



# Multithreaded Algorithms

Thomas Bamelis

KU Leuven Kulak

Academiejaar 2017-2018

# Overzicht

Inleiding

Basis van multithreading

Meeteenheden prestatie

Scheduling van threads

Analyseren van een algoritme

Matrix vermenigvuldiging

Merge sort

Besluit

# Overzicht

Inleiding

Basis van multithreading

Meeteenheden prestatie

Scheduling van threads

Analyseren van een algoritme

Matrix vermenigvuldiging

Merge sort

Besluit

# Multiprocessoren en threads

## Multiprocessors

- ▶ Meerder processors en/of cores per processor
- ▶ Meerdere instructies simultaan

## Threading

- ▶ Apart (parallel) uitgevoerd
- ▶ Heeft : ID, PC, registers en stack
- ▶ Deelt : code- en data sections en resources (e.g. file)

# Multiprocessoren en threads

## Multiprocessors

- ▶ Meerder processors en/of cores per processor
- ▶ Meerdere instructies simultaan

## Threading

- ▶ Apart (parallel) uitgevoerd
- ▶ Heeft : ID, PC, registers en stack
- ▶ Deelt : code- en data sections en resources (e.g. file)

# Multiprocessoren en threads

## **Multiprocessors**

- ▶ Meerder processors en/of cores per processor
- ▶ Meerdere instructies simultaan

## Threading

- ▶ Apart (parallel) uitgevoerd
- ▶ Heeft : ID, PC, registers en stack
- ▶ Deelt : code- en data sections en resources (e.g. file)

# Multiprocessoren en threads

## Multiprocessors

- ▶ Meerder processors en/of cores per processor
- ▶ Meerdere instructies simultaan

## Threading

- ▶ Apart (parallel) uitgevoerd
- ▶ Heeft : ID, PC, registers en stack
- ▶ Deelt : code- en data sections en resources (e.g. file)

# Multiprocessoren en threads

## Multiprocessors

- ▶ Meerder processors en/of cores per processor
- ▶ Meerdere instructies simultaan

## Threading

- ▶ Apart (parallel) uitgevoerd
- ▶ Heeft : ID, PC, registers en stack
- ▶ Deelt : code- en data sections en resources (e.g. file)



# Dynamic threading

Toegankelijke vorm van threading met enorm potentieel.  
Scheduler beslist hoeveel threads wanneer.  
IPV thread maken, thread suggereren.

**Nested parallelism** Een thread kan andere threads oproepen

**Parallel loop** Iedere iteratie in een for loop voert tegelijk uit

# Overzicht

Inleiding

Basis van multithreading

Meeteenheden prestatie

Scheduling van threads

Analyseren van een algoritme

Matrix vermenigvuldiging

Merge sort

Besluit

# Voorbeeld

Voorbeeld m.b.v. (slechte) recursieve Fibonacci (  $\Theta(F_n)$  )

Fib(n)

$F_0 = 0$   
 $F_1 = 1$   
 $F_i = F_{i-1} + F_{i-2}$  als  $i \geq 2$

```
1
2  if n <= 1
3      return n
4  else
5      x = Fib(n-1)
6      y = Fib(n-2)
7      return x + y
8
9
```

# Keywords

**spawn** Geeft aan dat de subroutine parallel kan worden uitgevoerd. Nested-parallism mogelijk (child kan andere threads oproepen).

**sync** Wachten tot alle children voltooiën (impliciet in iedere return)

# Keywords

**spawn** Geeft aan dat de subroutine parallel kan worden uitgevoerd. Nested-parallism mogelijk (child kan andere threads oproepen).

**sync** Wachten tot alle children voltooiën (impliciet in iedere return)

# Keywords

**spawn** Geeft aan dat de subroutine parallel kan worden uitgevoerd. Nested-parallism mogelijk (child kan andere threads oproepen).

**sync** Wachten tot alle children voltooiën (impliciet in iedere return)

# Parallel voorbeeld

## ‘Logical parallelism’

subroutine *kan*  
parallel uitvoeren

**Serialization** Threading  
keywoorden  
weglaten geeft  
sequentieel  
algoritme

P-Fib(n)

```
1
2 if n <= 1
3   return n
4 else
5   x = spawn P-Fib(n-1)
6   y = P-Fib(n-2)
7   sync
8   return x + y
```

# Voorstelling multithreaded algoritme

Gerichte kringloze graaf  $G(V,E)$   
(‘Computation dag’)

- ▶  $V$  de verzameling instructies (of strands)
- ▶  $E$  met  $(u,v) \in E$ :  *$u$  moet voor  $v$  uitvoeren.*

