

Gedae Runtime Kernel Performance Characterization

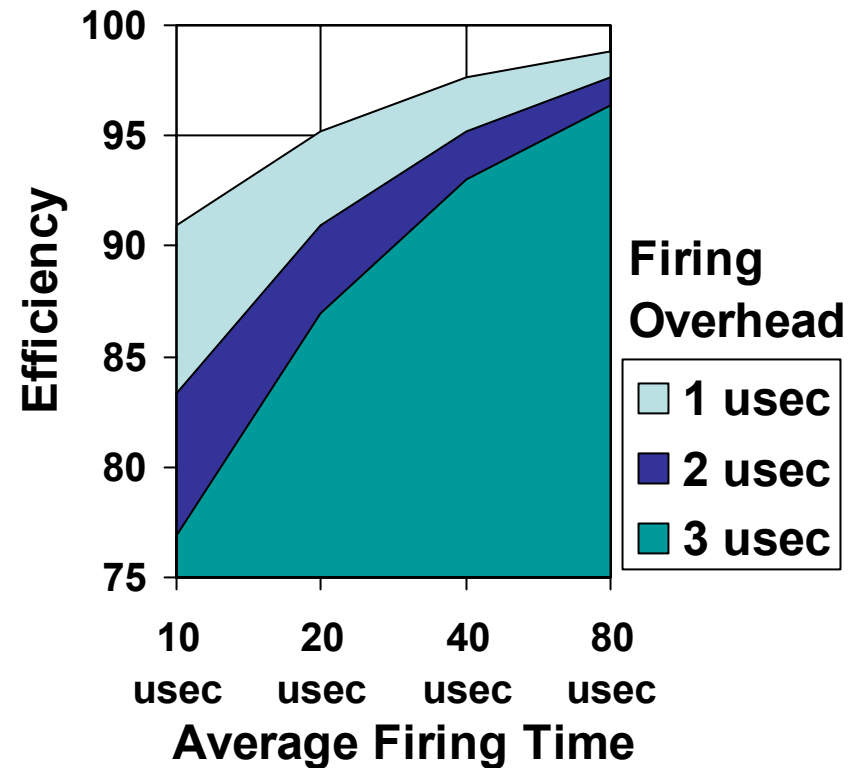
By Kerry B. Barnes

29 February 2003

Efficiency of Gedae Applications

- How efficient are Gedae applications compared to hand coding?
- What is the overhead of the Gedae Kernel?
- What is the cost of executing a primitive?
- What is the cost of dynamic vs. static scheduling?

Efficiency vs Firing Overhead and Average Firing Time



Two ways to define efficiency

- Efficiency = $\frac{\text{Handcoded execution time}}{\text{Gedae execution time}}$
- Efficiency = $\frac{\text{Time firing primitives}}{\text{Total execution time}}$
- The first estimate is what you really want to know
- The second definition has the advantage that it is based on measurable values

Overhead

- Defined
 - $\text{Overhead} = \text{Total execution time} - \text{Time firing primitives}$
 - $\text{Overhead/Fire} = \text{Overhead/Number of Box Firings}$
- Overhead/Fire provides a convenient measure of kernel efficiency that is independent of primitive execution times

Overhead

- Overhead functions exists for hand coded applications too!
 - Often ad-hoc implementation
 - Same problems solved repeatedly
 - Single point solutions can be efficient
- Gedae has following advantages:
 - Abstracts overhead functions to a small amount of code
 - Can be characterized
 - Effort on optimizing Gedae overhead functions benefit all Gedae applications

Objective



- The objective of this work is to characterize the overhead of the Gedae runtime kernel with the goal of determining how the kernel speed can be improved

Method

- Create timer functions
- Instrument code with timer functions
- Select benchmark graphs
- Run graphs and get timer results from instrumented code
- Determine optimization areas

