

Dynamic Processes Creation in MPI.NET

Fernando Afonso
Nicolas Maillard

Sumário

- Introdução
- Objetivo
- Estado da Arte
- Proposta MPI.NET-Spawn
- Avaliação de desempenho
- Conclusões
- Trabalhos Futuros

Introdução

- Message Passing Interface (**MPI**) é uma especificação de biblioteca para o desenvolvimento de aplicações paralelas.
 - Alto-desempenho;
 - Modelo de comunicação baseado em **troca de mensagens** (explícitas);
 - Geralmente utilizado em cluster.

Introdução

- Inicialmente a norma MPI especificou interfaces de programação (**APIs**) para as linguagens de programação Fortran e C.
- Posteriormente a norma MPI especificou a **API** MPI para a linguagem de programação C++.
- A norma MPI-2 especificou novos mecanismos:
 - **Criação dinâmica de processos;**
 - E/S paralela;
 - Operações remotas de memória.

Introdução

- A crescente utilização de linguagens de programação como Java e C# motivou a criação de bibliotecas com suporte a MPI para essas linguagens.
- Algumas dessas bibliotecas acrescentam funcionalidades como o envio de objetos e a simplificação das chamadas MPI.
- No entanto essas bibliotecas suportam somente as funcionalidades da norma MPI-1 ou algum subgrupo de suas funcionalidades.

Objetivo

Explorar a **criação dinâmica de processos** na biblioteca MPI.NET

- A comunicação entre processos de um programa MPI é realizada por meio de troca de mensagens explícitas.
- Para isso, MPI especifica diversas primitivas de comunicação (ponto a ponto, coletivas, bloqueantes e não bloqueantes).
- Por exemplo para realizar a troca de uma mensagem ponto a ponto, a primitiva `MPI_Send` e suas variações são utilizadas.

Estado da Arte

Programação com MPI

```
void main() {  
    int r, tag = 103;  
  
    MPI_Init(...) ;  
    MPI_Comm_rank(MPI_COMM_WORLD, &r);  
  
    if (r==0) {  
        val = 3 ;  
        MPI_Send(&val,...,1,tag,...);  
    }  
  
    if (r!=0)  
        MPI_Recv(&val, ..., 0, tag, ...);  
  
    MPI_Finalize();  
}
```


Estado da Arte

Programação com MPI

```
void main() {  
    int r, tag = 103;
```

A biblioteca MPI é inicializada

```
    MPI_Init(...);  
    MPI_Comm_rank(MPI_COMM_WORLD, &r);
```

```
    if (r==0) {  
        val = 3;  
        MPI_Send(&val,...,1,tag,...);  
    }
```

```
    if (r!=0)  
        MPI_Recv(&val, ..., 0, tag, ...);
```

```
    MPI_Finalize();  
}
```

Estado da Arte

Programação com MPI

```
void main() {  
    int r, tag = 103;
```

```
    MPI_Init(...);  
    MPI_Comm_rank(MPI_COMM_WORLD, &r);
```

```
    if (r==0) {  
        val = 3;  
        MPI_Send(&val,...,1,tag,...);  
    }
```

```
    if (r!=0)  
        MPI_Recv(&val, ..., 0, tag, ...);
```

```
    MPI_Finalize();  
}
```

O Ranking do processo é recuperado
na variável r

Estado da Arte

Programação com MPI

```
void main() {  
    int r, tag = 103;  
  
    MPI_Init(...);  
    MPI_Comm_  
  
    if (r==0) {  
        val = 3;  
        MPI_Send(&val,...,1,tag,...);  
    }  
  
    if (r!=0)  
        MPI_Recv(&val, ..., 0, tag, ...);  
  
    MPI_Finalize();  
}
```

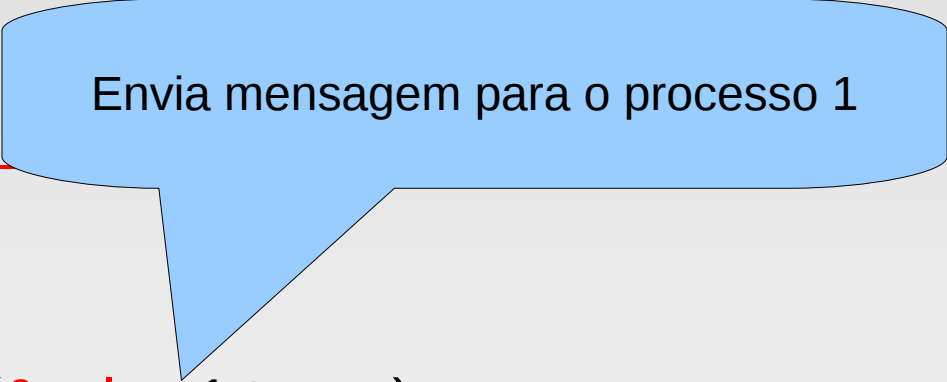


Se o Ranking é 0

Estado da Arte

Programação com MPI

```
void main() {  
    int r, tag = 103;  
  
    MPI_Init(...) ;  
    MPI_Comm_  
  
    if (r==0) {  
        val = 3 ;  
        MPI_Send(&val,...,1,tag,...);  
    }  
  
    if (r!=0)  
        MPI_Recv(&val, ..., 0, tag, ...);  
  
    MPI_Finalize();  
}
```

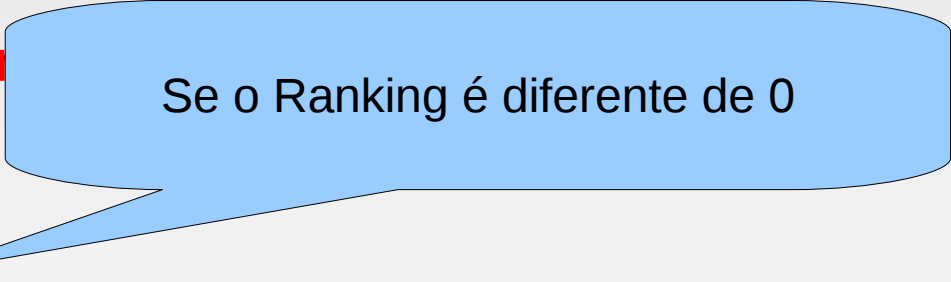


Envia mensagem para o processo 1

Estado da Arte

Programação com MPI

```
void main() {  
    int r, tag = 103;  
  
    MPI_Init(...) ;  
    MPI_Comm_rank(MPI_COMM_WORLD, &r);  
  
    if (r==0) {  
        val = 3 ;  
        MPI_Send(&val, 1, MPI_INT, 1, tag, MPI_COMM_WORLD);  
    }  
  
    if (r!=0) {  
        MPI_Recv(&val, 1, MPI_INT, 0, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);  
    }  
  
    MPI_Finalize();  
}
```

