Tarefa 1:

Rodando a primeira vez:

default:

```
**********
        **********
                **********
                        **********
```

Tempo de execução: 0.000165766 segundos

schedule(static):

```
**********
        **********
                **********
                        **********
```

Tempo de execução: 3.21679e-06 segundos

schedule(dynamic):

```
*  * * ** **** *** **  *** **  *** *
 * *  *  *  *    *  * * *    *
  * * *  *   * * * *     * ****
* * * ** * * *  * * * ** * * *  *
```

Tempo de execução: 6.4848e-06 segundos

schedule(guided):

```
        **********          *******  * *
**********
             *********         **  *
                  **********    ** *
```

Tempo de execução: 9.20333e-06 segundos

Segunda vez:

default:

```
**********

        **********

                **********

                        **********
```

Tempo de execução: 0.000168292 segundos

schedule(static):

```
**********

        **********

                **********

                        **********
```

Tempo de execução: 2.71946e-06 segundos

schedule(dynamic):

```
* * * * *** * *  **** **** *  ** * ***
 * * *   * *    *   * *  *   *
  * * * *  * *  * *  * * * *
* * * *  * ****  *   ***** * *
```

Tempo de execução: 5.73322e-06 segundos

schedule(guided):

```
            *********   *****   ** ***
                *******  ****
        **********             **
**********                 ***   *
```

Tempo de execução: 4.06615e-06 segundos

Terceira vez:

default:

****

        ****

                ****

                        ****

Tempo de execução: 0.0001639 segundos

schedule(static):

****

        ****

                ****

                        ****

Tempo de execução: 2.91504e-06 segundos

schedule(dynamic):

* * * * ** ** *** ** *** * ** * * ** *

 * * *   * *   * * * * *   *

  * * * * *   * * * *   * * ** *

* * * * * * ** *   * * * * * ** *

Tempo de execução: 5.73974e-06 segundos

schedule(guided):

**********   ***

        **********   ** **

                ****

              *********   **** *** *

Tempo de execução: 3.79793e-06 segundos

Tarefa 2 –

Primeira vez:

for 2147483648 steps pi = 0.85025571137293 in 3.03431408666074 secs

for 2147483648 steps pi = 1.23743450780128 in 2.53132573515177 secs

segunda vez:

for 2147483648 steps pi = 0.836705530101938 in 2.37544267904013 secs

for 2147483648 steps pi = 0.692237241804317 in 3.23874218389392 secs

terceira vez:

for 2147483648 steps pi = 1.20554698441116 in 3.26396093145013 secs

for 2147483648 steps pi = 1.30900966713299 in 2.48057495057583 secs

Tarefa 3 –

Primeira vez:

Tempo de execução com critical: 0.00091261 segundos

Tempo de execução com pré-alocação: 0.000151251 segundos

Segunda vez:

Tempo de execução com critical: 0.000967374 segundos

Tempo de execução com pré-alocação: 0.000162865 segundos

Terceira vez:

Tempo de execução com critical: 0.00118922 segundos

Tempo de execução com pré-alocação: 0.000118824 segundos

1. Testando Schedulers no OpenMP

- **Scheduler estático** (static) foi o mais rápido e eficiente, porque divide as tarefas de forma equilibrada entre as threads logo no início, sem precisar reorganizar as tarefas durante a execução.
- **Scheduler dinâmico** (dynamic) e **guided** foram um pouco mais lentos porque eles precisam dividir as tarefas conforme elas vão sendo executadas, o que cria um pouco de atraso (overhead).

- **Scheduler padrão** (default) foi o mais lento, provavelmente por não estar tão otimizado para esse tipo de trabalho.

Conclusão: O **static scheduler** é a melhor escolha quando as tarefas podem ser divididas de forma equilibrada desde o início.

---

2. Paralelização do Cálculo de Pi

- **Parallel for:** Teve tempos de execução que variaram, mostrando que pode haver desequilíbrios na forma como as tarefas são divididas entre as threads.
- **Tasks:** O uso de tasks mostrou uma leve melhora no tempo, porque as tarefas são divididas de forma mais flexível, adaptando-se melhor ao problema recursivo.

Conclusão: Para esse tipo de problema recursivo, **tasks** funcionaram um pouco melhor do que o uso de parallel for, porque permitem uma divisão mais eficiente do trabalho.

---

3. Efeitos Colaterais no Vetor

- **Seção crítica (critical):** Adicionou um atraso (overhead) significativo, porque as threads precisam esperar uma pela outra para modificar o vetor, o que diminui a velocidade.
- **Pré-alocação de memória:** Foi muito mais rápida, porque as threads não precisaram esperar umas pelas outras, já que o vetor foi pré-alocado e cada thread pôde trabalhar em sua parte ao mesmo tempo.

Conclusão: A **pré-alocação de memória** foi muito mais rápida e eficiente. Evitar seções críticas e sincronizações quando possível melhora muito o desempenho.

Resumo Geral:

- Para problemas que envolvem **recursão**, o uso de **tasks** geralmente é mais eficiente, porque as tarefas são divididas de forma mais flexível.
- Quando há **efeitos colaterais** (como várias threads modificando o mesmo vetor), é melhor **pré-alocar a memória** para evitar que as threads tenham que esperar umas pelas outras, o que melhora bastante o desempenho.