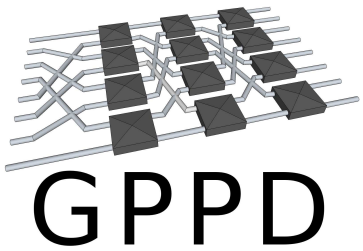
Online Mapping of MPI-2 Tasks to Processes and Threads

João Vicente F. Lima
Orientador: Prof. Dr. Nicolas Maillard
{joao.lima,nicolas}@inf.ufrgs.br

Universidade Federal do Rio Grande do Sul (UFRGS)

21 de agosto de 2009



Roteiro da apresentação

Roteiro da apresentação

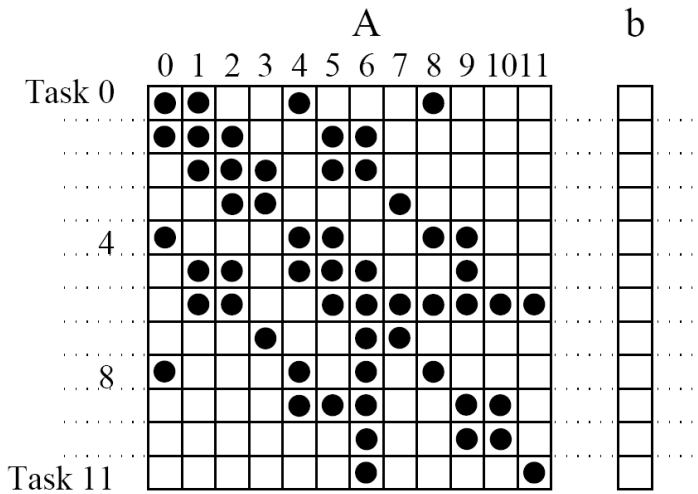
Introdução

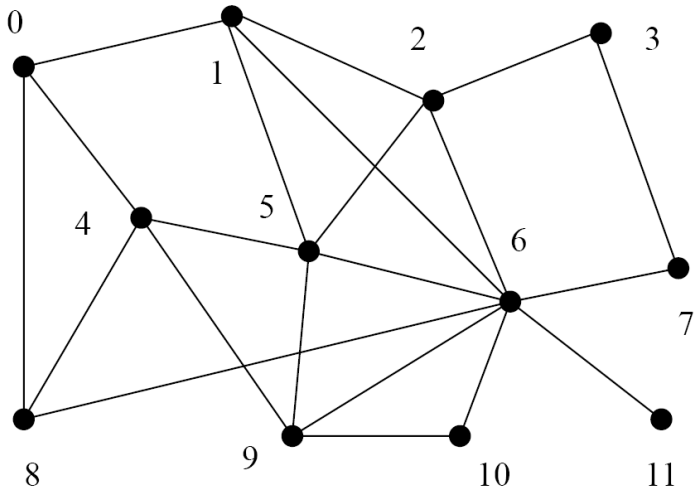
- Demanda crescente por desempenho ao longo dos anos
- No *hardware*:
  - arquiteturas vetoriais SIMD (*Single Instruction Multiple Data*)
  - sistemas MPPs (*Massively Parallel Processor*)
  - NOW (*Network of Workstations*)
  - *clusters* SMP (*Symmetric multiprocessor*)
  - Grades
- Mais recentemente:
  - *clusters* SMP *multi-core* (*many-core*)
  - *cluster of clusters*
- Mas, o desempenho não depende somente da arquitetura
- Um algoritmo paralelo também é decisivo

Roteiro da apresentação

Contexto

Algoritmos Paralelos Dinâmicos

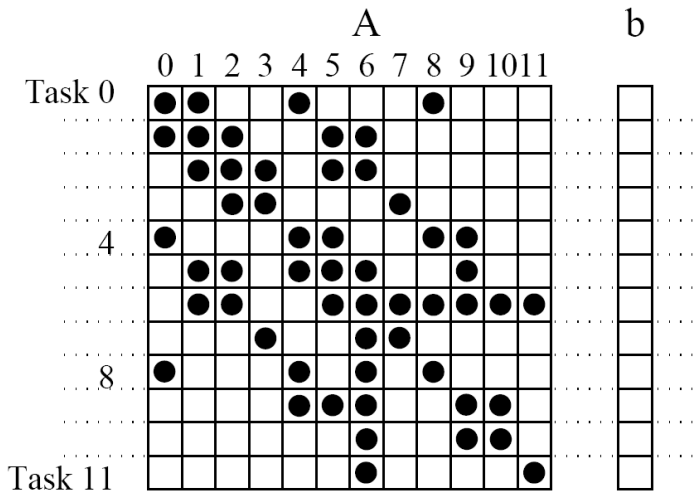


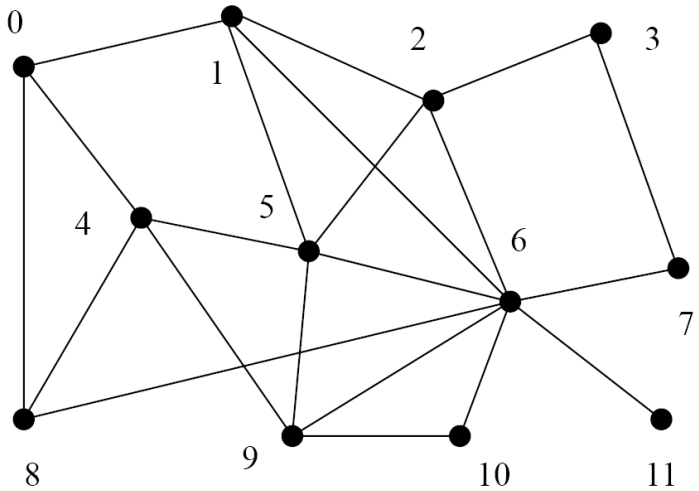


- O algoritmo, por sua vez, pode apresentar irregularidades

Contexto

Algoritmos Paralelos Dinâmicos

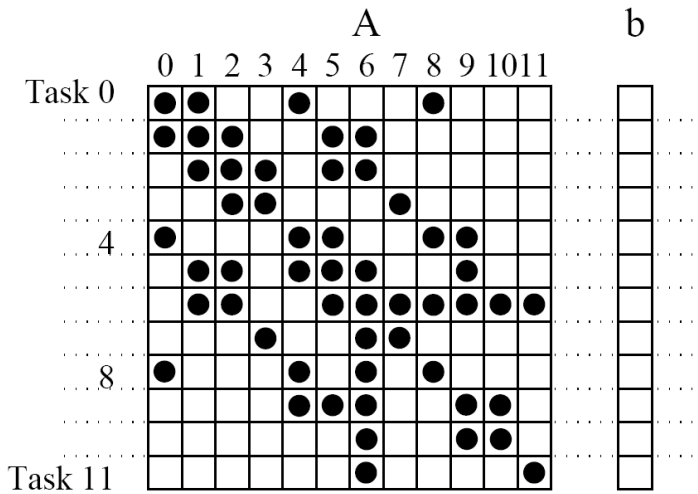


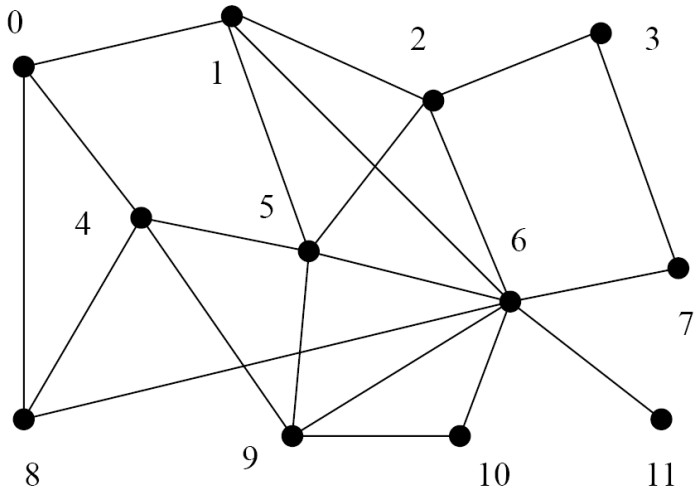


- O algoritmo, por sua vez, pode apresentar irregularidades
- Possível solução: um **algoritmo dinâmico**
  - prever a carga de tarefas com técnicas de programação
- Exemplos de técnicas: *Mestre/Escravo*, *Divisão-e-Conquista*
  - *Mestre/Escravo* é centralizada, ideal para problemas pequenos
  - *Divisão-e-Conquista* não determina o número de tarefas

Contexto

Algoritmos Paralelos Dinâmicos





- O algoritmo, por sua vez, pode apresentar irregularidades
- Possível solução: um **algoritmo dinâmico**
  - prever a carga de tarefas com técnicas de programação
- Exemplos de técnicas: *Mestre/Escravo*, *Divisão-e-Conquista*
  - *Mestre/Escravo* é centralizada, ideal para problemas pequenos
  - *Divisão-e-Conquista* não determina o número de tarefas
- O desempenho (plataforma/algoritmo) depende da **granularidade**

Contexto

Granularidade

- Razão entre processamento e comunicação
- Considera aspectos mais práticos (de plataforma)
  - recursos disponíveis
  - carga dos recursos
  - localidade
  - custos em comunicação e criação de tarefas
- Implementar um controle não é trivial, porque deve-se:
  - **descrever tarefa** - abstrata, processo ou *thread*
  - **controlar localidade** - reduzir custos de comunicação
  - **considerar carga de tabalho** - distribuir entre recursos
  - **controlar recursos** - inserção/remoção em tempo de execução