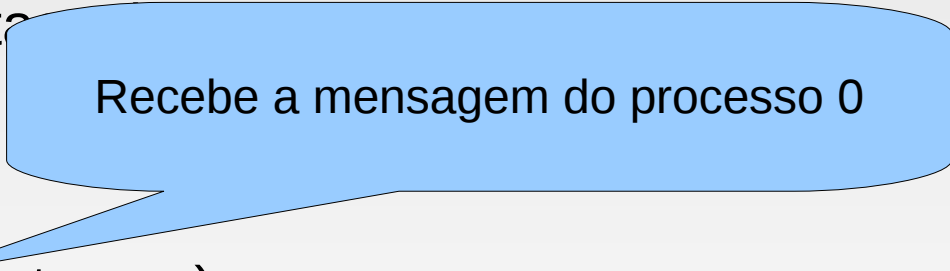


Se o Ranking é diferente de 0

Estado da Arte

Programação com MPI

```
void main() {  
    int r, tag = 103;  
  
    MPI_Init(...);  
    MPI_Comm_rank(MPI_COMM_WORLD, &r);  
  
    if (r==0) {  
        val = 3;  
        MPI_Send(&val,...,1,tag);  
    }  
  
    if (r!=0)  
        MPI_Recv(&val, ..., 0, tag, ...);  
  
    MPI_Finalize();  
}
```

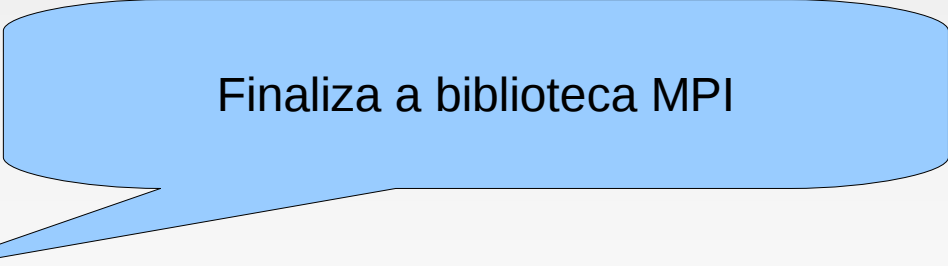


Recebe a mensagem do processo 0

Estado da Arte

Programação com MPI

```
void main() {  
    int r, tag = 103;  
  
    MPI_Init(...);  
    MPI_Comm_rank(MPI_COMM_WORLD, &r);  
  
    if (r==0) {  
        val = 3;  
        MPI_Send(&val,...,1,tag,...);  
    }  
  
    if (r!=0)  
        MPI_Recv(&val,  
  
    MPI_Finalize();  
}
```



Finaliza a biblioteca MPI

Estado da Arte

- Podemos classificar os projetos de bibliotecas MPI alternativas em duas categorias:
 - Implementações MPI inteiras: Pure MPI.NET, PMPI, PJMPI, JMPI...
 - Desempenho ruim;
 - Pequeno subgrupo de funcionalidades MPI.
 - Implementações sobre bibliotecas MPI existentes (chamadas nativas): mpiJava, MPI.NET, pyMPI, MPIRuby...
 - Desempenho bom;
 - Suporte completo as funcionalidades MPI-1.

Estado da Arte

- Dentre as bibliotecas MPI alternativas as que demonstram melhor desempenho são as bibliotecas mpiJava e MPI.NET.
- O desempenho dessas **bibliotecas** é bem similar ao desempenho da biblioteca MPI nativa.
- No entanto, enquanto a biblioteca MPI.NET abstrai as chamadas MPI a biblioteca mpiJava simplesmente mapeia chamadas C para Java.

MPI.NET

- A biblioteca MPI.NET pode ser utilizada por qualquer linguagem de programação da plataforma .Net.
- Dentre as abstrações oferecidas pela biblioteca destacam-se:
 - A capacidade de enviar objetos diretamente como se fossem tipos primitivos (tema de diversos projetos MPI para C++);
 - A redução significativa no número de parâmetros necessário pelas chamadas MPI.
 - A capacidade de introspecção.

Proposta MPI.NET-Spawn

- As primitivas responsáveis por criar e gerenciar processos dinamicamente no MPI são:
 - `MPI_Comm_spawn`: responsável por criar novos processos dinamicamente;
 - `MPI_Comm_spawn_multiple`: responsável por criar novos processos de múltiplas aplicações dinamicamente;
 - `MPI_Comm_get_parent`: responsável por recuperar o comunicador com o pai no processo criado.

Proposta MPI.NET-Spawn

- As interfaces da chamada `MPI_Comm_spawn` em C e C++ são respectivamente:
 - `int MPI_Comm_spawn (char *command, char *argv[], int maxprocs, MPI_Info info, int root, MPI_Comm comm, MPI_Comm *intercomm, int array_of_errcodes[]);`
 - Intercommunicator `MPI::Comm.Spawn` (char *command, char *argv[], int maxprocs, MPI_Info info, int root).
- Já na implementação realizada neste trabalho a interface no .Net pode ser tão simples como:
 - Intercommunicator `MPI.Spawn(String command);`

Proposta MPI.NET-Spawn

Utilização do Spawn

Programa pai

Um novo programa “filho” é inicializado
O comunicador é recuperado
na variável intercomm

```
...  
MPI_Init(...);  
...
```

```
MPI_Comm_spawn(...,&intercomm,...);
```

```
MPI_Send(&val,...,0,tag, intercomm);  
MPI_Recv(&val, ..., 0, tag,  
intercomm);
```

```
MPI_Finalize();  
}
```

Programa filho

```
void main() {  
...  
MPI_Init(...);  
...
```

```
MPI_Comm_get_parent(&parentcomm);
```

```
MPI_Recv(&val, ..., 0, tag,  
parentcomm);  
MPI_Send(&val,...,0,tag,  
parentcomm);
```

```
MPI_Finalize();  
}
```

Proposta MPI.NET-Spawn

Utilização do Spawn

■ Programa pai

```
void main() {  
    ...  
    MPI_Init(...);  
    ...  
}
```

```
MPI_Comm_spawn(...,&intercomm,...);
```

```
MPI_Send(&val,...,0,tag, intercomm);  
MPI_Recv(&val, ..., 0, tag,  
intercomm);
```

```
MPI_Finalize();  
}
```

