the knowledge of the underlying computing model for getting consistent and coherent view of the sensed data. The computing model uses uniprocessor, multiprocessor and acceleration using pipeline and data-path forwarding with Byzantine fault-tolerance. The pre-processing approach of the modified algorithm for sparse sensing gives better consistency and the application based calibration allowing coherent view of the data and at the same time reduces the total power consumption. This is analogous to the *needle in a hay stack*. The STACK implementation runs 4 times faster than the normal program based optimizations for static and dynamic scheduling.

Keywords: Sensor-centric data fusion; Algorithm design; Compressed Sensing (DCS); Sensor Fusion; Pre- and Post processing of Sensors; Cross-layer Protocols; Processor Computational Models.

1 Introduction

We study the optimization of data-centric algorithms, which need to process data locally and aggregate and optimize in a distributed way. Most of the time these families of algorithms are compared measure based on computational complexities in time and space, which allow studying its scalability. The lifetime of sensor network [1,8,13] is also an important measure to benchmark the performance of these algorithms. When using this measure as they are prone to error than a general deployment error correction needs to be included in terms of a Byzantine agreement algorithm.

S.C. Mukhopadhyay et al. (Eds.): New Developments and Appl. in Sen. Tech., LNEE 83, pp. 305–320.
springerlink.com

The computational model for the programs shown in Figure 1, Algorithm 1 and 2 assume sequential execution. The values of Flag and turn in algorithm decide which process enters the critical section allowing shared data modeling [3,4] for consistency [3] and atomic operations. Message passing model is used in a distributed power-aware topology; this is shown in Algorithm 3. The data coherency [3] cannot be achieved due to

- unknown number of processors
- independent inputs at different locations
- several programs executing at once, staring at different times, and operating at different speeds
- processor nondeterminism
- uncertain message delivery times
- unknown message ordering
- processor and communication failures

The value of UID is pre assigned and the value of the leader is based on having a reliable message passing FIFO available at each processing node element (PE).

Computational Model for Consistency:

- Pre-processing - reduce computation per application
- Dynamic Range - register bit usage
- Pipe-line (1-CPI) fine grain instruction level parallelization
- Double buffer for datapath support
- Dynamic power dissipation

The computational model is affected by software compiler optimizations, having sequential consistency [3], this is called program order. For embedded systems, most of the optimization is targeted for space as the target has resource limitations. In- order sequential consistency is performed by the compiler by using a window and appropriate scheduling of instructions to enhance performance. The pipe-line optimizations are done dynamically (out-of-order) [3] execution, which furthers performance which is not possible in the earlier case such as memory access and register allocations. Due to this data hazards are possible due to data coherency needs.

Processor and hardware support needs to support the high-level construct of the language in context with sequential consistency [1]. The non-blocking memory operations can be implemented using specific address ranges and explicit addresses can be assigned for shared memory operations. The processor can distinguish it by physical and virtual address space during dynamic execution. Unused op-code fields can be used to implement the latter case.

Even in applications where reasonable amounts of parallelism are available, significant serial, or nearly serial, code segments may occur. A problem where 90% of the program can obtain a speed-up of 10, and the remaining 10% has no parallelism, will actually run only about 5 times faster. In the hardware implementation, the same clock runs the memory that allows access to memory complete in one clock cycle. We show that the STACK model of execution gives superior performance than the sequential consistency model for data-path based algorithms, which effect sampling rate and power.

1.1 Computational Models

The modern processors use efficient Instruction Set Architecture (ISA) such as RISC processors. The performance measures defined as below

$$Perf = \frac{clock\ speed}{instruction\ count\ *\ cycles\ per\ instruction} \tag{1}$$

1.2 Van Numen

RISC [12] machines attempt to maximize performance by producing improvements in clock speed (factors of 2-5, typically) and major improvements in the cycles per instruction (factors of 5 to 10). They allow a slight increase in the instruction count (less than a factor of 2).