*Strand* stuk zonder parallelle keywords

Strands  $u$  en  $v$  ‘*in serie*’ indien direct pad  $(u,v) \in E$ ,  
anders *in parallel*



# Voorstelling multithreaded algoritme

Gerichte kringloze graaf  $G(V,E)$   
(‘Computation dag’)

- ▶  $V$  de verzameling instructies (of strands)
- ▶  $E$  met  $(u,v) \in E$ :  *$u$  moet voor  $v$  uitvoeren.*

*Strand* stuk zonder parallelle keywords

Strands  $u$  en  $v$  ‘*in serie*’ indien direct pad  $(u,v) \in E$ ,  
anders *in parallel*

# Voorstelling multithreaded algoritme

Gerichte kringloze graaf  $G(V,E)$   
(‘Computation dag’)

- ▶  $V$  de verzameling instructies (of strands)
- ▶  $E$  met  $(u,v) \in E$ :  *$u$  moet voor  $v$  uitvoeren.*

*Strand* stuk zonder parallelle keywords

Strands  $u$  en  $v$  ‘*in serie*’ indien direct pad  $(u,v) \in E$ ,  
anders *in parallel*

# Voorstelling multithreaded algoritme

Gerichte kringloze graaf  $G(V,E)$   
(‘Computation dag’)

- ▶  $V$  de verzameling instructies (of strands)
- ▶  $E$  met  $(u,v) \in E$ :  *$u$  moet voor  $v$  uitvoeren.*

*Strand* stuk zonder parallelle keywords

Strands  $u$  en  $v$  ‘*in serie*’ indien direct pad  $(u,v) \in E$ ,  
anders *in parallel*

# Soorten bogen

**Continuation boog**  $(u, u')$  Strand  $u$  die (in dezelfde thread) direct doorgaat naar volgende strand  $u'$

**Spawn boog**  $(u, v)$  Strand  $u$  'spawnt' strand  $v$  (mogelijks in andere thread)

**Call boog**  $(u, v)$  Strand  $u$  doet functieoproep naar functie  $v$  (in zelfde thread)

**Return boog**  $(u, x)$  Gespawnde strand  $u$  keert terug naar parentprocedure met  $x$  de eerste strand na de eerstvolgende sync na spawn  $u$

Iedere strand  $u$  die strand  $v$  spawnt, heeft ook cont. boog  $(u, u')$

# Soorten bogen

**Continuation boog**  $(u, u')$  Strand  $u$  die (in dezelfde thread) direct doorgaat naar volgende strand  $u'$

**Spawn boog**  $(u, v)$  Strand  $u$  'spawnt' strand  $v$  (mogelijks in andere thread)

**Call boog**  $(u, v)$  Strand  $u$  doet functieoproep naar functie  $v$  (in zelfde thread)

**Return boog**  $(u, x)$  Gespawnde strand  $u$  keert terug naar parentprocedure met  $x$  de eerste strand na de eerstvolgende sync na spawn  $u$

Iedere strand  $u$  die strand  $v$  spawnt, heeft ook cont. boog  $(u, u')$

# Soorten bogen

**Continuation boog**  $(u, u')$  Strand  $u$  die (in dezelfde thread) direct doorgaat naar volgende strand  $u'$

**Spawn boog**  $(u, v)$  Strand  $u$  'spawnt' strand  $v$  (mogelijks in andere thread)

**Call boog**  $(u, v)$  Strand  $u$  doet functieoproep naar functie  $v$  (in zelfde thread)

**Return boog**  $(u, x)$  Gespawnde strand  $u$  keert terug naar parentprocedure met  $x$  de eerste strand na de eerstvolgende sync na spawn  $u$

Iedere strand  $u$  die strand  $v$  spawnt, heeft ook cont. boog  $(u, u')$

# Soorten bogen

**Continuation boog**  $(u, u')$  Strand  $u$  die (in dezelfde thread) direct doorgaat naar volgende strand  $u'$

**Spawn boog**  $(u, v)$  Strand  $u$  'spawnt' strand  $v$  (mogelijks in andere thread)

**Call boog**  $(u, v)$  Strand  $u$  doet functieoproep naar functie  $v$  (in zelfde thread)

**Return boog**  $(u, x)$  Gespawnde strand  $u$  keert terug naar parentprocedure met  $x$  de eerste strand na de eerstvolgende sync na spawn  $u$

