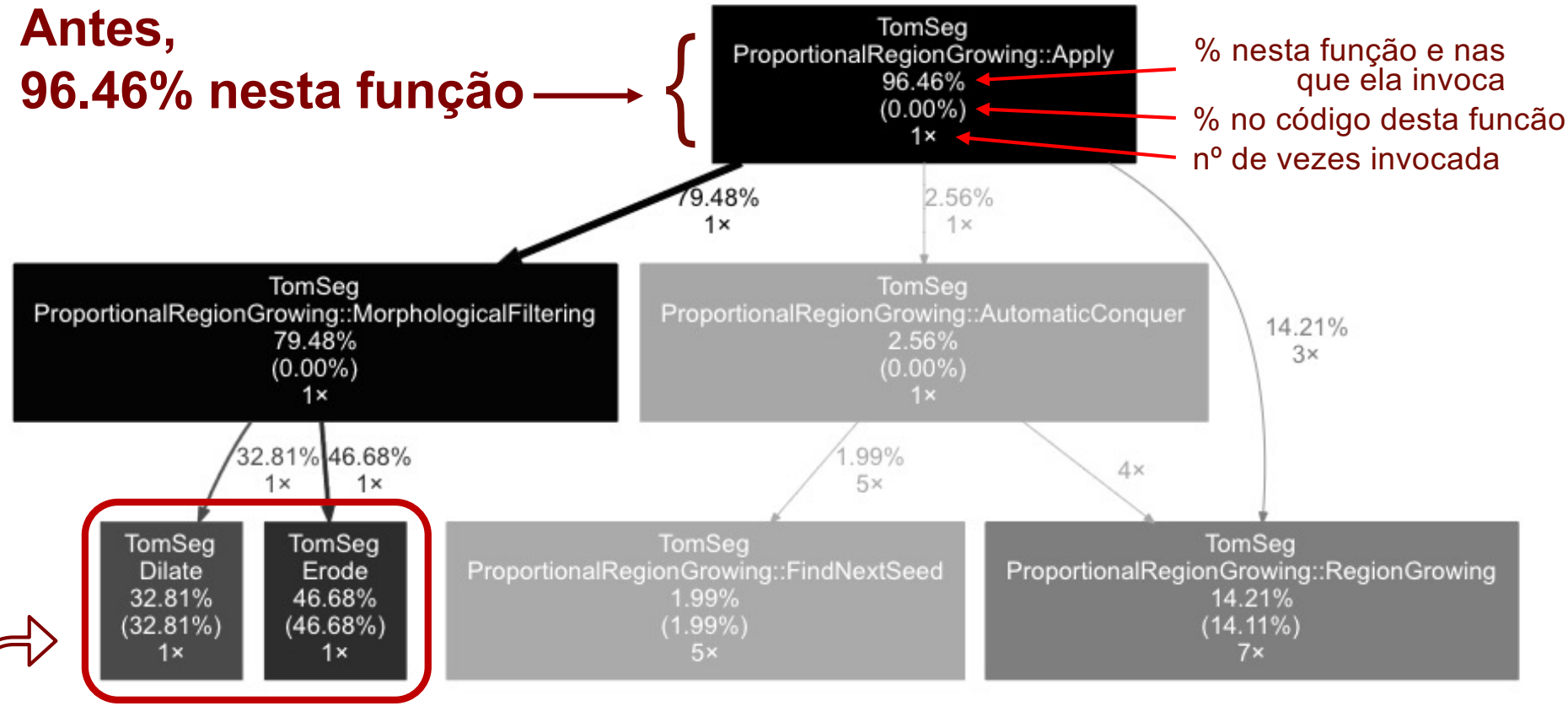
Estrutura do tema Avaliação de Desempenho (IA-32)

1. A avaliação de sistemas de computação
2. Técnicas de otimização de código (IM)
3. Técnicas de otimização de *hardware*
4. Técnicas de otimização de código (DM)
5. Outras técnicas de otimização
6. Medição de tempos

Análise de técnicas de otimização (*s/w*)

- técnicas de otimização de código (independ. máquina)
  - *já visto...*
- técnicas de otimização de código (depend. máquina)
  - dependentes do processador *(já visto...)*
  - dependentes da hierarquia da memória *(a ver numa próxima UC)*
    - a localidade espacial e temporal dum programa *(já visto...)*
    - quantificação da influência da *cache* no desempenho
- **outras técnicas de otimização**
  - na compilação: otimizações efetuadas pelo `Gcc`
  - na identificação dos "gargalos" de desempenho
    - *code profiling*
    - uso dum *profiler* para apoio à otimização
- **lei de Amdahl**

Code profiling: análise visual da melhoria de código duma função (antes)