Contexto

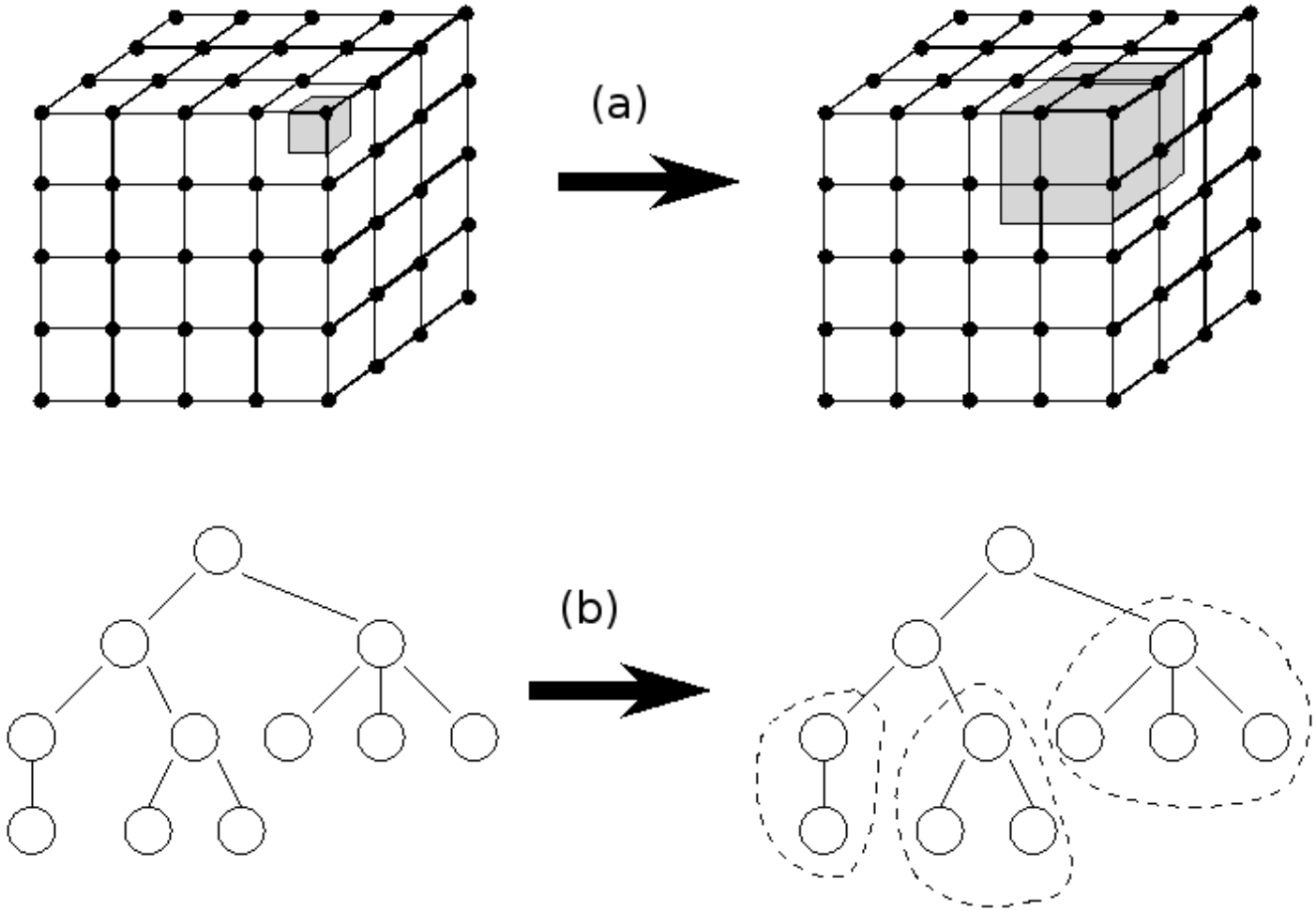
Granularidade

- Razão entre processamento e comunicação
- Considera aspectos mais práticos (de plataforma)
  - recursos disponíveis
  - carga dos recursos
  - localidade
  - custos em comunicação e criação de tarefas
- Implementar um controle não é trivial, porque deve-se:
  - **descrever tarefa** - abstrata, processo ou *thread*
  - **controlar localidade** - reduzir custos de comunicação
  - **considerar carga de tabalho** - distribuir entre recursos
  - **controlar recursos** - inserção/remoção em tempo de execução

Um ambiente de programação com suporte ao controle de granularidade simplifica a programação de algoritmos dinâmicos

Contexto

Granularidade



Roteiro da apresentação

Estado da Arte

Ambientes de Programação

- Memória Compartilhada: **Cilk**, **OpenMP** e **Intel©TBB**
  - **Cilk**: aplicações D&C para arquiteturas SMP
  - **OpenMP**: API com paralismo de laços, dados e tarefas (3.0)
  - **Intel©TBB** : biblioteca C++ para arquiteturas *multi-core*
- Memória Distribuída: **Cilk-NOW**, **Satin**, **KAAPI**, **AMPI** e **MPI**
  - **Cilk-NOW**: versão distribuída do **Cilk**
  - **Satin**: aplicações D&C para grades em Java
  - **KAAPI**: aplicações *multi-thread* e distribuídas
  - **AMPI**: implementação MPI para balanceamento de carga
  - **MPI**: padrão em PAD na programação com troca de mensagens

Estado da Arte

Conclusão

- Conclusão sobres os ambientes:
  - suporte para plataformas específicas
  - interfaces extensas, pouco uso em PAD
  - sem dinamismo (**AMPI**)
- Em **MPI**:
  - a norma define **tarefas**, mas as implementações usam **processos pesados**
  - implementações sem controle de granularidade
  - depende do programador (`MPI_Comm_spawn`/`pthread_create`)

Roteiro da apresentação

Proposta

libSpawn

Investigar as vantagens no controle *online* de granularidade em programas MPI dinâmicos

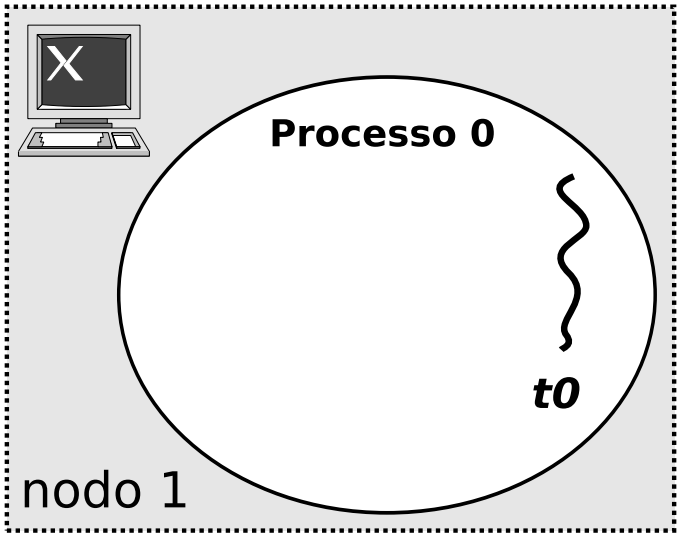
- **libSpawn** - biblioteca C++ que implementa o controle
- O controle processo/*thread* é **online** e transparente
- Implementa a comunicação entre tarefas
  - MPI não identifica *threads*

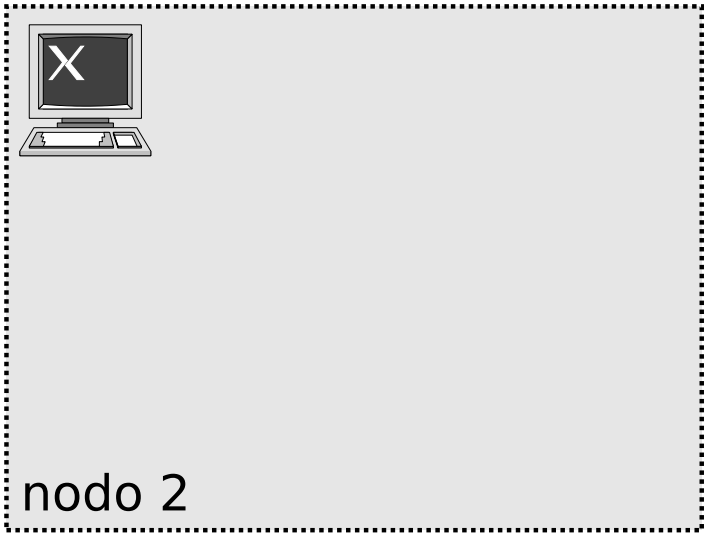
Parâmetros

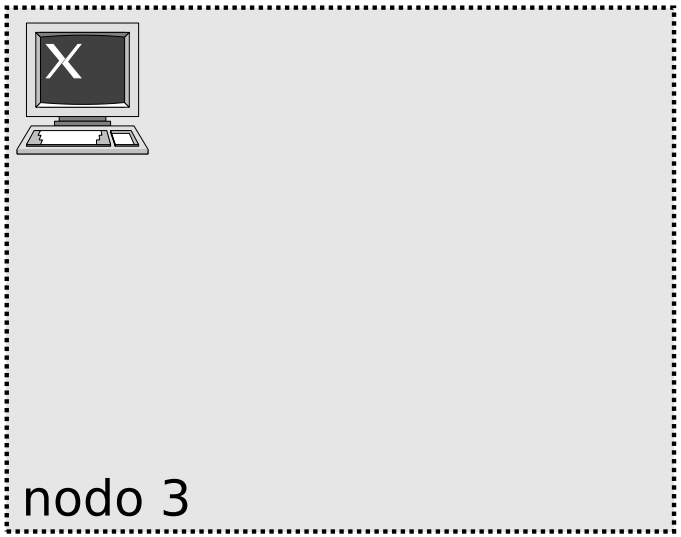
- **Cores da arquitetura** - *cores* disponíveis para execução
- **Carga de sistema** - avalia a carga de sistema em execução
- **Recursos de sistema** - limites de SO, como memória, descritores de arquivos, *threads*, ...

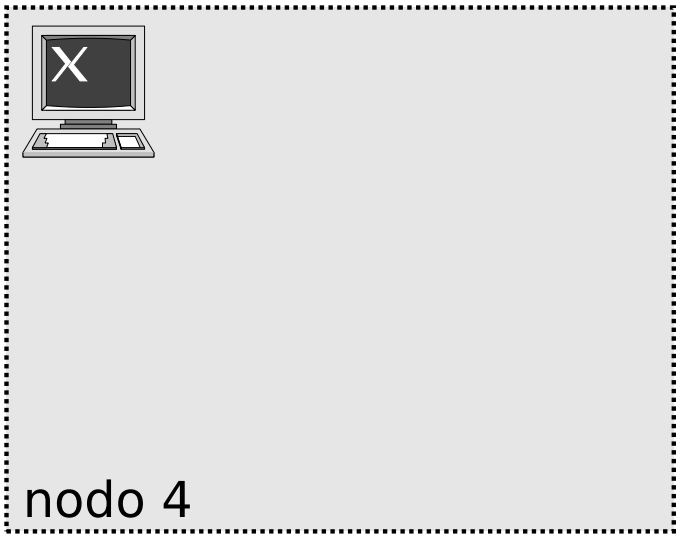
Proposta

Controle de Granularidade (1/6)