■ Programa filho

O programa “filho” recupera o comunicador
Com o pai na variável parent

```
MPI_Init(...);  
...
```

```
MPI_Comm_get_parent(&parentcomm);
```

```
MPI_Recv(&val, ..., 0, tag,  
parentcomm);  
MPI_Send(&val,...,0,tag,  
parentcomm);
```

```
MPI_Finalize();  
}
```

Proposta MPI.NET-Spawn

Utilização do Spawn

- Programa pai

```
void main() {
```

Uma mensagem é enviada ao filho

```
MPI_Comm_spawn(...,&intercomm,  
mm,  
  
MPI_Send(&val,...,0,tag, intercomm);  
MPI_Recv(&val, ..., 0, tag,  
intercomm);  
  
MPI_Finalize();  
}
```

- Programa filho

```
void main() {
```

O filho recebe uma mensagem do pai

```
MPI_Comm_get_parent(&parentcomm);  
tcomm  
  
MPI_Recv(&val, ..., 0, tag,  
parentcomm);  
MPI_Send(&val,...,0,tag,  
parentcomm);  
  
MPI_Finalize();  
}
```

Proposta MPI.NET-Spawn

Utilização do Spawn

■ Programa pai

```
void main() {  
    ...  
    MPI_Init(...);  
    ...
```

O pai recebe uma mensagem do filho

```
    MPI_Send(&val,...,0,tag, intercomm);  
    MPI_Recv(&val, ..., 0, tag,  
    intercomm);  
  
    MPI_Finalize();  
}
```

■ Programa filho

```
void main() {  
    ...  
    MPI_Init(...);  
    ...
```

O filho envia uma mensagem ao pai

```
    MPI_Recv(&val, ..., 0, tag,  
    parentcomm);  
    MPI_Send(&val,...,0,tag,  
    parentcomm);  
  
    MPI_Finalize();  
}
```


Proposta MPI.NET-Spawn

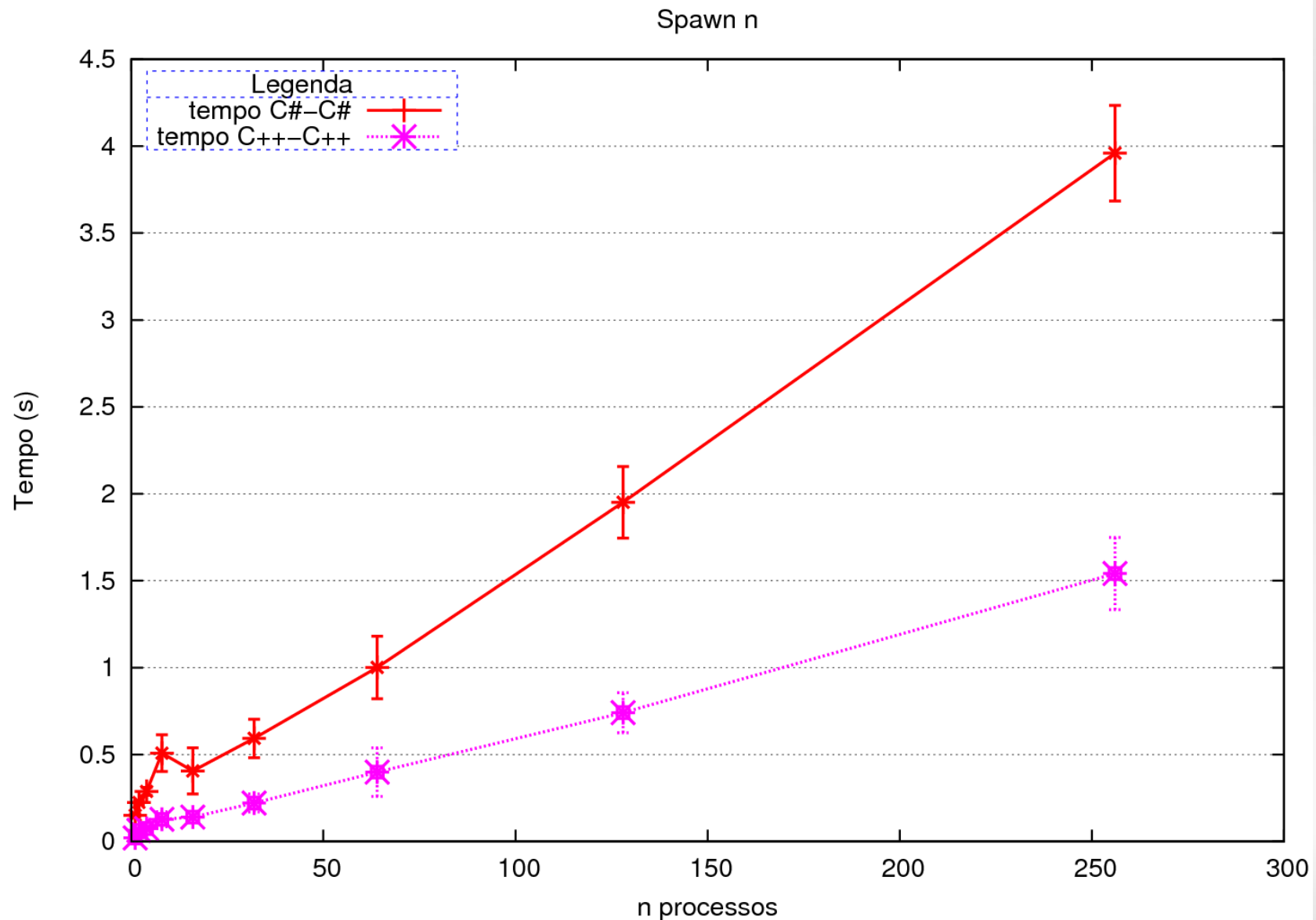
- Foram realizadas outras 4 sobrecargas desse método permitindo o uso pleno da funcionalidade especificada na norma.
- Foram implementadas também as primitivas `MPI_Comm_spawn_multiple` e `MPI_Comm_get_parent`.
- A implementação chama a função nativa em C.
 - É necessário converter os tipos diferentes.
 - É necessário proteger a memória do coletor de lixo.
 - É necessário recuperar o endereço das variáveis.