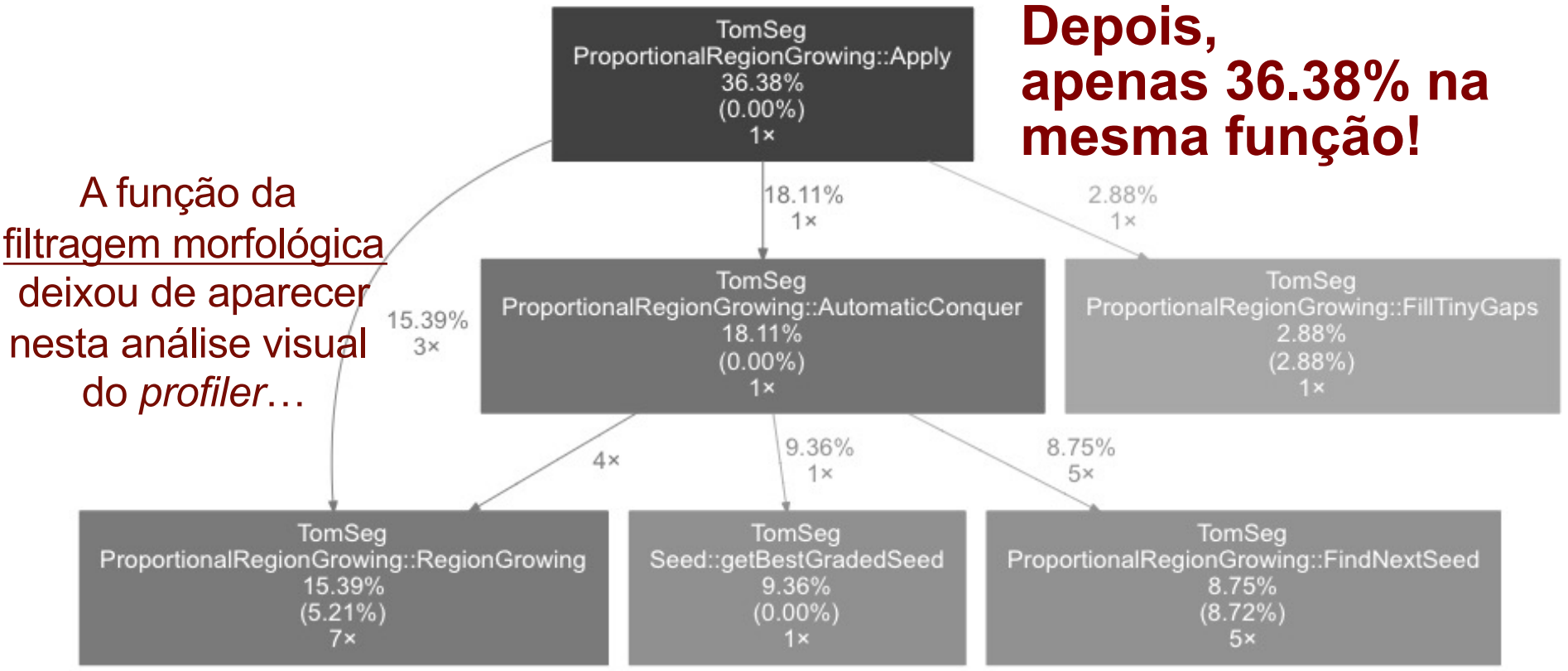


Quase 80% do tempo total é gasto nestas 2 funções da filtragem morfológica!

Conclusão: é aqui que se deve investir para melhorar a *performance* global

Figure 5.7.: Call-graph of the first version of *Propor. Region Growing* (DS3)

Code profiling: análise visual da melhoria de código duma função (depois)



Depois, apenas 36.38% na mesma função!

A função da filtragem morfológica deixou de aparecer nesta análise visual do *profiler*…

Figure 5.9.: Call-graph from the last version of *Propor. Region Growing* (DS3)

Code profiling e visualização: dicas para a sua realização

Mais comum para aplicações em ambiente GNU/Linux:

- `gprof` : disponível em `gcc/binutils`
- `gprof` requer compilação com `-pg` para gerar o ficheiro `gmon.out` contendo os dados para o *profiler*
- `gprof` lê a informação em `gmon.out` e produz um relatório em `main.gprof`
- `gprof2dot` lê `main.gprof` e gera uma representação visual dos dados tal como apresentado nos slides anteriores
- `gprof2dot` disponível em https://github.com/jrfonseca/gprof2dot
- para visualizar a imagem do *call graph* usar o comando `xdot`

https://developer.ridgerun.com/wiki/index.php/Profiling_GNU/Linux_applications

Lei de Amdahl

O ganho no desempenho – *speedup* –
obtido com a melhoria do tempo de execução de uma parte do sistema,
está limitado pela fração de tempo que essa parte do sistema pode ser usada.

$$Speedup_{overall} = \frac{Tempo\_exec_{antigo}}{Tempo\_exec_{novo}} = \frac{1}{\Sigma\ (f_i\ /\ s_i)}$$

f_i - fracções com melhoria s_i
s_i - *speedup* de cada fracção

Ex.1: Se **10%** de um programa executa **90x** mais rápido

***Overall speedup* = 1.11**

Ex.2: Se **90%** de um prog executa **90x** mais rápido

***Overall speedup* = 9.09**

Paralelismo:
se N_{proc} Ξ ***speedup***, trocar s_i por N_{proc}