Timer Functions

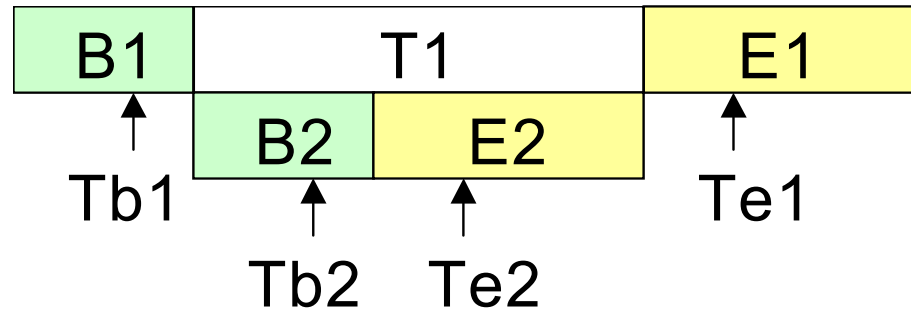
- New Functions
 - `embBeginTimer`
 - `embEndTimer`
- Gather elapse time statistics of code between begin and end

• Usage

```
Func1 () {  
    embBeginTimer(F1_TIMER);  
    ... code to be timed ...  
    embEndTimer(F1_TIMER);  
}
```

Timer Function Characterization

```
for (i=0; i<1000; i++)  
{  
    embBeginTimer (TIMER1);  
    embBeginTimer (TIMER2);  
    embEndTimer (TIMER2);  
    embEndTimer (TIMER1);  
}
```



$$T_t = T_{e2} - T_{b2}$$
$$T_{pt} = T_{e1} - T_{b2} - T_t$$

Timers are used to characterize themselves

Timer Characterization Results

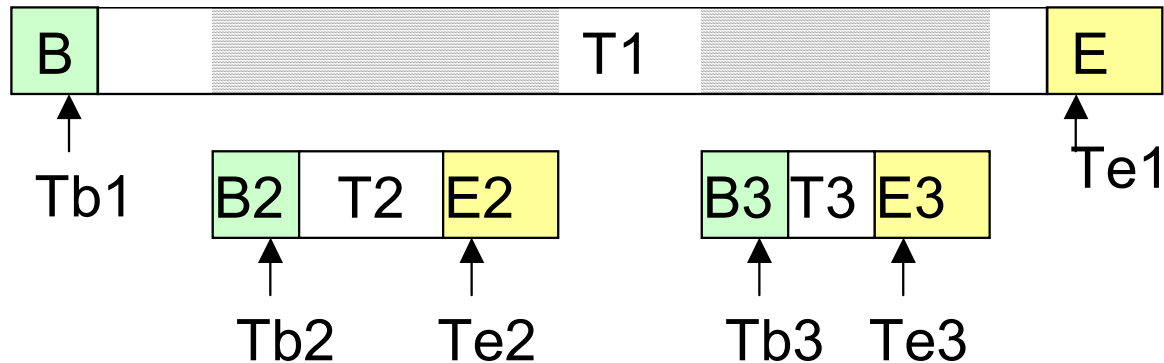
Processor	Time	Mean	Stdv	Min	Max
Linux	Tpt	1.5	0.5	1.0	2.0
	Tt	0.8	4.0	0.0	125
NT	Tpt	3.6	0.4	3.3	9.5
	Tt	1.8	1.8	1.7	57.0
Solaris	Tpt	1.4	0.7	0.0	3.0
	Tt	0.5	0.5	0.0	1.0
Altivec1	Tpt	1.4	0.02	1.4	2.0
	Tt	0.6	0.005	0.5	0.7
Altivec2	Tpt	1.9	0.2	1.7	2.6
	Tt	0.7	0.1	0.7	1.5
Altivec3	Tpt	1.9	0.07	1.9	3.9
	Tt	0.7	0.02	0.7	0.8

The Altivec-X processors are target BSP processors

Altivec1 has the most consistent timer times

We are not prepared to compare target BSPs at this time!