Avaliação de Desempenho

- Foram executados alguns benchmarks sintéticos para verificar o desempenho da biblioteca.
 - Spawn-n: na criação de n processos com tarefas vazias;
 - Cálculo do i -ésimo elemento da sequência de Fibonacci: em uma aplicação que crie muitos processos com tarefas pequenas;
 - Cálculo do fractal de Mandelbrot: em uma aplicação com a relação número de processos/tamanho das tarefas balanceada.

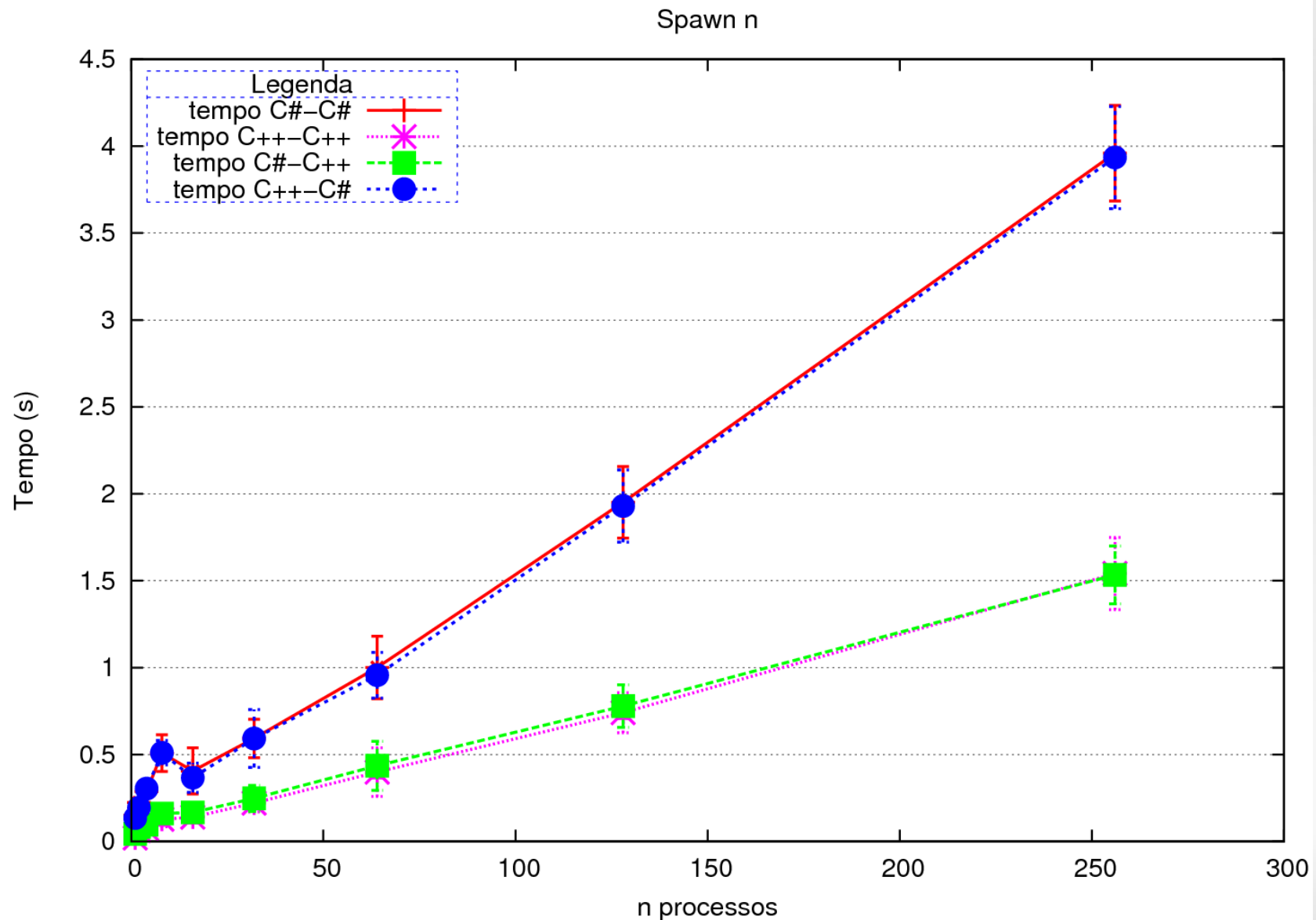
Avaliação de Desempenho

Spawn-n



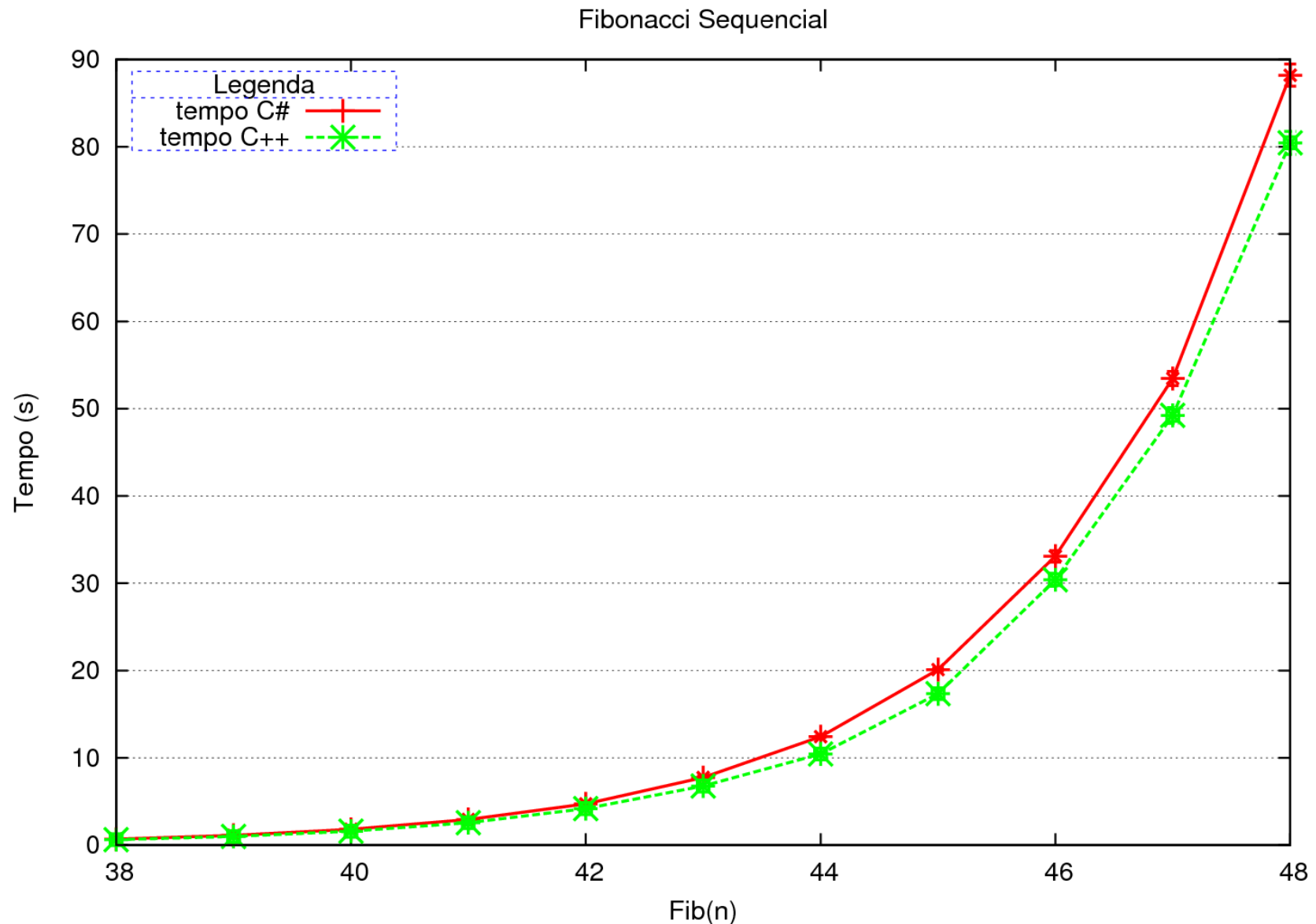
Avaliação de Desempenho

Spawn-n



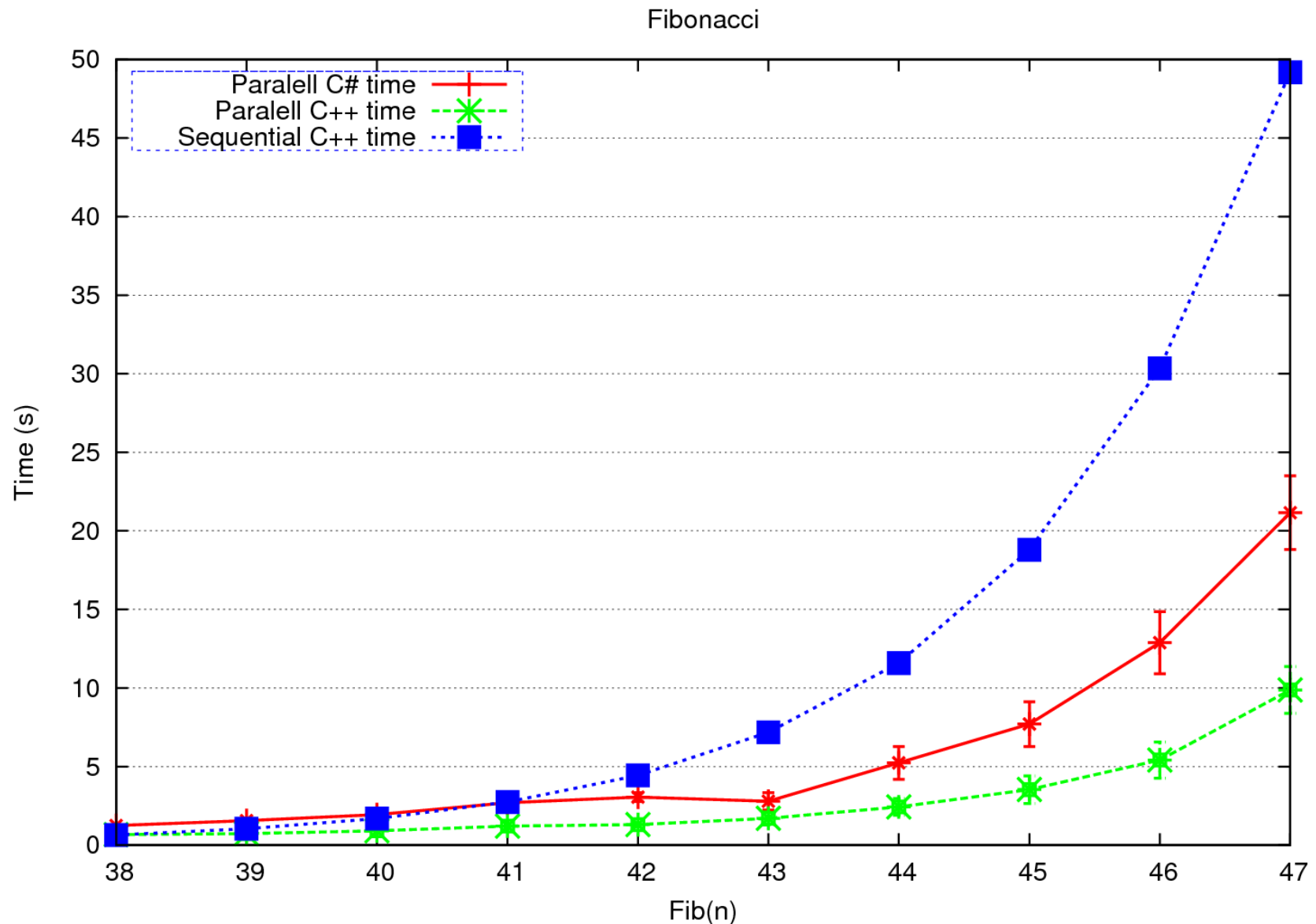
Avaliação de Desempenho

Fibonacci Sequencial



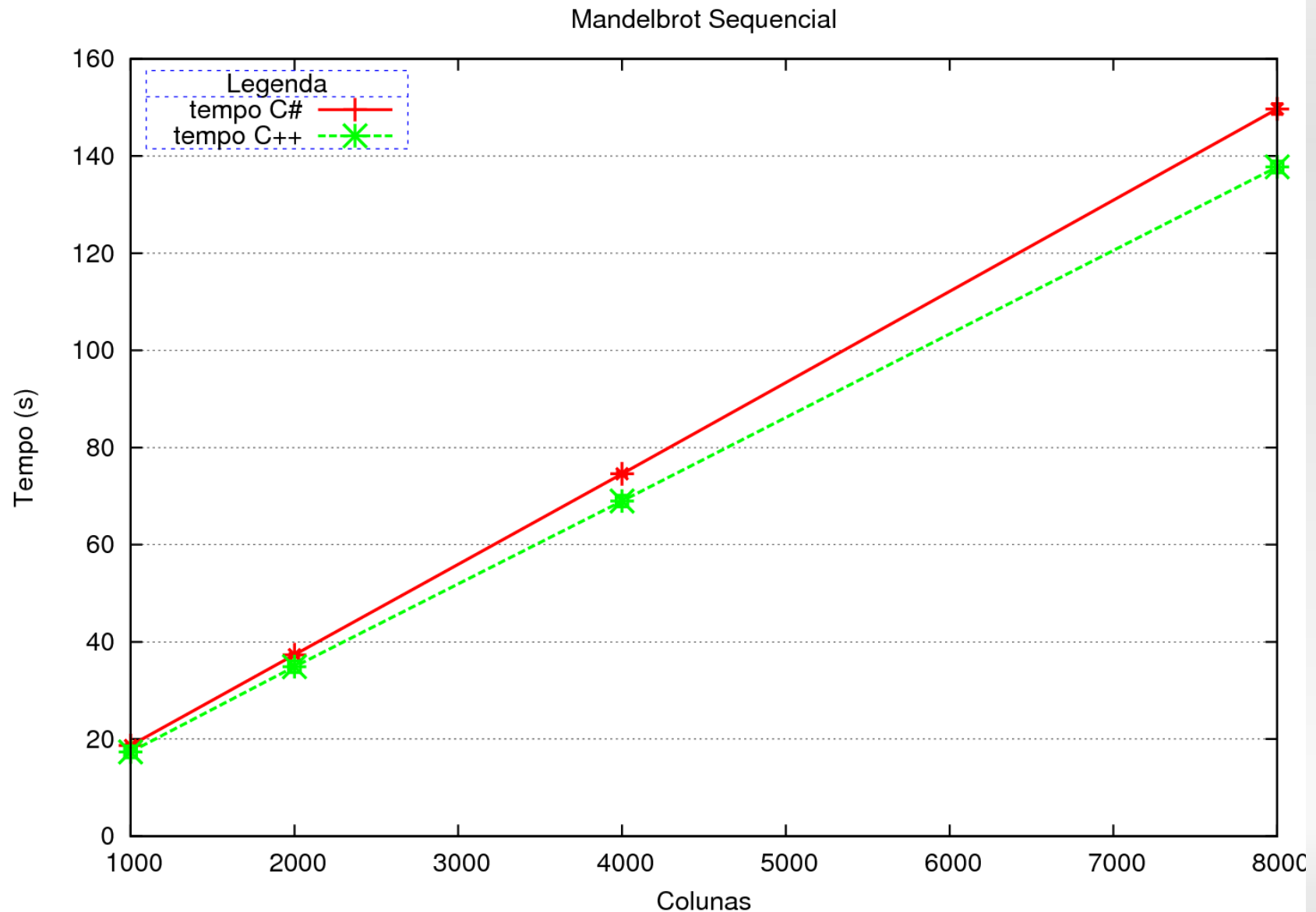
Avaliação de Desempenho

Fibonacci Paralelo