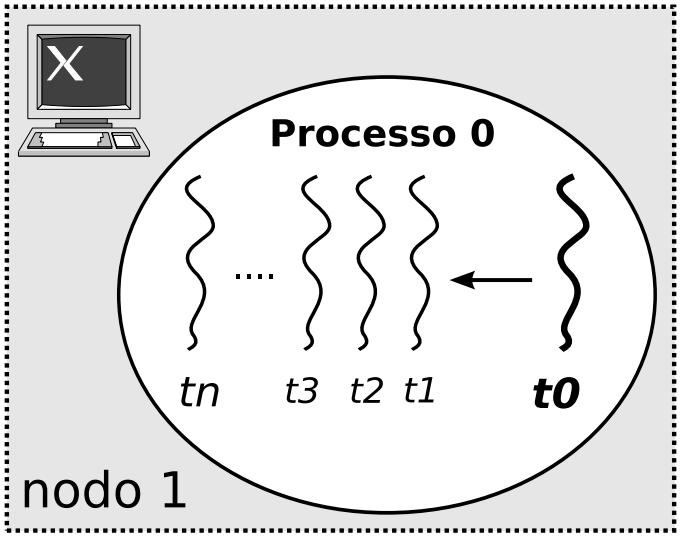


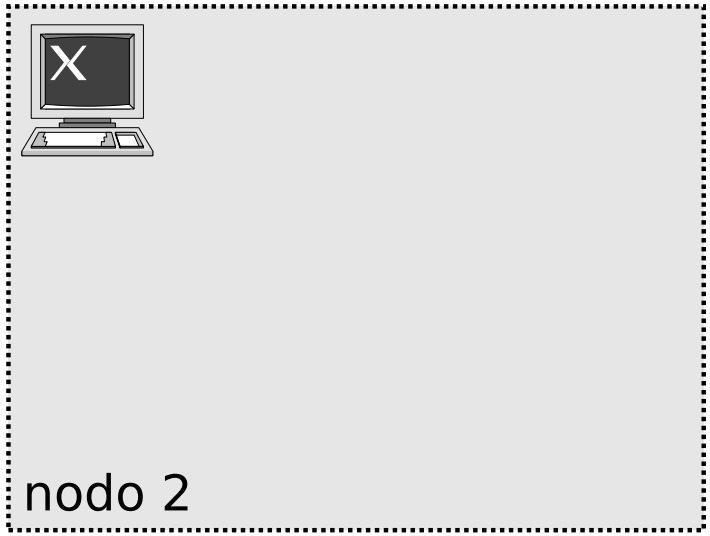


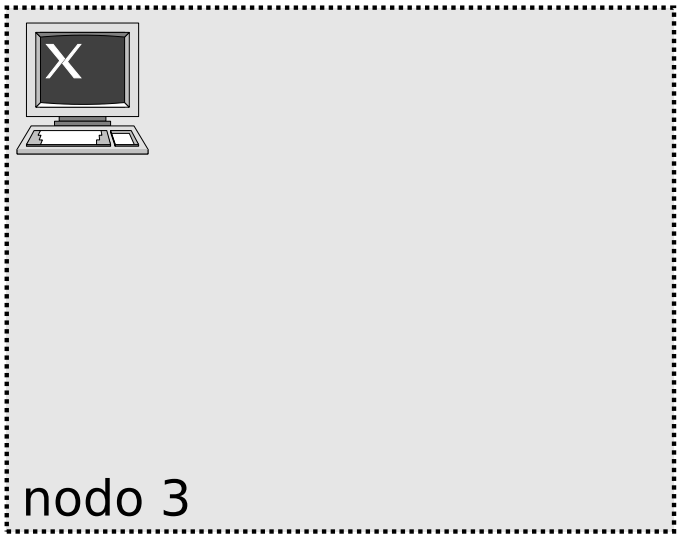


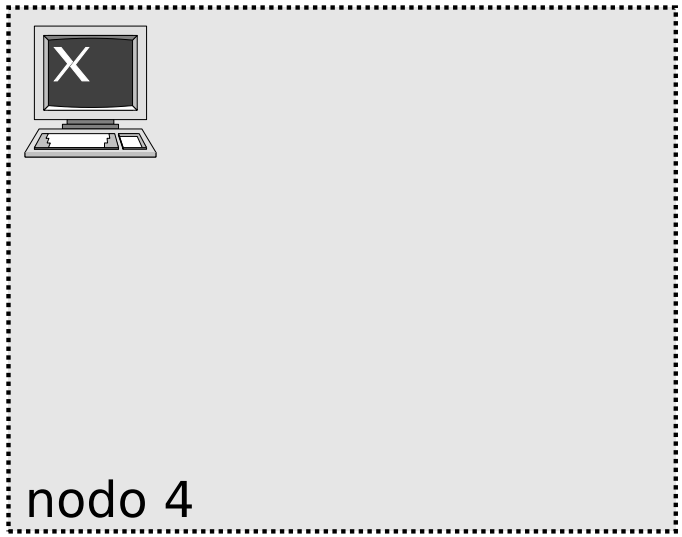
Proposta

Controle de Granularidade (2/6)



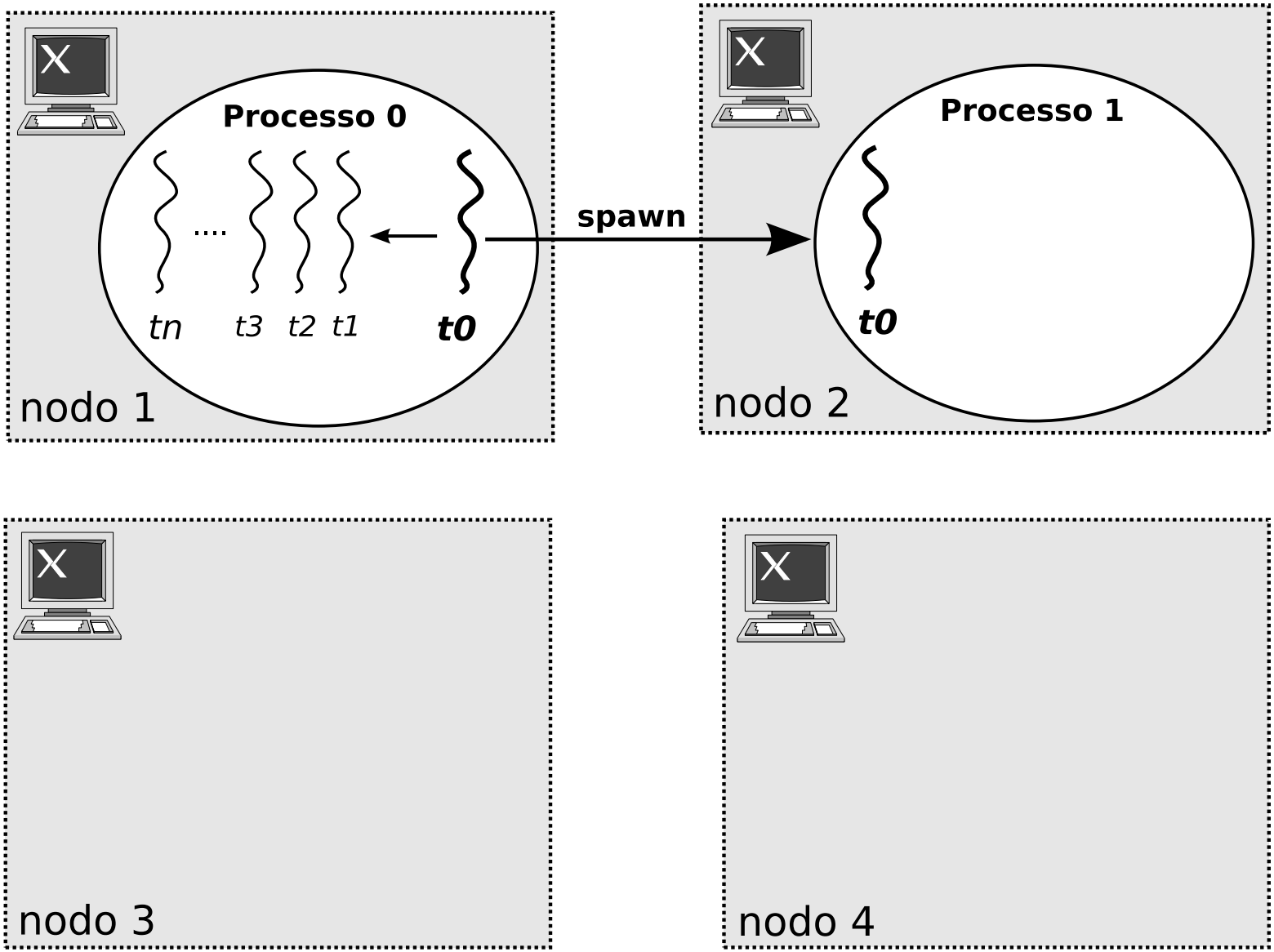






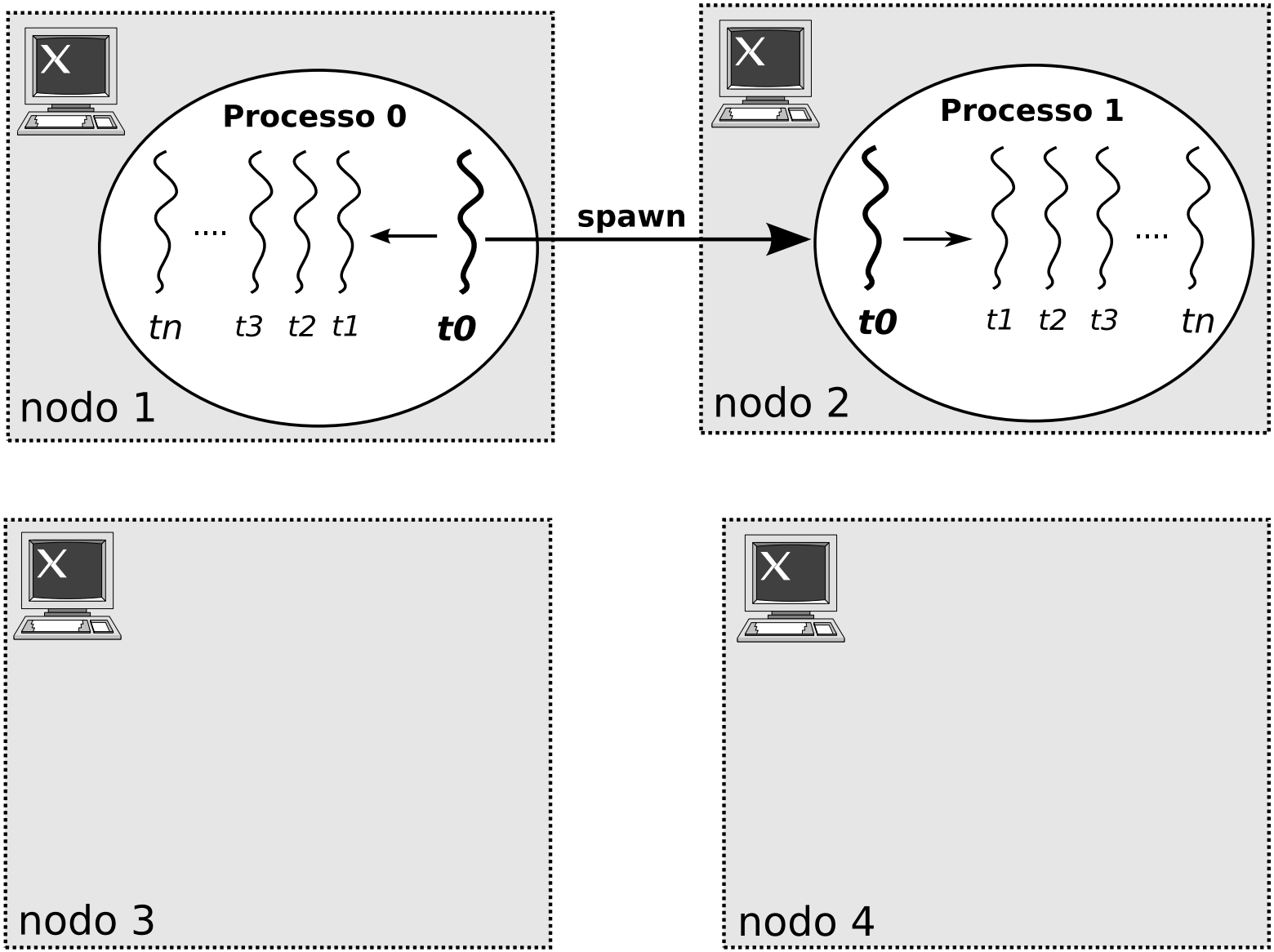
Proposta

Controle de Granularidade (3/6)



