Automatic Energy-Hotspot Detection and Elimination in Real-Time Deeply Embedded Systems

17.01.2023

Andreas Gräfensteiner

Friedrich-Alexander-Universität Erlangen-Nürnberg

Content

Motivation

Motivation

Up until now:

- Dynamic Voltage and Frequency Scaling or ultra-low-power
- While useful also quite complicated

Motivation

Up until now:

- Dynamic Voltage and Frequency Scaling or ultra-low-power
- While useful also quite complicated

Now: Instead of changing hardware => Optimizing software code

- Structure code in the most energy efficient way
- Finding Energy Hotspots and remove them

=> All just through changing the order of the commands

Energy-Hotspots

Energy-Hotspots

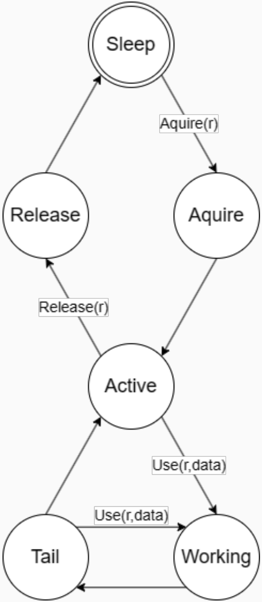
- Specific areas within deeply embedded systems
- Higher energy consumption compared to the overall energy usage patterns of the system
- Identification based on distinctive inefficiencies
- Categorization into three types:
  - Tail, Sleep and Active

Why to avoid them

- Enhanced Performance
- Extended Battery Life
- Improved Reliability
- Cost Efficiency
- Environmental Impact

Different Types of Energy Hotspot

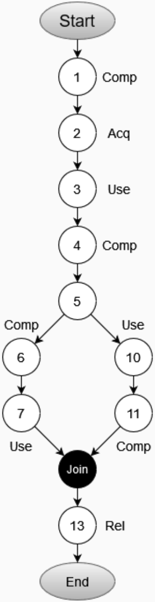
Machine r



Sleep
Aquire(r)
Release
Aquire
Release(r)
Active
Use(r,data)
Use(r,data)
Tail
Working

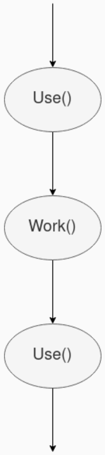
The code

```
1. int i = 0;
2. Acquire(r);
3. Use(r,i);
4. int j = Rand();
5. if(j > 1) {
6.    j = 1;
7.    Use(r,j);
8. }
9. else {
10.   Use(r,j);
11.   i = 1;
12. }
13. Release(r);
```



$\text{Hotspot}_{\text{Tail}}$

- Focus on energy inefficiency of a delay between two consecutive Use() statements after the execution of the first



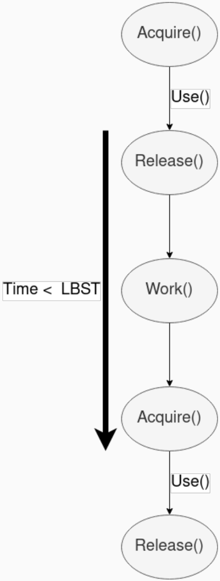
Hotspot$_{Sleep}$

- Inefficient transition between sleep and active state

=> Significant energy overhead for Acquire() and Release()

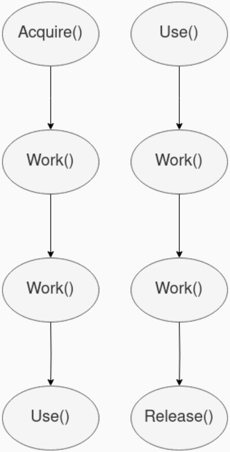
- Can be calculated using Lower Bound on Sleep Time (LBST)

$$LBST = \frac{E_{Rel} + E_{Acq}}{Pow_A - Pow_S}$$



Hotspot$_{Active}$

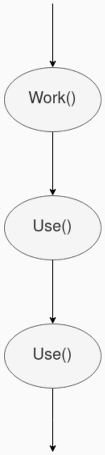
- Two different Variants
- Energy inefficiency due to interval between Aquire()/Use() and Use()/Release()
- The prolonged idleness or activity leads to energy wastage



Improvements for Energy Hotspots

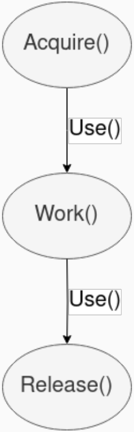
$\text{Hotspot}_{\text{Tail}}$

- use statements should be moved towards each other
- Can be prevented by time restriction and dependencies



Hotspot$_{Sleep}$

- Optimization of the transition between active and sleep states according to LBST
- Adjustment of the code sequences

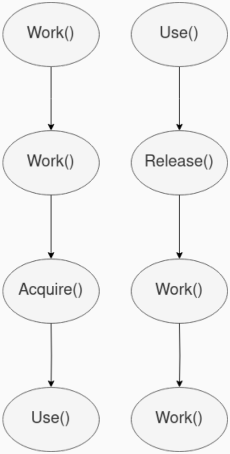


$\text{Hotspot}_{\text{Active}}$

- Bring Acquire closer to Use() and Use() closer to Release()

=> Reduces inefficient resource utilization

- Careful this can lead to a new $\text{Hotspot}_{\text{Sleep}}$



Conclusion

Conclusion

Advantages:

- Easy to implement and use for small programs
- Useable without any hardware modifications
- Reduced Energy costs
- Could be automated with further research

Disadvantages:

- Code needs to be more structured
- Increased development time
- The MCFG of the code needs to be known
- It gets quite complicated for complex programs

Conclusion

- At the moment usefull for parts or small projects
- Could be used by the industry with further research