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

KU LEUVEN kulak

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 parallalism 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 ($\Theta(F_n)$)

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

Fib(n)

```

if n <= 1
  return n
else
  x = Fib(n-1)
  y = Fib(n-2)
  return x + y


```

Keywords

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

sync Wachten tot alle children voltooien (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 voltooien (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 voltooien (impliciet in iedere return)

Parallel voorbeeld

‘Logical parallelism’ subroutine *kan* parallel uitvoeren

Serialization Threading keywoorden weglaten geeft sequentieel algoritme

P-Fib(n)

```

if n <= 1
   return n
else
   x = spawn P-Fib(n-1)
   y = P-Fib(n-2)
   sync
   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 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'

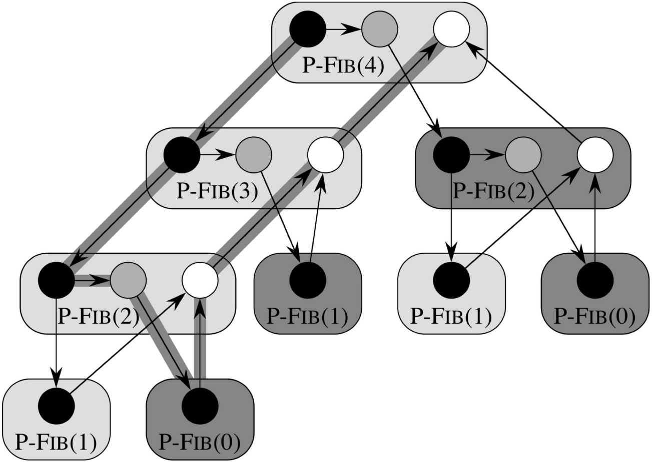
Spawn boog (u, v) Strand u 'spawnt' strand v (mogelijks in andere thread)

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')

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

```
P-Fib(n)
/*Bu*/
if n <= 1
return n
else
x = /*Eu*/ spawn /*Bv*/ P-Fib(n-1)
  /*Ev*/
/*Bu'*/
y = /*Eu'*/ /*Bv'*/ P-Fib(n-2)
/*Ev'*/
sync
return /*Bx*/ x + y /*Ex*/
```

Bogen $(i \leqslant 2)$**:**
-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

```
P−Fib(n)
/*Bu*/
if n <= 1
return n
else
x = /*Eu*/ spawn /*Bv*/ P−Fib(n−1)
  /*Ev*/
/*Bu'*/
y = /*Eu'*/ /*Bv'*/ P−Fib(n−2)
/*Ev'*/
sync
return /*Bx*/ x + y /*Ex*/
```

Bogen ($i \leqslant 2$):

-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

```
P-Fib(n)
/*Bu*/
if n <= 1
return n
else
x = /*Eu*/ spawn /*Bv*/ P-Fib(n-1)
  /*Ev*/
/*Bu'*/
y = /*Eu'*/ /*Bv'*/ P-Fib(n-2)
/*Ev'*/
sync
return /*Bx*/ x + y /*Ex*/
```

Bogen ($i \leqslant 2$):
-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

```
P-Fib(n)
/*Bu*/
if n <= 1
return n
else
x = /*Eu*/ spawn /*Bv*/ P-Fib(n-1)
  /*Ev*/
/*Bu'*/
y = /*Eu'*/ /*Bv'*/ P-Fib(n-2)
/*Ev'*/
sync
return /*Bx*/ x + y /*Ex*/
```

Bogen $(i \leqslant 2)$**:**
-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

```
P-Fib(n)
/*Bu*/
if n <= 1
return n
else
x = /*Eu*/ spawn /*Bv*/ P-Fib(n-1)
  /*Ev*/
/*Bu'*/
y = /*Eu'*/ /*Bv'*/ P-Fib(n-2)
/*Ev'*/
sync
return /*Bx*/ x + y /*Ex*/
```

Bogen ($i \leqslant 2$):

-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

```
P-Fib(n)
/*Bu*/
if n <= 1
return n
else
x = /*Eu*/ spawn /*Bv*/ P-Fib(n-1)
  /*Ev*/
/*Bu'*/
y = /*Eu'*/ /*Bv'*/ P-Fib(n-2)
/*Ev'*/
sync
return /*Bx*/ x + y /*Ex*/
```

Bogen ($i \leqslant 2$):

-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