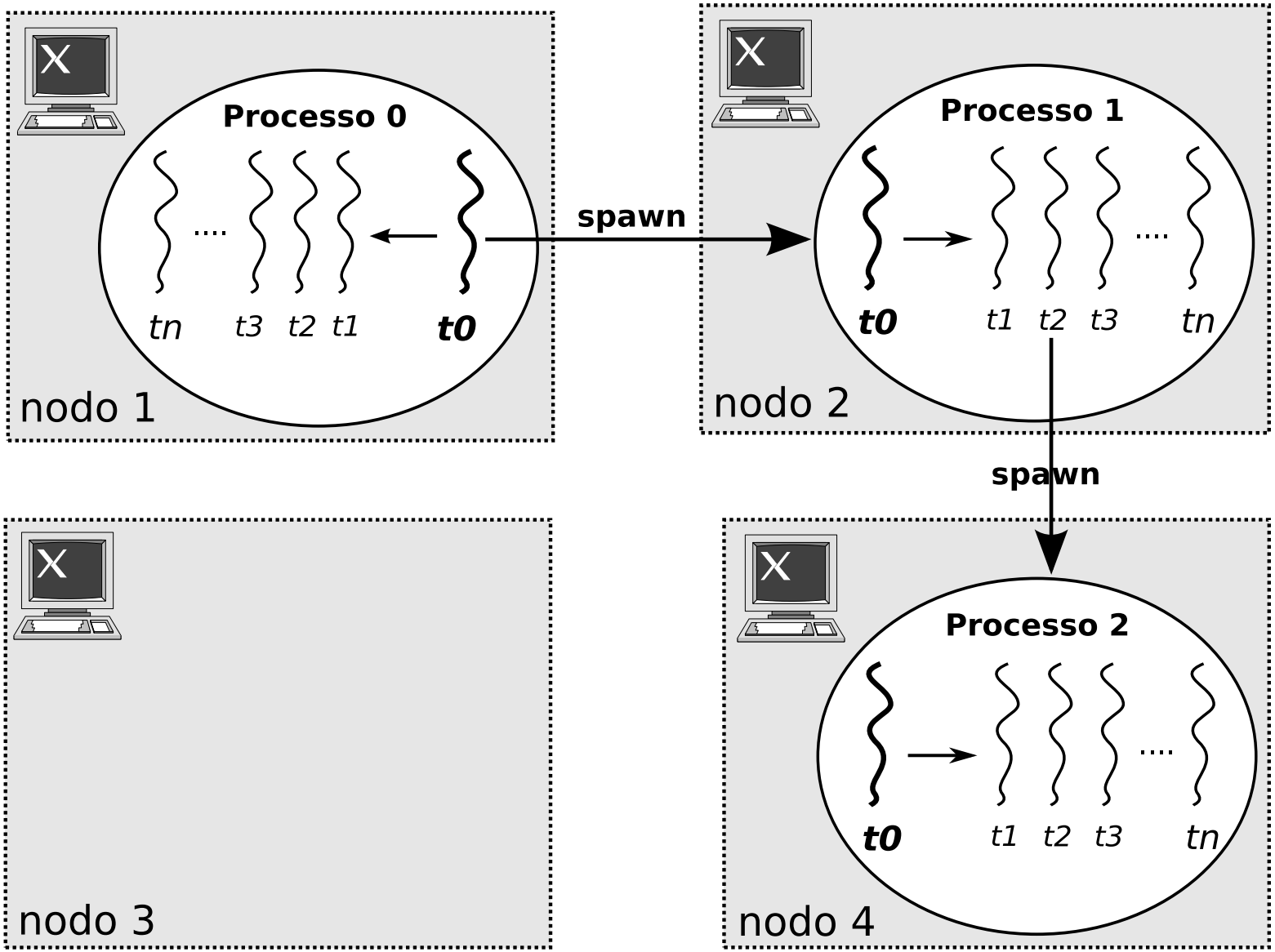
Proposta

Controle de Granularidade (4/6)



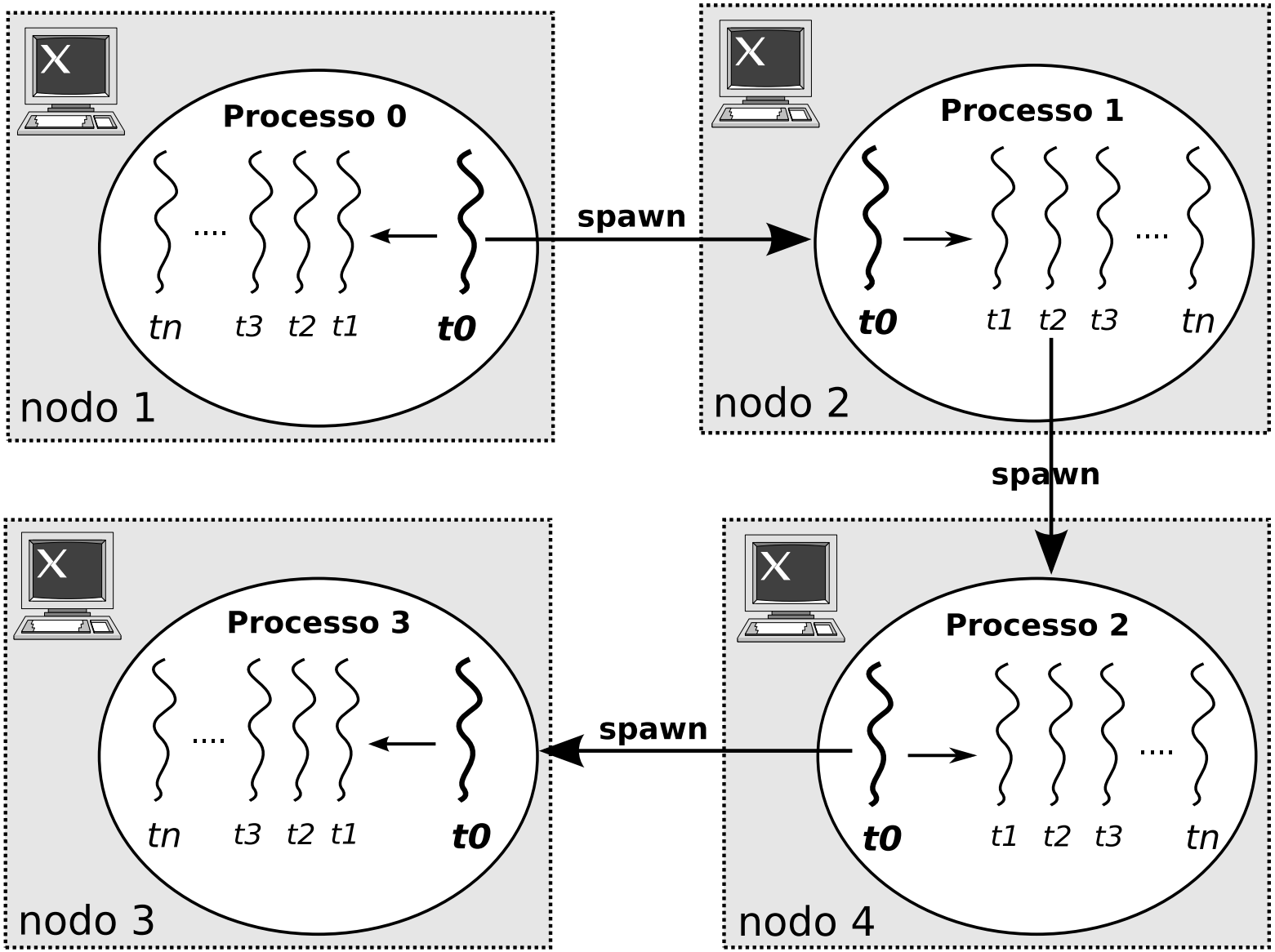
Proposta

Controle de Granularidade (5/6)



Proposta

Controle de Granularidade (6/6)



Proposta

Comunicação de Tarefas

- Soluciona impasses na comunicação com *threads*
- Classes especializadas de acordo com o tipo de tarefa (processo/*thread*)
- Suporta as funções de mensagens:
  - bloqueantes: `Send`/`Recv`
  - não bloqueantes: `Isend`/`Irecv`
- Mensagens são armazenadas no comunicador (`Comm`)
  - duas listas para mensagens `Send` e `Recv`
  - o comunicador controla as mensagens
- Comunicações em memória compartilhada

Proposta

Características de Granularidade

- **Descrição de tarefa**
  - pode ser tanto um processo quanto uma *thread*
- **Localidade**
  - aumenta quando cria novas *threads* localmente
  - controla as comunicações entre tarefas
- **Carga de Trabalho**
  - cada processo verifica a carga de sistema, para decidir entre processo/*thread*
- **Gerenciamento de recursos**
  - depende da implementação MPI utilizada (MPICH2)
  - recursos são os endereços contidos em `hostfile`
- **Criação de tarefas**
  - reduz o custo com a criação de *threads*

Proposta

Limitações

- Um `Spawn` não gera tarefas híbridas (*threads* e processos)
- O `Spawn` não é coletivo (`COMM_SELF`)
- Variáveis globais podem ocasionar inconsistências
- Não suporta comunicações coletivas
- Sem gerenciamento de recursos
- A utilização de outras *threads* não foi testada

Roteiro da apresentação

Resultados - Experimentos

- **Objetivo**: avaliar a proposta em relação a processos
- Quatro *benchmarks* sintéticos:
  - **Fibonacci**
  - **Mergesort**
  - N-Queens
  - Caixeiro Viajante (TSP)
- Plataforma: Grid'5000 (http://www.grid5000.fr)
  - convênio entre GPPD e INRIA como equipe associada
  - *sites* Toulouse, Sophia, Rennes
  - agregados com 2 CPUs Dual Core, 4 *cores* por nó
  - rede Gigabit Ethernet
  - 4GB de memória (Sophia), 8GB nas demais