Iedere strand  $u$  die strand  $v$  spawnt, heeft ook cont. boog  $(u, u')$

# Soorten boog

**Continuation boog**  $(u, u')$  Strand  $u$  die (in dezelfde thread) direct doorgaat naar volgende strand  $u'$

**Spawn boog**  $(u, v)$  Strand  $u$  'spawnt' strand  $v$  (mogelijks in andere thread)

**Call boog**  $(u, v)$  Strand  $u$  doet functieoproep naar functie  $v$  (in zelfde thread)

**Return boog**  $(u, x)$  Gespawnde strand  $u$  keert terug naar parentprocedure met  $x$  de eerste strand na de eerstvolgende sync na spawn  $u$

Iedere strand  $u$  die strand  $v$  spawnt, heeft ook cont. boog  $(u, u')$



# Soorten bogen

**Continuation boog**  $(u, u')$  Strand  $u$  die (in dezelfde thread) direct doorgaat naar volgende strand  $u'$

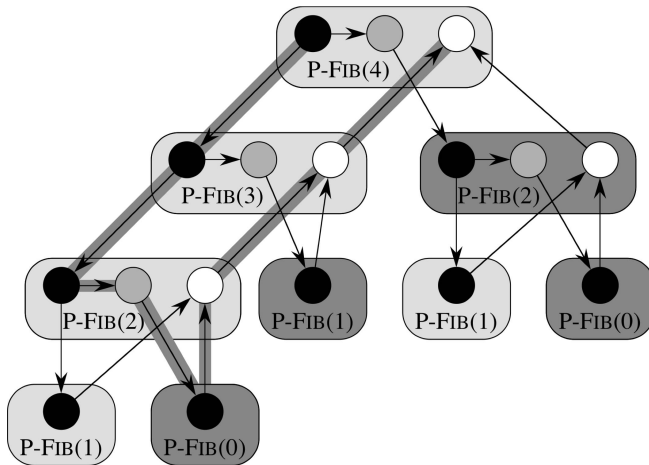
**Spawn boog**  $(u, v)$  Strand  $u$  'spawnt' strand  $v$  (mogelijks in andere thread)

**Call boog**  $(u, v)$  Strand  $u$  doet functieoproep naar functie  $v$  (in zelfde thread)

**Return boog**  $(u, x)$  Gespawnde strand  $u$  keert terug naar parentprocedure met  $x$  de eerste strand na de eerstvolgende sync na spawn  $u$

Iedere strand  $u$  die strand  $v$  spawnt, heeft ook cont. boog  $(u, u')$

# P-Fib(4)



- ▶ Bol: strand
- ▶ Hor. pijl: cont. boog
- ▶ Ver./Dig. pijl (neerwaarts): spawn of call boog
- ▶ Ver./Dig. pijl (opwaarts): return boog

# Strands in P-Fib

Bu / Eu : Begin strand u / End strand u

```

1  P-Fib (n)
2  /* Bu */
3  if n <= 1
4  return n
5  else
6  x = /* Eu */ spawn /* Bv */ P-Fib (n-1)
      /* Ev */
7  /* Bu' */
8  y = /* Eu' */ /* Bv' */ P-Fib (n-2)
9  /* Ev' */
10 sync
11 return /* Bx */ x + y /* Ex */

```

## Bogen

$(i \leq 2)$ :

-Spawn

$(u, v)$

-Cont  $(u, u')$

-Call  $(u', v')$

-Return

$(v, x) (v', x)$

\* sync in

return →

Parallel

keyword

# Strands in P-Fib

Bu / Eu : Begin strand u / End strand u

```

1  P-Fib (n)
2  /* Bu */
3  if n <= 1
4  return n
5  else
6  x = /* Eu */ spawn /* Bv */ P-Fib (n-1)
      /* Ev */
7  /* Bu' */
8  y = /* Eu' */ /* Bv' */ P-Fib (n-2)
9  /* Ev' */
10 sync
11 return /* Bx */ x + y /* Ex */

```

## Bogen

$(i \leq 2)$ :

-Spawn

$(u, v)$

-Cont  $(u, u')$

-Call  $(u', v')$

-Return

$(v, x) (v', x)$

\* sync in

return →

Parallel

keyword

# Strands in P-Fib

Bu / Eu : Begin strand u / End strand u

```

1  P-Fib (n)
2  /* Bu */
3  if n <= 1
4  return n
5  else
6  x = /* Eu */ spawn /* Bv */ P-Fib (n-1)
      /* Ev */
7  /* Bu' */
8  y = /* Eu' */ /* Bv' */ P-Fib (n-2)
9  /* Ev' */
10 sync
11 return /* Bx */ x + y /* Ex */