```
P-Fib(n)
/*Bu*/
if n <= 1
return n
else
x = /*Eu*/ spawn /*Bv*/ P-Fib(n-1)
  /*Ev*/
/*Bu'*/
y = /*Eu'*/ /*Bv'*/ P-Fib(n-2)
/*Ev'*/
sync
return /*Bx*/ x + y /*Ex*/
```

Bogen ($i \leqslant 2$):
-Spawn (u, v)
-Cont (u, u')
-Call (u', v')
-Return (v, x) (v', x)
$*$ sync in return –> Parallel keyword

Ideale parallelle 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 parallelle 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 parallelle 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 parallelle 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_∞

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_∞

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_∞

Prestatie meten

Hoe kwaliteit meten van een algoritme?

work

- Tijd om op 1 processor uit te voeren

∗ *bij 1 tijdseenheid per strand, aantal knopen*

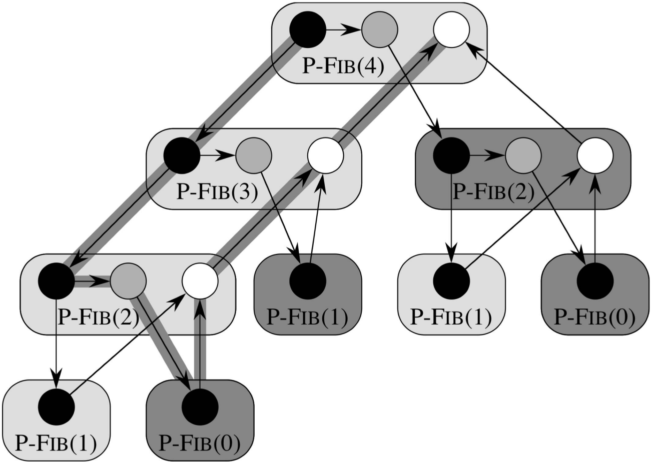
span

- 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_∞

Span



Span is de dikke lijn.

Ondergrenzen

Work en span zorgen voor ondergrenzen.

work law

- $T_P \geqslant T_1/P$

  P processoren $\Rightarrow$ P werkeenheden / tijdseenheid
  $\Rightarrow$ PT_P werkeenheden in T_P tijd
  EN
  Totaal werk = work $\Rightarrow PT_P \geqslant T_1$

span law

- $T_P \geqslant T_\infty$

  P processoren systeem altijd trager of even vlug als ∞ processoren. *(∞ kan P na-apen)*

Ondergrenzen

Work en span zorgen voor ondergrenzen.

work law

- $T_{\text{P}} \geqslant T_1/P$

  P processoren $\Rightarrow$ P werkeenheden / tijdseenheid
  $\Rightarrow PT_{\text{P}}$ werkeenheden in T_{P} tijd
  EN
  Totaal werk = work $\Rightarrow PT_{\text{P}} \geqslant T_1$

span law

- $T_{\text{P}} \geqslant T_\infty$

  P processoren systeem altijd trager of even vlug als ∞ processoren. *(∞ kan P na-apen)*

Ondergrenzen

Work en span zorgen voor ondergrenzen.

work law

- $T_{\mathrm{P}} \geqslant T_1/P$

  P processoren $\Rightarrow$ P werkeenheden / tijdseenheid
  $\Rightarrow$ PT_{P} werkeenheden in $\mathrm{T_P}$ tijd
  EN
  Totaal werk = work $\Rightarrow PT_{\mathrm{P}} \geqslant T_1$

span law

- $T_{\mathrm{P}} \geqslant T_{\infty}$

  P processoren systeem altijd trager of even vlug als ∞ processoren. *(∞ 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 $T_1/T_{\mathrm{P}} = \Theta(P)$

Perfect linear speedup $T_1/T_{\mathrm{P}} = P$

Speedup

$\rightarrow$ ***Speedup***: hoeveel sneller met P processoren dan 1
uitgedrukt met: T_1/T_{P}
Met bovengrens P (work law)

Linear speedup $T_1/T_{\mathrm{P}} = \Theta(P)$

Perfect linear speedup $T_1/T_{\mathrm{P}} = P$

Speedup

$\rightarrow$ ***Speedup***: hoeveel sneller met P processoren dan 1
uitgedrukt met: T_1/T_{P}
Met bovengrens P (work law)

Linear speedup $T_1/T_{\mathrm{P}} = \Theta(P)$

Perfect linear speedup $T_1/T_{\mathrm{P}} = P$

Speedup

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

Linear speedup $T_1/T_\mathrm{P} = \Theta(P)$

Perfect linear speedup $T_1/T_\mathrm{P} = P$

Parallelism

→ ***Parallelism***: hoeveel voordeel door multi-threading uitgedrukt met: T_1 / T_∞

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 parallellisme, geen perfecte lineariteit mogelijk. (want $T_Q \geqslant T_\infty$)

Parallelism

→ ***Parallelism***: hoeveel voordeel door multi-threading uitgedrukt met: T_1 / T_∞

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 parallellisme, geen perfecte lineariteit mogelijk. (want $T_Q \geqslant T_\infty$)

Parallelism

→ ***Parallelism***: hoeveel voordeel door multi-threading uitgedrukt met: T_1 / T_∞