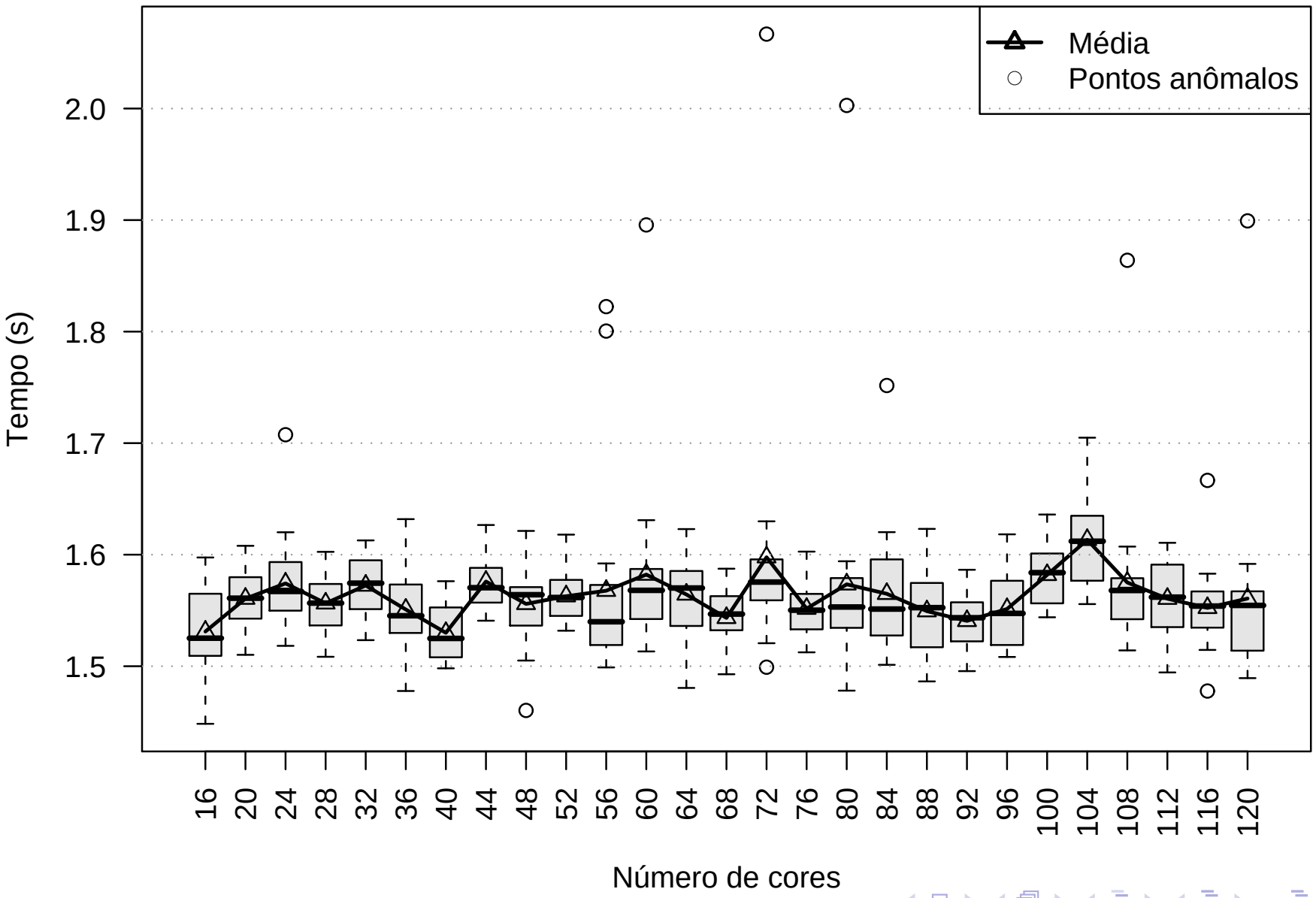
Resultados - Gráficos Utilizados

- Dois gráficos são mostrados por experimento
  - facilitar a visualização devido à diferença de escala
- **Tempos libSpawn** : médias e simetria com *boxplot*
  - *quartile* inferior (Q1) - 25% menores que Q2
  - mediana (Q2)
  - *quartile* superior (Q3) - 25% maiores que Q2
  - *outliers* - resultados anômalos
  - linha inferior - $Q1 - 1,5 * IQR$
  - linha superior - $Q3 + 1,5 * IQR$
- **Tempos Processos X libSpawn**
  - médias da amostra
  - desvio padrão em linha vertical
- Tamanho das amostras: 20 execuções

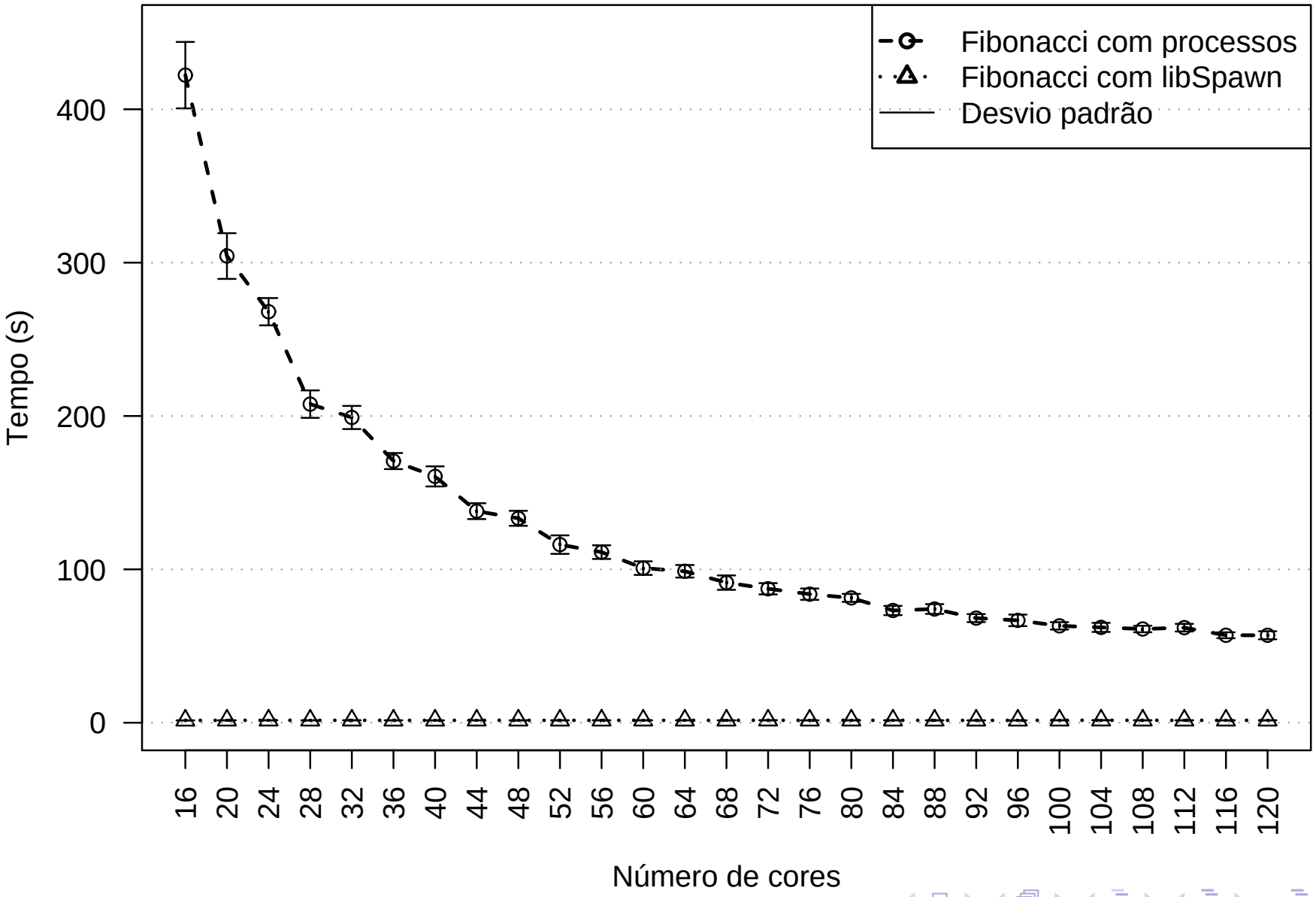
Resultados - Fibonacci

- **Entrada**: 20
- **Recursos**: de 16 a 120 *cores*
- **Tarefas**: 13.529

Resultados - Fibonacci com libSpawn



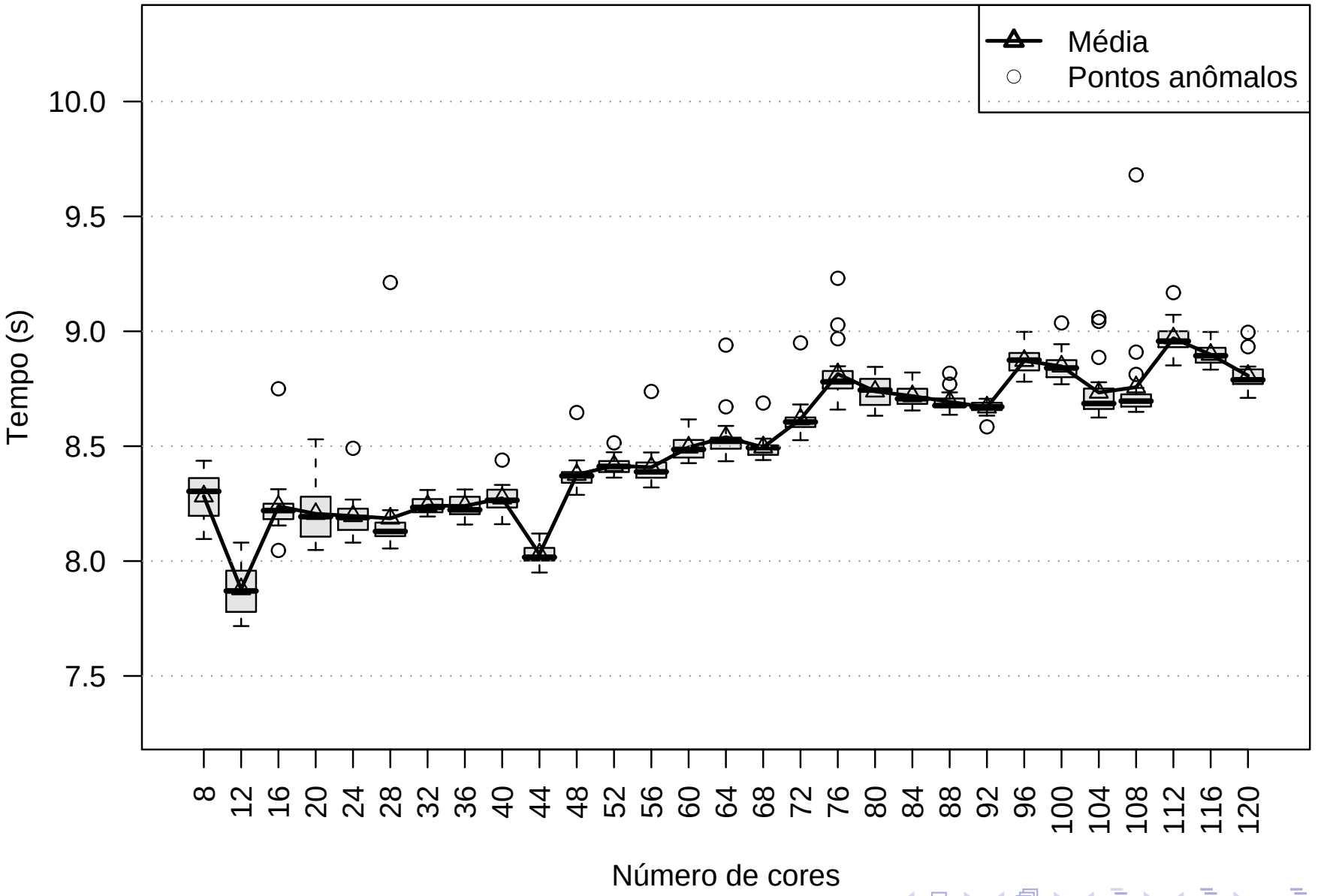
Resultados - Fibonacci processos X libSpawn



