OpenMP

Eng. Software / PSPD

Prof. Fernando W Cruz

Agenda

1. Algumas aplicações são mais difíceis de paralelizar por conta da dependência de resultados/valores gerados por cada thread.
   - Exemplo: Série de Fibonnacci... qual a melhor forma de paralelizar?
2. Paralelizando soma de vetores
3. Paralelizando totalização de valores em vetores

1. Série de Fibonnacci

```c
#include <stdio.h>

int fib(int n) {
    if (n<3)/* supondo n>0 */
        return 1;
    else
        return (fib(n-1) + fib(n-2));
} /* fim-fib */

int main(void) {
    int n;
    printf("Valor de n : ");
    scanf("%d", &n);
    #pragma omp parallel
    {
        printf("%d\n", fib(n));
    }
    return 0;
} /* fim-main */
```

Funciona???

```c
#include <stdio.h>

int fib(int n) {
    if (n<3)/* supondo n>0 */
        return 1;
    else
        return (fib(n-1) + fib(n-2));
} /* fim-fib */

int main(void) {
    int n;
    printf("Valor de n : ");
    scanf("%d", &n);
    #pragma omp parallel
    {
        printf("%d\n", fib(n));
    }
    return 0;
} /* fim-main */
```

```c
#include <stdio.h>
#include <omp.h>

long fib(int n) {
    return(n < 3? 1 : fib(n-1) + fib(n-2));

} /* fim-fib */

int main(void) {
    int n=10;
    #pragma omp parallel
    {
        int t=omp_get_thread_num();
        printf("%d: %ld\n", t, fib(n-t));
    }
    return 0;
} /* fim-main */
```

2. Soma de vetores

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define TAM 5

int main(void) {
    int a[TAM],b[TAM],c[TAM];
    int i;
    for (int i=0; i<TAM; i++) {
        a[i]=rand()%11;
        b[i]=rand()%5;
    }
    for (int i=0; i<TAM; i++) {
        c[i] = a[i] + b[i];
        printf("c[%d]=%d\n", i, c[i]);
    } /* fim-for */
    return 0;
} /* fim-main */
```

Soma de vetores… funciona?

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define TAM 5

int main(void) {
    int a[TAM],b[TAM],c[TAM];
    int i;
    for (int i=0; i<TAM; i++) {
        a[i]=rand()%11;
        b[i]=rand()%5;
    }
    for (int i=0; i<TAM; i++) {
        c[i] = a[i] + b[i];
        printf("c[%d]=%d\n", i, c[i]);
    } /* fim-for */
    return 0;
} /* fim-main */
```

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define TAM 5

int main(void) {
    int a[TAM],b[TAM],c[TAM];
    int i;
    for (int i=0; i<TAM; i++) {
        a[i]=rand()%11;
        b[i]=rand()%5;
    }
    #pragma omp parallel
    {
        for (int i=0; i<TAM; i++) {
            c[i] = a[i] + b[i];
            printf("c[%d]=%d\n", i, c[i]);
        } /* fim-for */
    }
    return 0;
} /* fim-main */
```

Soma de vetores - ajustando limites

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define TAM 5

int main(void) {
    int a[TAM],b[TAM],c[TAM];
    int i;
    for (int i=0; i<TAM; i++) {
        a[i]=rand()%11;
        b[i]=rand()%5;
    }
    for (int i=0; i<TAM; i++) {
        c[i] = a[i] + b[i];
        printf("c[%d]=%d\n", i,
c[i]);
    } /* fim-for */
    return 0;
} /* fim-main */
```

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <omp.h>
#define TAM 5

int main(void) {
    int a[TAM],b[TAM],c[TAM];
    int i;
    for (int i=0; i<TAM; i++) {
        a[i]=rand()%11;
        b[i]=rand()%5; }
    #pragma omp parallel
    {
        int id, i, nthreads, istart, iend;
        id = omp_get_thread_num();
        nthreads = omp_get_num_threads();
        istart   = id*TAM / nthreads;
        iend     = (id+1) * TAM / nthreads;
        if (id == nthreads - 1)
            iend = TAM;
        for (int i=istart; i<iend; i++) {
            c[i] = a[i] + b[i];
            printf("c[%d]=%d\n", i, c[i]);
        } /* fim-for */
    }
    return 0;
} /* fim-main */
```

Soma de vetores com omp parallel for

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <omp.h>

#define TAM 5

int main(void) {
    int a[TAM],b[TAM],c[TAM];
    int i;
    for (i=0; i<TAM; i++) {
        a[i]=rand()%11;
        b[i]=rand()%5;
    }
    #pragma omp parallel for
      for (i=0; i<TAM; i++) {
        c[i] = a[i] + b[i];
        printf(“c[%d]=%d\n”, i, c[i]);
      } /* fim-for */

    return 0;
} /* fim-main */
```

