# IM420 Sistemas Embarcados de Tempo Real

## Notas de Aula – Semana 12

**Prof. Denis Loubach**
dloubach@fem.unicamp.br

Programa de Pós-Graduação em Eng. Mecânica / Área de Mecatrônica
Faculdade de Engenharia Mecânica - FEM
Universidade Estadual de Campinas - UNICAMP


$1^o$ Semestre de 2018

# Tópicos

# *Google's Tensor Processing Unit could advance Moore's Law 7 years into the future*

*Forget the CPU, GPU, and FPGA, Google says its Tensor Processing Unit, or TPU, advances machine learning capability by a factor of three generations.*

Fonte: `http://www.pcworld.com/article/3072256/google-io/googles-tensor-processing-unit-said-to-advance-moores-law-seven-years-into-the-future.html`
18 de maio, 2016

# Visão Geral

Muitas atividades precisam ser levadas em consideração no desenvolvimento de sistemas de tempo real:

- Requisitos do sistema
- Entradas e saídas
- *Deadlines* (*hard* ou *soft*)
- Eventos e resposta a eventos
- Padrão de chegada dos eventos e frequência
- Tarefas que precisam ser concorrentes (conceito de multi-tarefa)
- Escalonabilidade do sistema
- Utilização de protocolos de para comunicação inter-tarefas

# Visão Geral (cont...)

Uma questão importante consiste no fato de como decompor o software de aplicação em **tarefas concorrentes**

Técnicas e linguagens de modelagem (*e.g.,* UML e SysML) têm sido utilizadas

No final, um sistema de tempo real é considerado escalonável se todas as tarefas do sistema cumprirem seus *deadlines*

# Recomendações para identificação de concorrência

**Abordagem *outside-in***

- Diagrama de contexto em alto nível (identificação de entradas/saídas)
- Identificação de fluxos de comunicação
- Primeira decomposição da aplicação
  - Identificação de potenciais tarefas
  - Interação com o mundo de fora (interfaces de comunicação)
- Essa primeira decomposição ainda precisa ser refinada para identificar ou mesclar outras possíveis tarefas no sistema

# Principais conceitos

## Unidades de concorrência

Uma unidade de concorrência pode ser uma **tarefa** ou um **processo**. Pode ser ainda qualquer *thread* de execução escalonável que compete por tempo de processamento na CPU

Embora uma *ISR* não seja escalonável para rodar concorrentemente com outras rotinas, ela também precisa ser considerada no projeto para concorrência

Isso porque *ISRs* seguem a política de preempção e são unidades de execução competindo por tempo de processamento na CPU

# Principais conceitos (cont...)

O objetivo primário da decomposição de processos consiste na **otimização de execuções em paralelo**

Visando maximizar a performance e tempo de resposta em aplicações de tempo real

Se este processo for concluído de forma correta, o resultado será um sistema que cumpre todos seus *deadlines* de maneira robusta

Se não, os *deadlines* estarão comprometidos e o projeto final do sistema não será aceitável

# Execução Paralela *vs.* Pseudo-paralela

Num sistema de tempo real, **tarefas concorrentes** podem ser escalonadas para executar (rodar) num ambiente considerando **uni**- ou **multi**-processador

# Execução Paralela *vs.* Pseudo-paralela (cont...)

Em **uni-processador** (*single-processor*), o sistema pode atingir uma **execução concorrente**, gerando impressão de paralelismo que de fato é um **pseudo-paralelismo**

Neste caso, a aplicação é decomposta em múltiplas tarefas visando maximizar a utilização da CPU

Note que num sistema uni-processado existe somente um *program counter* (também conhecido como ponteiro de instruções)

Assim, somente uma instrução pode ser executada por vez

Neste contexto, muitas aplicações utilizam um escalonador com capacidade de execução multi-tarefa para execução de mais de uma tarefa

Então, o termo **execução concorrente** é empregado

# Execução Paralela *vs.* Pseudo-paralela (cont...)

Em **multi-processador** (*multi-processor*), o sistema pode atingir uma **execução verdadeiramente paralela**

Por exemplo, se duas CPUs são utilizadas no sistema, duas tarefas concorrentes podem executar em paralelo, ou seja, ao mesmo tempo

Esse paralelismo é possível porque existem dois *program counters* (um para cada CPU) sendo utilizados