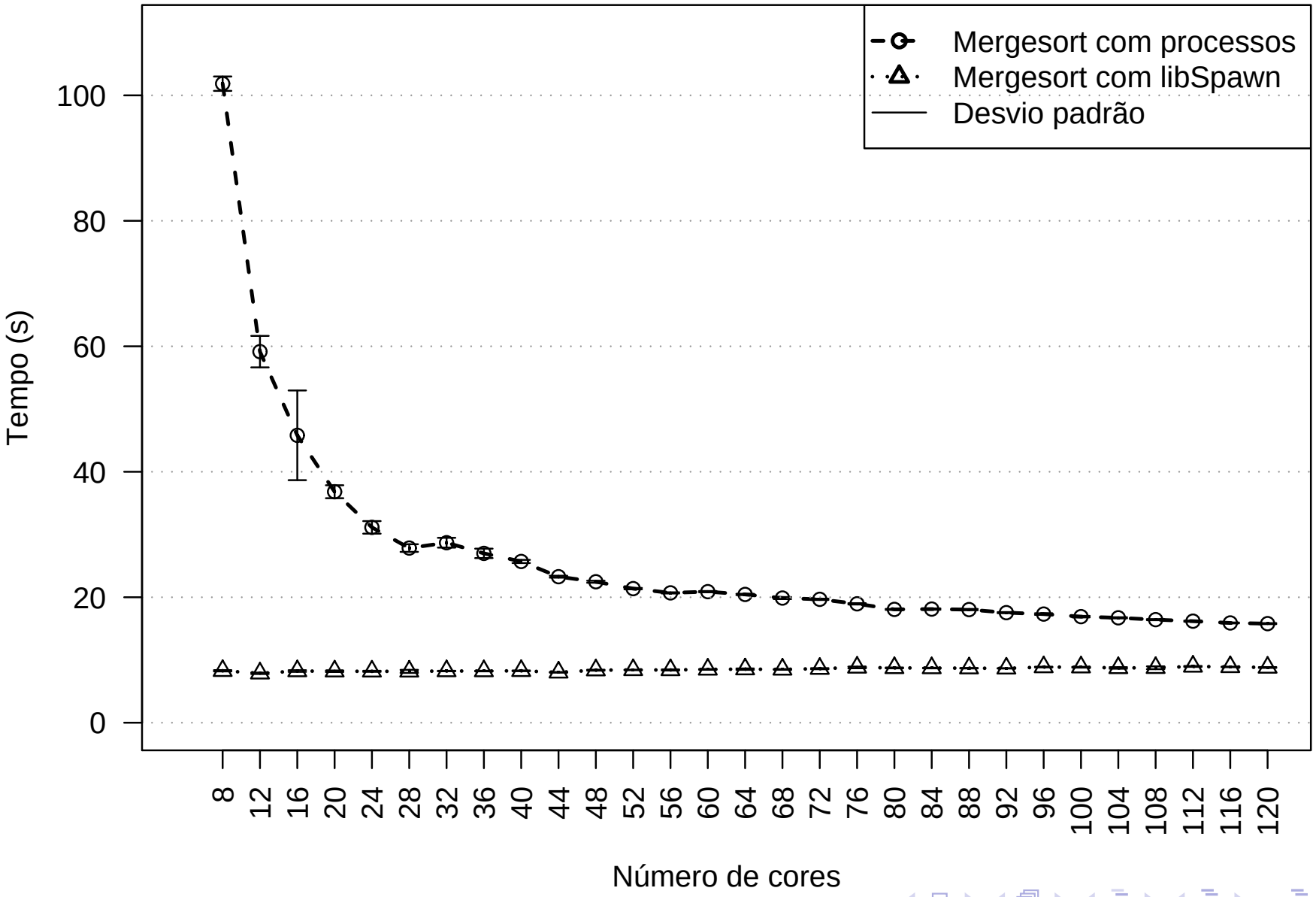
Resultados - Mergesort

- **Entrada**: 2 milhões de números aletórios (1)
- **Recursos**: de 8 a 120 *cores*
- **Tarefas**: 10.812

Resultados - Mergesort com libSpawn



Resultados - Mergesort processos X libSpawn



Resultados - Conclusão

- Os experimentos avaliam os custos, principalmente:
  - criação de tarefas
  - localidade
  - comunicação
- Ganhos significativos devido a:
  - redução em criação de tarefas (*threads*)
  - redução na comunicação
- Observações
  - ganhos menores no Mergesort devido ao formato MPMD
  - tempos anômalos devido à falta de escalonamento de tarefas
  - O pior tempo na libSpawn não é maior que o melhor caso com processos
- Em suma, o controle proporciona ganhos significativos

Roteiro da apresentação

Conclusão

Conclusão

- O trabalho tratou de controlar a granularidade em MPI
- Esse controle se concretiza em uma biblioteca (libSpawn)
- Ganhos significativos com *benchmarks* sintéticos
- Redução na criação de tarefas e comunicação
- Tempos anômalos devido à falta de escalonamento (processo/*thread*)

- Trabalhos futuros:
  - escalonamento em dois níveis (*cores*/ nó)
  - novos experimentos com *benchmarks* dinâmicos
  - experimentos em diferentes plataformas (Grades, *cluster of clusters*)

Referências

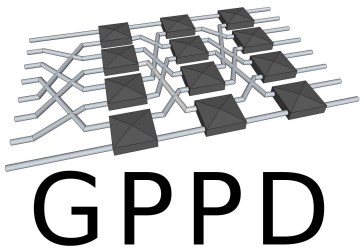
- Forum, The MPI (1995). A Message-Passing Interface Standard. Technical Report CS-94-230, University of Tennessee, Knoxville, Tennessee.
- Forum, The MPI (1997). MPI-2: Extensions to the Message-Passing Interface. Technical Report CDA-9115428, University of Tennessee, Knoxville, Tennessee.
- Grama, A., Gupta, A., Karypis, G., and Kumar, V. (2003). *Introduction to Parallel Computing*. Addison-Wesley, 2th edition.
- Lima, J. V. F., Maillard, N. (2008). Controle de Granularidade com threads em Programas MPI Dinâmicos. *WSCAD'08 - Workshop em Sistemas Computacionais de Alto Desempenho*.
- Pezzi, G. P., Cera, M. C., Mathias, E. N., Maillard, N., and Navaux, P. O. A. (2006). Escalonamento Dinâmico de programas MPI-2 utilizando Divisão e Conquista. *WSCAD'06 - Workshop em Sistemas Computacionais de Alto Desempenho*, 7:71-78.
- Rünger, G. (2006). Parallel Programming Models for Irregular Algorithms. *Lecture Notes in Computational Science and Engineering - Parallel Algorithms and Cluster Computing*, 52:3-23.

Online Mapping of MPI-2 Tasks to Processes and Threads

João Vicente F. Lima
Orientador: Prof. Dr. Nicolas Maillard
{joao.lima,nicolas}@inf.ufrgs.br

Universidade Federal do Rio Grande do Sul (UFRGS)

21 de agosto de 2009



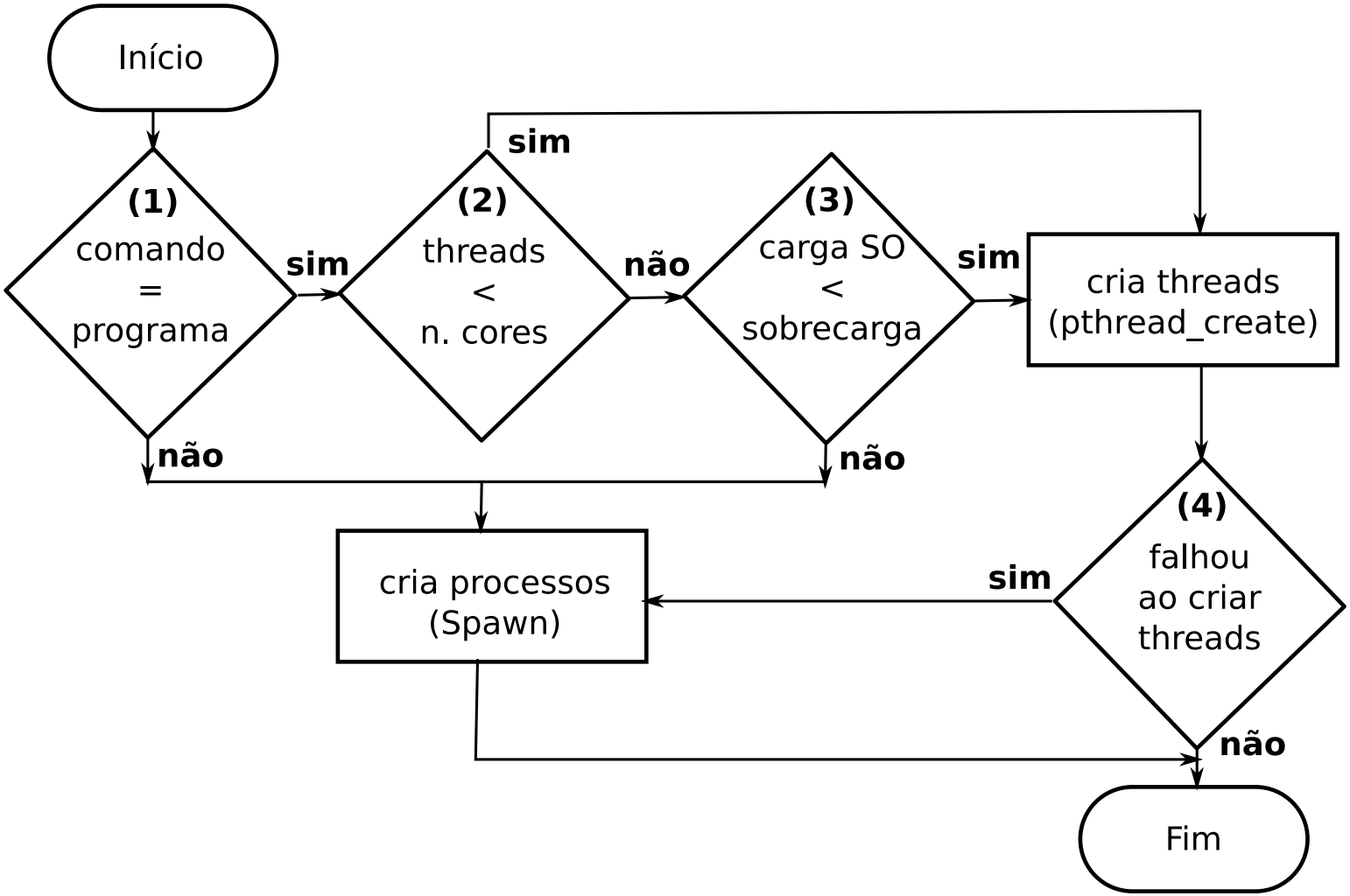
Extra - Estado da Arte

Suporte à Granularidade

- **Descrição de tarefa**
  - MPI é sempre um processo
  - nos outros, a tarefa é abstrata e executa como *thread*
- **Localidade**
  - Satin, KAAPI e AMPI aumentam a localidade
  - MPI depende da implementação ou do programador (`MPI_Comm_spawn`/`pthread_create`)
- **Carga de trabalho**
  - somente OpenMP e MPI dependem da implementação
- **Gerenciamento de recursos**
  - o MPI não especifica, mas permite inclusão por cliente/servidor
- **Criação de tarefas**
  - AMPI não suporta