Example Timer Calls



$$T2 = Te2 - Tb2 - Tt$$

$$T3 = Te3 - Tb3 - Tt$$

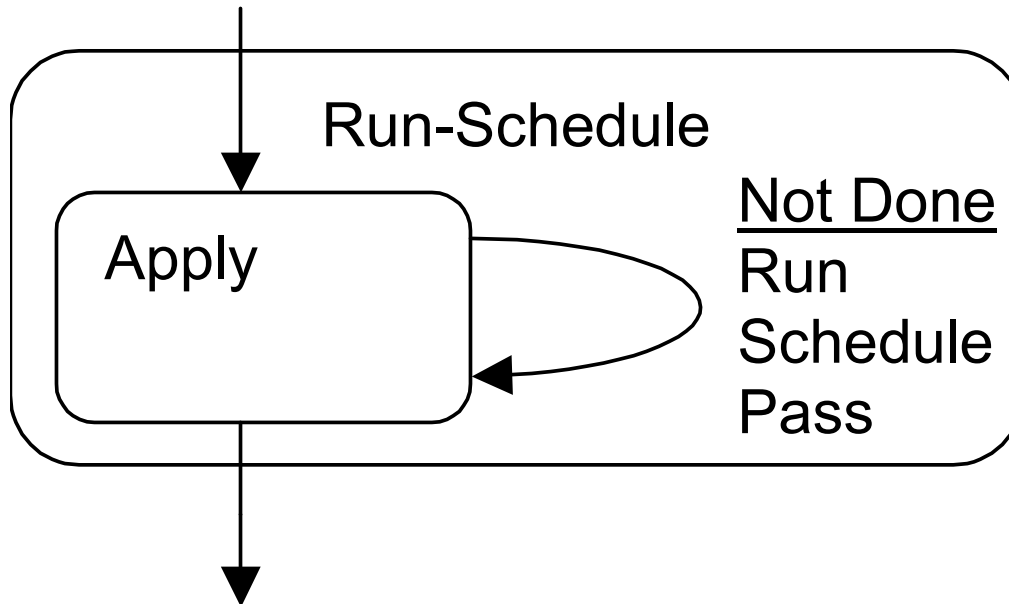
$$T1 = Te1 - Tb1 - Tt - T2 - T3 - 2 * Tpt$$

Ellapse time does not include nested timer calls

Instrumenting Kernel

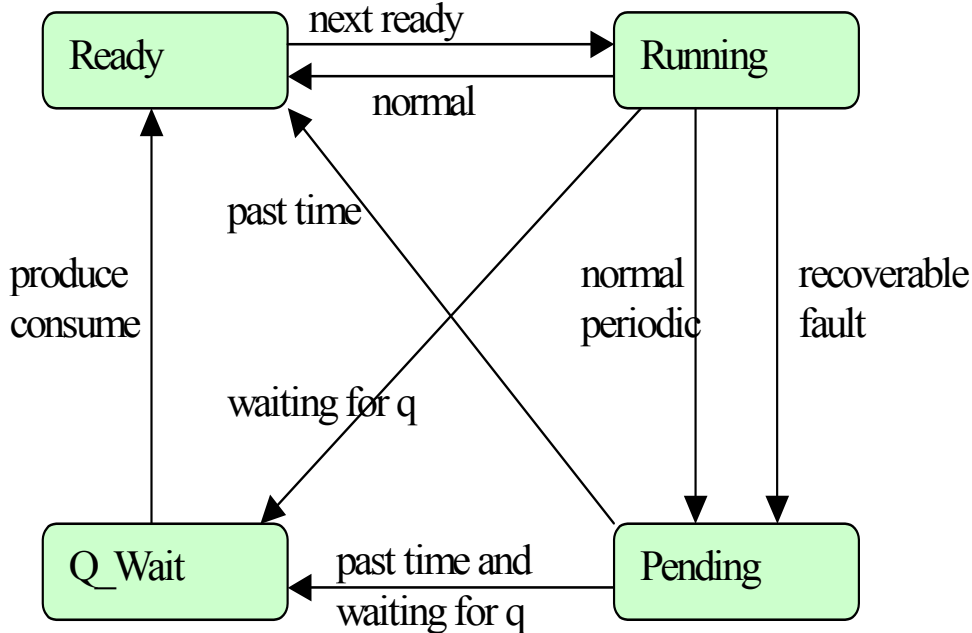
- Static Scheduler
- Dynamic Scheduler
 - Run Dynamic Scheduler
 - Static Schedule State Changes
 - Queue Management Functions
- Segmentation
 - Dynamic Queue State Changes
 - Reset and EOS Functions
- Distributed Control not yet instrumented

Static Scheduler



- 3 Timers: run-schedule, apply and run-schedule-pass
- The run-sched-pass function is the code that must be executed between Apply method calls

