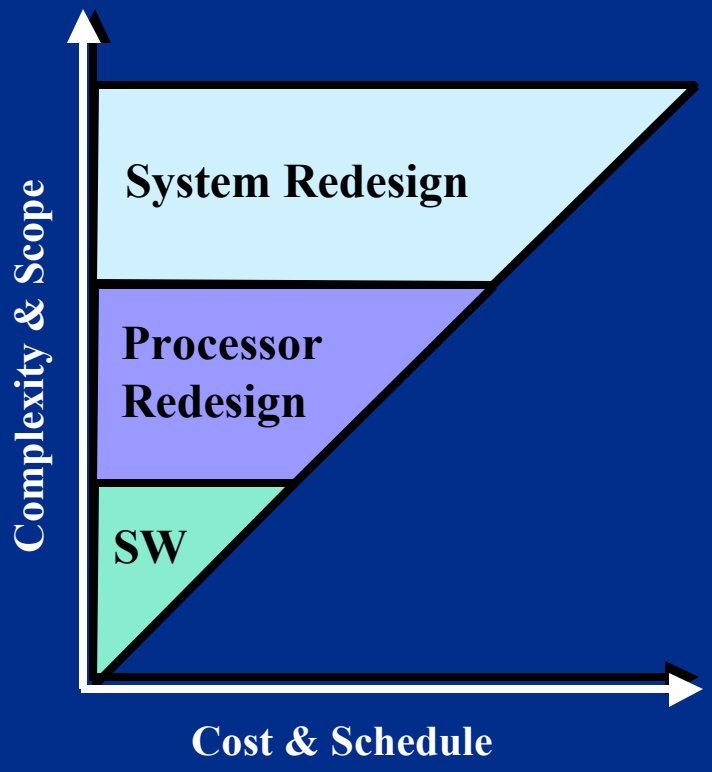
SOFTWARE CENTRIC OPTIMIZATION OF A REAL-TIME EMBEDDED SYSTEM

Dr. Max Lee
Raytheon Company
McKinney, TX
(972) 952-3499
max_w_lee@raytheon.com

Additional Author: Marshall Moluf

OPTIMIZATION

- Allow addition of incremental functionality into system already at capacity
- Need is often discovered during integration when schedule is critical

