niversidade Federal do Rio Grande do Sul

PARALELISMO NA LINGUAGEM HASKELL

Vagner Franco Pereira
Rodrigo Machado
Lucas Mello Schnorr

INTRODUÇÃO (Objetivos)

- Linguagens funcionais puras possuem características interessantes (como transparência referencial) que podem ser úteis para o desenvolvimento de algoritmos paralelos.

- Este trabalho visa ser um estudo de caso da aplicabilidade da linguagem funcional pura Haskell para computação paralela.

- Para tanto, propomos
  - Implementação de algoritmos sequenciais e paralelos em Haskell
  - Análise do esforço de paralelização e do desempenho obtido pelos algoritmos paralelos em Haskell
  - Comparação (do desempenho) de duas

Linguagens Funcionais (Haskell)

- Linguagem Funcional Pura
- Memória Gerenciada - Coletor de lixo
- Avaliação preguiçosa (*Lazy evaluation*)
- Mônadas

PARALELISMO NA LINGUAGEM HASKELL

1 Control.Parallel.Strategies

- Interface para desenvolvedor expressar paralelismo
- Principais estratégias : *rpar , rseq*
- Criação de sparks para gerar paralelismo
- Gerencia implicitamente a dependência de dados

1 Control.Parallel.Strategies

```
runEval $ do
   a <- rpar (f x)
   b <- rseq (f y)
   rseq a
   return (a,b)
```

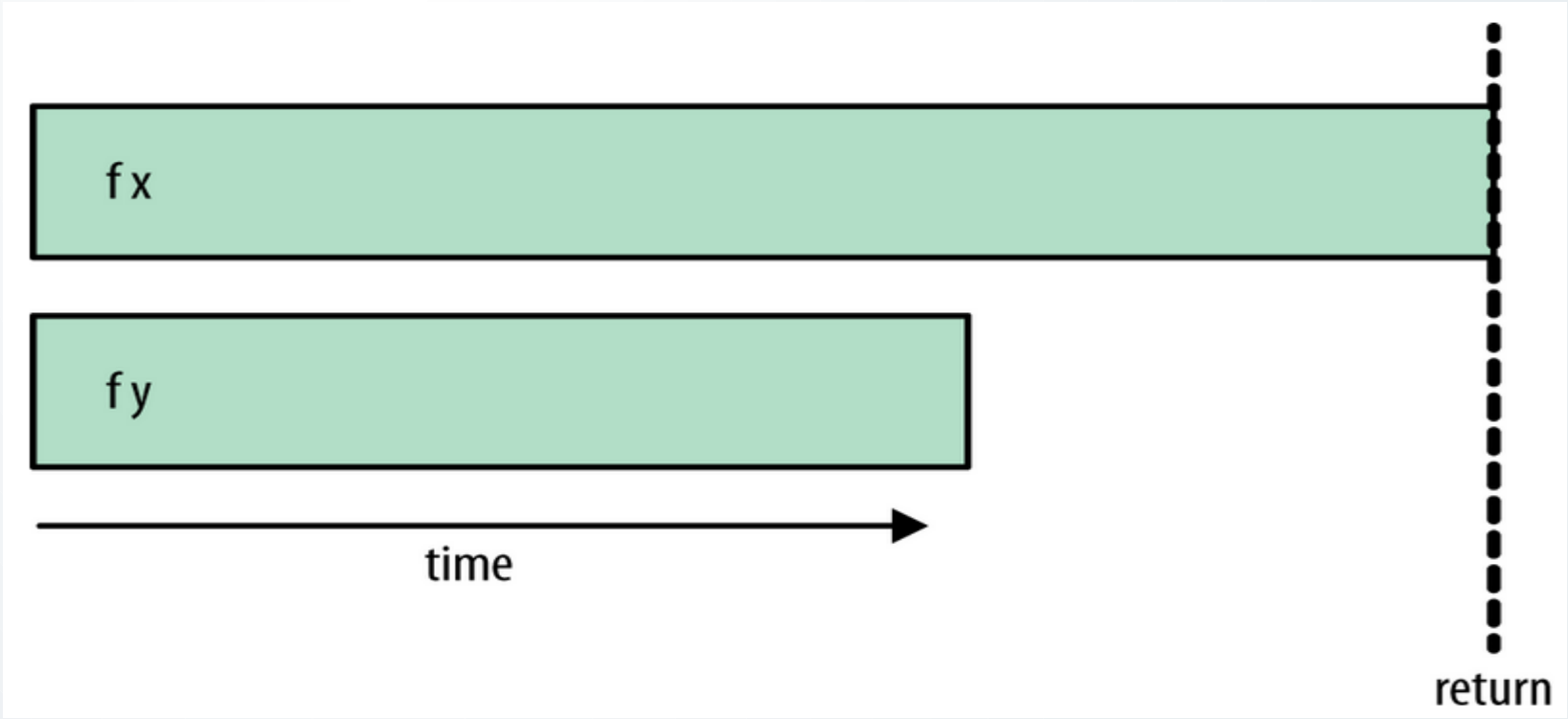


Imagem retirada do livro: Parallel and Concurrent

4 Metodologia

- **Cálculo do fractal de Mandelbrot** : implementação sequencial e paralela em Haskell e em Ansi C

