SARMTI Implementation on KONTRON STAR VX Platform with 3rd generation Intel® Core™ Processors

22 October 2013

Gedae

Intel

NA Software

KONTRON

Table of Contents

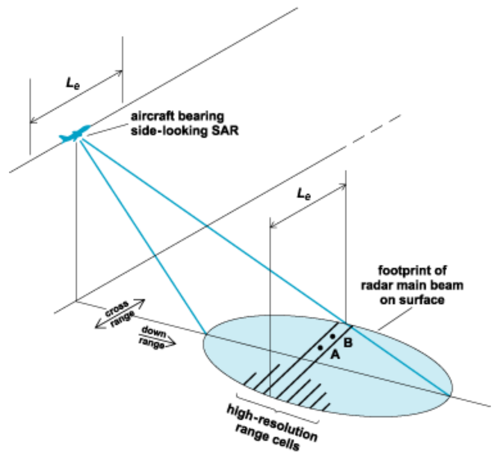
Introduction

The objective of this effort was to explore the parallel implementation of NA Software SARMTI algorithm on KONTRON's STAR VX hardware platform with multiple blades each with one Intel 3rd generation Intel® Core™ processor. A secondary object was to verify the benefits of using Gedae's Development Environment both in terms of productivity and software efficiency. So the measures used are the processing time of the hand code produced by hand with the support of Intel experts vs the processing time of the same algorithms developed using Gedae, and the code size of the handcoded software vs the code size of the version implemented in Gedae's Idea language. The paper will additionally include measures of function speed for various functions, some qualitative assessments of the processor load and measure of data transfer rates on the PCIe bus in KONTRON's STAR VX hardware platform.

NA Software SARMTI Algorithm

NA Software (NAS), an Affiliate member of the Intel Alliance, develops and licenses advanced radar algorithms to defense organizations and corporations around the world. NAS also produces and sells low-level DSP software and signal processing libraries to board manufacturers that sell DSP products into the defense and aerospace sector. This software enables defense contractors to deliver their solutions and products to their customers quicker and more efficiently.

This study is based on a radar algorithm developed by NAS called SARMTI. The geometry of an example radar system is shown in the diagram to the right with a sidewards looking radar system mounted on an aircraft. As the plane flies along the radar system is used to illuminate the ground.



L_e
aircraft bearing side-looking SAR
L_e
footprint of radar main beam on surface
cross range
down range
B
A
high-resolution range cells

Figure 1 - Geometry of Example RADAR

Radar surveillance systems are usually designed to either operate in a SAR (Synthetic Aperture Radar) mode or a MTI (Moving Target Indication) mode. The SAR mode produces imagery and the MTI mode produces detections of moving targets on the ground. There are many SAR sensors in operation around the world but few MTI systems that can detect slow moving targets on the ground due to their expense and large computational workload. The SARMTI algorithm by NAS uses a SAR sensor to detect moving targets. This algorithm has the following advantages:

- Any SAR radar system can be turned into a MTI system.
- Detection of moving targets to a high probability of detection and low false alarm rate is possible even when targets and clutter (ground) returns are mixed.

- Accurate estimation of target position, velocity and acceleration is possible. This is not possible with a conventional MTI system when target and clutter returns are mixed.
- The radar system can give both high-resolution imagery and target information at the same time.
- Target recognition based processing is also possible even when target and clutter returns are mixed.

An example of the output of the SARMTI algorithm with real radar data from an operational SAR system is given below:

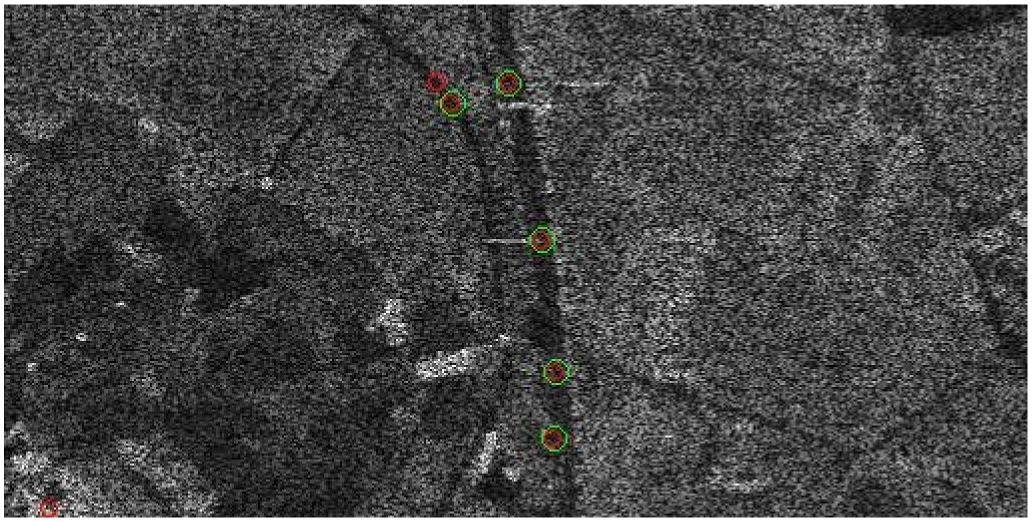


Figure 2 - Output of SARMTI Algorithm for Operational SAR System

The picture above shows a SAR image of a road with five vehicles on it. The SAR image was produced by a SAR processor written by NAS, the green circles show the ground truth information (the real moving targets). The red circles represent the output from the SARMTI processor. In this example the SARMTI processor has detected and positioned all five moving targets correctly with one false alarm.

Ecosystem

Each one of the participants in this study brings a critical capability to the table. This section summarizes those capabilities.

Intel

Intel has a family of processors that bring great value to the defense industry. The performance of these processors supplies the defense industry with the computer power it needs to develop advanced defense systems. Intel continues to develop a deeper understanding of the lower SWAP and high performance requirements defense industry and to provide processors that meet those requirements.

KONTRON

KONTRON's STAR VX system is a pre-qualified hardware platform that brings the power of multiple Intel processors with high speed data fabric to the defense industry. It brings a balance between the IPC between boards and the compute power on the boards.

The basic structure of the system used in this benchmarking is shown in figure 3. It consists of 6 blades each with a single Intel quad core processor with the blades interconnected by a PCIe bus. The system also has a 10GETH connection not shown in this diagram.

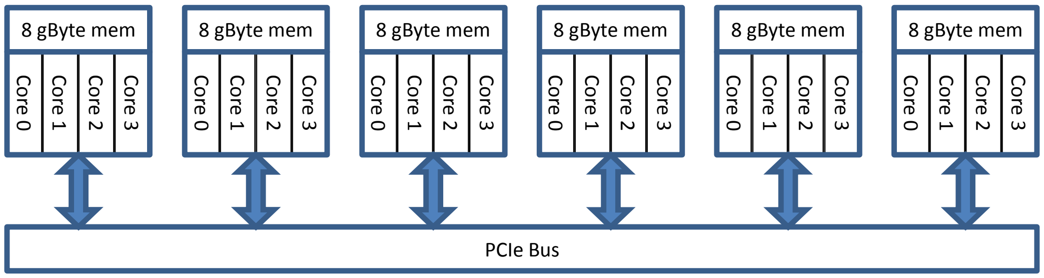


Figure 3 - Architecture of StarVX System

Gedae

The defense business has changed. Competition is tough and budgets are smaller. Customers will not accept paper designs and fund proof of concept (POC) development like they did in the past. Contractors have to build POC demonstrators, fielded demonstrators, production prototypes and even production systems using BD and IR&D budgets. The challenge is that software development dominates the risk, schedule and cost of sensor system development. While hardware development has progressed dramatically in the last 20 years, software development has only made incremental progress.

Gedae's offering revolutionizes software development. Gedae's compiler extends the von Neumann programming model. When considering what Gedae is, Gedae is in concept to a compiler. Gedae has no relationship to a layered software approach. The von Neumann programming model converts a program written in a von Neumann language into a sequence of instructions for a von Neumann processor with 1 memory and 1 CPU. Effectively the collection of instructions is the model of the hardware that the von Neumann compiler targets. Gedae's compiler targets a collection of von Neumann processors, memories and data paths (the Gedae processor). Gedae adds, among other things, data transfer instructions to the von Neumann set of instructions. We call this the Gedae instruction set. Gedae