Isto permite que duas instruções diferentes sejam executadas ao mesmo tempo

# Execução Paralela *vs.* Pseudo-paralela (cont...)

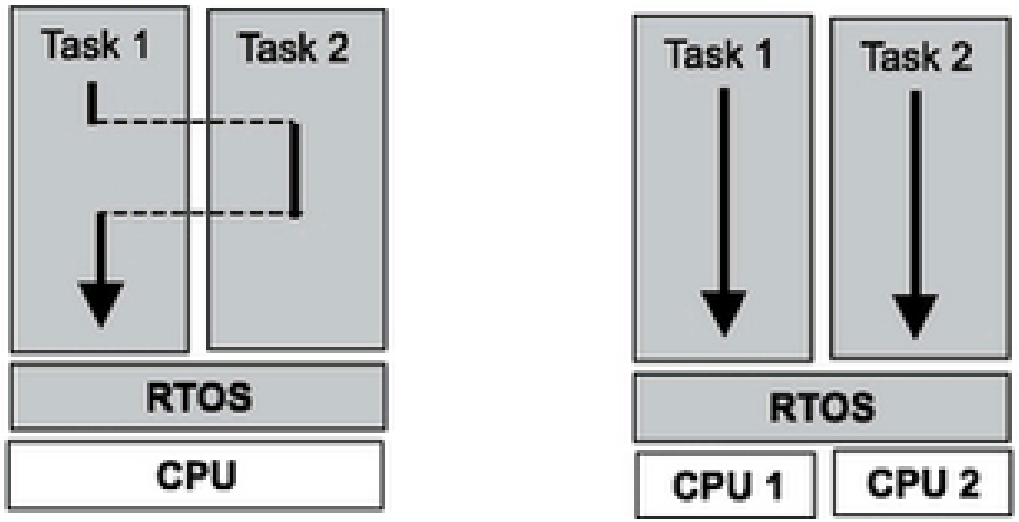


Figura: Execução pseudo-paralela *vs.* paralela

# Execução Paralela *vs.* Pseudo-paralela (cont...)

No caso da utilização de um sistema de múltiplas CPUs, o RTOS pode ser **distribuído**

Isto significa que vários componentes (ou cópias de componentes do RTOS) podem ser executados em CPUs diferentes

Neste tipo de sistema, múltiplas tarefas podem ser atribuídas para rodar em cada CPU, assim como realizado num sistema uni-processado

Mesmo que duas ou mais CPUs permitam uma execução com paralelismo verdadeiro, cada CPU (internamente) ainda estará executando segundo uma concorrência

# Como identificar tarefas críticas e urgentes

Note a diferença entre criticidade e urgência

**Tarefa crítica**

Tarefas cujas falhas são desastrosas

Neste caso, os *deadlines* podem ser curtos ou longos, mas devem sempre ser cumpridos, caso contrário o sistema não estará de acordo com os requisitos estabelecidos

**Tarefa urgente**

Tarefa cujos tempos de *deadline* são relativamente curtos

Neste caso, entretanto, cumprir ou não o *deadline* pode ou não ser crítico

# Análise de Escalonabilidade

Depois que o sistema embarcado foi decomposto em tarefas e em *ISRs*, as tarefas precisam ser escalonadas para rodar e executar as funcionalidades do sistema

Uma análise de escalonabilidade (*schedulability analysis*) determina se todas as tarefas podem ser escalonadas para rodar e cumprir seus *deadlines*

Isto deve considerar o algoritmo de escalonamento utilizado

Notar também que esta análise leva em conta somente aspectos temporais, ou seja, as funcionalidades do sistema não são verificadas

# Análise de Escalonabilidade (cont...)

Exemplo de análise de escalonabilidade considerando o *Rate Monotonic Analysis - RMA* [1]

Algoritmo de escalonamento: *Rate Monotonic Scheduling- RMS*

- contempla escalonamento preemptivo
- atribuição de prioridades fixas de acordo com o período da tarefa (função monotônica da taxa de execução da tarefa)
- ou seja, menor o período, maior a prioridade

# RMA

Uma série de restrições se aplicam para a RMA básica:

1. todas as tarefas devem ser periódicas
2. as tarefas devem ser independentes umas das outras e sem qualquer interação
3. os *deadlines* das tarefas são determinados pelo início do próximo período
4. cada tarefa possui tempo de execução constante
5. todas as tarefas possuem o mesmo nível de criticidade
6. tarefas aperiódicas são limitadas à inicialização do sistema e recuperação de falhas, além disso essas tarefas não possuem *hard deadlines*

# Teste de escalonabilidade básico para RMA

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq U(n) = n(2^{\frac{1}{n}} - 1) \quad (1)$$

onde:

$1 \leq i \leq n$
$C_i$ = *worst-case execution time* associado a tarefa periódica $i$
$T_i$ = período associado a tarefa $i$
$n$ = número de tarefas

$U(n)$ é o fator de utilização do processador

O lado direito da equação modela o limite teórico do fator de utilização do processador