- **Simulação n-Corpos gravitacional** : implementação paralela em Haskell do algoritmo quadrático e do Barnes Hut

4 Metodologia

- **Ambiente de execução**:
  A máquina beagle com dois processadores de oito núcleos em cada um deles. Frequência de 2000 MHz

- **Compiladores utilizados**: GHC, GCC

- **Número de réplicas nos experimentos**: 50

Fractal de Mandelbrot

```
iteracaoPontoMandelbrot::Iteracao->NroComplexo->NroComplexo->Pixel
iteracaoPontoMandelbrot  0 complexo complexoIterado=  (realPart (complexo),imagPart (complexo),0)
iteracaoPontoMandelbrot iterador complexo complexoIterado
                | magnitude complexoIterado > 2 =(realPart (complexo),imagPart (complexo),iterador)
                | otherwise = iteracaoPontoMandelbrot(iterador-1) complexo  zn
                where
                        zn = (complexoIterado^2)+(complexo)
```

Código Sequencial

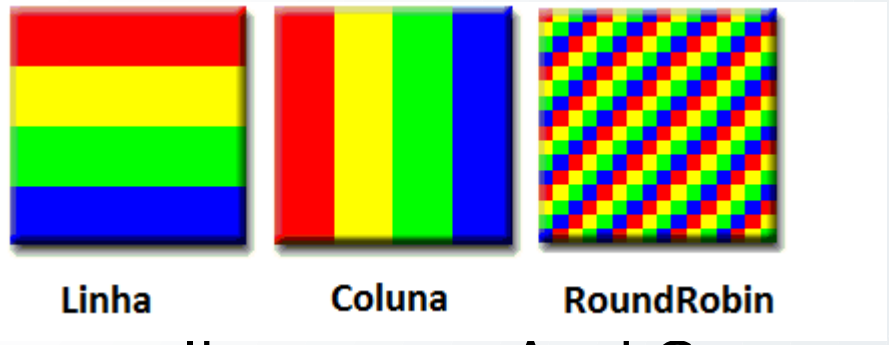
```
mandelbrot l = map(\pixel->iteracaoMandelbrot numeroIteracaoes pixel pixel ) l
```

Código

```
mandelbrot :: [[NroComplexo]]->Iteracao->Eval [Pixel]
mandelbrot [] _  =return []
mandelbrot (listaPonto:plano) numeroIteracaoes = do
        p1 <-rdeepPar(map(\pixel->iteracaoPontoMandelbrot numeroIteracaoes pixel pixel ) listaPonto)
        p2 <- mandelbrot plano numeroIteracaoes
        return (p1++p2)
```