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 parallellisme, geen perfecte lineariteit mogelijk. (want $T_Q \geqslant T_\infty$)

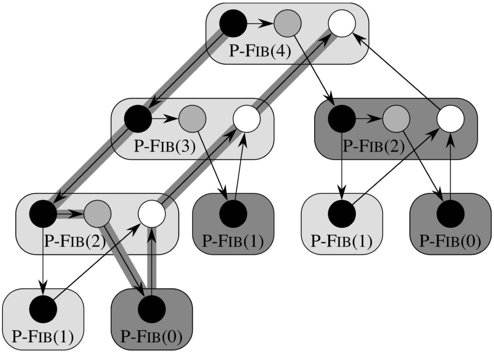
Parallelism

→ ***Parallelism***: hoeveel voordeel door multi-threading uitgedrukt met: T_1/T_∞

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 parallellisme, geen perfecte lineariteit mogelijk. (want $T_Q \geqslant T_\infty$)

Vb parallellisme



work = T_1 = 17

span = T_∞ = 8 (dikke lijn)

parallelism = T_1/T_∞ = 2,125

$\rightarrow$ max 2,125 sneller

Slackness

Verhouding tussen parallellisme algoritme en computer met P processors

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

'Hoeveel meer/minder parallellisme dan processors'

Onder 1 $\rightarrow$ meer processors dan parallellisme $\rightarrow$ niet perfect lineair

Boven 1 $\rightarrow$ minder processors dan parallellisme $\rightarrow$ mogelijks perfect lineair
$\Rightarrow$Processors zijn hierbij de limiterende factor

Overzicht

KU LEUVEN kulak

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:
$\rightarrow$ te veel zorgt voor trashing
$\rightarrow$ te weinig voor onderbenutting

KU LEUVEN kulak

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.

Perfomance greedy schedular

Stelling 27.1
Gegeven een ideale parallelle computer met P processors, een greedy scheduler en een algoritme met span = T_∞ en work = T_1
$\Rightarrow T_P \leqslant \frac{T_1}{P} + T_\infty$

Bovengrens work law = $T_P \leqslant \frac{T_1}{P}$ en span law $T_P \leqslant T_\infty$
$\rightarrow$ Niet slecht.

Bewijs stelling 27.1

1) **Complete stap**
P processors $\Rightarrow P \frac{werk}{stap}$
Stel aantal complete stappen $> \lfloor T_1/P \rfloor$
$\Rightarrow$ dan is het total werk minstens

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

*$want: a \mod n = a - n\lfloor a/n \rfloor$
**$want: 0 \leqslant a \mod n < n$

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

Bewijs stelling 27.1

1) **Complete stap**
   P processors $\Rightarrow P\frac{werk}{stap}$
   Stel aantal complete stappen $> \lfloor T_1/P \rfloor$
   $\Rightarrow$ dan is het total werk minstens

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

   **want:* $a \mod n = a - n\lfloor a/n \rfloor$
   ***want:* $0 \leqslant a \mod n < n$

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

KU LEUVEN kulak

Bewijs stelling 27.1

1) **Complete stap**
P processors $\Rightarrow P\frac{werk}{stap}$
Stel aantal complete stappen $> \lfloor T_1/P \rfloor$
$\Rightarrow$ dan is het total werk minstens

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

**want:* $a \mod n = a - n\lfloor a/n \rfloor$
***want:* $0 \leqslant a \mod n < n$

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

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.
$\rightarrow$ G' subgraaf uit te voeren voor de incomplete stap
$\rightarrow$ G" uit te voeren erna
$\rightarrow$ Startknoop cruciaal pad geen inkomende bogen = uitvoerbaar (anders niet start)
$\Rightarrow$ Incomplete stap voert alle zulke bogen uit in G' (want minder dan P strands en greedy)
$\Rightarrow$ Lengte cruciale pad G" 1 korter dan van G'
$\Rightarrow$ Span = Span - 1 na stap
$\Rightarrow$ Aantal incomplete stappen $\leqslant$ 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.
$\rightarrow$ G' subgraaf uit te voeren voor de incomplete stap
$\rightarrow$ G" uit te voeren erna
$\rightarrow$ Startknoop cruciaal pad geen inkomende bogen = uitvoerbaar (anders niet start)
$\Rightarrow$ Incomplete stap voert alle zulke bogen uit in G' (want minder dan P strands en greedy)