1.3 Distributed Architecture

A synchronous network system consists of a collection of computing elements located at the nodes of a directed network graph. We refer to these computing elements as processing elements (PES), which suggests that they are pieces of sensor hardware. In order to define synchronous network system start with a directed graph $G = (V, E)$. We use the letter n to denote $\mid V \mid$, the number of nodes in the network digraph.

1.4 FPGA

The hardware implementation uses a data driven pipelining, the clock rate can be calculate as

$$CLK = Active\ event\ Q + \ Longest\ delay + Setup\ time + ClkSkew \tag{2}$$

$$Power = \frac{1}{2} Capacitive\ Load \times \ Voltage^2 \times Freq\ Switch \tag{3}$$

The clock cycles shared between execution units and the memory unit, which allows transferring data from memory in one clock cycle. The actual implementation, which needs to calculate the longest delay while completing a calculation determines the clock period, it tends to be longer to accommodate all the variations.

1.5 Computational IP Cores

FPGA design framework [7] allows using many available algorithms in a form of a library protected by intellectual rights. Pre-processing of data needs many steps which can use these algorithm libraries, such as Scaling [7], Interpolation [7] and Mixing [7]for video oriented data streams.

Algorithm 1. Process P1

```
1: states_i :
2: flag[0] = 0;
3: flag[1] = 0;
4: turn;

5: msgs_i :
6: P0: flag[0] = 1; P1: flag[1] = 1;
7: turn = 1;
8:
9: while (flag[1] == 1 && turn == 1) do
10:
11:    // busy wait
12:
13:
14: end while
15: // critical section
16: ...
17: // end of critical section
18: flag[0] = 0;
```

Algorithm 2. Process P2

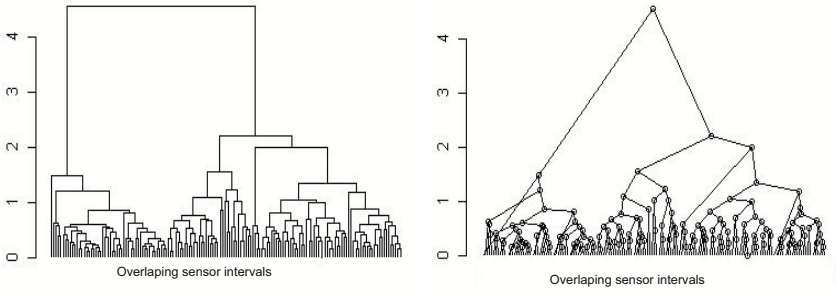
```
1: states_i :
2: flag[0] = 0;
3: flag[1] = 0;
4: turn;

5: msgs_i :
6: P1: flag[0] = 1; P1: flag[1] = 1;
7: turn = 0;
8:
9: while (flag[0] == 1 && turn == 1) do
10:
11:    // busy wait
12:
13:
14: end while
15: // critical section
16: ...
17: // end of critical section
18: flag[1] = 0;
```

Algorithm 3. Process P1,P2,...Pn

```
1: Each process begins by sending its UID to its clockwise neighbor.
2: Each process checks its UID (u) against the one it just received (v),
3: if v > u then
4:    the process sends v on to the next process
5: end if
6: if v = u then
7:    the process is chosen and sends out a leader message
8: end if
```

Fig. 1. Program for atomicity



(a) Sparse sensor readings **(b) Averaging cost function**

Fig. 2. Sensor-centric Data aggregation using atomic averaging

1.5.1 Scaling

The raw data need to be scaled to a metric before an application can further process, this family of algorithms use register size, integer with fixed bits or floating point scaling to ensure metric.

1.5.2 Interpolation

Sampling rate helps to define how to represent the input signal and interpolate it, when calculating its dynamic range.

1.5.3 Mixing

Data stream which represents images often needs an algorithm to fuse data, it uses a pre-processing step on every pixel before the fusion can be performed.

2 Data Pre-processing – Non Blocking

2.1 Double Buffering

The synchronization methods used by IPC [5] are generally fall into the category of waited and non-waited. If the algorithm is designed with data-path approach using buffering then less computationally complex algorithm can be implemented which is wait free. The porting issues, which is processor dependent, is the availability of atomic operations to access and operate on memory variables atomically. Algorithm 4, DoubleBuffer() shows how the value of *latest* is shared between many Readers and continuously updated by one writer. Line 4 used the flag pointing to the buffer pairs which contain the latest update, and the non-blocking write uses number of readers $(N + 1)$ if all are in use, its new value as shown in line 12.