5 Mandelbrot

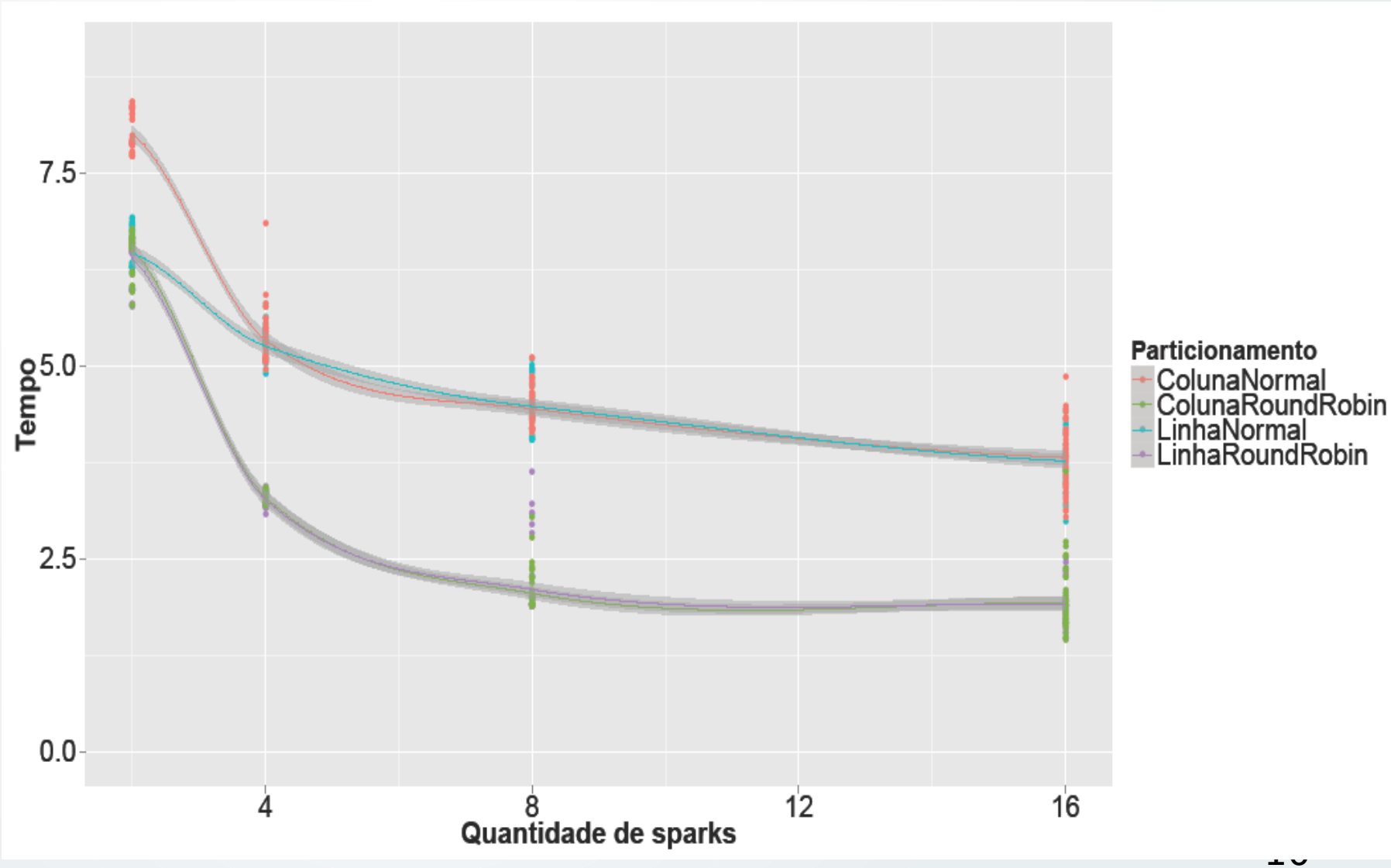
- Efeito das diferentes formas de particionamento no tempo de cálculo do fractal de Mandelbrot