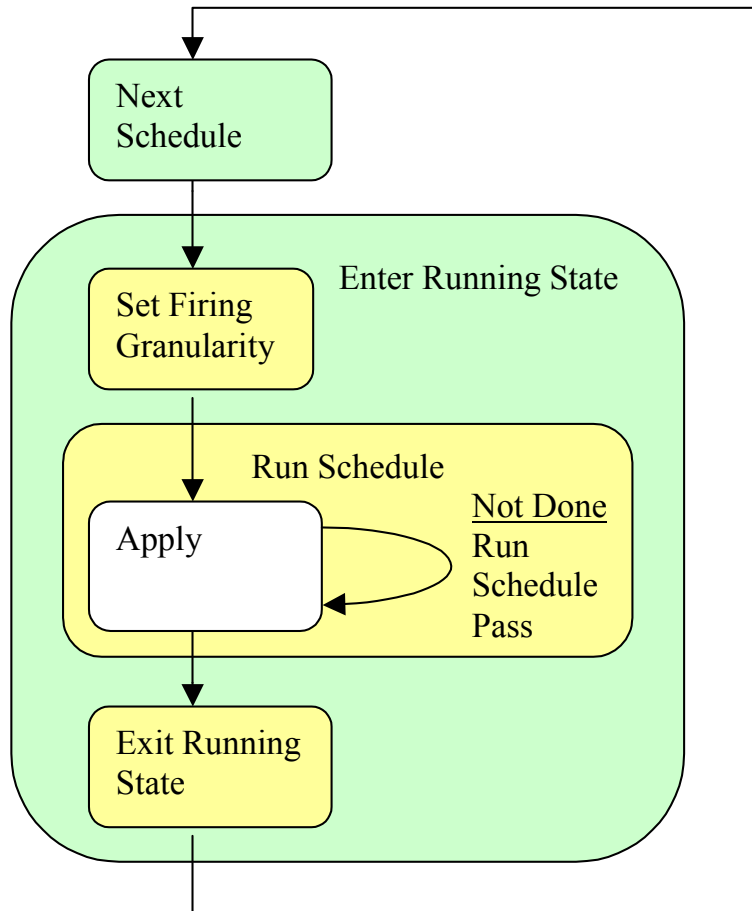
Dynamic Scheduler States



Timers

- enter-ready-state
- exit-ready-state
- enter-running-state
- exit-running-state
- enter-pending-periodic
- enter-pending-retry
- exit-pending

Dynamic Scheduler – State Transitions from Above



Timers

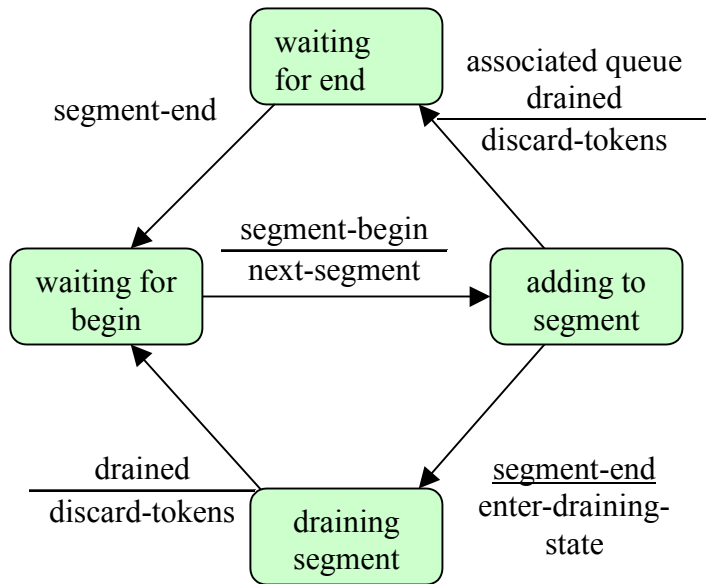
- next-schedule
 - exit-pending
 - enter-ready
 - enter-ready
 - exit-ready
- enter-running-state
 - set-firing-granularity
 - run-schedule
 - exit-running-state
 - enter-pending-periodic
 - enter-ready
 - enter-pending-retry