The design of double buffering allows to simultaneously have multiple readers access the same buffer. Writer can only interfere with the reader when they both choose to use the same row. This can occur in two cases. The first case can occur when a reader is interrupted after it has chosen a row (after line 3 in Algorithm 4),

but before it updates the use count. The writer then executes, and can potentially choose the same row as the reader. The second case occurs when the writer is interrupted after it has chosen a row (line 9 in Algorithm 4). If this row happens to be *Latest*, then the reader can also choose to read from the same row. So, it is possible for the readers and the writer to select the same row i. However, the reader will read from the buffer indicated by C1[i], while the writer will use the opposite one. As the writer updates C1[i] only after the complete message is written, and the reader always increments the use count before reading Cl[i], we can guarantee that the writer and readers cannot interfere with each other in this algorithm, even if they happen to use the same row.

Algorithm 4. DoubleBuffer

```
 1: Reader()
 2: ridx = Latest
 3: inc ReaderCnt[rindx]
 4: cl = Cl[ridx]
 5: read Buff[ridx][cl]
 6: dec ReaderCnt[ridx]
 7: Writer()
 8: for (ii=Latest;;ii++)
 9: if ReaderCnt(ii mod NRows]==0 then
10:    break;
11: end if
12: cl= not Cl[ii]
13: write Buff[ii][cl];
14: Cl[ii]=c;
15: Latest=ii;
```

3 Algorithms for Data Coherency

As different sensors are connected to each node, the nodes have to periodically measure the values for the given parameters which are correlated. The inexpensive sensors may not be calibrated, and need processing of correlated data, according to intra and inter sensor variations. The pre-processing algorithms allow to accomplish two functions, one to use minimal number of measurement at each sensor, and the other to represent the signal in its loss-less sparse representation, which allows application level views. Figure 2(a) and 2(b) shows the cluster tree of inter-sensor intervals for a batch of sensors from Table 1.

3.1 Compressed Sensing (CS)

The pre-processing steps used by CS allows to sample at rates lower than the Nyquist rate, it recovers the original signal by rescaling and interpolation, which are save as compressed co-efficient during pre-processing.

– Sampling rate of i.i.ds [8,13]
– Aggregation window of the ensemble
– The delta ranges possibly measured for the complete ensemble - calibration

The signal measured if it can be represented at a sparse representation, then this technique is called the sparse basis as shown in equation (4), of the measured signal. The technique of finding a representation with a small number of significant coefficients is often referred to as Sparse Coding. When sensing locally many techniques have been implemented such as the Nyquist rate [1], which define the minimum number of measurements needed to faithfully reproduce the original signal. Using CS it is further possible to reduce the number of measurement for a set of sensors with correlated measurements [8,9].

$$x = \sum \vartheta(n)\Psi_n = \sum \vartheta(n_k)\Psi_{n_k}, \tag{4}$$

Consider a real-valued signal $x \in R^N$ indexed as x(n), $n \in 1, 2, ..., N$. Suppose that the basis $\Psi = [\Psi_1, ..., \Psi_N]$ provides a K-sparse representation of x; that is, where x is a linear combination of K vectors chosen from, Ψ, n_k are the indices of those vectors, and $\vartheta(n)$ are the coefficients; the concept is extendable to tight frames. Alternatively, we can write in matrix notation $x = \Psi\vartheta$, where x is an $N \times 1$ column vector, the sparse basis matrix is $N \times N$ with the basis vectors Ψ_n as columns, and $\vartheta(n)$ is an $N \times 1$ column vector with K nonzero elements. Using $\| \, . \, \|_p$to denote the ℓ_p norm, we can write that $\| \, \vartheta \, \|_p = K$; we can also write the set of nonzero indices $\Omega 1, ..., N$, with $|\Omega| = K$. Various expansions, including wavelets [6], Gabor bases [6], curvelets [6], are widely used for representation and compression of natural signals, images, and other data.