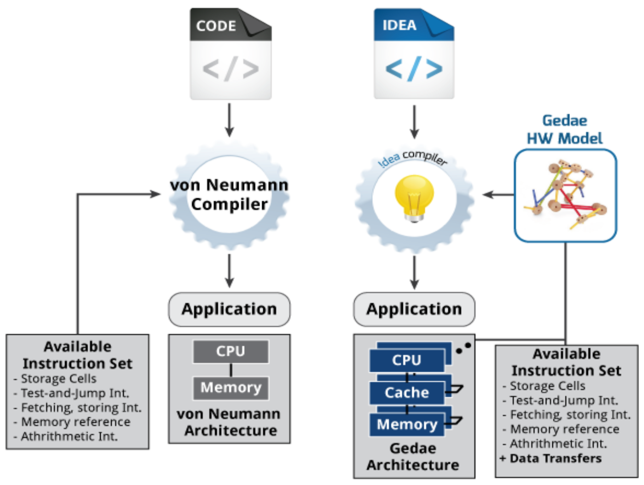


Figure 4 - The differences and similarities between the von Neumann and Gedae programming models

extends the von Neumann compiler concept to build software for the Gedae processor using the Gedae instruction set. In addition to the instruction set, Gedae's compiler also uses a hardware model of the targeted computer system that describes the topology of processors, memories and data paths.

The technology was reduced to practice with a development environment, and a dataflow language and compiler in 1995. The product's value was proven on production programs and generated revenue. But it became apparent that the only successful business model required the product have 1) a language that could be used by all software developers regardless of their skill level, 2) a general purpose language that could be used to build all the software (signal, image and data processing, and control) for a sensor system, 3) a compiler that generated an application at least as efficient as one written by experts and 4) the compiler had to work on any processor and any system of processors.

The result was not anticipated.

1. The language is abstract. It describes the behavior of the software but doesn't dictate directly a sequence of instructions. It can't because the Gedae processor is parallel and so the instructions are not executed in sequence. But it does use syntax and semantics common in mainstream computer languages such as algebra, loops, conditionals, functions and data structures. Since Gedae's Idea language extends von Neumann languages it is general purpose.
2. The compiler can be reused on a broad class of targets without modification. A hardware model describes the topology of processors, memories and data paths, and registers the Gedae instruction set for a particular processor. The compiler constrains the application to run on the topology described and uses the registered instruction set.
3. The compiler automation produces code that is more efficient than even the most skilled programmers can practically write by hand. Data movement and memory efficiency are critical to performance. In order to meet the requirement to achieve high efficiency the compiler had to optimize the layout of data in memory, the schedule of memory use, the data transfers and the concurrency.
4. The language is even more abstract than expected. In order for the compiler to optimize the application, it takes responsibility for the sequence of instructions, the layout of data in memory including reuse of memory and multi-rate processing. As a result the language abstraction is pure in the sense that it must allow the compiler to determine data rates, data sizes and sequence without ambiguity.
5. Before the compiler can successfully build the software, it had to verify the software is correct. So, the compiler links together the entire application and verifies that it can build software that runs and duplicates the specified behavior – informing the developer of the precise source and cause of any ambiguities that exist.

As a result, Gedae's Idea language is more abstract than MATLAB's language in the sense that the developer is not directly responsible for implementing the multirate processing and the Idea language is more complete because it uses array algebra instead of matrix algebra.

Gedae fixes the core issues of software development. The risk, schedule and cost of development are dramatically reduced making it possible to successfully react to the changing state of the defense industry and produce POC demonstrators, field demonstrators, production prototypes and production systems using IR&D and BD budgets.

Description of Gedae Development Environment

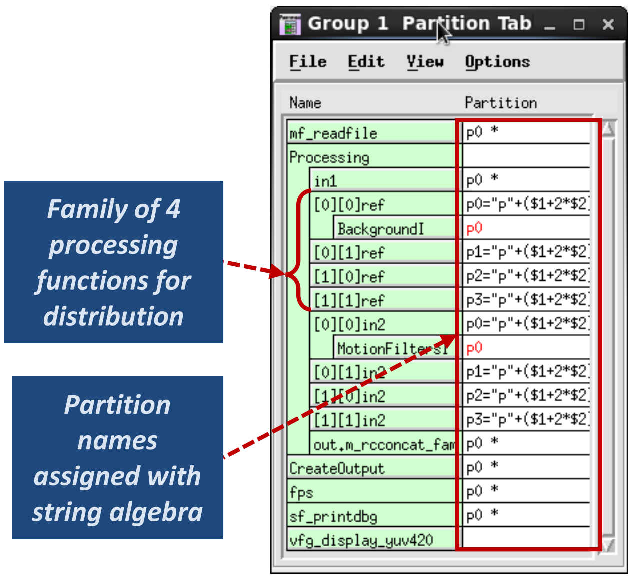
This section is a brief overview of Gedae's Development environment. More detail can be found at www.gedae.com.