```

## Bogen

$(i \leq 2)$ :

-Spawn

$(u, v)$

-Cont  $(u, u')$

-Call  $(u', v')$

-Return

$(v, x) (v', x)$

\* sync in

return →

Parallel

keyword

# Strands in P-Fib

Bu / Eu : Begin strand u / End strand u

```

1  P-Fib (n)
2  /* Bu */
3  if n <= 1
4  return n
5  else
6  x = /* Eu */ spawn /* Bv */ P-Fib (n-1)
      /* Ev */
7  /* Bu' */
8  y = /* Eu' */ /* Bv' */ P-Fib (n-2)
9  /* Ev' */
10 sync
11 return /* Bx */ x + y /* Ex */

```

## Bogen

$(i \leq 2)$ :

-Spawn

$(u, v)$

-Cont  $(u, u')$

-Call  $(u', v')$

-Return

$(v, x) (v', x)$

\* sync in

return →

Parallel

keyword

# Strands in P-Fib

Bu / Eu : Begin strand u / End strand u

```

1  P-Fib (n)
2  /* Bu */
3  if n <= 1
4  return n
5  else
6  x = /* Eu */ spawn /* Bv */ P-Fib (n-1)
      /* Ev */
7  /* Bu' */
8  y = /* Eu' */ /* Bv' */ P-Fib (n-2)
9  /* Ev' */
10 sync
11 return /* Bx */ x + y /* Ex */

```

## Bogen

$(i \leq 2)$ :

-Spawn

$(u, v)$

-Cont  $(u, u')$

-Call  $(u', v')$

-Return

$(v, x) (v', x)$

\* sync in

return →

Parallel

keyword

# Strands in P-Fib

Bu / Eu : Begin strand u / End strand u

```

1  P-Fib (n)
2  /* Bu */
3  if n <= 1
4  return n
5  else
6  x = /* Eu */ spawn /* Bv */ P-Fib (n-1)
      /* Ev */
7  /* Bu' */
8  y = /* Eu' */ /* Bv' */ P-Fib (n-2)
9  /* Ev' */
10 sync
11 return /* Bx */ x + y /* Ex */

```

## Bogen

$(i \leq 2)$ :

-Spawn

$(u, v)$

-Cont  $(u, u')$

-Call  $(u', v')$

-Return

$(v, x) (v', x)$

\* sync in

return →

Parallel

keyword



# Strands in P-Fib

Bu / Eu : Begin strand u / End strand u

```

1  P-Fib (n)
2  /* Bu */
3  if n <= 1
4  return n
5  else
6  x = /* Eu */ spawn /* Bv */ P-Fib (n-1)
      /* Ev */
7  /* Bu' */
8  y = /* Eu' */ /* Bv' */ P-Fib (n-2)
9  /* Ev' */
10 sync
11 return /* Bx */ x + y /* Ex */

```

## Bogen

$(i \leq 2)$ :

-Spawn

$(u, v)$

-Cont  $(u, u')$

-Call  $(u', v')$

-Return

$(v, x) (v', x)$

\* sync in

return  $\rightarrow$

Parallel

keyword

# Ideale parallele computer

- ▶ Iedere processor even vlug
- ▶ Sequentially consistent: Alsof 1 instructie-cyclus van alle processoren maar 1 geheugentoegang nodig was
- ▶ Geen scheduling kost (in realiteit minimaal)

# Ideale parallele computer

- ▶ Iedere processor even vlug
- ▶ Sequentially consistent: Alsof 1 instructie-cyclus van alle processoren maar 1 geheugentoegang nodig was
- ▶ Geen scheduling kost (in realiteit minimaal)

# Ideale parallele computer

- ▶ Iedere processor even vlug
- ▶ Sequentially consistent: Alsof 1 instructie-cyclus van alle processoren maar 1 geheugentoegang nodig was
- ▶ Geen scheduling kost (in realiteit minimaal)

# Ideale parallele computer

- ▶ Iedere processor even vlug
- ▶ Sequentially consistent: Alsof 1 instructie-cyclus van alle processoren maar 1 geheugentoegang nodig was
- ▶ Geen scheduling kost (in realiteit minimaal)

# Overzicht

Inleiding

Basis van multithreading

Meeteenheden prestatie

Scheduling van threads

Analyseren van een algoritme

Matrix vermenigvuldiging

Merge sort

Besluit

# Prestatie meten

## Hoe kwaliteit meten van een algoritme?

### *work*

- ▶ Tijd om op 1 processor uit te voeren

\* *bij 1 tijdseenheid per strand, aantal knopen*

### *span*

- ▶ De tijd van het meest tijdsintensieve pad

\* *bij 1 tijdseenheid per strand, lengte langste (critical) pad*

$T_P$  = tijd op  $P$  processors  
work =  $T_1$  en span =  $T_\infty$

# Prestatie meten

Hoe kwaliteit meten van een algoritme?