Algorithms 5, 6 and 7 allow applications to select how it views the state of the nature. FloodMinVal() algorithm uses a k-agreement [5], which allow to represent real-time floating values with lower-bound intervals. Figure 2, uses an averaging algorithm from R-Systems, general tree structure to calculate the dendrogram for the data-set in Table 1. The algorithm gives a good lower bound which has the values between $(1.6, 2.25)$. The power-aware algorithms use clustering which allows to read correlated value, due to lack of calibration in small sensors, the algorithm should be able to update only higher confidence values seen. The algorithm FloodMinRange() allows maximizing the coherency of the previous algorithm FloodMinVal() by calibrating between overlapping ranges, which are active during the cluster formation. FloodMinRange(), Line 12 allows to find the minimum of the overlapping ranges, which is a better representation of the signals value in terms of the sensors current sampling and density of coverage. To check the validity and the effectiveness, we use regression analysis using off-line statistical methods in section 5.

3.2 Coherency Cost Function of Sparse Representation

A single measured signal of finite length, which can be represented in its sparse representation, by transforming into all its possible basis representations. The number of basis for the for each level j can be calculated from the equation as

$$A_{j+1} = A_j^2 + 1 \tag{5}$$

So staring at $j = 0$, $A_0 = 1$ and similarly, $A_1 = 1^2 + 1 = 2$, $A_2 = 2^2 + 1 = 5$ and $A_3 = 5^2 + 1 = 26$ different basis representations.

Let us define a framework to quantify the sparsity of ensembles of correlated signals $x_1, x2, ..., xj$ and to quantify the measurement requirements. These correlated signals can be represented by its basis from equation (5). The collection of all possible basis representation is called the sparsity model.

$$x = P\theta \tag{6}$$

Where P is the sparsity model of K vectors ($K << N$) and θ is the non zero coefficients of the sparse representation of the signal. The sparsity of a signal is defined by this model P, as there are many factored possibilities of $x = P\theta$. Among the factorization the unique representation of the smallest dimensionality of θ is the sparsity level of the signal x under this model.

3.3 Proposed Algorithm Definition

The basis of the sensing algorithm design is presented, which is further adapted for data-centric with sensor based optimizations for large deployments.

Theorem 1. *Each Process Element (PES) maintains a variable min-val, originally set to its own initial value. For each of* $\frac{f}{k} + 1$ *rounds, the PES all broadcast their minvals, then each process resets its min-val to the minimum of its old min-val and all the values in its incoming messages. At then end, the decision value is min-val.*

Algorithm 5. FloodMinVal

1: $states_i$:
2: rounds $\in \aleph$, initially 0
3: decision $\in$ V $\cup$ unknown, initially unknown
4: min-val $\in$ V, initially i's value

5: $msgs_i$:
6: **if** rounds $\leq \frac{f}{k}$ **then**
7: send min-val to all other processes
8: **end if**

9: $trans_i$:
10: rounds := rounds+ 1
11: let m_j be the message from j, for each j from which a message arrives
12: min-val := $min(min - val \cup m_j : j \neq i)$
13: **if** rounds= $\frac{f}{k} + 1$ **then**
14: decision := min-val
15: **end if**

Algorithm 6. FloodMinRange

$states_i$:
rounds $\in \aleph$, initially 0
decision $\in$ V $\cap$ unknown, initially unknown
min-range $\in$ V, initially i's value

$msgs_i$:
if rounds $N - \tau$ **then**
  send min-range to all other processes
end if

$trans_i$:
rounds := rounds + 1
let m_j be the message from j, for each j from which a message arrives
min-range := $min(min - range \cap m_j : j \neq i)$
if rounds= $N - \tau$ **then**
  decision := min-range
end if

Algorithm 7. BestBasis

$states_i$:
Mark all elements on the bottom level J
Let $j = J$
Let $k = 0$

$msgs_i$:
Compare the cost value v_1 of elements k on level $j-1$ (couting from the left on the level) to the sum v_2 of the cost values of the elements $2k$ and $2k+1$ on the level.
if $v_1 \leq v_2$ **then**
  all marks below element k on the level $j-1$ are deleted, and element k is marked.
end if
if $v_1 \geq v_2$ **then**
  the cost value v_1 of element k is replaced with v_2.
end if

$trans_i$:
$k = k+1$. If there are more elements on level j if $k < 2^{2j-1} - 1$, go to step 6.
$j = j - 1$. If $j > 1$, goto step 4.
The marked basis has the lowest possible cost value, which is the value currently assigned to the top element.