The Idea Language

Gedae's Idea language is similar in abstraction to the MATLAB language making algorithm and application developers productive. The Idea language has the entire set of usual control features such as loops and conditionals and a sophisticated array algebra language. One of the more important rather unique features is the explicit support of data streams. The effect of the streaming features is that functions can be written once and the compiler is responsible for implementing the multi-rate behavior accommodating a regular stream of data combined with parameters that change asynchronously – specifically on state machine transitions. In fact, if the code is used to compute a parameter value that doesn't change the compiler will compute the values during compilation and set them as parameters.

Mapping Functions to Multiple Cores – Compiler Settings

A table that lists all the functions in the application is provided to the user so the user can define sets of functions (a mathematic set partitioning). An example is shown in figure 5 to the right. We call this the partition table. Each set will be mapped to a different processing core in the Gedae processor model. The table is hierarchically organized according to the functional break down of the application and can easily accommodate tens of thousands of functions. The set assignments can be done using equations and string algebra to accommodate large and/or complex mappings. In the example shown, the indices on the right hand side of the variables, e.g. [0][1]ref, are family range variables and indicate that there are a family of functions. In the expressions for the partition names the $1 and $2 refer to these range variables, i.e. [$1][$2]ref.


Group 1 Partition Tab
File
Edit
View
Options
Family of 4 processing functions for distribution
Partition names assigned with string algebra

Name	Partition
mf_readfile	p0 *
Processing	
in1	p0 *
[0][0]ref	p0="p"+($1+2*$2
BackgroundI	p0
[0][1]ref	p1="p"+($1+2*$2
[1][0]ref	p2="p"+($1+2*$2
[1][1]ref	p3="p"+($1+2*$2
[0][0]in2	p0="p"+($1+2*$2
MotionFiltersI	p0
[0][1]in2	p1="p"+($1+2*$2
[1][0]in2	p2="p"+($1+2*$2
[1][1]in2	p3="p"+($1+2*$2
out.m_rcconcat_fam	p0 *
CreateOutput	p0 *
fps	p0 *
sf_printdbg	p0 *
vfg_display_yuv420	

Figure 5 - Example of function partitioning table used to specifiy implementation

There are a complete set of tools for specifying and analyzing the implementation. The defaults are good. Specifying the implementation and letting the compiler build optimized code generally take more than 10X less effort than using traditional approaches and dramatically reduces faults.

Collection and Analyzing Execution Trace Data

The compiler collects trace events at the developer's request (a toggle button). The collection is very detailed and the trace tool provides the user with tools to manipulate the table for purposes of analysis. An example trace is shown in figure 6. The same data is displayed both on a per processor display (p0 to p3, the client line is the development environment) and function by function. The black bars are events that are processing application data. The red and green bars are overhead on the processing core

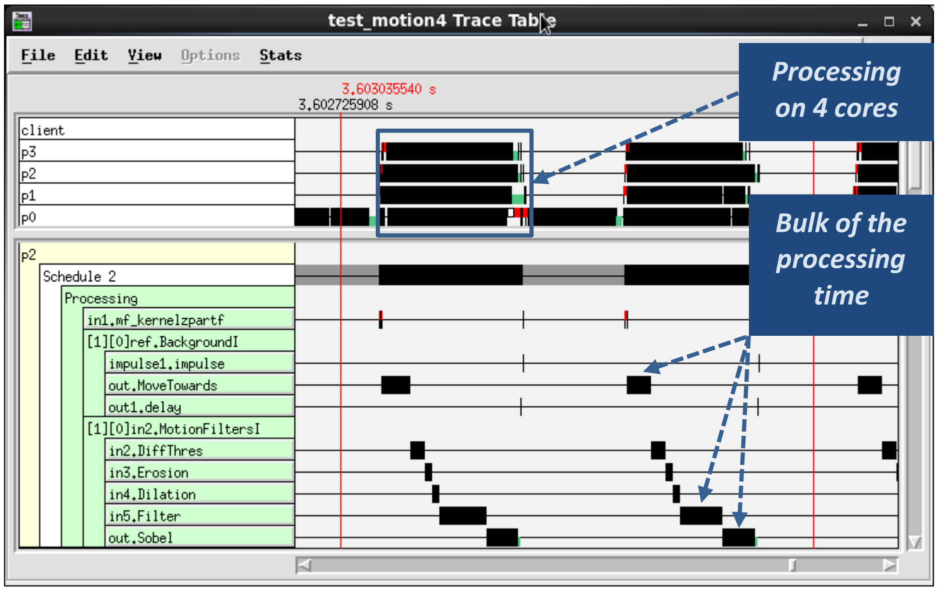
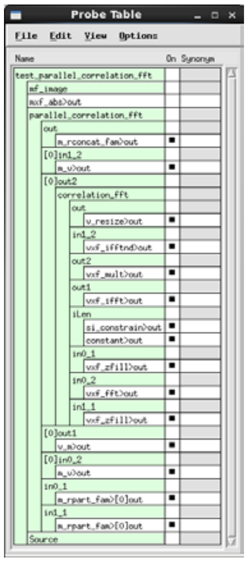