$\Rightarrow$ Lengte cruciale pad G" 1 korter dan van G'
$\Rightarrow$ Span = Span - 1 na stap
$\Rightarrow$ Aantal incomplete stappen $\leqslant$ 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.
$\rightarrow$ G' subgraaf uit te voeren voor de incomplete stap
$\rightarrow$ G" uit te voeren erna
$\rightarrow$ Startknoop cruciaal pad geen inkomende bogen = uitvoerbaar (anders niet start)
$\Rightarrow$ Incomplete stap voert alle zulke bogen uit in G' (want minder dan P strands en greedy)
$\Rightarrow$ Lengte cruciale pad G" 1 korter dan van G'
$\Rightarrow$ Span = Span - 1 na stap
$\Rightarrow$ Aantal incomplete stappen $\leqslant$ span $= T_\infty$

Bewijs stelling 27.1

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

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

$\square$

Gevolg 27.2

Stel zelfde aannames stelling 27.1, dan is T_P maximum 2 keer de optimale tijd.

Bewijs

Stel T_P^* optimale tijd

$\Rightarrow T_P \leqslant \frac{T_1}{P} + T_\infty \leqslant 2 \cdot \max(\frac{T_1}{P}, T_\infty)$ (27.1)

$$\begin{cases} T_P \leqslant 2 \cdot \max(\frac{T_1}{P}, T_\infty) & \\ T_P^* \geqslant \max(\frac{T_1}{P}, T_\infty) & \text{work en span law} \end{cases}$$

$\Rightarrow T_P \leqslant 2T_P^*$

$\square$

Gevolg 27.2

Stel zelfde aannames stelling 27.1, dan is T_P maximum 2 keer de optimale tijd.

Bewijs

Stel T_P^* optimale tijd

$\Rightarrow T_P \leqslant \frac{T_1}{P} + T_\infty \leqslant 2 \cdot \max(\frac{T_1}{P}, T_\infty)$ (27.1)

$$\begin{cases} T_P \leqslant 2 \cdot \max(\frac{T_1}{P}, T_\infty) \\ T_P^* \geqslant \max(\frac{T_1}{P}, T_\infty) & \text{work en span law} \end{cases}$$

$\Rightarrow T_P \leqslant 2T_P^*$

$\square$

Gevolg 27.3

Stel zelfde aannames stelling 27.1 .
Als $P << \frac{T_1}{T_\infty}$ = slackness, dan $T_P \approx \frac{T_1}{P}$
Dus de speedup $\approx P$ en dus bijna perfect lineair.
$<<\approx$ 10 keer zo groot, dan is T_∞ in $T_P \leqslant \frac{T_1}{P} + T_\infty$ kleiner dan 10% van $\frac{werk}{processor}$

Bewijs Stel $P << \frac{T_1}{T_\infty}$

$\Rightarrow T_\infty << \frac{T_1}{P}$

$\Rightarrow \frac{T_1}{P} + T_\infty \approx \frac{T_1}{P}$

$$\begin{cases} \frac{T_1}{P} + T_\infty \approx \frac{T_1}{P} & \\ T_P \leqslant \frac{T_1}{P} + T_\infty & \text{(27.1)} \\ T_P \geqslant \frac{T_1}{P} & \text{work law} \end{cases}$$

$\Rightarrow T_P \approx \frac{T_1}{P} \Rightarrow \frac{T_1}{T_P} \approx P$ $\square$

Gevolg 27.3

Stel zelfde aannames stelling 27.1 .
Als $P << \frac{T_1}{T_\infty} =$ slackness, dan $T_P \approx \frac{T_1}{P}$
Dus de speedup $\approx P$ en dus bijna perfect lineair.
$<<\approx$ 10 keer zo groot, dan is T_∞ in $T_P \leqslant \frac{T_1}{P} + T_\infty$ kleiner dan 10% van $\frac{werk}{processor}$

Bewijs Stel $P << \frac{T_1}{T_\infty}$
$\Rightarrow T_\infty << \frac{T_1}{P}$
$\Rightarrow \frac{T_1}{P} + T_\infty \approx \frac{T_1}{P}$

$$\begin{cases} \frac{T_1}{P} + T_\infty \approx \frac{T_1}{P} & \\ T_P \leqslant \frac{T_1}{P} + T_\infty & (27.1) \\ T_P \geqslant \frac{T_1}{P} & \text{work law} \end{cases}$$

$\Rightarrow T_P \approx \frac{T_1}{P} \Rightarrow \frac{T_1}{T_P} \approx P$ $\square$

Overzicht

- Inleiding
- Basis van multithreading
- Meeteenheden prestatie
- Scheduling van threads
- Analyseren van een algoritme
- Matrix vermenigvuldiging
- Merge sort
- Besluit

Overzicht

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

KU LEUVEN kulak

Afsluitende frame

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