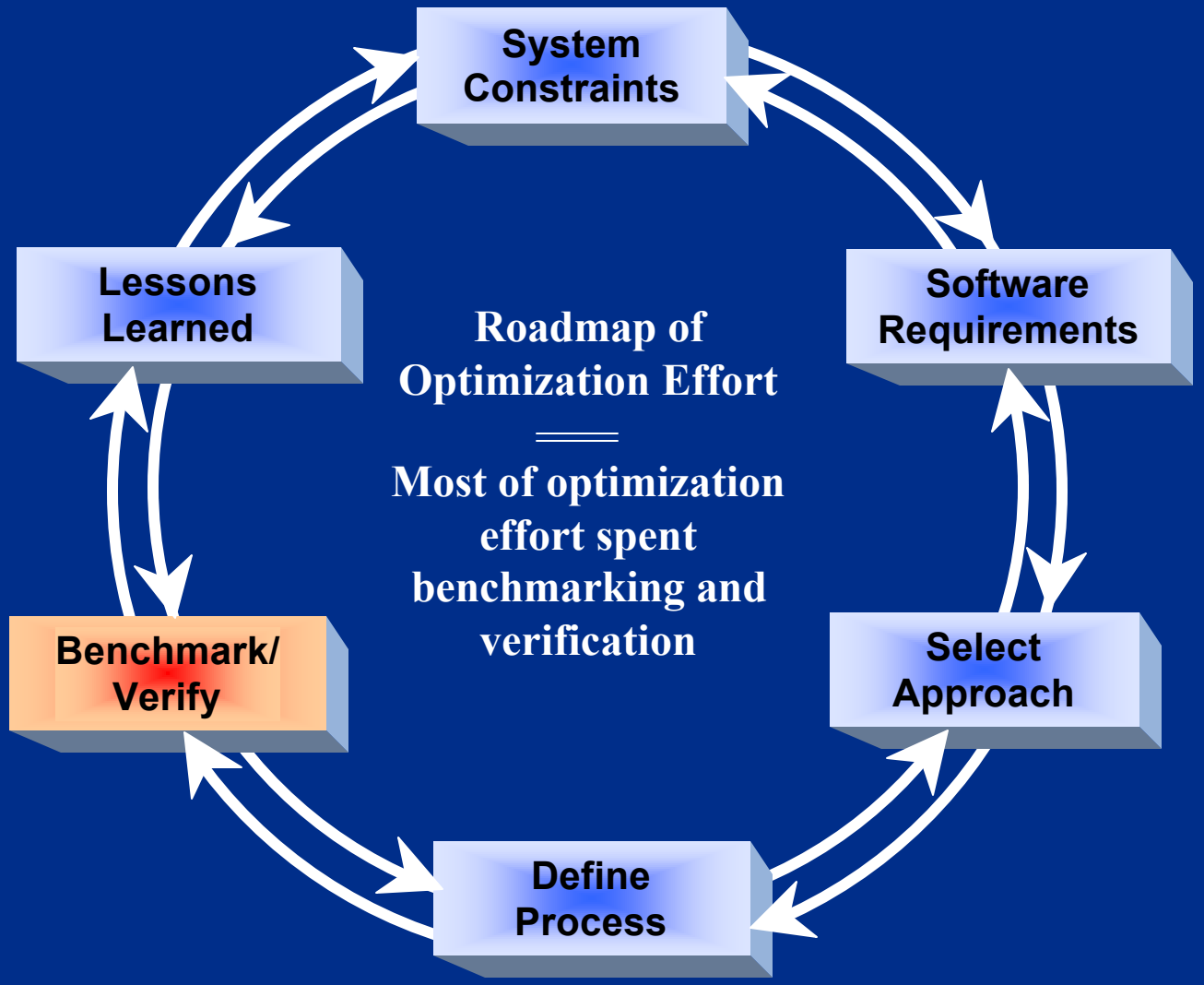


- Subsystem modified, but external interfaces unchanged
- Processor centric upgrades both computer hardware and software
- Software centric using existing processor
  - Execute faster
  - Use less memory
  - Reduce input/output blockages
  - Allow I&T to continue

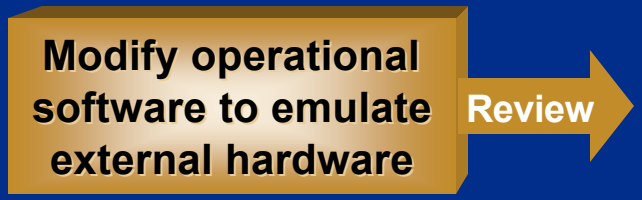
SYSTEM AND ENVIRONMENT

- Airborne system using multiple sensors and processors
- Systems being integrated and flight tests on-going
- Processor fixed -- 40 MHz, environmentally certified, VME board, RISC
- Over 35,000 lines of real-time C code had been developed and integrated
- Memory usage and input/output bandwidth were acceptable
- Execution time for full functionality projected to be greater than 100%
- *Need was to reduce software execution time in one specific processor in order to add the remaining real-time requirements*
- Software centric approach to optimization minimizes program risk
- In-process optimizations were needed ASAP for continuing flight test (few coding changes with large benefits, low risks)

SOFTWARE CENTRIC ENHANCEMENTS



ESTABLISH BASELINE



- Used NT workstations (target processors unavailable for optimization)
- Used single driver for multiple computer software units
  - Modified real-time software to replicate typical scenarios
  - Exercised representative modes/states
  - Emulated hardware interfaces and interrupts
  - Captured telemetry data
- Loaded real-time software in environment similar to operational
- Established benchmarks by executing mission scenarios
- Tools: C++ compiler, Debugger, Profiler, WinDiff, Understand for C
- Used Profiler to
  - Measure procedure hit counts
  - Measure procedure execution times

VERIFY FUNCTIONALITY

- Used NT workstations
- Minimized risk, cost, and schedule by only implementing coding changes (no design changes, no algorithm changes, same computational intervals)
- Optimized code for performance in high CPU utilization routines identified by benchmarks
- Profiled optimized code against baseline to validate
  - Hit counts were identical
  - Telemetry files were identical
- NT Profiler timing measurements were inadequate
  - Timing measurements were milliseconds and improvements were microseconds/nanoseconds
  - Timing measurements were for NT instead of RISC target processor
- Target processor test bed needed to verify performance enhancements

MEASURE ENHANCEMENTS

- Used target processor test bed (RISC and VME board)
- Reused existing software driver (without Profiler)
- Revised coding to minimize executions of software-implemented instructions
- More efficient conditionals
- Combine loops and unroll short loops
- Pointer addressing for multiple-indexed arrays
- In-line short procedures instead of passing arguments
- Global literals instead of constant parameters
- Tailor macros for specific purpose
- Assembly language for high use routines to
  - Eliminate extra compiler inserted instructions
  - Minimize stack push/pull operations

OPTIMIZATION EXAMPLES
USING ARITHMETIC VERSUS LOGICAL STATEMENTS

- **Using arithmetic statements to replace multiple logical if-statements**
  - **use**
```
if(((1<<I) & (1<<0|1<<5|1<<11|1<<14|1<<20)) != 0)
```
    or
```
if(((1<<I) & ~(1<<0|1<<5|1<<11|1<<14|1<<20)) == 0)
```
    Note: Optimizing compiler reduces constant to 0x00104821
  - **instead of** `if(I==0||I==5||I==11||I==14||I==20)` **where 0 £ I £ 31**