Figure 6 - Execution trace showing all evens on all CPUs in the system

associated with transferring data and controlling currency of processors. There are a collection of analysis tools that allow the developer to determine which functions are using the most time, for analyzing and optimizing the cost of data transfers and for determining the cause of any processor idle time. In the example shown p1, p2 and p3 are idle for a period of time when a function that is not running in parallel is occupying p0.

Collecting Probe Data and Regression Testing

The compiler collects probe data at the developer's request. A table listing all of the outputs of functions is presented to the developer. See figure 7. The developer selects those data items he/she would like to probe. The example in figure 7 shows all outputs of the processing algorithm are selected. Gedae implements the algorithm with probe functions that write data to files in a directory of probe data for all the outputs specified. Gedae also creates a dictionary so the probe data can be queried for analysis during development and then used as regression test data during the course of partitioning, mapping and optimizing the implementation for the target hardware. The developer uses the Gedae regression testing tools to verify that the outputs of the target version have



Name	On	Synonym
test_parallel_correlation_fft		
mf_image		
mxf_abs>out		
parallel_correlation_fft		
out		
m_rconcat_fam>out	■	
[0]in1_2		
m_v>out	■	
[0]out2		
correlation_fft		
out		
v_resize>out	■	
in1_2		
vxf_ifftnd>out	■	
out2		
vxf_mult>out	■	
out1		
vxf_ifft>out	■	
iLen		
si_constrain>out	■	
constant>out	■	
in0_1		
vxf_zfill>out	■	
in0_2		
vxf_fft>out	■	
in1_1		
vxf_zfill>out	■	
[0]out1		
v_m>out	■	
[0]in0_2		
m_v>out	■	
in0_1		
m_rpart_fam>[0]out	■	
in1_1		
m_rpart_fam>[0]out	■	
Source		

Figure 7 - Probe Table

not changed or at least are within suitable limits. Generally errors are the responsibility of the compiler (Gedae, Inc.), but in some cases functions optimized for the CPU used when implementing on a target give slightly different results. The developer must make the decision as to whether these differences are acceptable. This is more likely an issue if a co-processor like an FPGA or GPU is targeted than if it is an Intel GPP or a TI DSP. Probe display and compare functions provided by the Gedae uses standard comparison techniques and “MATLAB like” displays. The developer is free to develop both display and comparison functions required in special situations. In addition, Gedae provides menu items to export probe data in .mat formats for analyzing the data using MATLAB.

Summary

The Gedae Idea Development Environment (GIDE) is a complete and powerful development capability.

Description of the SARMTI Parallelization Strategy

NA Software with advice from Intel hand coded an optimized parallel version of the software to demonstrate the ability to achieve HPEC performance on a KONTRON hardware platform with multiple Intel quad core processors. The version was developed for a 4 core Intel processor and the timing numbers for that version are included in this report. In addition, NA Software implemented a simple round robin scheme to expand the processing to 6 quad core processors. The disadvantage of this scheme it that round-robin dimension contained 7 iterations that do not fit well on 6 processors. The timing numbers for that experiment are also reported in the paper.

It follows that with a description of the structure of the data tokens being processed and describes how the tokens are decomposed for distribution among the processing cores.

