Gedae Runtime Kernel Performance Characterization

Kerry Barnes (kerry@gedae.com), William Lundgren (bill@gedae.com)
Gedae, Inc.
18000 Horizon Way, Suite 200, Mt. Laurel, NJ 08054

Objective

The objective of this work is to characterize the execution overhead of the Gedae runtime kernel with the goal of determining how the overhead can be reduced. The kernel was instrumented to measure the time of various functions including firing function boxes, changing schedule state, and managing dynamic queues. A set of benchmark graphs was chosen to exercise the kernel. The results of the measurements made by running these graphs are presented showing Altivec target processor overhead of 1-3 µsec per function firing and an efficiency of over 90% for the benchmarks. While this overhead is low, the results suggest several promising areas for reducing the overhead further.

The runtime kernel controls the running of an application by scheduling function box execution, managing dynamic queues and propagating segment[1] boundaries. Scheduling can be divided into static scheduling and dynamic scheduling. In static scheduling the function boxes are fired in a predetermined order. Dynamic scheduling involves choosing which static schedule to fire based on availability of data in dynamic queues. All of these control activities are overhead to the work of the graph, which is firing the boxes that implement the graph algorithm.

[1] A segment is a finite length stream of data within a continuous data stream. Successive segments are processed independently with reset, parameter setting and clean-up. Each segment can be processed by a different subgraph (mode).

The control functions implemented by the runtime kernel are not unique to Gedae and need to be implemented for most applications. Because the runtime kernel abstracts these functions into a relatively small amount of code, Gedae provides the opportunity to focus on improving the efficiency of this code.

Method

The following method was used to characterize the runtime kernel:

1. A set of timer functions was created to accurately measure the elapsed time of different sections of the code.
2. The runtime kernel code was instrumented with these timer functions. For these tests, 44 sections of code were timed.
3. Three benchmark graphs were chosen to exercise the runtime kernel.
4. The benchmark graphs were executed on 6 different processor types, and the timer statistics for these graphs were obtained.
5. The results were analyzed to determine what areas of the runtime kernel most need improvement.

The three benchmark graphs are:

1. rt_stap (Real Time Space Time Adaptive Processing) a Mitre benchmark
2. e_comm – Offset quadrature phase shift key system simulation
3. nr – A noise removal algorithm for speech data

The graphs use the following runtime kernel features:

	rt_stap	e_comm	Nr
static scheduler	x	x	X
dynamic scheduler		x	X
segmentation			X

The 6 processors include three host processors – a Linux PC, a Solaris UltraSPARC and an NT PC processor – and three target Altivec DSP processors. The Altivec processors are representative of the type of processor on which embedded Gedae applications run.

Results

The table below shows the summary results for all 3 graphs running on the 6 different processor types. The efficiency is the total time firing function boxes (the useful work) over the total execution time. The overhead is the average time required to fire a function box.

Table 1: Summary Efficiency (%) and Overhead (µsec/box firing)

Graph:	rt_stap		e_comm		Nr	
Proc	**Eff**	**Ov**	**Eff**	**Ov**	**Eff**	**Ov**
Linux	99	1.5	73	4.7	68	9.2
Solaris	98	14	79	32	90	22
NT	99	1.6	79	6.0	89	16
Altivec1	97	1.7	91	2.7	95	1.8
Altivec2	93	1.1	92	2.8	92	2.0
Altivec3	98	0.5	96	1.9	90	2.0

When comparing processors of the same type the important number is the overhead. While high efficiency is generally desirable, one processor having a higher efficiency than another may be the result of function boxes being slower rather than the overhead on that processor being smaller.

Conclusion

The runtime kernel performance was measured on three representative graphs and all achieved an efficiency of over 90% on the target DSP platforms. While this efficiency is good, the performance measurements showed which areas of the kernel code could be improved to achieve even greater efficiency. Analysis of the 44 sections of code timed showed that the following categories are promising areas for performance improvement:

1. static scheduler – the code required to fire a function box in a static schedule
2. copy functions – copy functions to manage dynamic queue data
3. schedule state functions – functions that change a schedule's state
4. queue control functions – the interface between the queues and the static schedule state

These 4 categories account for the majority of the overhead in all three graphs. For Altivec1 (which is typical) the percentage of the overhead used by each category is:

Table 2: Percentage of Overhead Accounted for by Different Functions

Function:	rt_stap	e_comm	nr
static scheduler	90	20	18
copy functions		18	9
schedule state		36	26
queue control		10	15
Total	90	84	68

Three ideas for improving the runtime kernel performance are:

1. Code-generate multiple box firings into a single box firing to reduce the static scheduler overhead.

2. Avoid copying data from dynamic queues into schedule memory by manipulating the pointers in the box state vectors to point into the dynamic queues.
3. Optimize the functions for managing the schedule-heaps to improve the efficiency of the schedule state change functions

More graphs should be added to the benchmark suite to gain confidence in choosing the areas for performance enhancements. The method can easily be applied to other graphs and will be made generally available to users to do so.