Produce and consume - state transitions from below

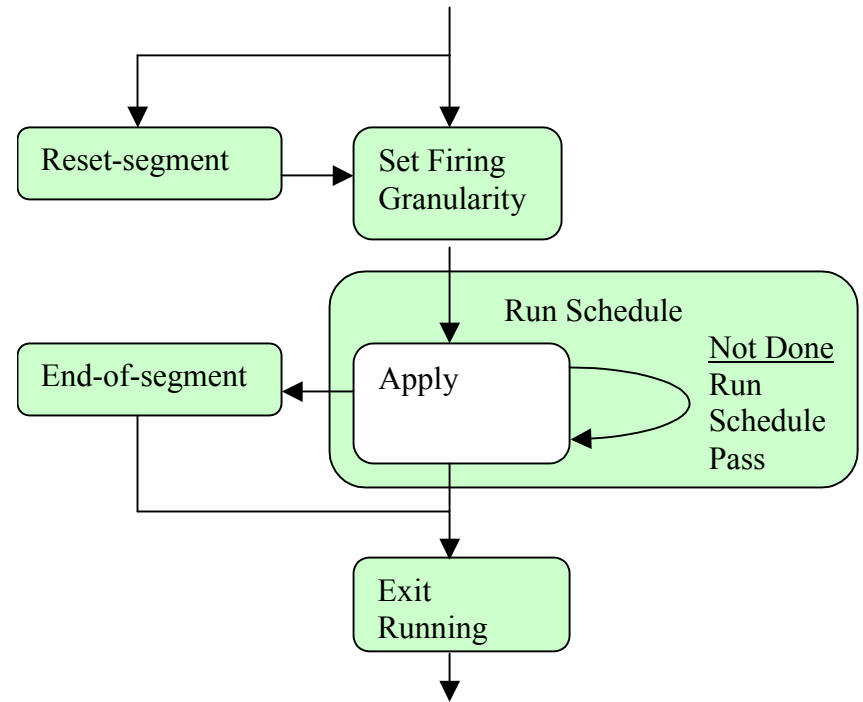
Timers

- produce
 - produce1
 - update-scheduler-on-produce
 - enter-ready-state
 - produce-copy
- write-dqs
 - ready-to-write
 - write-dq
 - write-dq-copy1
 - write-dq-copy2
 - produce1
- consume
 - update-scheduler-on-consume
 - enter-ready-state
 - read-dq
 - ready-to-read
 - read-dq-copy1
 - read-dq-copy2
 - consume