The Parallelization Strategy

Only the amount of computation not the details of the algorithm are important to the parallelization strategy. The algorithm, as expected, does lots of FFTs and vector arithmetic and repeats them for various parameterizations so the computational burden is very high. The most important details are the structure of the data (vector, matrix, array3d etc.) and the data that is processed as an atomic operation. For example, it is important to know that each row of data in a matrix is stored in contiguous memory. Then computing a vector operation like an FFT on each row is straight forward. On the other hand it is often quite inefficient to do a vector operation on the columns since that data is strided in memory. One usually transposes the data in memory to put the columns in contiguous memory to improve efficiency of the computation.

The Structure of SARMTI Processing

The SARMTI algorithm takes an RxC complex valued data input and runs an identical algorithm, with different parameterizations, 7 times to produce output target locations.

The hand coded version divides the workload onto the processors by dividing as evenly as possible the 7 processing steps onto the P processors. Each of the 7 process steps is then further parallelized by dividing the workload onto the 4 cores.

The Gedae Idea version processes these 7 steps sequentially and parallelizes the algorithm by parallelizing the Process blocks onto the P x 4 processing cores. A diagram of the software processing architecture is shown in figure 8. The hand coded version divides the processing as shown by the red vertical lines. The load balancing is not very good as shown since 2 processors are idle 1/3 of the time. The Gedae Idea version divides the processing as shown by the green horizontal lines. The dimension of the data within the processing is much bigger than 7 and more evenly divides the processing for better load balancing.

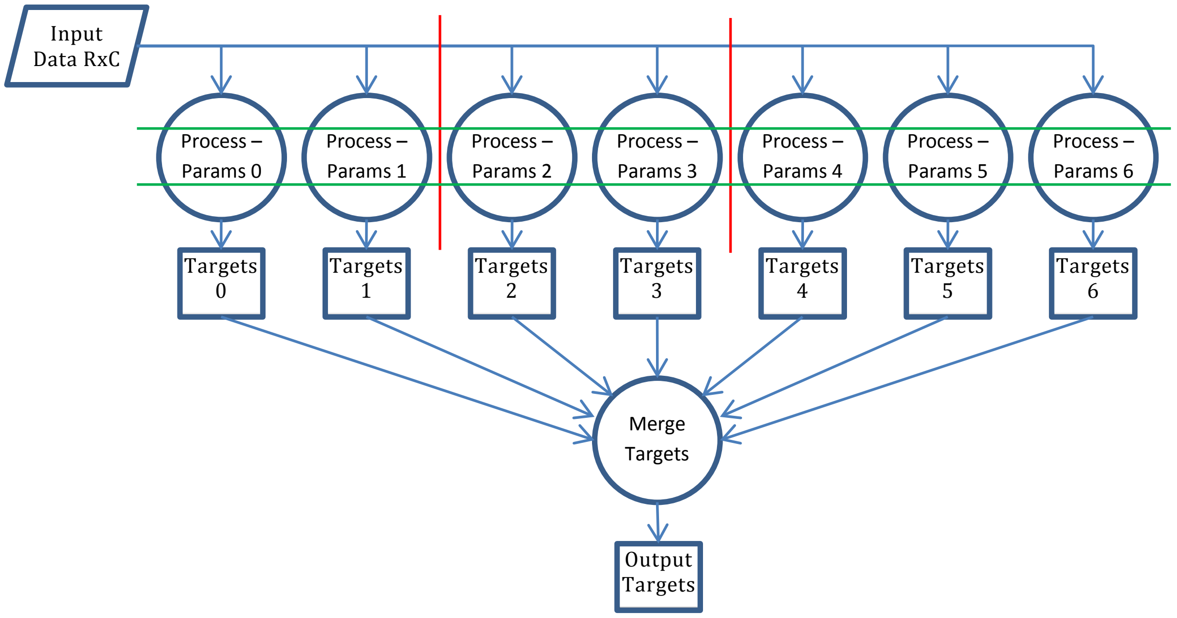
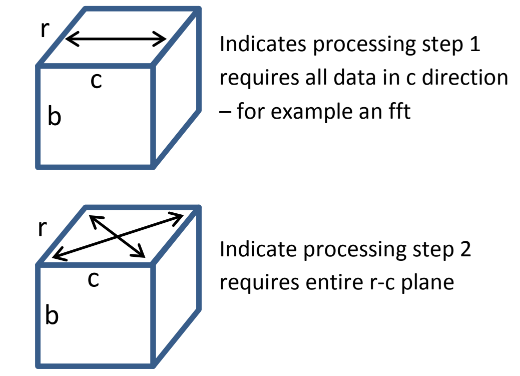


Figure 8 - Algorithm Architecture

The Structure of SARMTI Data

Within each one of the processing blocks the data is a 3d array. In the diagrams in figure 9 the cubes are displayed with the fastest moving index horizontal – c dimension (so that dimension is in contiguous memory), the second fastest back into the image – r dimension (show as a diagonal) and the slowest index is vertical – b dimension. These are illustrated on the left side of figure n.

