

Bacharelado em Ciência da Computação - DCC/IM-UFRJ
Programação Paralela e Distribuída
Prof. Gabriel P. Silva
3º Trabalho – 18/10/2019

1) Verifique se os seguintes laços são vetorizáveis ou não. Caso positivo indique quais os comandos devem ser utilizados para sua execução de forma correta e também melhor desempenho considerando o conjunto de instruções AVX-512. **(1,0 ponto)**

a)

```
for ( k=0 ; k<n ; k++ ) {  
    x[k] = q + y[k]*( r*z[k+10] + t*z[k+11] );  
}
```

b)

```
ii = n;  
ipntp = 0;  
do {  
    ipnt = ipntp;  
    ipntp += ii;  
    ii /= 2;  
    i = ipntp - 1;  
  
    for ( k=ipnt+1 ; k<ipntp ; k=k+2 ) {  
        i++;  
        x[i] = x[k] - v[k ]*x[k-1] - v[k+1]*x[k+1];  
    }  
} while ( ii>0 );
```

c)

```
for ( i=1 ; i<n ; i++ ) {  
    x[i] = z[i]*( y[i] - x[i-1] );  
}
```

d)

```
for ( i=1 ; i<n ; i++ ) {  
    for ( k=0 ; k<i ; k++ ) {  
        w[i] += b[k][i] * w[(i-k)-1];  
    }  
}
```

e)

```
for ( k=0 ; k<n ; k++ ) {  
    x[k] = u[k] + r*( z[k] + r*y[k] ) +  
        t*( u[k+3] + r*( u[k+2] + r*u[k+1] ) +  
        t*( u[k+6] + r*( u[k+5] + r*u[k+4] ) ) );  
}
```

f)

```
for ( k=0 ; k<n ; k++ ) {  
    vx[k] = 0.0;  
    xx[k] = 0.0;  
    ix[k] = (long) grd[k];  
    xi[k] = (double) ix[k];  
    ex1[k] = ex[ ix[k] - 1 ];  
    dex1[k] = dex[ ix[k] - 1 ];  
}
```

```
for ( k=0 ; k<n ; k++ ) {  
    vx[k] = vx[k] + ex1[k] + ( xx[k] - xi[k] )*dex1[k];  
    xx[k] = xx[k] + vx[k] + flx;  
    ir[k] = xx[k];  
    rx[k] = xx[k] - ir[k];  
    ir[k] = ( ir[k] & 2048-1 ) + 1;  
    xx[k] = rx[k] + ir[k];  
}
```

```
}
```

```
for ( k=0 ; k<n ; k++ ) {  
    rh[ ir[k]-1 ] += 1.0 - rx[k];  
    rh[ ir[k] ] += rx[k];  
}
```

```
g) t = 0.0037;  
s = 0.0041;  
kn = 6;  
jn = n;
```

```
for ( k=1 ; k<kn ; k++ ) {  
    for ( j=1 ; j<jn ; j++ ) {  
        za[k][j] = ( zp[k+1][j-1] +zq[k+1][j-1] -zp[k][j-1] -zq[k][j-1] ) *  
                    ( zr[k][j] +zr[k][j-1] ) / ( zm[k][j-1] +zm[k+1][j-1]);  
        zb[k][j] = ( zp[k][j-1] +zq[k][j-1] -zp[k][j] -zq[k][j] ) *  
                    ( zr[k][j] +zr[k-1][j] ) / ( zm[k][j] +zm[k][j-1]);  
    }  
}
```

```
for ( k=1 ; k<kn ; k++ ) {  
    for ( j=1 ; j<jn ; j++ ) {  
        zu[k][j] += s*( za[k][j] *( zz[k][j] - zz[k][j+1] ) -  
                        za[k][j-1] *( zz[k][j] - zz[k][j-1] ) -  
                        zb[k][j] *( zz[k][j] - zz[k-1][j] ) +  
                        zb[k+1][j] *( zz[k][j] - zz[k+1][j] ) );  
        zv[k][j] += s*( za[k][j] *( zr[k][j] - zr[k][j+1] ) -  
                        za[k][j-1] *( zr[k][j] - zr[k][j-1] ) -  
                        zb[k][j] *( zr[k][j] - zr[k-1][j] ) +  
                        zb[k+1][j] *( zr[k][j] - zr[k+1][j] ) );  
    }  
}
```

```
for ( k=1 ; k<kn ; k++ ) {  
    for ( j=1 ; j<jn ; j++ ) {  
        zr[k][j] = zr[k][j] + t*zu[k][j];  
        zz[k][j] = zz[k][j] + t*zv[k][j];  
    }  
}
```

```
h) k = 1;  
while (k <= maxit && error > tol) {  
    error = 0.0;  
    /* copia a nova soluo sobre a antiga */  
    for (j=0; j<m; j++)  
        for (i=0; i<n; i++)  
            uold[i + m*j] = u[i + m*j];  
  
    /* computa stencil, erro residual e atualiza */  
    for (j=1; j<m-1; j++)  
        for (i=1; i<n-1; i++){  
            resid =(  
                ax * (uold[i-1 + m*j] + uold[i+1 + m*j])  
                + ay * (uold[i + m*(j-1)] + uold[i + m*(j+1)])  
                + b * uold[i + m*j] - f[i + m*j]  
            ) / b;
```

```

/* Solução atualizada */
u[i + m*j] = uold[i + m*j] - omega * resid;

/* Erro residual acumulado */
error = error + resid*resid;
}
/* Verificação do erro */
k++;
error = sqrt(error) /(n*m);

} /* while */