Segmentation

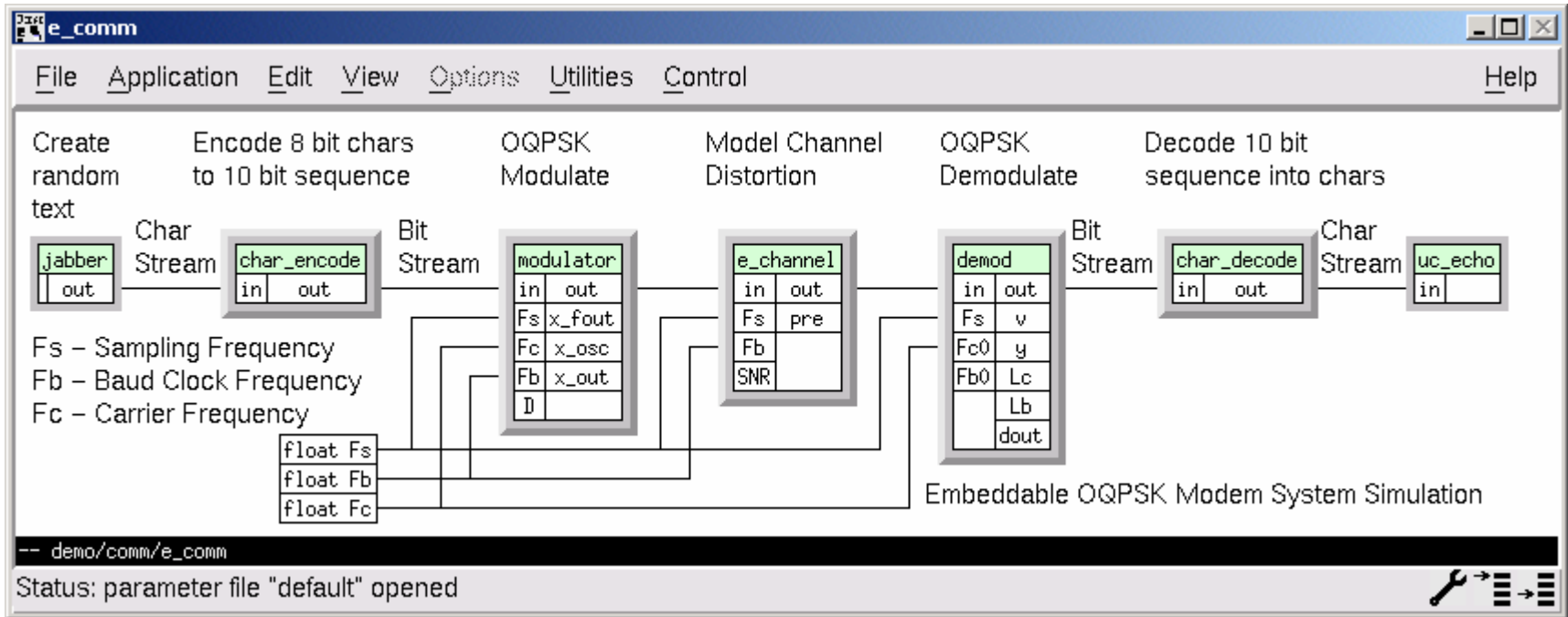


Segmentation State Diagram



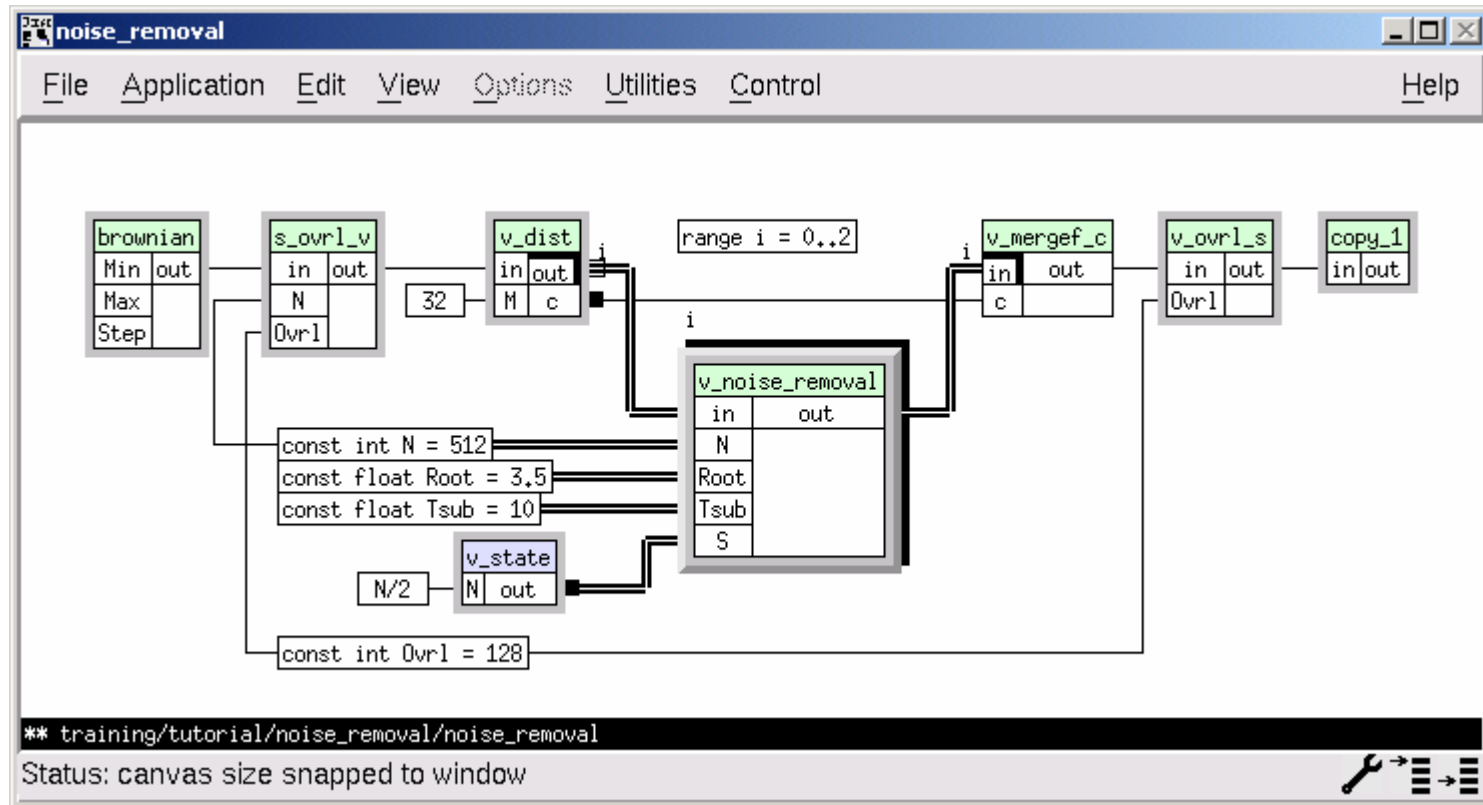
Enter-running-state function with segmentation