The c vectors and r x c matrices are in contiguous memory and the c dimension strides through memory



r
c
b
Indicates processing step 1 requires all data in c direction – for example an fft
r
c
b
Indicate processing step 2 requires entire r-c plane

Figure 9 - Processing dependencies

Processing steps of data in the data cube have different requirements as to what data the algorithm needs to access. Some algorithms need to atomically access each of the rxb vectors with c data elements. Some need to access one of the b planes with rxc data points. These are illustrated in figure 9. Note that while the two processing dependencies could be handled by the second division of data – it gives better load balancing to divide the relatively large r direction onto the small number rather than divide the data in the b direction. Note also that converting between data division in step 1 and 2 will be almost cost free because the processor cores will share the same memory just different parts of the memory.

Part of the SARMTI algoritm is illustrated in figure 10 for a P (3 in the figure) processor system. The algorithm begins by broadcasting an rxc input image to P processors and then to the 4 cores within each of the three processors. This two phase process uses DMA IPC to communicate the image to the P processors and allows zero copy shared memory to be used for broadcast to the 4 cores. An rxc image has all the information needed to generate a bxrxc data cube. In Step 2 vectors are generated a row at a time. The output data has to be decomposed along boundaries that keep the entire data set processed by a function in contiguous local memory. Because the b dimension is relatively small dividing the processing in the b dimension onto the 4xP cores gives poor load balancing. The memory layout that most equally partitions the cube is to divide the b dimension onto 3 processors and the row dimension onto 4 cores. By contast the data in step 3 requires a complete plane of r by c so both the processor and the core decomposition is along the b dimension. In this case the only choice is to divide the b dimension onto the 4P cores. The data doesn't have to be exchanged among memories in Step 3 because the data cube that spans the full r by c plane is all stored in the memory of the processor and is only separated into separate memories as it is moved through the processing hierarchical cache memories attached to each processing core. The compiler optimizes the transfer and also concurrency of the processing to insure that the transfer is as efficient as possible. In addition, since the compiler implements the transfer there is no need to use an IPC layer that has abstraction beyond lowest most efficient interface. In Step 4 the results of Step 3 are summed into a single rxc image. This summation is done by first summing the images across the cores and then summing those partial sums into a final image. The IPC for summing across the core uses zero copy shared memory transfers while the summing across processors uses more DMA IPC. Figure 7 is actually a simplified version of the algorithm. In reality there are more subcubes generated and other interesting divisions of the data. However in all cases the Idea algorithm merely describes how the data is to be partitioned and the appropriate IPC (DMA or shared memory) is added by the Gedae compiler.