3. Acumulando valores - tratar RC

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <omp.h>
#define TAM 1000

int main(void) {
    int a[TAM],b[TAM],c[TAM];
    int i;
    int total=0;

    for (int i=0; i<TAM; i++) {
        a[i]=rand()%11;
        b[i]=rand()%5;
    } /* fim-for */

    for (int i=0; i<TAM; i++) {
          c[i] = a[i] + b[i];
          total +=c[i];
        } /* fim-for */
    printf("Total = %d\n", total);
    return 0;
} /* fim-main */
```

Funciona???

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <omp.h>
#define TAM 1000

int main(void) {
    int a[TAM],b[TAM],c[TAM];
    int i;
    int total=0;

    for (int i=0; i<TAM; i++) {
        a[i]=rand()%11;
        b[i]=rand()%5;
    } /* fim-for */

    for (int i=0; i<TAM; i++) {
          c[i] = a[i] + b[i];
          total +=c[i];
        } /* fim-for */
    printf("Total = %d\n", total);
    return 0;
} /* fim-main */
```

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <omp.h>
#define TAM 1000

int main(void) {
    int a[TAM],b[TAM],c[TAM];
    int i;
    int total=0;

    for (int i=0; i<TAM; i++) {
        a[i]=rand()%11;
        b[i]=rand()%5;
    } /* fim-for */
    #pragma omp parallel for
    for (int i=0; i<TAM; i++) {
          c[i] = a[i] + b[i];
          total +=c[i];
        } /* fim-for */
    printf("Total = %d\n", total);
    return 0;
} /* fim-main */
```

Funciona com eficiência?

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <omp.h>

#define TAM 500000

int main(void) {
    int a[TAM],b[TAM],c[TAM];
    int i;
    int total = 0;
    int soma_local;

    for (int i=0; i<TAM; i++) {
        a[i]=0; //rand()%11;
        b[i]=1; //rand()%5;
    }
    #pragma omp parallel private(soma_local, i)
    {
        int soma_local=0;
        #pragma omp for schedule(dynamic, 1)
        for (int i=0; i<TAM; i++) {
            c[i] = a[i] + b[i];
            soma_local +=c[i];
        } /* fim-for */
        #pragma omp atomic
            total +=soma_local;
    } /* fim pragma parallel */
    printf("Total = %d\n", total);
    return 0;
} /* fim-main */
```

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <omp.h>

#define TAM 500000

int main(void) {
    int a[TAM],b[TAM],c[TAM];
    int i;
    int total=0, sum_local;

    for (int i=0; i<TAM; i++) {
        a[i]=1; //rand()%11;
        b[i]=0; //rand()%5;
    } /* fim-for */
    #pragma omp parallel private (i, sum_local)
    {
        int sum_local = 0;
        #pragma omp for
        for (int i=0; i<TAM; i++) {
            c[i] = a[i] + b[i];
            sum_local +=c[i];
        } /* fim-for */
        #pragma omp critical
        {
            total += sum_local;
        }
    } /* fim-parallel */
    printf("Total = %d\n", total);
    return 0;
} /* fim-main */
```

#pragma parallel com reduction(+:var)

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <omp.h>

#define TAM 500000

int main(void) {
    int a[TAM],b[TAM],c[TAM];
    int i;
    int total=0;

    for (int i=0; i<TAM; i++) {
        a[i]=1; //rand()%11;
        b[i]=0; //rand()%5;
    } /* fim-for */
    #pragma omp parallel for private (i) reduction(+:total)
        for (int i=0; i<TAM; i++) {
            c[i] = a[i] + b[i];
            total +=c[i];
        } /* fim-for */
    printf("Total = %d\n", total);
    return 0;
} /* fim-main */
```

OpenMP

Eng. Software / PSPD

Prof. Fernando W Cruz