Benchmark Graph e_comm



Dynamic Schedule consisting of 19 primitives divided into 4 static schedules connected by 3 dynamic queues

Benchmark graph noise_removal



Dynamic scheduling with segmentation – 55 primitives, 1 external state variable, 5 static schedules, 5 dynamic queues

Results

Graph:	rt_stap_easy		e_comm		noise_removal	
	Efficiency %	Overhead (usec/fire)	Efficiency %	Overhead (usec/fire)	Efficiency %	Overhead (usec/fire)
Linux	99	1.4	72	4.7	68	9.2
Solaris	98	1.4	80	32	90	22
Nt	99	1.6	79	6.0	89	16
Altivec1	97	1.7	90	2.7	95	1.8
Altivec2	93	1.1	92	2.8	92	2.0
Altivec3	70.	18.0	40	41.0	41	34.0

- 1-3 usec overhead for Altivec1 and Altivec2.
- Altivec3 overhead times are 18 to 41 usec. What's going on?

Typical Weighted Timer Histogram for Altivec1

Timer update-on-produce: 80894

Bin	Count	Perc	
2.384e-07:	0	0.00%	
4.768e-07:	27336	20.02%	*****
9.537e-07:	52509	76.90%	*****
1.907e-06:	1046	3.06%	**
3.815e-06:	3	0.02%	*
7.629e-06:	0	0.00%	
1.526e-05:	0	0.00%	
3.052e-05:	0	0.00%	

Typical Weighted Timer Histogram for Altivec3

Timer update-on-produce: 58739

Bin	Count	Perc	
2.384e-07:	0	0.00%	
4.768e-07:	5909	0.72%	*
9.537e-07:	5734	1.39%	*
1.907e-06:	5724	2.78%	**
3.815e-06:	7337	7.14%	****
7.629e-06:	22866	44.50%	*****
1.526e-05:	11169	43.47%	*****
3.052e-05:	0	0.00%	

Results – rt_stap_easy altivec1

Timer	Called	Total	Mean	Stdv	Min	Max	Perc	PerFire
apply	7592362	3.761e+02	4.954e-05	7.821e-03	3.600e-07	2.428e+00	96.70%	4.954e-05
run-sched-pass	7592361	1.156e+01	1.523e-06	8.573e-07	8.000e-08	2.205e-04	2.97%	1.523e-06
run-schedule	7346338	1.283e+00	1.747e-07	6.751e-07	1.200e-07	1.950e-04	0.33%	1.690e-07

Run-sched-pass takes most of the overhead time

Results – e_comm altivec1

Gedae, Inc.
01000111 01000101 01000100 01000001 01000101
01001110 01001101 01000100 01000001 01000101
01001110 01001101 01000100 01000001 01000101
01000111 01000101 01000100 01000001 01000101