## *work*

- ▶ Tijd om op 1 processor uit te voeren

\* *bij 1 tijdseenheid per strand, aantal knopen*

## *span*

- ▶ De tijd van het meest tijdsintensieve pad

\* *bij 1 tijdseenheid per strand, lengte langste (critical) pad*

$T_P$  = tijd op  $P$  processors  
work =  $T_1$  en span =  $T_\infty$



# Prestatie meten

Hoe kwaliteit meten van een algoritme?

## ***work***

- ▶ Tijd om op 1 processor uit te voeren

\* *bij 1 tijdseenheid per strand, aantal knopen*

## ***span***

- ▶ De tijd van het meest tijdsintensieve pad

\* *bij 1 tijdseenheid per strand, lengte langste (critical) pad*

$T_P$  = tijd op  $P$  processors  
work =  $T_1$  en span =  $T_\infty$

# Prestatie meten

Hoe kwaliteit meten van een algoritme?

## ***work***

- ▶ Tijd om op 1 processor uit te voeren

\* *bij 1 tijdseenheid per strand, aantal knopen*

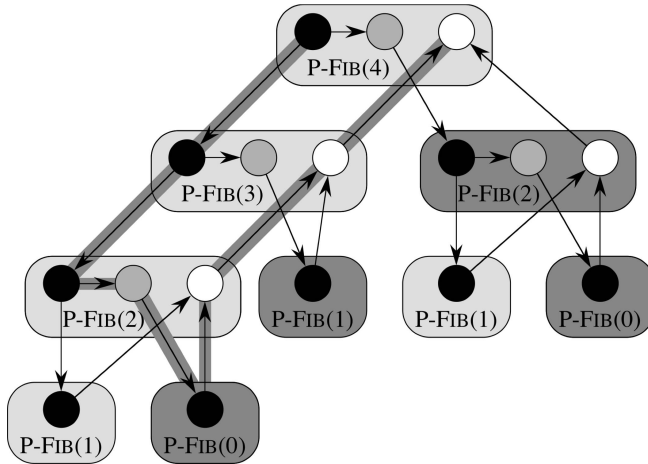
## ***span***

- ▶ De tijd van het meest tijdsintensieve pad

\* *bij 1 tijdseenheid per strand, lengte langste (critical) pad*

$T_P$  = tijd op  $P$  processors  
work =  $T_1$  en span =  $T_\infty$

# Span



Span is de  
dikke lijn.

# Ondergrenzen

Work en span zorgen voor ondergrenzen.

*work law*

- $T_P \geq T_1 / P$

P processoren  $\Rightarrow$  P werkeenheden / tijdseenheid

$\Rightarrow PT_P$  werkeenheden in  $T_P$  tijd

EN

Totaal werk = work  $\Rightarrow PT_P \geq T_1$

*span law*

- $T_P \geq T_\infty$

P processoren systeem altijd trager of even vlug als  $\infty$  processoren. ( $\infty$  kan P na-apen)

# Ondergrenzen

Work en span zorgen voor ondergrenzen.

*work law*

- $T_P \geq T_1 / P$

P processoren  $\Rightarrow$  P werkeenheden / tijdseenheid

$\Rightarrow PT_P$  werkeenheden in  $T_P$  tijd

EN

Totaal werk = work  $\Rightarrow PT_P \geq T_1$

*span law*

- $T_P \geq T_\infty$

P processoren systeem altijd trager of even vlug als  $\infty$  processoren. ( $\infty$  kan P na-apen)

# Ondergrenzen

Work en span zorgen voor ondergrenzen.