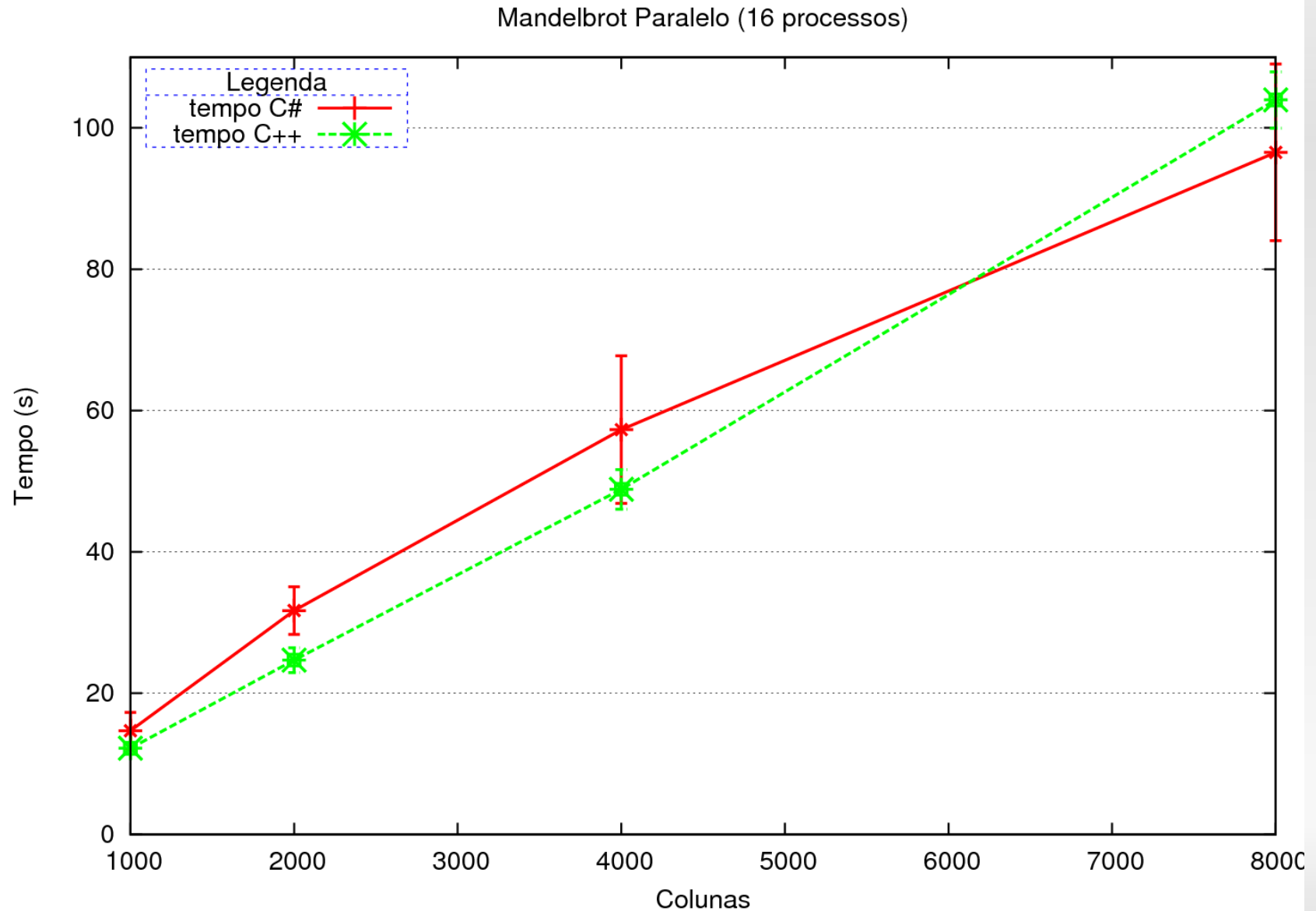
Avaliação de Desempenho

Mandelbrot Sequencial



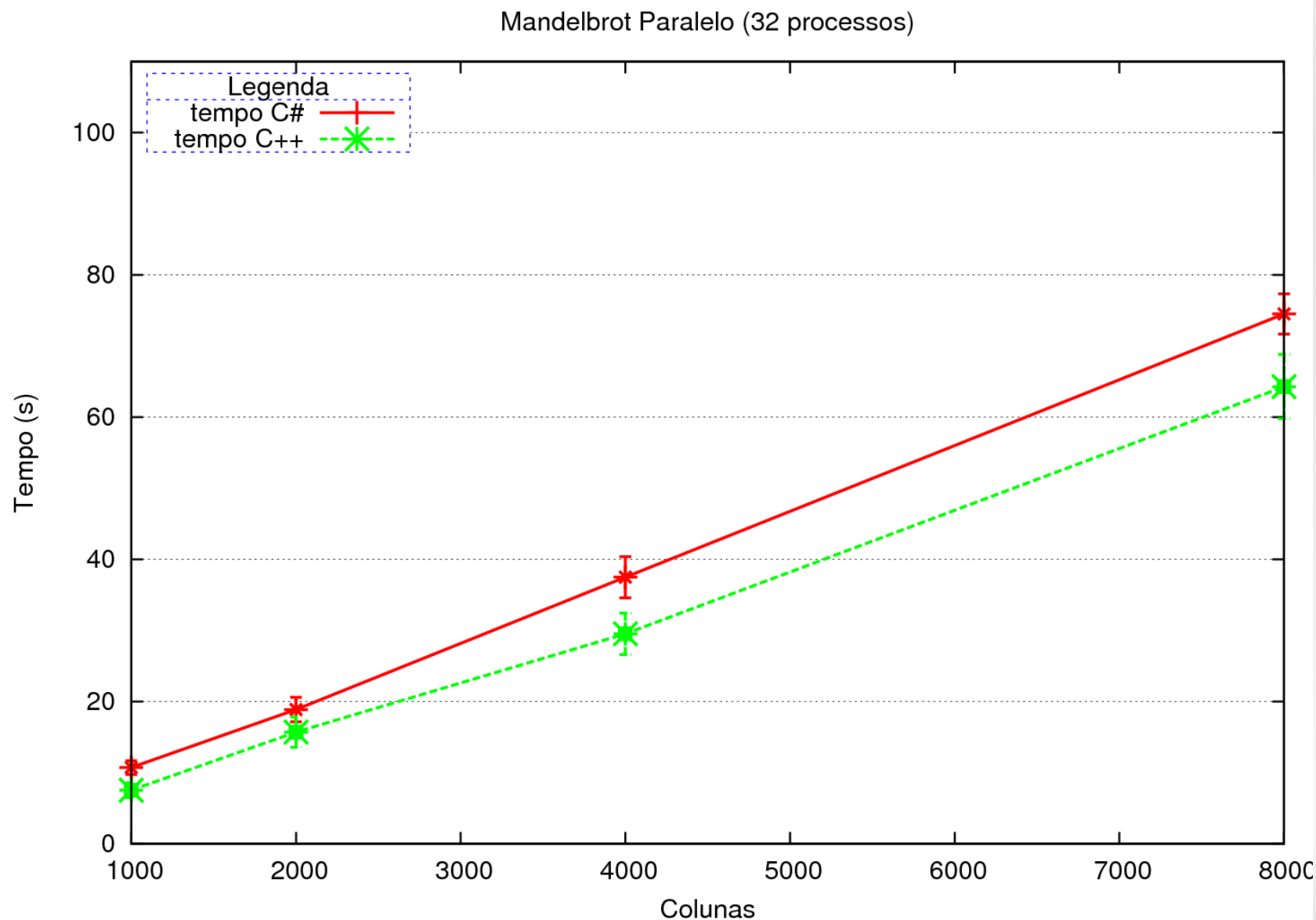
Avaliação de Desempenho

Mandelbrot Paralelo



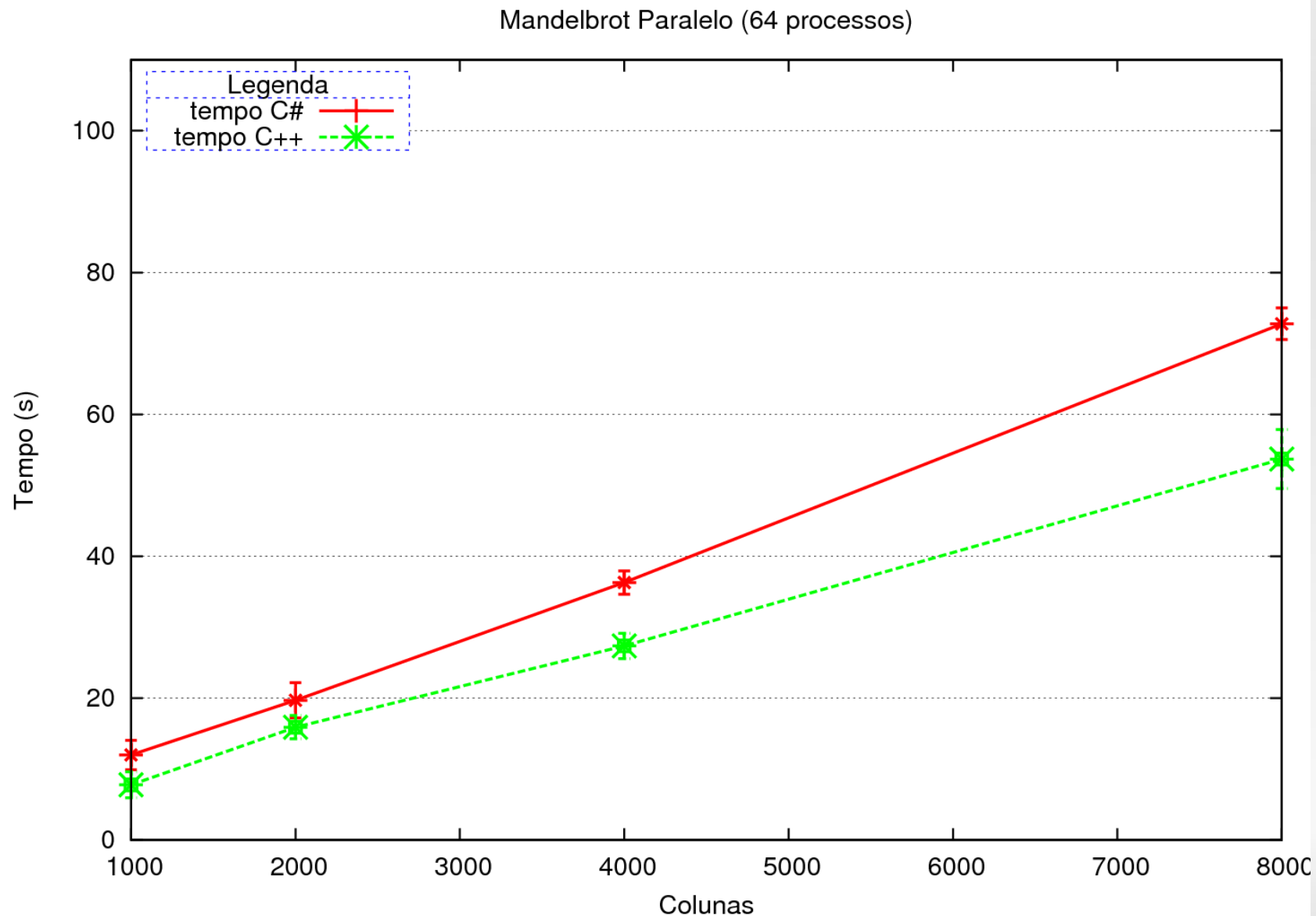
Avaliação de Desempenho

Mandelbrot Paralelo



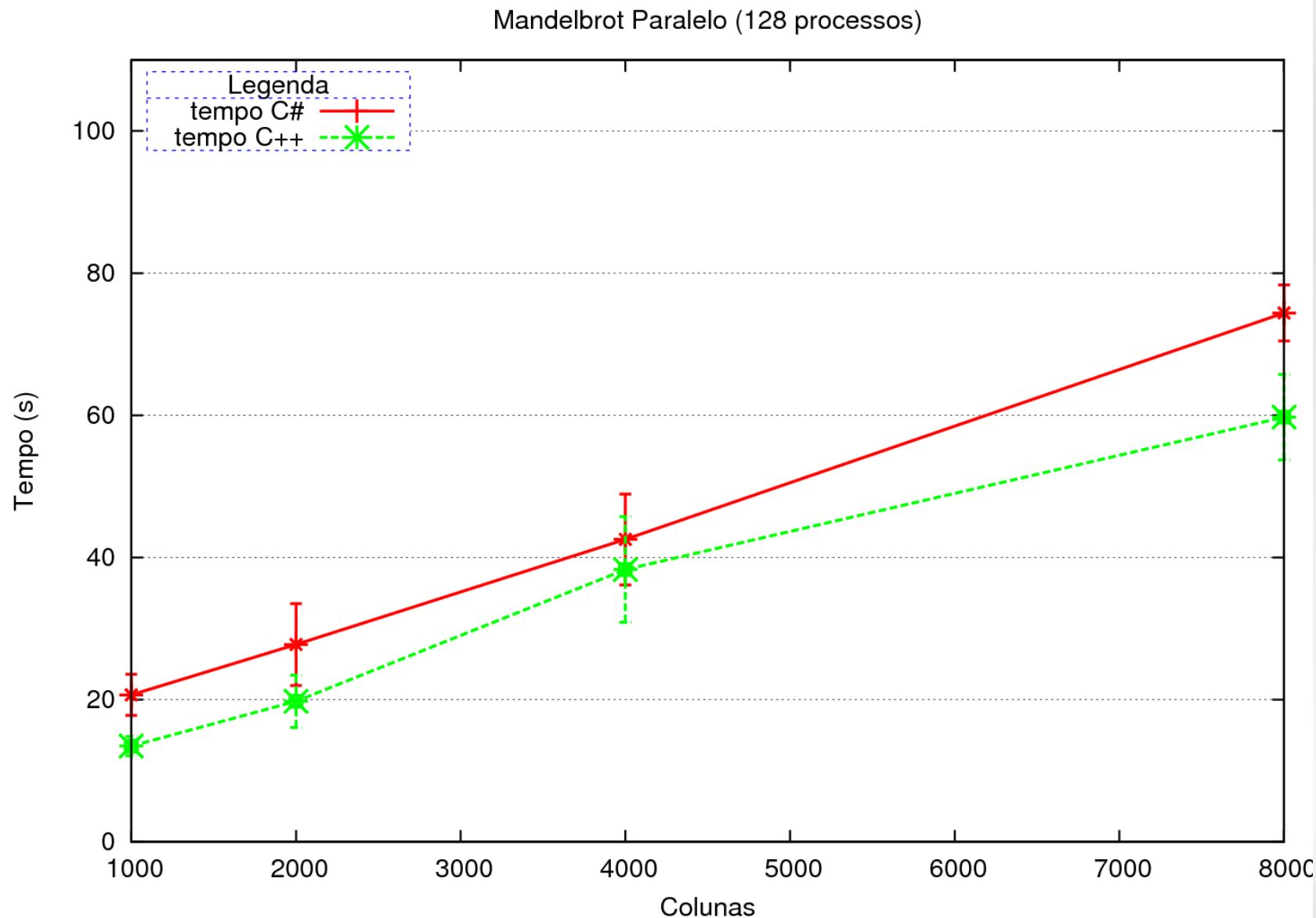
Avaliação de Desempenho

Mandelbrot Paralelo



Avaliação de Desempenho

Mandelbrot Paralelo



Conclusões

- Embora seja viável suportar a norma MPI-2 em outras linguagens não suportadas pela norma nem todas aplicações irão obter desempenho aceitável.
- A **biblioteca** não introduz *overhead* significativo sobre a biblioteca nativa.
- A **linguagem de programação** introduz *overhead* significativo sobre determinadas aplicações.
- A inicialização dos processos se mostrou um grande problema de desempenho.

Trabalhos Futuros

- Testar otimizações do .Net para tentar melhorar o desempenho tanto da linguagem quanto da inicialização dos processos.
 - Pré-compilar os programas;
 - Executar sem a máquina virtual;
 - Outras otimizações de *runtime*.
- Executar novos benchmarks sintéticos.

Dynamic Processes Creation in MPI.NET

Perguntas?