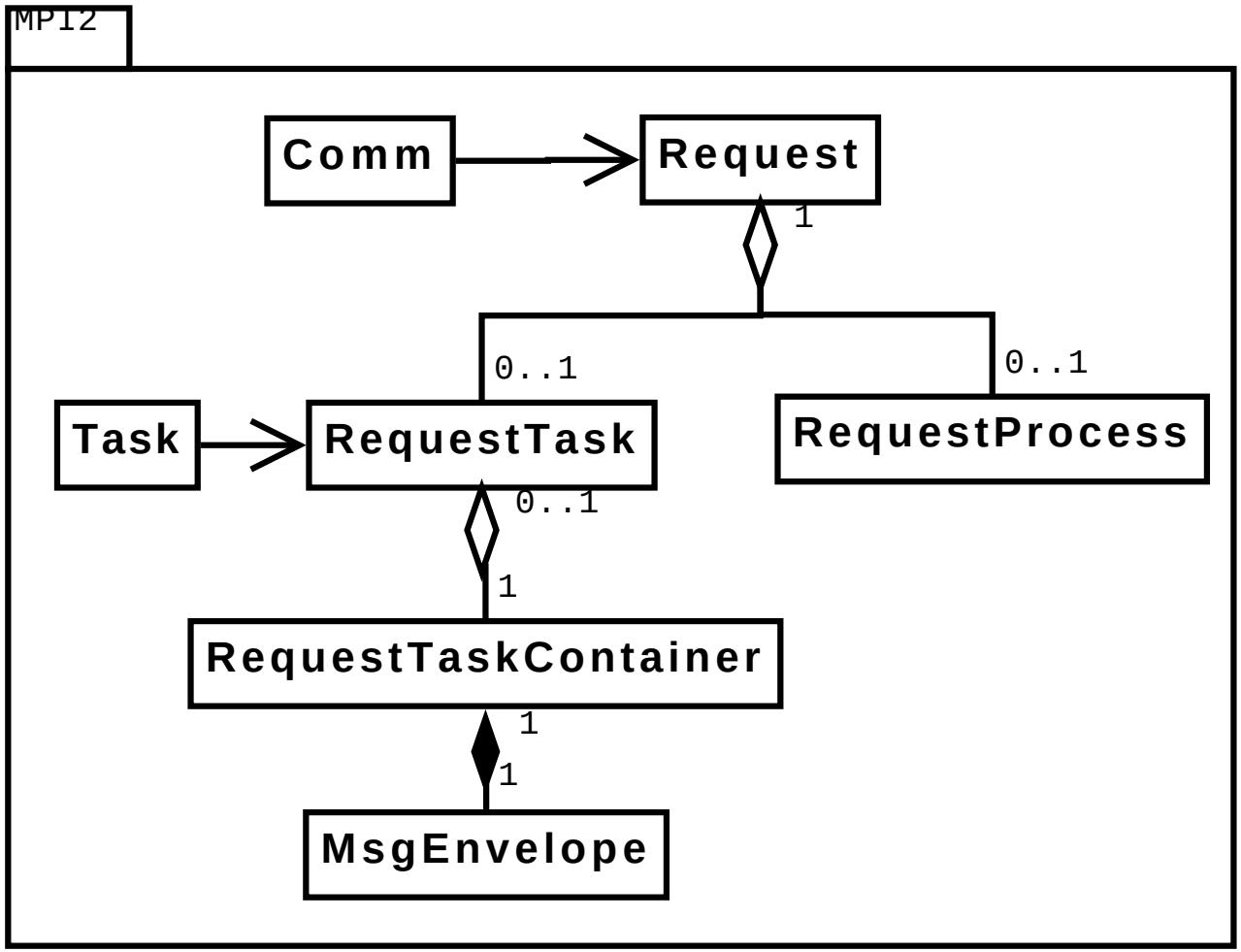
Extra - Proposta

Controle de Granularidade



Extra - Proposta

Comunicação de Tarefas



Extra - Proposta

Comunicação de Tarefas

- **Comm** - armazena as listas de mensagens
- **Request** - pedido de comunicação
- **RequestTask** - métodos de comunicação entre *threads*
- **RequestProcess** - métodos de comunicação entre processos
- **RequestTaskContainer** - representa uma mensagem
- **MsgEnvelope** - descreve a identificação da mensagem

Extra - Proposta

Comunicação de Tarefas

- Envio de mensagem bloqueante em `Send`

MPI2::Comm.Send(...)

```
1: void Send(...) {
2:   Request *request;
3:   if (isCommThreads())
4:     request = new RequestTask(...);
5:   else
6:     request = new RequestProcess(...);
7:   request.Wait();
8: }
```

Extra - Proposta

Comunicação de Tarefas

- Envio de mensagem bloqueante em `Send`

MPI2::Comm.Send(...)

```
1: void Send(...) {
2:   Request *request;
3:   if (isCommThreads())
4:     request = new RequestTask(...);
5:   else
6:     request = new RequestProcess(...);
7:   request.Wait();
8: }
```

Extra - Proposta

Comunicação de Tarefas

- Envio de mensagem bloqueante em `Send`

MPI2::Comm.Send(...)

```
1: void Send(...) {
2:   Request *request;
3:   if (isCommThreads())
4:     request = new RequestTask(...);
5:   else
6:     request = new RequestProcess(...);
7:   request.Wait();
8: }
```

Extra - Proposta

Comunicação de Tarefas

- Envio de mensagem bloqueante em `Send`

MPI2::Comm.Send(...)

```
1: void Send(...) {
2:   Request *request;
3:   if (isCommThreads())
4:     request = new RequestTask(...);
5:   else
6:     request = new RequestProcess(...);
7:   request.Wait();
8: }
```

Extra - Proposta

Comunicação de Tarefas

- Envio de mensagem bloqueante em `Send`

MPI2::Comm.Send(...)

```
1: void Send(...) {
2:   Request *request;
3:   if (isCommThreads())
4:     request = new RequestTask(...);
5:   else
6:     request = new RequestProcess(...);
7:   request.Wait();
8: }
```

Extra - Proposta

Comunicação de Tarefas

- Envio de mensagem não bloqueante em `Isend`

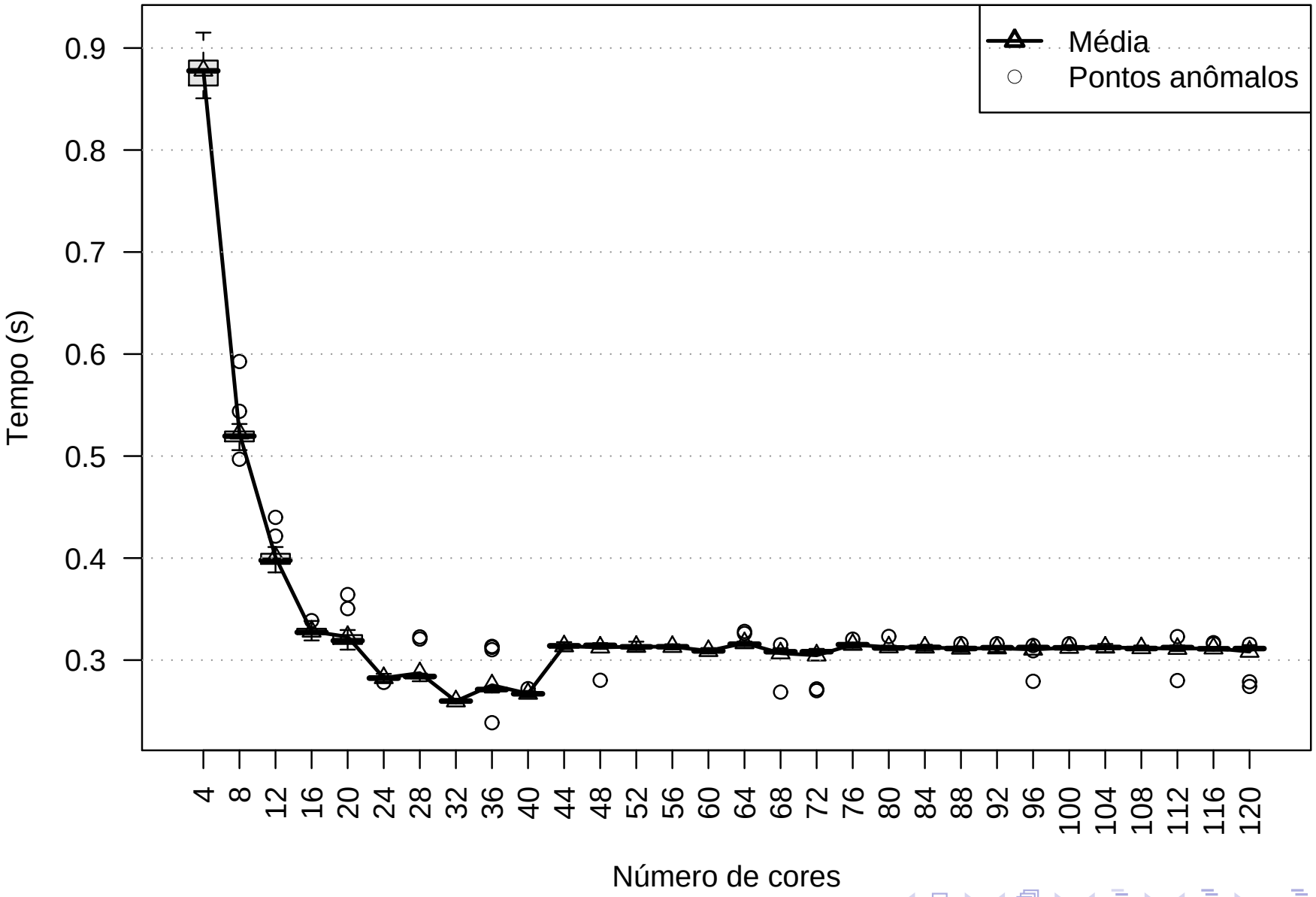
MPI2::Comm.Isend(...)

```
1: void Isend(...) {
2:   Request *request;
3:   if (isCommThreads())
4:     request = new RequestTask(...);
5:   else
6:     request = new RequestProcess(...);
7:   return *request;
8: }
```