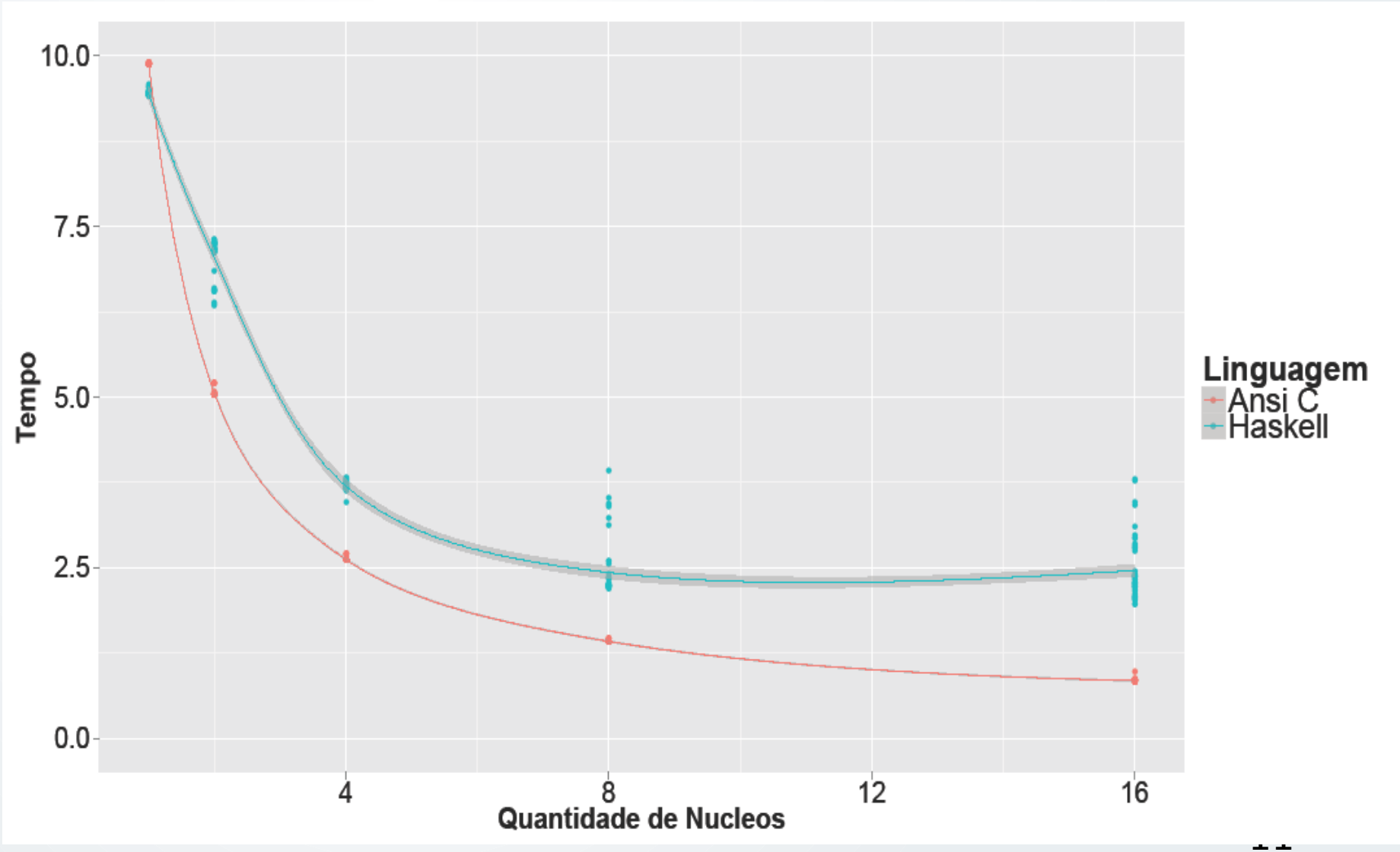
- Comparação com a linguagem Ansi C

- Para os resultados a quantidade de números complexos foi fixada em 4194304

Mandelbrot-Resultados



Mandelbrot-Resultados



imulação n-Corpos Gravitacional

- Simula a movimentação de um conjunto de corpos no espaço sob influência da gravidade mútua
- Apresenta dependência de dados
- Força gravitacional:

$$F = G\frac{m_1 m_2}{r^2}$$

mulação n-Corpos Gravitacional quadrático

- Complexidade computacional: $O(n^2)$
- Paralelismo é feito particionando-se a lista de corpos
- Particionamento é trivial uma vez que o tempo de cálculo das forças é teoricamente igual em todos os corpos

6 Barnes-Hut

- Complexidade computacional: $O(nlogn)$
- Paralelismo é realizado em duas etapas: particionando a lista e na criação da árvore
- Particionamento da lista é trivial
- Particionamento na criação da árvore é dinâmico, pois não se sabe quantos corpos pertencem a cada nodo da árvore