Se o somatório (para um determinado conjunto de tarefas) for menor ou igual ao fator de utilização, este conjunto é **escalonável**

O valor de $U$ diminui a medida que $n$ aumenta e converge para aproximadamente 69% quando $n$ tende a infinito

# Teste de escalonabilidade básico para RMA (cont...)

Exemplo:

| Tarefas | Tempo de Execução | Período [ms] |
|---|---|---|
| Tarefa 1 | 20 | 100 |
| Tarefa 2 | 30 | 150 |
| Tarefa 3 | 50 | 300 |

Tabela: Conjunto de tarefas para RMA

De acordo com a Eq. 1, o fator de utilização do processador pode ser calculado como mostrado abaixo:

$$\frac{20}{100} + \frac{30}{150} + \frac{50}{300} = U(3) = 3(2^{\frac{1}{3}} - 1)$$

$56.66\% \leq U(3) = 77.98\%$

Conjunto de tarefas é escalonável segundo RMS

# Teste de escalonabilidade estendido para RMA

O RMA básico é considerado muito restritivo para aplicações práticas

A segunda premissa do RMA básico é considerada ainda impraticável, uma vez que as tarefas possuem inter-comunicação

Além disso, métodos de sincronização são parte de vários projetos de sistemas de tempo real

A utilização de métodos de sincronização inter-tarefas implica que algumas tarefas podem ficar bloqueadas em função da tentativa de acesso aos recursos compartilhados

# Teste de escalonabilidade estendido para RMA (cont...)

Considerando estes fatos, a RMA estendida leva em conta a possibilidade de sincronização das tarefas

$$\sum_{j=1}^{m} \left( \frac{C_j}{T_j} \right) + \frac{B_m}{T_m} \leq U(m) = m(2^{\frac{1}{m}} - 1) \tag{2}$$

onde:

$1 \leq m \leq n$
$C_j$ = *worst-case execution time* associado a tarefa periódica $j$
$T_j$ = período associado a tarefa $j$
$B_m$ = maior período de bloqueio que pode ser sofrido pela tarefa $m$
$n$ = número de tarefas

Para este caso do RMA, considerar ainda que as tarefas precisam estar ordenadas por prioridade, sendo $j = 1$ a tarefa de maior prioridade

# Teste de escalonabilidade estendido para RMA / Exemplo

Ainda considerando o conjunto de tarefas da Tabela 1 e inserindo dois recursos compartilhados, como ilustrado na figura abaixo

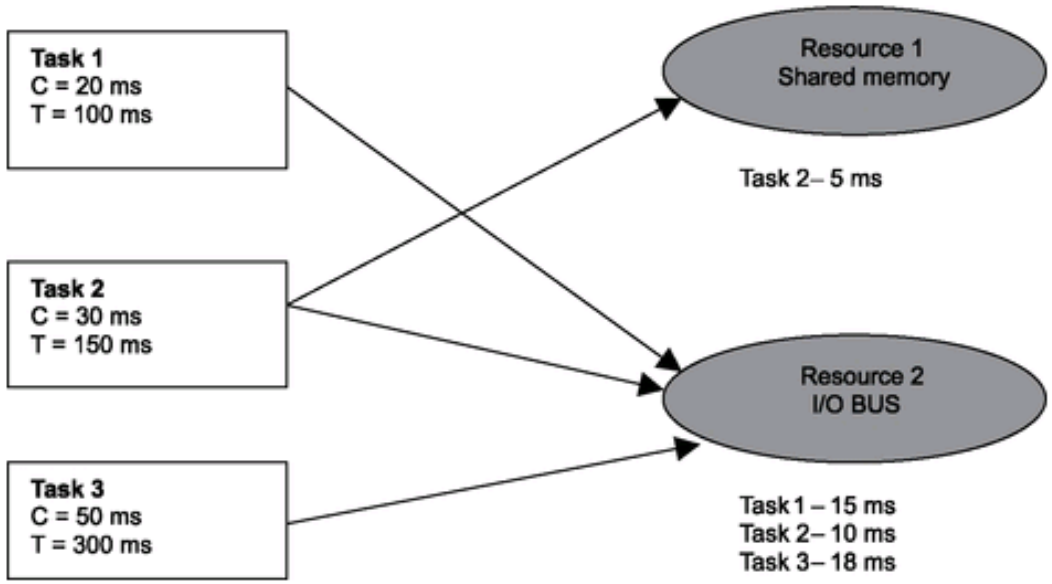


Figura: RMA estendido, recurso compartilhado

# Teste de escalonabilidade estendido para RMA / Exemplo (cont...)

Neste exemplo, os dois recursos representam:

- uma memória compartilhada (recurso #1)
- um barramento de I/O (recurso #2)

- A tarefa #1 utiliza o recurso #2 por 15ms ($BR_{2T_1} = 15$)
- A tarefa #2
  - utiliza o recurso #1 por 5ms ($BR_{1T_2} = 5$)
  - utiliza o recurso #2 por 10ms ($BR_{2T_2} = 10$)
- A tarefa #3 utiliza o recurso #2 por 18ms ($BR_{2T_3} = 18$)

$BR_{2T_1}$ Tempo de bloqueio do recurso 2 pela tarefa 1

# Teste de escalonabilidade estendido para RMA / Exemplo (cont...)

A Equação 2 se subdivide em três equações separadas que precisam ser verificadas contra o limite de utilização

# Teste de escalonabilidade estendido para RMA / Exemplo (cont...)

**Primeira equação ($m = 1$):** tanto a tarefa #2 como a #3 podem bloquear a tarefa #1 com o recurso #2, pega-se o maior valor neste caso

$$\sum_{j=1}^{m} \left(\frac{C_j}{T_j}\right) + \frac{B_m}{T_m} \leq U(m) = m(2^{\frac{1}{m}} - 1)$$
$$\sum_{j=1}^{1} \left(\frac{C_j}{T_j}\right) + \frac{B_1}{T_1} \leq U(1) = 1(2^{\frac{1}{1}} - 1)$$
$$\frac{C_1}{T_1} + \frac{B_1}{T_1} \leq U(1) = 1(2^{\frac{1}{1}} - 1)$$
$$\frac{C_1}{T_1} + \frac{\max\{BR_{2T_2}, BR_{2T_3}\}}{T_1} \leq U(1) = 1(2^{\frac{1}{1}} - 1)$$
$$\frac{20}{100} + \frac{\max\{10,18\}}{100} \leq U(1) = 1(2^{\frac{1}{1}} - 1)$$
$$\frac{20}{100} + \frac{18}{100} \leq U(1) = 1(2^{\frac{1}{1}} - 1)$$

$$38\% \leq U(1) = 100\%$$

Tarefa #1 é escalonável

# Teste de escalonabilidade estendido para RMA / Exemplo (cont...)

**Segunda equação ($m = 2$):** a tarefa #2 pode ser bloqueada pela tarefa #3 por $BR_{2T_3} = 18\text{ms}$

$$\sum_{j=1}^{m} \left(\frac{C_j}{T_j}\right) + \frac{B_m}{T_m} \leq U(m) = m(2^{\frac{1}{m}} - 1)$$
$$\sum_{j=1}^{2} \left(\frac{C_j}{T_j}\right) + \frac{B_2}{T_2} \leq U(2) = 2(2^{\frac{1}{2}} - 1)$$
$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{B_2}{T_2} \leq U(2) = 2(2^{\frac{1}{2}} - 1)$$
$$\frac{20}{100} + \frac{30}{150} + \frac{18}{150} \leq U(2) = 2(2^{\frac{1}{2}} - 1)$$

$$52\% \leq U(2) = 82.84\%$$

Tarefa #2 é escalonável

# Teste de escalonabilidade estendido para RMA / Exemplo (cont...)

**Terceira equação ($m = 3$):** note que $B_n$ é sempre igual a zero, neste exemplo $n = 3$

O fator de bloqueio da tarefa de menor prioridade é sempre igual a 0

Isso porque nenhuma outra tarefa irá bloqueá-la, ao invés esta tarefa será preemptada se necessário

$$\sum_{j=1}^{m} \left(\frac{C_j}{T_j}\right) + \frac{B_m}{T_m} \leq U(m) = m(2^{\frac{1}{m}} - 1)$$

$$\sum_{j=1}^{3} \left(\frac{C_j}{T_j}\right) + \frac{B_3}{T_3} \leq U(3) = 3(2^{\frac{1}{3}} - 1)$$

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} \leq U(3) = 3(2^{\frac{1}{3}} - 1)$$

$$\frac{20}{100} + \frac{30}{150} + \frac{50}{300} \leq U(3) = 3(2^{\frac{1}{3}} - 1)$$

$56.67\% \leq U(3) = 77.98\%$

Tarefa #3 é escalonável

# Informação ao leitor

Notas de aula baseadas nos seguintes textos:


C. L. Liu and J. W. Layland, “Scheduling algorithms for multiprogramming in a hard-real-time environment,” *J. ACM*, vol. 20, pp. 46–61, Jan. 1973.


Q. Li and C. Yao, *Real-Time Concepts for Embedded Systems*.
CMP books, 2003.