*work law*

- $T_P \geq T_1 / P$

P processoren  $\Rightarrow$  P werkeenheden / tijdseenheid

$\Rightarrow PT_P$  werkeenheden in  $T_P$  tijd

EN

Totaal werk = work  $\Rightarrow PT_P \geq T_1$

*span law*

- $T_P \geq T_\infty$

P processoren systeem altijd trager of even vlug als  $\infty$  processoren. ( $\infty$  kan  $P$  na-apen)

# Speedup

→ **Speedup**: hoeveel sneller met  $P$  processoren dan 1  
uitgedrukt met:  $T_1/T_P$

Met bovengrens  $P$  (work law)

Linear speedup  $T_1/T_P = \Theta(P)$

Perfect linear speedup  $T_1/T_P = P$

# Speedup

→ **Speedup**: hoeveel sneller met  $P$  processoren dan 1  
uitgedrukt met:  $T_1/T_P$

Met bovengrens  $P$  (work law)

Linear speedup  $T_1/T_P = \Theta(P)$

Perfect linear speedup  $T_1/T_P = P$



# Speedup

→ **Speedup**: hoeveel sneller met  $P$  processoren dan 1  
uitgedrukt met:  $T_1/T_P$   
Met bovengrens  $P$  (work law)

**Linear speedup**  $T_1/T_P = \Theta(P)$

Perfect linear speedup  $T_1/T_P = P$

# Speedup

→ **Speedup**: hoeveel sneller met  $P$  processoren dan 1  
uitgedrukt met:  $T_1/T_P$   
Met bovengrens  $P$  (work law)

**Linear speedup**  $T_1/T_P = \Theta(P)$

**Perfect linear speedup**  $T_1/T_P = P$

# Parallelism

→ **Parallelism**: hoeveel voordeel door multi-threading uitgedrukt met:  $T_1/T_\infty$

3 interpretaties:

1. *Ratio*: gemiddeld werk per stap in langste pad vergeleken met gemiddelde werk per stap van  $T_1$  (= work en langste pad = span)
2. *Bovengrens*: maximum speedup
3. *Mogelijkheid perfect lineair*: Indien # processoren  $Q$  groter is dan parallelisme, geen perfecte lineariteit mogelijk. (want  $T_Q \geq T_\infty$ )

# Parallelism

→ **Parallelism**: hoeveel voordeel door multi-threading uitgedrukt met:  $T_1/T_\infty$

3 interpretaties:

1. *Ratio*: gemiddeld werk per stap in langste pad vergeleken met gemiddelde werk per stap van  $T_1$  (= work en langste pad = span)
2. *Bovengrens*: maximum speedup
3. *Mogelijkheid perfect linear*: Indien # processoren  $Q$  groter is dan parallelisme, geen perfecte lineariteit mogelijk. (want  $T_Q \geq T_\infty$ )

# Parallelism

→ **Parallelism**: hoeveel voordeel door multi-threading uitgedrukt met:  $T_1/T_\infty$

3 interpretaties:

1. *Ratio*: gemiddeld werk per stap in langste pad vergeleken met gemiddelde werk per stap van  $T_1$  (= work en langste pad = span)
2. *Bovengrens*: maximum speedup
3. *Mogelijkheid perfect linear*: Indien # processoren  $Q$  groter is dan parallelisme, geen perfecte lineariteit mogelijk.  
(want  $T_Q \geq T_\infty$ )

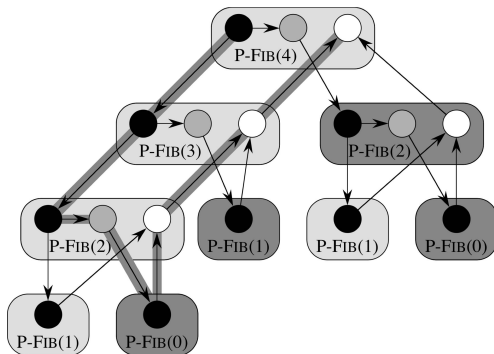
# Parallelism

→ **Parallelism**: hoeveel voordeel door multi-threading uitgedrukt met:  $T_1/T_\infty$

3 interpretaties:

1. *Ratio*: gemiddeld werk per stap in langste pad vergeleken met gemiddelde werk per stap van  $T_1$  (= work en langste pad = span)
2. *Bovengrens*: maximum speedup
3. *Mogelijkheid perfect lineair*: Indien # processoren  $Q$  groter is dan parallelisme, geen perfecte lineariteit mogelijk.  
(want  $T_Q \geq T_\infty$ )

# Vb parallelisme



work =  $T_1 = 17$

span =  $T_\infty = 8$   
(dikke lijn)

parallelism =  
 $T_1/T_\infty = 2,125$

→ max 2,125  
sneller

# Slackness

Verhouding tussen parallellisme algoritme en computer met P processors

$$\frac{\text{parallellisme}}{P} = \frac{T_1/T_\infty}{P} = \frac{T_1}{PT_\infty}$$

*‘Hoeveel meer/minder parallellisme dan processors’*

Onder 1 → meer processors dan parallellisme → niet perfect lineair

Boven 1 → minder processors dan parallellisme → mogelijks perfect lineair

⇒ Processors zijn hierbij de limiterende factor



# Overzicht

Inleiding

Basis van multithreading

Meeteenheden prestatie

Scheduling van threads

Analyseren van een algoritme

Matrix vermenigvuldiging

Merge sort

Besluit

# Scheduling

Algemeen : Een scheduler beslist of er een thread aangemaakt wordt bij een spawn en mapped op static string.  
Beslissing hangt af van momentele belasting computer.