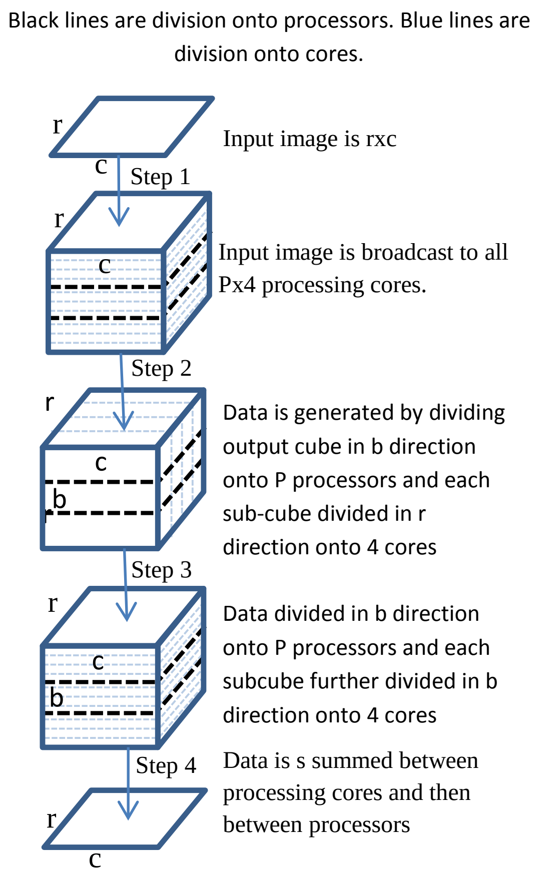

Black lines are division onto processors. Blue lines are division onto cores.
r
c
Input image is rxc
Step 1
r
c
Input image is broadcast to all Px4 processing cores.
Step 2
r
c
b
Data is generated by dividing output cube in b direction onto P processors and each sub-cube divided in r direction onto 4 cores
Step 3
r
c
b
Data divided in b direction onto P processors and each subcube further divided in b direction onto 4 cores
Step 4
Data is s summed between processing cores and then between processors
r
c

Figure 10 - Decomposition of data tokens for parallel processing

Results

The results demonstrate the efficiency of applications produced by the Gedae compiler, the reduction of cost and the compression of the development schedule.

NASoftware – Hand Optimized C Code

The hand written code had about 10,000 semi-colons. There were about 1,000 semi-colons in functions that were free of state data, memory allocation and threading. The remain 9,000 lines of code were largely dedicated to infrastructure to implement IPC, threading, memory allocation and management, execution timing, and debugging tools.

The results of the hand coding are shown below. The implementation of this code for a single processor with 4 cores was done by expert developers with advice from Intel experts. The problem of implementing and optimizing performance on multiple processors was going to require serious surgery on the code to accommodate the corner turn of the data using the backplane. NA Software only had sufficient resources to implement a version that distributed the time sequence of processing the 7 data cubes across the 6 processors. This effort was quite successful but had the handicap of not very good load balancing. The timing results of the distribution are show in table 1 below. The 5 experiments are for various data sizes. The ratio of the size of the data set for each demonstration relative to the size of the data set for demonstration 1 is shown in table 7 as well.

Table 1- Timing Result for Hand Optimized Code

Hand Coded Timing Results [in seconds]		**Demonstration #**				
		1	**2**	**3**	**4**	**5**
Data Ratio		1				
[N] Procs x [M]Cores	**1x1**	6.80	14.20	28.30	72.60	285.50
	1x4	2.05	4.42	8.61	21.50	87.60
	2x4	1.17	2.39	4.73	12.00	48.60

Gedae, Inc. – Gedae Idea Language and Compiler

The 1,000 lines of “pure” C code were left untouched in the Gedae implementation. The 9,000 lines of code were converted into about Gedae Idea code was 750 lines of code. The code reduction was partly due to the abstraction of the Idea language but mostly due to the removal of the IPC, threading, memory management, execution timing and debugging code. That code is implemented for an application by Gedae’s compiler. Those instrumentations are used during development and not included in the final version of the software unless the developer needs access to selected information in the deployed application.

The timing results are shown in table 2. The result is that the code generated by the Idea compiler runs about 50% faster than the hand coded version. The increased processing throughput is likely a result of several factors. For one thing, since Idea doesn’t support recursion, some of the code that recursively looked for connected regions in a Boolean image was relaced by a bwlabel kernel that labels the connected region and accesses the memory in a regular pattern whereas the recursive algorithm accesses it in an irregular pattern. But that effect was likely quite small. The primary increases in processing speed were due to the smaller code size, the optimized IPC and concurrency control, and probably most important the optimization of the complete memory plan. While the Idea compiler creates C code that it passes to the C compiler – it doesn't leave any memory allocation or planning to the developer or the C compiler. Since it owns all the memory, the Idea compiler is able to do global optimization of the data layout in memory and the schedule of memory use over time.

Table 2 - Timing Results for Code Optimized and Implemented by Gedae

Gedae Timing Results [in seconds]		**Demonstration #**				
		1	**2**	**3**	**4**	**5**
[N] Procs x [M]Cores	**1x1**	4.03	8.71	17.50	42.50	172.40
	1x4	1.32	2.95	5.91	14.70	57.80
	2x4	0.74	1.63	3.25	8.20	31.80

Conclusions

The conclusion is that automation gives a compressed development schedule, reduced development cost and more efficient code. In other comparison programs it has been shown that it also gives generally better quality code – though the code written by NA Software was very high quality.