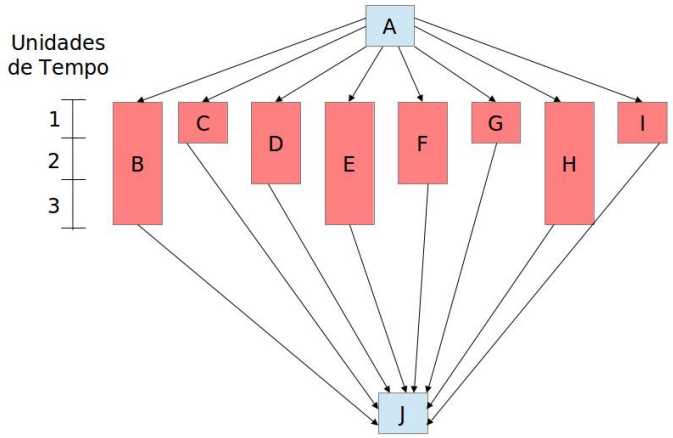
# Lista de Revisão I

## Programação Concorrente e Distribuída

Rodrigo Caetano Rocha

2 de setembro de 2015

1. Marque verdadeiro ou falso para as afirmativas abaixo e justifique a sua resposta.
   (a) De acordo com a taxonomia de Flynn, uma GPU é uma arquitetura MIMD.
   (b) Em uma arquitetura UMA, cada processador gasta o mesmo tempo para acessar a memória RAM.
   (c) Uma CPU multicore é uma arquitetura MIMD com memória distribuída.
   (d) A programação de máquinas de memória compartilhada é geralmente realizada através de bibliotecas como OpenMP e TBB.
   (e) A escalabilidade de arquiteturas many-cores é limitada pelo acesso compartilhada à memória.
   (f) As máquinas de arquitetura MISD são implementadas em diversos tipos de processadores da atualidade.
   (g) GPUs são aceleradores gráficos que compartilham a memória principal com a CPU multicores.
2. Dada a aplicação abaixo:
   (a) Determinar o work, span, speedup e tempo de execução para 4 processadores (P=4).
   (b) Escalonar as tarefas de maneira paralela em um quad-core.
   (c) Como melhorar o desempenho da aplicação? Re-escalone o código se necessário.
   (d) Qual o número ideal de processadores?
   (e) Qual o speedup considerando a Lei de Amdahl?
3. Dado o código abaixo que implementa o cálculo fatorial, acrescente os #pragmas necessários para paralelizar o código utilizando OpenMP. Explique a sua solução.



```
int factorial(int number){
   int fac = 1;
   for(int n=2; n<=number; ++n){
      fac *= n;
   }
   return fac;
}
```

4. Dado o código abaixo que implementa uma convolução de imagens 2D coloridas (ex.: filtragem de imagens para detecção de bordas), identifique as dependências de dados e aponte quais operações podem ser executadas em paralelo (ex.: Quais loops podem ser executados de forma independente? Quais operações dentro do loop mais interno podem ser executadas em paralelo? O código pode ser modificado para aumentar o paralelismo da aplicação?)

```
for(int x = 0; x < w; x++){
   for(int y = 0; y < h; y++){
      double red = 0.0, green = 0.0, blue = 0.0;
      // multiplica cada valor do filtro com o pixel correspondente
      for(int filterX = 0; filterX < filterWidth; filterX++){
         for(int filterY = 0; filterY < filterHeight; filterY++){
            int imageX = (x - filterWidth / 2 + filterX + w) % w;
            int imageY = (y - filterHeight / 2 + filterY + h) % h;
            red += image[imageX][imageY].r * filter[filterX][filterY];
            green += image[imageX][imageY].g * filter[filterX][filterY];
            blue += image[imageX][imageY].b * filter[filterX][filterY];
         }
      }

      // arredonda valores maiores que 0 e 255
      result[x][y].r = min(max(int(factor * red + bias), 0), 255);
      result[x][y].g = min(max(int(factor * green + bias), 0), 255);
      result[x][y].b = min(max(int(factor * blue + bias), 0), 255);
   }
}
```

5. Dado o código abaixo que implementa um algoritmo para remover inteiros duplicados (substituindo por -1), acrescente os #pragmas necessários para paralelizar o código utilizando OpenMP. Explique sua solução.

```
void removerDuplicados(int *v, int n){
   int i, j;
   for(i=0; i < n; i++){
      for(j=i+1; j < n; j++){
         if(v[i] == v[j]){
            v[j] = -1;
         }
      }
   }
}
```

6. Dado o código abaixo que implementa o algoritmo de ordenação QuickSort, acrescente os #pragmas necessários para paralelizar o código utilizando OpenMP. Explique sua solução.

```
void quickSort (int numList[],  int nLower, int nUpper){
   if (nLower < nUpper){
      // cria çõparties
```

```
4       int nSplit = partition (numList, nLower, nUpper);
5       // executa o quicksort recusivamente em cada metade dos dados
6       quickSort (numList, nLower, nSplit - 1);
7       quickSort (numList, nSplit + 1, nUpper);
8    }
9 }
```

7. Dado o código abaixo que implementa o algoritmo de ordenação BubbleSort, identifique as dependências de dados e aponte quais operações podem ser executadas em paralelo. (ex: Quais loops podem ser executados de forma independente? Quais operações dentro do loop mais interno podem ser executadas em paralelo? O código pode ser modificado para aumentar o paralelismo da aplicação? Qual a estratégia de paralelismo que poderia ser utilizada?)

```
1 void BubbleSort(int V[], int N){
2    int i,j;
3    for(i = 0; i < N; i++)
4       for(j = 0; j < N-1; j++)
5          if(v[j] > v[j+1])
6             swap(v[j],v[j+1];
7 }
```

8. Dado o código abaixo que implementa o algoritmo para encontrar o segundo maior valor em um vetor com valores não repetidos, identifique as dependências de dados e aponte quais operações podem ser executadas em paralelo. (ex: Quais operações dentro do loop podem ser executadas em paralelo? O código pode ser modificado para aumentar o paralelismo da aplicação? Qual a estratégia de paralelismo que poderia ser utilizada? Como o algoritmo pode ser melhorado?)

```
1  int segundoMaior(int *v, int n){
2     int i, maior, segundo;
3     maior = segundo = v[0];
4     for(i=1; i < n; i++){
5        if(v[i] > maior){
6           segundo = maior;
7           maior = v[i];
8        }
9     }
10    return(segundo);
11 }
```

9. Apresente as principais diferenças (vantagens e desvantagens) de paralelismo multicore (memória compartilhada) e cluster de computadores (memória distribuída).

10. Descreva brevemente os padrões paralelos fork-join, map, reduction, scan e stencil. Compare os padrões reduction e scan. Como é possível ter ganho de desempenho com o padrão scan em uma máquina paralela?