3.4 Sensor Centric Algorithm

DCS allows to enable distributed coding algorithms to exploit both intra-and inter-signal correlation structures. In a sensor network deployment, a number of sensors measure signals that are each individually sparse in the some basis and also correlated [6,9] from sensor to sensor. If the separate sparse basis are projected onto the scaling and wavelet [8] functions of the correlated sensors(common coefficients), then all the information is already stored to individually recover each of the signal at the joint decoder. This does not require any pre-initialization between sensors. The expanded wavelet optimization and its cost-functions are shown in Figure 3(a) and 3(b).

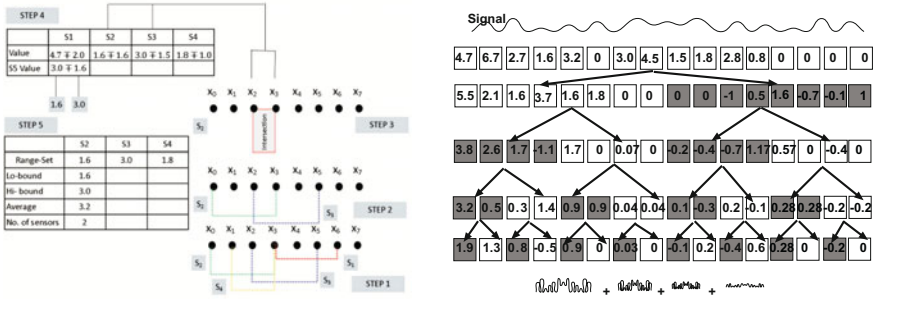
3.4.1 Joint Sparsity Representation

For a given ensemble X, we let $P_F(X) \subseteq P$ denote the set of feasible location matrices $P \in P$ for which a factorization $X = P\Theta$ exits. We define the joint sparsity levels of the signal ensemble as follows. The joint sparsity level D of the signal ensemble X is the number of columns of the smallest matrix $P \in P$. In these models each signal x_j is generated as a combination of two components: (i) a common component z_C, which is present in all signals, and (ii) an innovation component z_j, which is unique to each signal. These combine additively, giving

$$x_j = z_C + z_j, j \in \forall \tag{7}$$

$$X = P\Theta \tag{8}$$

We now introduce a bipartite graph $G = (V_V, V_M, E)$, as shown in Figure 4, that represent the relationships between the entries of the value vector and its measurements. The common and innovation components K_C and K_j, $(1 < j < J)$, as well as the joint sparsity $D = K_C + \sum K_J$.



(a) Sparse sensor readings **(b) Basis representation with maximal overlap**

Fig. 3. Sensor-centric Data Fusion during the aggregation step using sparse STACK model.

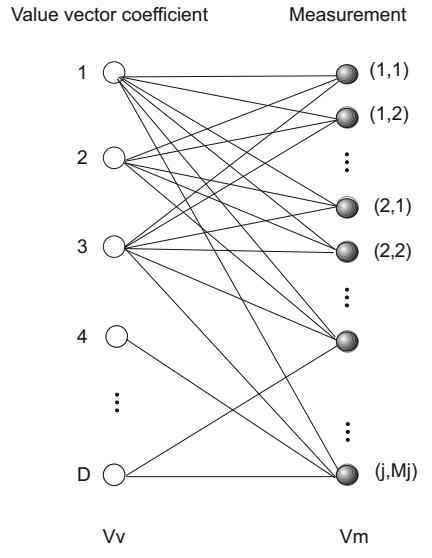


Fig. 4. Bipartite graphs representing fused aggregation for data coherency

The set of edges E is defined as follows:

- The edge E is connected for all K_c if the coefficients are not in common with K_j.
- The edge E is connected for all K_j if the coefficients are in common with K_j.