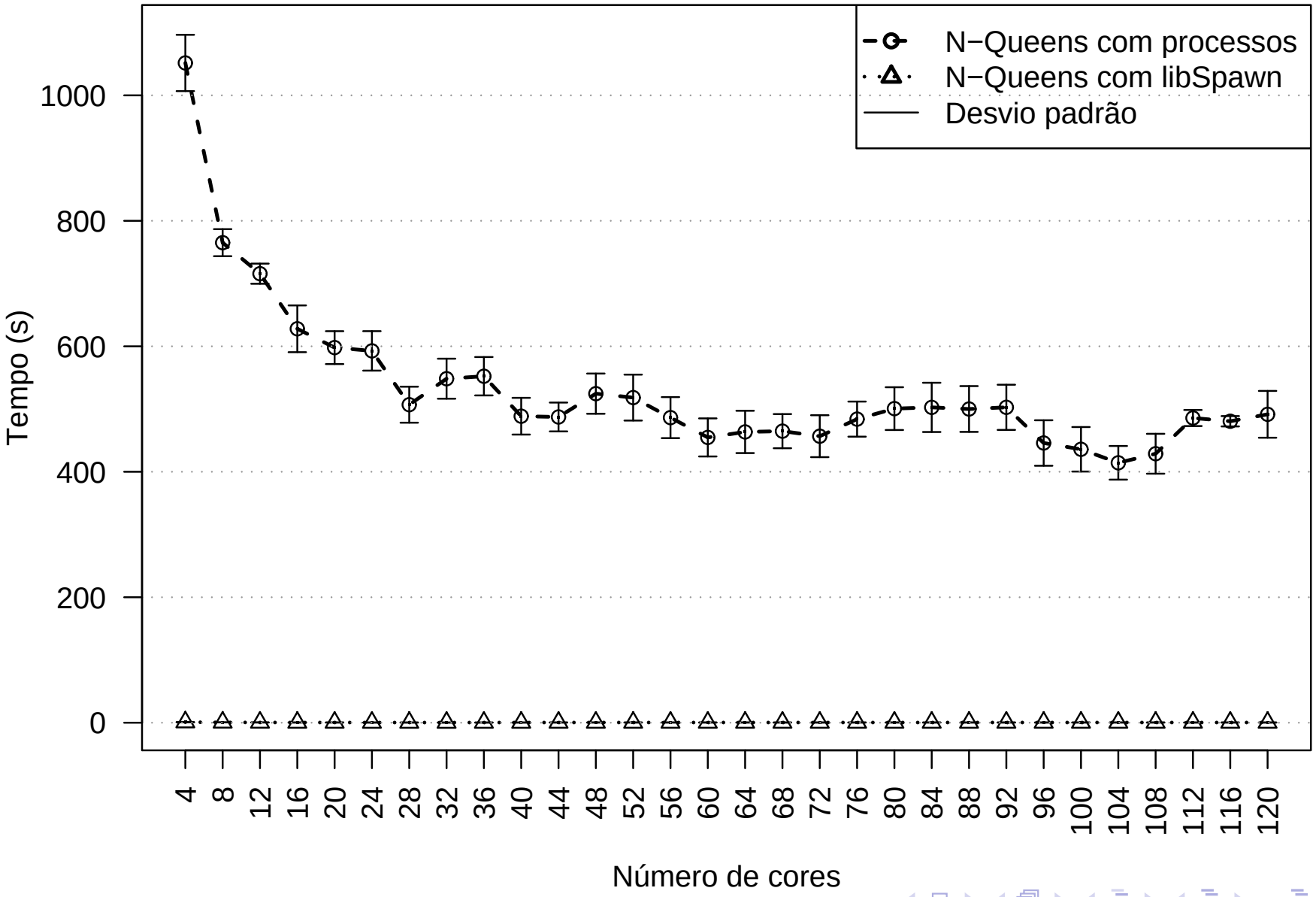
Extra - N-Queens

- **Entrada**: tabuleiro de 10 X 10
- **Recursos**: de 4 a 120 *cores*
- **Tarefas**: 34.814

Extra - N-Queens com libSpawn



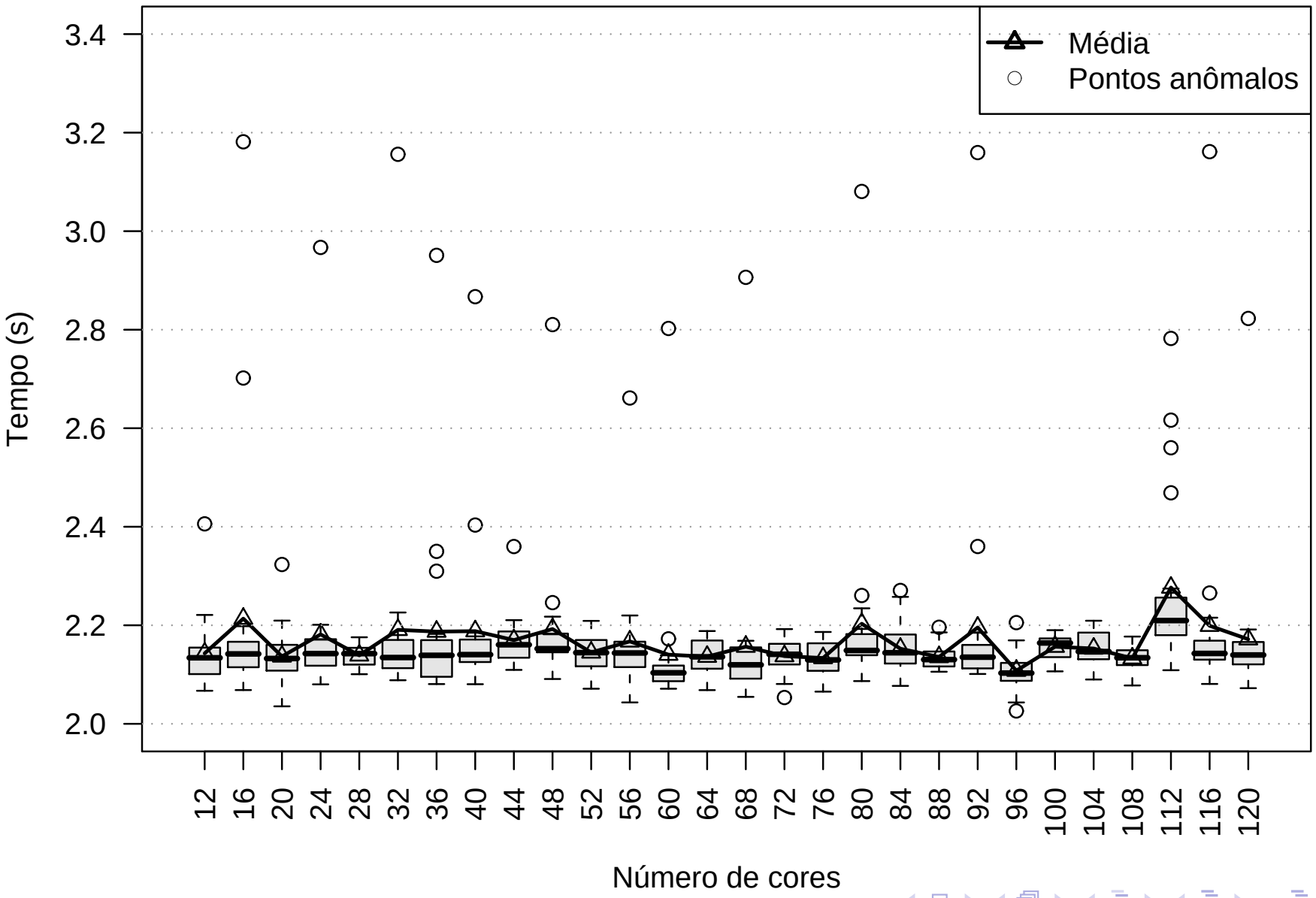
Extra - N-Queens processos X libSpawn



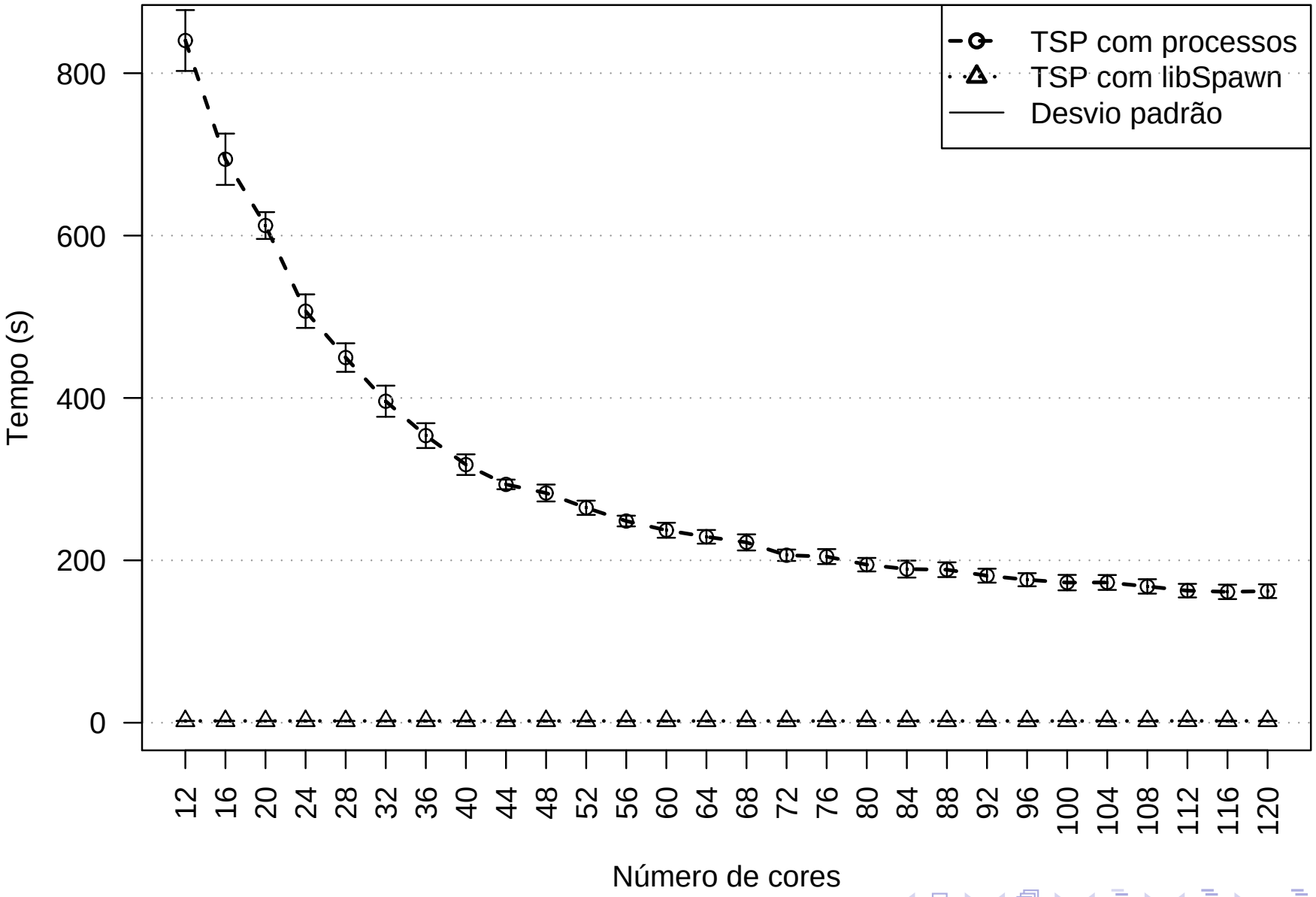
Extra - TSP

- **Entrada**: 10 cidades
- **Recursos**: de 12 a 120 *cores*
- **Tarefas**: 31

Extra - TSP com libSpawn



Extra - TSP processos X libSpawn



Online Mapping of MPI-2 Tasks to Processes and Threads

João Vicente F. Lima
Orientador: Prof. Dr. Nicolas Maillard
{joao.lima,nicolas}@inf.ufrgs.br

Universidade Federal do Rio Grande do Sul (UFRGS)

21 de agosto de 2009