▷ Waarom?

Strands efficiënt parallel uitvoeren (teveel zorgt voor trashing)

# Kenmerken beschouwde scheduler

**Centralized** De scheduler weet op ieder moment de load van de computer.

**Greedy** De scheduler creëert zoveel mogelijk threads bij iedere stap.

**Complete stap** Er zijn  $P$  strands klaar om uit te voeren op ieder tijds stap. Minder is incomplete.

# Performance greedy scheduler

## Stelling 27.1

Gegeven een ideale parallelle computer met  $P$  processors, een greedy scheduler en een algoritme met span =  $T_\infty$  en work =  $T_1$

$$\Rightarrow T_P \leq \frac{T_1}{P} + T_\infty$$

Bovengrens work law =  $T_P \leq \frac{T_1}{P}$  en span law  $T_P \leq T_\infty$

→ Niet slecht.

# Bewijs stelling 27.1

## 1) Complete stap

$P$  processors  $\Rightarrow P \frac{\text{werk}}{\text{stap}}$

Stel aantal complete stappen  $> \lfloor T_1/P \rfloor$

$\Rightarrow$  dan is het total werk minstens

$$\begin{aligned} P \cdot (\lfloor T_1/P \rfloor + 1) &= P \cdot \lfloor T_1/P \rfloor + P \\ &= T_1 - (T_1 \bmod P) + P \\ &> T_1 \end{aligned} \quad (1)$$

\*want:  $a \bmod n = a - n \lfloor a/n \rfloor$

\*\*want:  $0 \leq a \bmod n < n$

**CONTRADICTIE:** meer werk dan  $T_1$   
 $\Rightarrow$  aantal complete stappen  $\leq \lfloor T_1/P \rfloor$

## Bewijs stelling 27.1

### 1) Complete stap

$P$  processors  $\Rightarrow P \frac{\text{werk}}{\text{stap}}$

Stel aantal complete stappen  $> \lfloor T_1/P \rfloor$

$\Rightarrow$  dan is het total werk minstens

$$\begin{aligned} P \cdot (\lfloor T_1/P \rfloor + 1) &= P \cdot \lfloor T_1/P \rfloor + P \\ &= T_1 - (T_1 \bmod P) + P \\ &> T_1 \end{aligned} \quad (1)$$

*\*want:  $a \bmod n = a - n\lfloor a/n \rfloor$*

*\*\*want:  $0 \leq a \bmod n < n$*

**CONTRADICTIE:** meer werk dan  $T_1$   
 $\Rightarrow$  aantal complete stappen  $\leq \lfloor T_1/P \rfloor$

## Bewijs stelling 27.1

### 1) Complete stap

$P$  processors  $\Rightarrow P \frac{\text{werk}}{\text{stap}}$

Stel aantal complete stappen  $> \lfloor T_1/P \rfloor$

$\Rightarrow$  dan is het total werk minstens

$$\begin{aligned} P \cdot (\lfloor T_1/P \rfloor + 1) &= P \cdot \lfloor T_1/P \rfloor + P \\ &= T_1 - (T_1 \bmod P) + P \\ &> T_1 \end{aligned} \quad (1)$$

\*want:  $a \bmod n = a - n\lfloor a/n \rfloor$

\*\*want:  $0 \leq a \bmod n < n$

**CONTRADICTIE:** meer werk dan  $T_1$   
 $\Rightarrow$  aantal complete stappen  $\leq \lfloor T_1/P \rfloor$

# Bewijs stelling 27.1

## 2) **Incomplete stap**

Stel graaf  $G$  de graaf die het algoritme voorstelt:

Maak alle bogen gewicht 1 door langere bogen op te splitsen.