```

2) O segundo trabalho consiste em paralelizar o algoritmo de ordenação chamado RankSort ou Enumeration Sort: **(3,0 pontos)**

- Considere n números **todos** distintos.
 - Para cada número determinamos o seu rank, isto é, a quantidade de números que são menores que ele. Esse rank então determina a posição do número na sequência ordenada.
 - Um algoritmo seqüencial implementando esse método pode ser assim:
 - i. Suponha n números num vetor $a[0]; a[1]; \dots; a[n - 1]$.
 - ii. Primeiro $a[0]$ é comparado com todos para determinar a quantidade de números menores que ele.
 - iii. Digamos que esta quantidade seja x , então $a[0]$ é armazenado no vetor $b[x]$. Fazemos então o mesmo com $a[1]$ e depois sucessivamente com todos os outros.
 - iv. O vetor b assim obtido será a resposta.
 - v. O algoritmo sequencial é $O(n^2)$.
 - Elabore um programa paralelo em OpenMP usando esse método. Justifique o aumento de desempenho esperado com esta versão executando com múltiplos processadores.
 - Execute o código com 1, 2, 4, 8 e 16 processadores e avalie o speed-up obtido.
 - Apresente um relatório com código fonte, resultados e comentários sobre todo esse processo.
- 3) Elabore um programa utilizando diretivas do OpenMP que calcule o total de números primos entre 2 e N usando o método do crivo de Eratóstenes, ou seja, marcando em um vetor todos os múltiplos de 2, todos os múltiplos de 3, todos os múltiplos de 5, etc. Utilize uma versão **recursiva** e considere o uso de **tasks**. Execute o código com 1, 2, 4, 8 e 16 processadores e um valor de N que resulte em um tempo de execução sequencial da ordem de 300 segundos para um processador. Avalie o tempo de execução, speed-up e eficiência obtidos. Elabore um relatório com o código fonte, resultados e comentários. **(3,0 pontos)**
- 4) Implemente uma versão paralela em OpenMP da simulação 2-D de transferência de calor em uma placa de metal $N \times N$, utilizando o método Gauss-Seidel SOR (*successive over-relaxation*). Este método funciona realizando M iterações, nas quais é calculada a temperatura de cada ponto de uma placa de metal como a média de cada um dos seus vizinhos. A seguir, tem-se a versão sequencial do método. **(3,0 pontos)**

```

#define TAM N+2

double temp[TAM][TAM], mat[TAM][TAM];

void sor(void)
{
    int i,j,k;
    for (k = 0; k < M; k++) {
        for (i = 1; i < TAM - 1; i++) {
            for(j = 1; j < TAM - 1; j++)

```

```

        temp[i][j] = 0.25*(mat[i-1][j] + mat[i+1][j] +
        mat[i][j-1] + mat[i][j+1]);
    }
    memcpy (mat, temp, sizeof (mat));
}
}

```

Uma primeira abordagem para a paralelização do método seria fazer com que cada iteração dividisse a placa de metal em faixas, fizesse com que cada thread calculasse a temperatura de uma parte e colocasse o resultado numa matriz auxiliar. Depois cada thread copiaria sua parte para a matriz inicial. Desta forma, o uso de barreiras é muito intenso. **O seu objetivo é implementar o método utilizando apenas a barreira para sincronizar as threads no final da execução.**

Dados para o problema:

- Número de iterações: 1024
- Tamanho da placa de metal: 1022
- A temperatura inicial da placa é 20°C
- A temperatura da fonte de calor localizada no ponto 800x800 da placa é 100°C
- As bordas estão sempre com a temperatura igual a 20°C
- Organize as matrizes de forma a otimizar o acesso à cache e à memória. Considere a possibilidade do uso de tasks e instruções vetoriais.
- Execute o código com 1, 2, 4, 8 e 16 processadores e avalie o tempo, speed-up e eficiência obtidos.
- Apresente um relatório com código fonte, resultados e comentários sobre todo esse processo.