A further optimization can be performed to reduce the number of measurement made by each sensor, the number of measurement is now proportional to the maximal overlap of the inter sensor ranges and not a constant as shown in equation (4). This is calculated by the common coefficients K_c and K_j, if there are common coefficients in K_j then one of the K_c coefficient is removed and the common Z_c is added, these change does not effecting the reconstruction of the original measurement signal x.

3.5 Distributed Fused Parameter Dictionary

The sample sensor measurements of Table 1 and its transformed basis are shown in Figure 3 (a) and Figure 3 (b), illustrate all its possible basis representations. The cast-function [7] searches to find an optimal (grey rectangles) best basis matching the least number of coefficients to represent the signal without overlaps. The lowest range is calculated by selecting consecutive significant coefficients $(1.3, 1.7)$, which determine the maximal overlap for the sensor intervals. This best basis dictionary is stored in the hashed location of the application's search tree.

4 STACK Model Validation

4.1 Lower Bound Validation Using Covariance

The Figure 3(b) shows lower bound of the overlapped sensor i.i.d. of $S_1 - S_8$, as shown it is seen that the lower bound is unique to the temporal variations of S_2. In our analysis we will use a general model which allows to detect sensor faults. The binary model can result from placing a threshold on the real-valued readings of sensors. Let m_n be the mean normal reading and m_f the mean event reading for a sensor. A reasonable threshold for distinguishing between the two possibilities would be $0.5(\frac{m_n+m_f}{2})$. If the errors due to sensor faults and the fluctuations in the environment can be modeled by Gaussian distributions with mean 0 and a standard deviation σ, the fault probability p would indeed be symmetric. It can be evaluated using the tail probability of a Gaussian [9], the Q-function [9], as follows:

$$p = Q\frac{\left((0.5(\frac{m_n+m_f}{2}) - m_n)\right)}{\sigma} = Q\left(\frac{m_f - m_n}{2\sigma}\right) \tag{9}$$

From the measured i.i.d. value sets we need to determine if they have any faulty sensors. This can be shown from equation (9) that if the correlated sets can be distinguished from the mean values then it has a low probability of error due to sensor faults, as sensor faults are not correlated. Using the statistical analysis package R, we determine the correlated matrix of the sparse sensor outputs as shown This can be written in a compact matrix form if we observe that for this case the co-variance matrix is diagonal, this is,

$$\Sigma = \begin{pmatrix} \rho_1 & 0 & .. & 0 \\ 0 & \rho_2 & .. & 0 \\ : & : & \searrow & : \\ 0 & 0 & .. & \rho_d \end{pmatrix} \tag{10}$$

The correlated co-efficient are shown matrix (11) the corresponding diagonal elements are highlighted. Due to overlapping reading we see the resulting matrix shows that S_1 and S_2 have higher index. The result sets is within the desired bounds of the previous analysis using DWT. Here we not only prove that the sensor are not faulty but also report a lower bound of the optimal correlated result sets, that is we use S_2 as it is the lower bound of the overlapping ranges.

Table 1. Sparse representation of sensor values

Sensors	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
$i.i.d._1$	2.7	0	1.5	0.8	3.7	0.8	2.25	1.3
$i.i.d._2$	4.7	1.6	3	1.8	4.7	1.6	3	1.8
$i.i.d._3$	6.7	3.2	4.5	2.8	5.7	2.4	3.75	2.3