6 Barnes-Hut

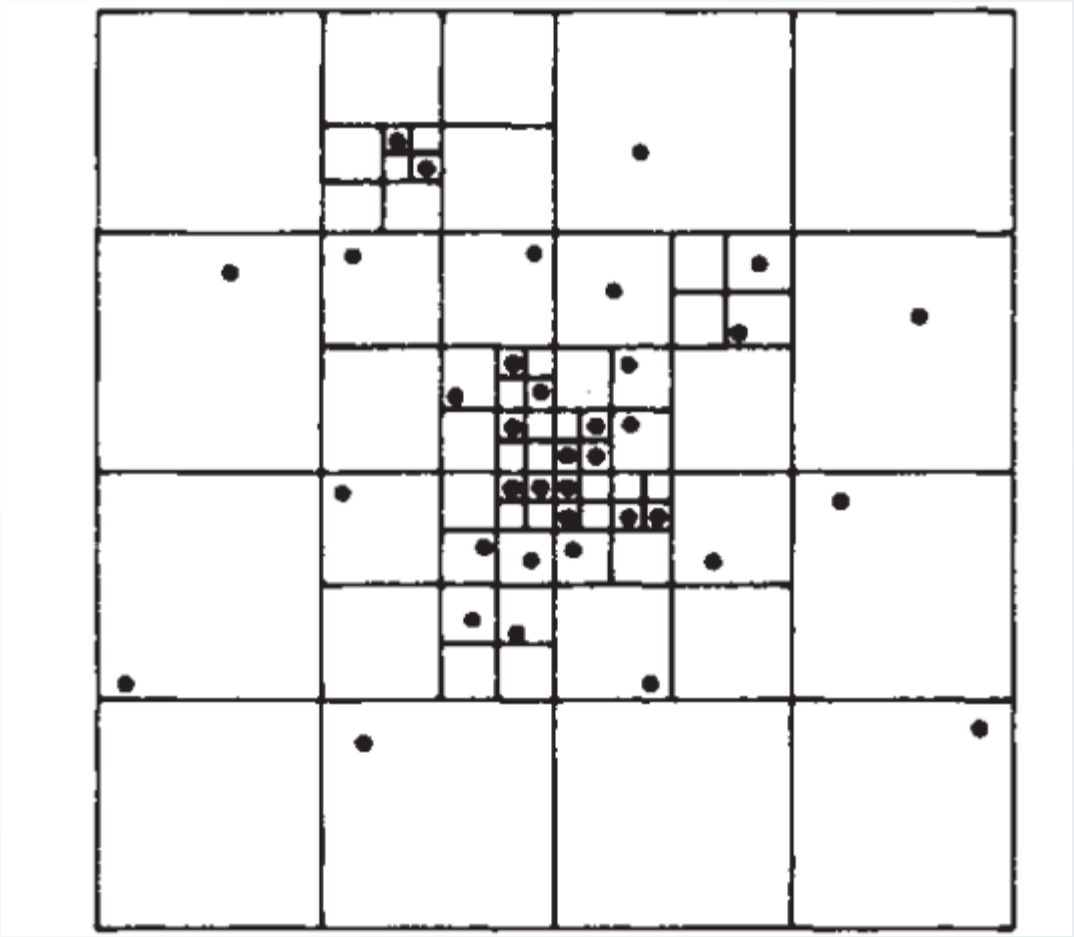


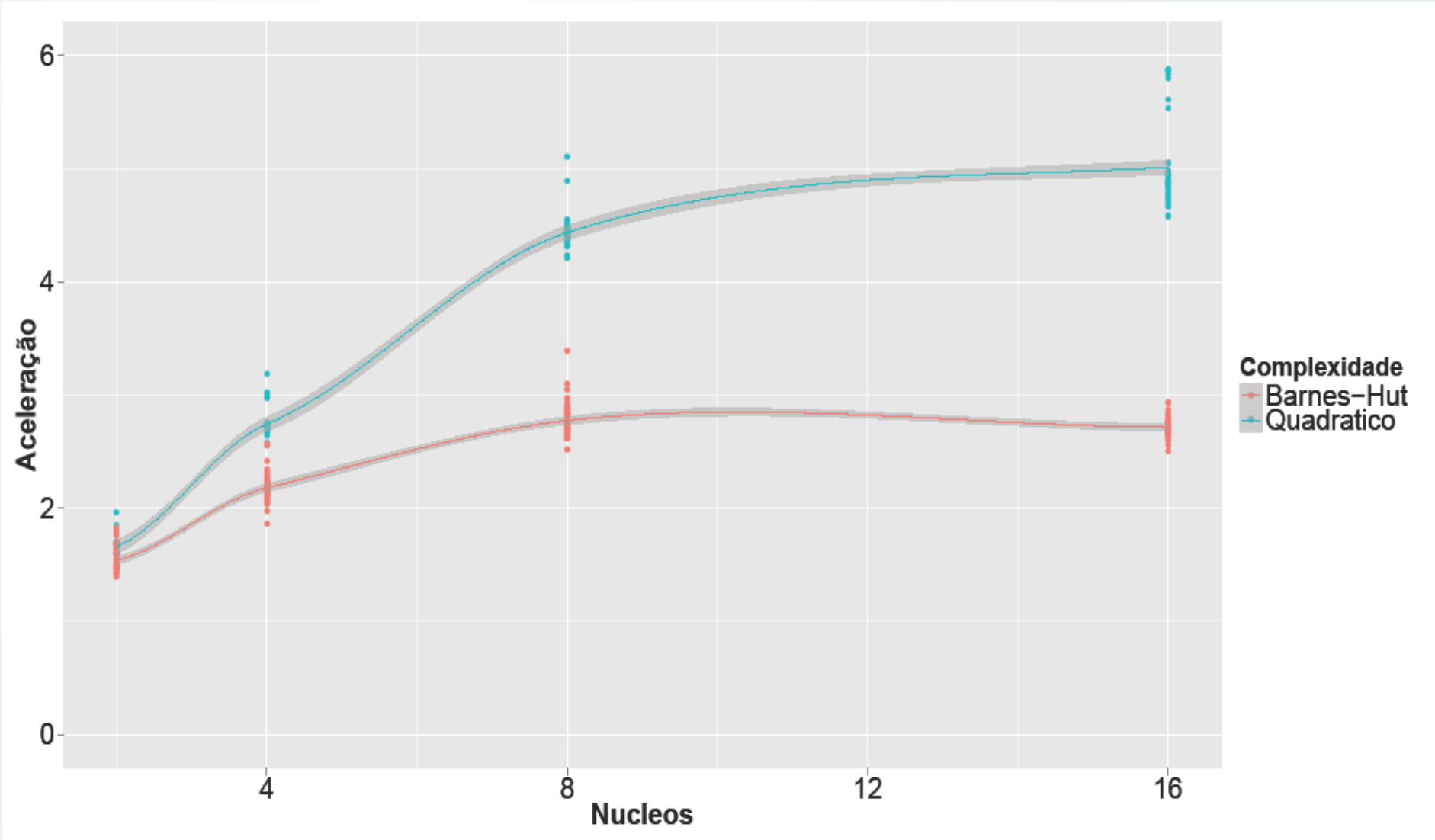
Imagem retirada do artigo: A hierarchical O(N log N) force-calculation algorithm

imulação n-Corpos Gravitacional

Número de corpos = 600
Número de passos = 100
Cada passo representa 60 minutos

Tempos Médios	Núcleos				
	1	2	4	8	16
Quadrático	26.35s	15.92s	9.61s	5.94s	5.22s
Barnes-Hut	0.85s	0.55s	0.39s	0.31s	0.30s

Simulação n-Corpos Gravitacional



7 Conclusão

- Haskell oferece abstrações de alto nível que simplificam paralelismo
- A biblioteca Strategies abstrai do programador a necessidade de controle da dependência de dados
- Independente da linguagem utilizada, é necessário pensar no particionamento e granularidade das tarefas
- O gerenciamento de memória acaba sendo um fator significativo no desempenho

7 Conclusão

- Haskell tem tempos superiores a Ansi C, porém a diferença não é muito acentuada
- Haskell oferece diversas bibliotecas para paralelismo (que não foram exploradas em profundidade neste trabalho)
- Somente paralelizar um algoritmo ruim pode não ser suficiente (caso da simulação n-corpos).