- **Using integer parameter to replace complex logical OR if-statements**
  - **use**
```
A = ( ((1<<20)<<C1) | ((1<<10)<<C2) | (1<<C3) ) &
    ( ((1<<20)<<4) | ((1<<10)<<6) | (1<<2) ) ;
if (A != 0)
```
    Notes: Optimizing compiler performs constant calculations. The if-statement must be used several times before this optimization becomes efficient!
  - **instead of** `if(C1==4 || C2==6 || C3==2)` **where 0 £ C1,C2,C3 £ 9**

- **Using integer parameter to replace complex logical AND if-statements**
  - **use**
```
B = ( ((1<<20)<<D1) | ((1<<10)<<D2) | (1<<D3) ) &
    ~( ((1<<20)<<7) | ((1<<10)<<3) | (1<<5) ) ;
if (B == 0)
```
    Notes: Optimizing compiler performs constant calculations. The if-statement must be used several times before this optimization becomes efficient!
  - **instead of** `if(D1==7 && D2==3 && D3==5)` **where 0 £ D1,D2,D3 £ 9**

OPTIMIZATION EXAMPLES
REDUCING SW IMPLEMENTED INSTRUCTIONS

- **Using reciprocals to eliminate divides**
  - use
```
if((Y+Z) < -0.5) ;
```
  - instead of
```
if(1.0/(Y+Z) < -2.0 ) ;
```

- **Using multiplies to replace divides**
  - use
```
X = A/(B*C*D) ;
```
  - instead of
```
X = A/B/C/D ;
```

- **Using squares to replace square roots**
  - use
```
if(Z*Z > R2) ;
```
  - instead of
```
if(Z > SQRT(R2)) ;
```
where Z > 0

- **Using pointer for double index arrays to eliminate integer multiply for addresses**
  - use
```
ptr = &A[0][0] ;
for(L=0 ; L<M*N ; L++)
*ptr++ = 0.0 ;
```
  - instead of
```
for(I=0 ; I<M ; I++)
for (J=0 : J<N ; J++)
    A[I][J] = 0.0 ;
```
where M is constant
where N is constant

OTHER OPTIMIZATION EXAMPLES

- Using pointers instead of indexing into an array of link-list pointers
  - use
```
for(ptr=&t_index[event][0] ; *ptr != NULL_TASK ; ptr++)
    t_list[*ptr].ocur_events[ix] |= bit ;
```
  instead of
```
for(J=0 ; t_index[event][J] != NULL_TASK ; J++)
    t_list[t_index[event][J]].ocur_events[ix] |= bit ;
```

- Combining sequential, dependent if-statements (especially when high utilization)
  - use
```
if(++ctr18 > 2)
{
    ctr18 = 0 ;
    if(++ctr6 > 5) ctr6 = 0 ;
}
```
  - instead of
```
ctr18 = (++ctr18>=3) ? 0 : ctr18 ;
ctr6 = ( (ctr18==0)&&(++ctr6>=6) ) ? 0 : ctr6 ;
```

- Use structure copy instead of setting individual parameters (for large structures)
- Eliminate waits by interlacing programs between HW command-response
- Move the setting of un-indexed parameters outside of for-loops
- Reduce format conversions by minimizing mixed mode

LESSONS LEARNED AND RESULTS

- Second development team can optimize code in parallel with integration team
- Identify limitations of compiler for target processor
- Modify code to minimize executions of software-implemented instructions
- Use strengths of compiler for target processor
- Target processor test bed required because NT Profiler inadequate for timing
- Had high confidence of functionality from NT hit counts and telemetry
- Had high confidence of performance from test bed timing measurements
- 8% throughput improvement, only 7 of 47 CSU modified
- Delivered several interim low-risk / large-gain optimizations in time to benefit system integration
- Completed effort over 4 months

CONTACT INFORMATION

Max W. Lee, max_w_lee@raytheon.com, 972.952.3499

Dr. Lee is a software engineering Fellow. He has been with Raytheon since 1983 and has managed signal processing programs and manufacturing organizations. Before joining Raytheon, he managed electronic warfare requirements and software development groups, and implemented/integrated software for several inertial navigation and avionics systems.

D.E.	Engineering Management, SMU
M.S.	Electrical Engineering, University of Missouri at Rolla
B.S.	Electrical Engineering, Auburn University

Marshall L. Moluf, mmoluf@raytheon.com, 972.952.4500

Mr. Moluf has been with Raytheon since 1999. He has developed, integrated, and tested software for several embedded processors and is currently developing software for airborne systems.

B.S.	Computer Engineering, Kansas State