$$\Sigma = \begin{pmatrix} \overrightarrow{4.0} & 3.20 & 3.00 & 2.00 & 2.00 & 1.60 & 1.5 & 1.0 \\ 3.2 & \overrightarrow{2.56} & 2.40 & 1.60 & 1.60 & 1.28 & 1.20 & 0.80 \\ 3.0 & 2.40 & \overrightarrow{2.250} & 1.50 & 1.50 & 1.20 & 1.125 & 0.75 \\ 2.0 & 1.60 & 1.50 & \overrightarrow{1.00} & 1.00 & 0.80 & 0.75 & 0.5 \\ 2.0 & 1.60 & 1.50 & 1.00 & \overrightarrow{1.00} & 0.80 & 0.75 & 0.5 \\ 1.6 & 1.28 & 1.20 & 0.80 & 0.80 & \overrightarrow{0.64} & 0.60 & 0.4 \\ 1.5 & 1.20 & 1.125 & 0.75 & 0.75 & 0.60 & \overrightarrow{0.5625} & 0.375 \\ 1.0 & 0.80 & 0.750 & 0.50 & 0.50 & 0.40 & 0.375 & \overrightarrow{0.250} \end{pmatrix} \tag{11}$$

The sensor data are correlated due to variations in deployment they are difficult to calibrate. The pre-processing of data will needs to correct the coefficients before applying the fusion function. As this constraint the design of the algorithm, we need custom sensor hardware. FPGA allows to design data-paths(see Table 2), which keep the algorithm design independent of any pre-processing step. Once such example is double buffering which allows slow data stream from flash memory to settle before the next data-set is applied.

5 Algorithm Acceleration

Table 2 shows all the optimizations available for algorithm performance tuning. The efficiency of the algorithm[4] depends on the Instruction count and CPI [12]. These parameters are determined by the processor design, once the type of tool sets is chosen then further optimization of the program size and speed are targeted. The notion of throughput of algorithm execution and its design dependency on the data-path demands is studied for all the computation models.

5.1 Static Program Order Model

The compiler uses basic block optimization techniques, which can use Instruction Level Parallelism (ILP). Optimization of this type uses infinite resources such as window size and base register counts. The expected throughput may be reduced, as it is dependent on the target. Program consistency and shared data coherency is available in the higher level program construct and can be addressed with simplicity. Some data-path optimizations are possible, memory dependencies, which are architecture dependent, are not addressed at this level, this step allow for fine grain instruction level parallelism.

5.2 Dynamic Out-of-Order Model

The performance measures computed with processor-based architecture and optimization from Table 2, Figure 5 illustrates that there is increase in clock rate

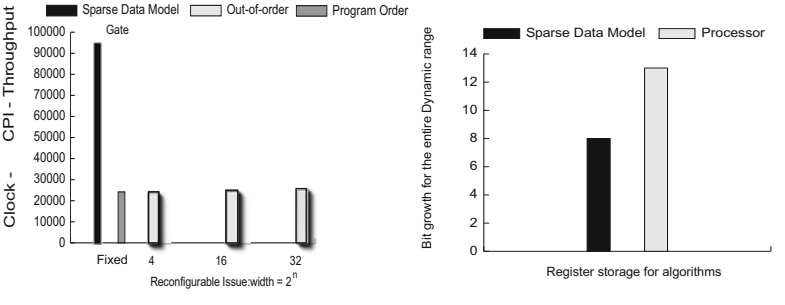
if the pipeline [1,12] is used. In our case, the pipeline depth is varied during re-configuration. The simulation results show that total increase is around 10% using dynamic optimizations. As long theyre available registers, the ILP can execute in parallel by increasing the issue rate gradually. RISC uses Tomusulu's algorithm [12] for register reservation to hide any load and read latencies. Processor based IPC mechanism and data cache allows to accelerate the data-path depending on the architecture. General purpose data forwarding, pipeline optimizations including branch predictions and speculation using out-of-order execution are supported in the hardware. The pipeline utilizes temporal parallelism.

5.3 STACK

Temporal and spacial data coherency is achieved by using the pre-processing and compressed sensing techniques. The algorithm allows to compressed signal without any information loss and further enhances the working data range of the algorithm. As there are no delays in load and store instruction in the hardware register language [7], it can further accelerate 40% of the instructions as shown in Figure 5(a), which uses dynamic range for bit growth and register allocation as shown in Figure 5(b) and equation (12). fine grain parallelism. At the same time, it can take advantage of instruction level parallelism, which is pipeline optimization technique used in the previous models.

$$Bitgrowth \leq \lg_2(\sum_{k=0}^{k<l} |f[k]|) \tag{12}$$

Where k is the number of co-efficient used by the algorithm, $|f[k]|$, gives the total possible range.