→  $G'$  subgraaf uit te voeren voor de incomplete stap

→  $G''$  uit te voeren erna

→ Startknoop cruciaal pad geen inkomende bogen = uitvoerbaar (anders niet start)

⇒ Incomplete stap voert alle zulke bogen uit in  $G'$  (want minder dan  $P$  strands en greedy )

⇒ Lengte cruciale pad  $G''$  1 korter dan van  $G'$

⇒ Span = Span - 1 na stap

⇒ Aantal incomplete stappen  $\leq$  span =  $T_\infty$



# Bewijs stelling 27.1

## 2) **Incomplete stap**

Stel graaf  $G$  de graaf die het algoritme voorstelt:

Maak alle bogen gewicht 1 door langere bogen op te splitsen.

→  $G'$  subgraaf uit te voeren voor de incomplete stap

→  $G''$  uit te voeren erna

→ Startknoop cruciaal pad geen inkomende bogen = uitvoerbaar (anders niet start)

⇒ Incomplete stap voert alle zulke bogen uit in  $G'$   
(want minder dan  $P$  strands en greedy )

⇒ Lengte cruciale pad  $G''$  1 korter dan van  $G'$

⇒ Span = Span - 1 na stap

⇒ Aantal incomplete stappen  $\leq$  span =  $T_\infty$

# Bewijs stelling 27.1

## 2) **Incomplete stap**

Stel graaf  $G$  de graaf die het algoritme voorstelt:

Maak alle bogen gewicht 1 door langere bogen op te splitsen.

→  $G'$  subgraaf uit te voeren voor de incomplete stap

→  $G''$  uit te voeren erna

→ Startknoop cruciaal pad geen inkomende bogen = uitvoerbaar (anders niet start)

⇒ Incomplete stap voert alle zulke bogen uit in  $G'$  (want minder dan  $P$  strands en greedy )

⇒ Lengte cruciale pad  $G''$  1 korter dan van  $G'$

⇒  $\text{Span} = \text{Span} - 1$  na stap

⇒ Aantal incomplete stappen  $\leq \text{span} = T_\infty$

# Bewijs stelling 27.1

$$T_P = \# \text{ complete} + \# \text{ incomplete stappen} = \lfloor T_1/P \rfloor + T_\infty$$

$$\Rightarrow T_P \leq \lfloor T_1/P \rfloor + T_\infty$$



## Gevolg 27.2

Stel zelfde aannames stelling 27.2, dan is  $T_P$  maximum 2 keer de optimale tijd.

### Bewijs

Stel  $T_P^*$  optimale tijd

$$\Rightarrow T_P \leq \frac{T_1}{P} + T_\infty \leq 2 \cdot \max\left(\frac{T_1}{P}, T_\infty\right) \quad (27.1)$$

$$\begin{cases} T_P \leq 2 \cdot \max\left(\frac{T_1}{P}, T_\infty\right) \\ T_P^* \geq \max\left(\frac{T_1}{P}, T_\infty\right) \end{cases} \quad \text{work en span law}$$

$$\Rightarrow T_P \leq 2T_P^*$$



## Gevolg 27.2

Stel zelfde aannames stelling 27.2, dan is  $T_P$  maximum 2 keer de optimale tijd.

### Bewijs

Stel  $T_P^*$  optimale tijd

$$\Rightarrow T_P \leq \frac{T_1}{P} + T_\infty \leq 2 \cdot \max\left(\frac{T_1}{P}, T_\infty\right) \quad (27.1)$$

$$\begin{cases} T_P \leq 2 \cdot \max\left(\frac{T_1}{P}, T_\infty\right) \\ T_P^* \geq \max\left(\frac{T_1}{P}, T_\infty\right) \end{cases} \quad \text{work en span law}$$

$$\Rightarrow T_P \leq 2T_P^*$$



# Gevolg 27.3

# Overzicht

Inleiding

Basis van multithreading

Meeteenheden prestatie

Scheduling van threads

Analyseren van een algoritme

Matrix vermenigvuldiging

Merge sort

Besluit

# Overzicht

Inleiding

Basis van multithreading

Meeteenheden prestatie

Scheduling van threads

Analyseren van een algoritme

Matrix vermenigvuldiging

Merge sort

Besluit



# Frame-titel

Tekst.

# Overzicht

Inleiding

Basis van multithreading

Meeteenheden prestatie

Scheduling van threads

Analyseren van een algoritme

Matrix vermenigvuldiging

Merge sort

Besluit

# Frame-titel

Tekst.

# Overzicht

Inleiding

Basis van multithreading

Meeteenheden prestatie

Scheduling van threads

Analyseren van een algoritme

Matrix vermenigvuldiging

Merge sort

Besluit

# Afsluitende frame

Dynamic programming voorziet niet enkel een betere manier, maar zelfs een bijna optimale manier.