Timer	Called	Total	Mean	Stdv	Min	Max	Perc	PerFire
apply	667628	1.737e+01	2.601e-05	2.177e-03	1.590e-07	7.901e-01	90.50%	2.601e-05
run-sched-pass	821288	3.649e-01	4.443e-07	1.167e-06	3.900e-08	1.502e-04	1.90%	5.465e-07
read-dq-copy1	102408	3.392e-01	3.313e-06	5.552e-05	1.990e-07	1.294e-03	1.77%	5.081e-07
exit-running-state	153660	1.693e-01	1.102e-06	1.207e-06	5.990e-07	1.312e-04	0.88%	2.535e-07
next-schedule	153660	1.669e-01	1.086e-06	1.184e-06	8.800e-07	1.316e-04	0.87%	2.500e-07
enter-running-state	153660	1.285e-01	8.365e-07	1.144e-06	5.610e-07	1.320e-04	0.67%	1.925e-07
exit-ready-state	153660	1.026e-01	6.676e-07	3.410e-07	3.990e-07	1.123e-04	0.53%	1.536e-07
update-on-consume	102408	9.840e-02	9.609e-07	2.084e-07	6.390e-07	3.504e-05	0.51%	1.474e-07
update-on-produce	102589	9.652e-02	9.408e-07	1.157e-06	5.200e-07	1.173e-04	0.50%	1.446e-07
enter-ready-state	153661	8.908e-02	5.797e-07	5.875e-07	3.990e-07	1.153e-04	0.46%	1.334e-07
run-schedule	153660	4.083e-02	2.657e-07	1.420e-06	1.200e-07	1.270e-04	0.21%	6.116e-08

Results – noise_removal altivec1



Timer	Called	Total	Mean	Stdv	Min	Max	Perc	PerFire
apply	425702	1.484e+01	3.486e-05	5.015e-05	2.800e-07	2.353e-04	95.14%	3.486e-05
run-sched-pass	474822	1.393e-01	2.934e-07	5.873e-07	4.000e-08	3.880e-05	0.89%	3.272e-07
write-dq-copy2	16373	7.289e-02	4.452e-06	4.078e-07	4.080e-06	3.484e-05	0.47%	1.712e-07
read-dq-copy1	32746	6.523e-02	1.992e-06	1.769e-06	2.000e-07	3.468e-05	0.42%	1.532e-07
update-on-consume	82376	6.122e-02	7.432e-07	3.350e-07	4.400e-07	1.676e-05	0.39%	1.438e-07
update-on-produce	49632	5.646e-02	1.138e-06	4.279e-07	5.600e-07	4.243e-06	0.36%	1.326e-07
next-schedule	49120	5.559e-02	1.132e-06	3.442e-07	8.420e-07	3.209e-05	0.36%	1.306e-07
exit-running-state	49120	4.133e-02	8.415e-07	2.710e-07	4.400e-07	2.084e-05	0.26%	9.710e-08
enter-running-state	49120	3.974e-02	8.091e-07	2.565e-07	4.420e-07	3.068e-05	0.25%	9.336e-08
exit-ready-state	49120	3.105e-02	6.322e-07	2.431e-07	3.600e-07	3.112e-05	0.20%	7.295e-08
enter-ready-state	49121	2.959e-02	6.023e-07	2.089e-07	3.600e-07	1.628e-05	0.19%	6.950e-08
consume	82376	2.439e-02	2.960e-07	2.039e-07	1.200e-07	3.060e-05	0.16%	5.728e-08
produce1	49121	2.104e-02	4.284e-07	1.169e-07	2.800e-07	1.548e-05	0.13%	4.943e-08

Conclusion – Important Timers

- Run-sched-pass
- Copy functions
 - Write-dq-copy2
 - Read-dq-copy1
- Queue management
 - Update-scheduler-on-produce
 - Update-scheduler-on-consume
- Schedule state
 - Enter-running
 - Exit-running
 - Enter-ready
 - Exit-ready
 - Next-schedule

Conclusion

Function	rt_stap_easy	e_comm	noise_removal
run-sched-pass	90%	20%	18%
copy functions		18%	9%
schedule state		36%	26%
queue management		10%	15%
Total	90%	84%	68%

Four categories account for most of the overhead time

Conclusion – suggested enhancements

- Code generation of static schedule
- Eliminate copies by directly adjusting pointers
- Improve schedule state change functions by optimizing SchedHeap functions

We can comprehensively measure the overhead and systematically optimize the most costly components

Conclusion – Future work

- Determine reason for slow Altivec3 times
- Make instrumented version of library available for general use
- Add instrumentation to measure distribution overhead
- Add additional benchmark graphs
- Use values measured by instrumentation in gsim simulation