(a) Algorithm Throughput **(b) Dynamic Range and Bitwidth**

Fig. 5. STACK model algorithm performance versus static program order and dynamic out-of-order pipe-line optimizations

Table 2. Execution Optimization Parameters

Tools	Instruction Count	Clocks cycles per instruction (CPI)	Clock Rate	Program Consistency	Program and Data Chorency
Program	✓				
Compiler	✓	✓			
Instruction-set	✓	✓	✓		
Computational Model			✓	✓	✓
Technology Software/Hardware			✓		

6 Summary

Due to IP core requirements and dedicated algorithm implementations, STACK model gives the most flexibility compared to high-level software and hardware based IPC synchronization. The requirements for error correction by data coherency program are needed when sequential and atomic operations cannot be guaranteed in concurrent executions. Due to implementation in the hardware for the same algorithm, the program footprint is smaller keeping overheads low and avoiding the need for synchronization libraries to be included. The computational model choice of hardware implementation for a given gate count gives a higher bound on performance for the same clock rate compared to the Van Numen model. Higher performance for the same clock rate equates to lower power thus enhancing the lifetime of the sensor network.

Acknowledgments

One of the authors would like to thank the members of the Intel Parallel Processing lab at IIIT-Hyderbad and specially Nayan Mujadiya for extending his time and effort during CS4200 with Prof. Govindarajulu.

References

1. Iyer, V., Iyengar, S.S., Rama Murthy, G., Srinathan, K., Rakee, Srinivas, M.B.: Intelligent Networks Sensor Processing of Information using Key Management. In: Proc. 4rd International Conference on Sensing Technology - ICST, Leece, Italy (2010)
2. Baron, D., Duarte, M.F., Wakin, M.B., Sarvotham, S., Baraniuk, R.G.: Distributed Compressive Sensing. In: Proc: Preprint, Rice University, Texas, USA (2005)
3. Adve, S.V., Gharachorloo, K.: Shared Memory Consistency Models: A Tutorial
4. Brook, R.R., Sitharama Iyengar, S.S.: Robust Distributed Computing and Sensing Algorithm. ACM, New York (1996)
5. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann, San Francisco (1996)
6. Jensen, A., la Cour-Harbo, A.: Ripples in Mathematics, p. 246. Springer, Heidelberg (2001); Softcover ISBN 3-540-41662-5

7. Digital Signal Processing with Field Programmable Gate Array, UWe Meyer-Baese. Springer, Heidelberg (May 2001)
8. INSPIRE-DB: Intelligent Networks Sensor Processing of Information using Resilient Encoded-Hash DataBase. In: 2010 Fourth International Conference on Sensor Technologies and Applications (2010)
9. Krishnamachari, B., Member, IEEE, Sitharama Iyengar, S., Fellow, IEEE: Distributed Bayesian Algorithms for Fault-Tolerant Event Region Detection in Wireless Sensor Networks. IEEE Transactions on Computers 53(3) (March 2004)
10. Iyer, V., Sitharama Iyengar, S., Rammurthy, G., Srinivas, M.B.: SenseSIM: Sensor Network Simulator. In: ISSNIP, Melbourne, Austrlia (2009)
11. Slepian, D., Wolf, J.: Noiseless coding of correlated information sources (1973)
12. Hennessy, J.L., Horowitz, M.A.: An Overview of the MIPS-X-MP Project, Stanford University, Technical Report No. 86-300 (1986)
13. Iyer, V., Sitharama Iyengar, S., Murthy, G.R., Parameswaran, N., Singh, D., Srinivas, M.B.: Effects of channel SNR in Mobile Cognitive Radios and Coexisting Deployment of Cognitive Wireless Sensor Networks. In: 29th IEEE International Performance Computing and Communications Conference, IPCCC 2010, Albuquerque, New Mexico, USA (December 9-11, 2010)