Případová studie akcelerace pomocí GPGPU: enumerativní ověřování modelu

Jiří Barnat

Technologie CUDA

- Levná a dostupná HW akcelerace
- Masově rozšířená
- Vyžaduje specifický způsob paralelizace algoritmů

Anotace této případové studie

Enumerativní ověřování modelu

- Formální verifikace **asynchronních paralelních systémů**
- Výpočetně a prostorově náročná aplikace
- Nevyžaduje aritmetiku s pohyblivou desetinnou čárkou
- P-úplné algoritmy (nelze snadno efektivně paralelizovat)

Předchozí úspěšné realizace paralelizace v ParaDiSe Parallel & Distributed Systems Laboratory

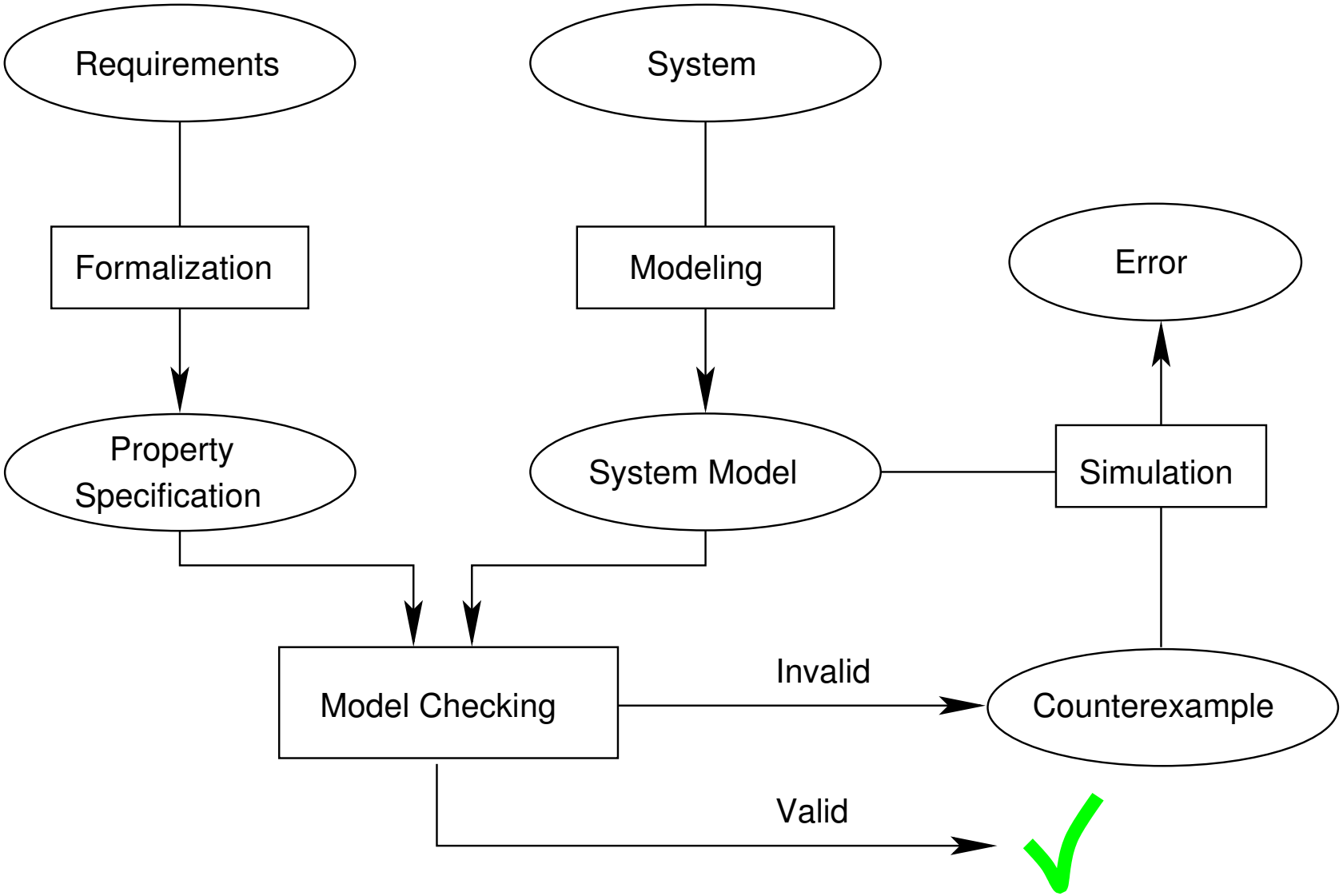
- Prostředí s distribuovanou pamětí (výpočetní klastry)
- Prostředí se sdílenou pamětí (vícejaderné výpočetní servery)

Otázka

- **Je možné akcelerovat s využitím GP GPU?**

Enumerativní ověřování modelu

Schéma metody ověřování modelu

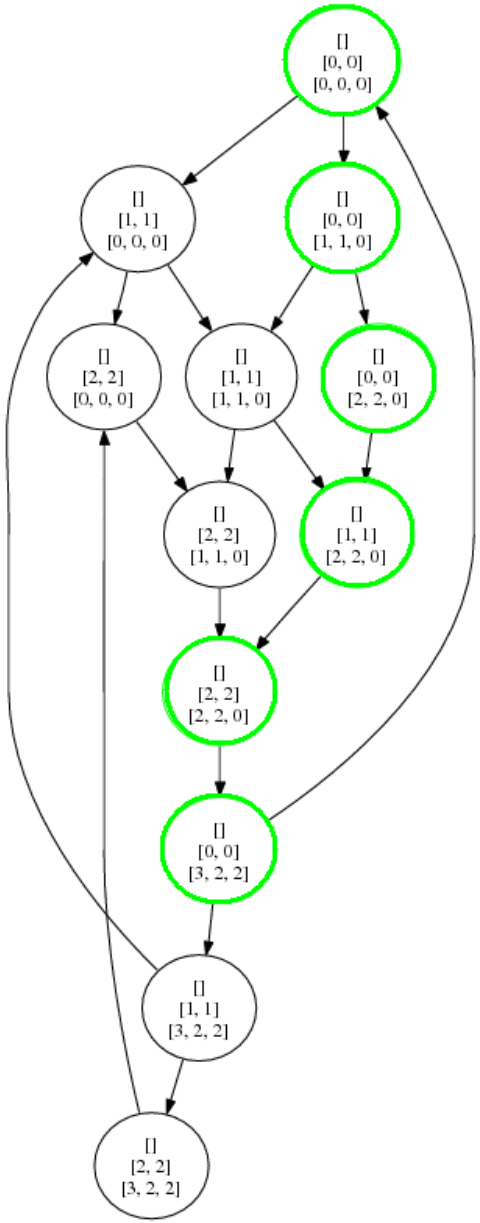


Paralelní systém & Graf stavového prostoru

```
channel {byte} c[0];

process A {
   byte a;
   state q1,q2,q3;
   init q1;
   trans
   q1→q2 { effect a=a+1; },
   q2→q3 { effect a=a+1; },
   q3→q1 { sync c!a; effect a=0; };
}

process B {
   byte b,x;
   state p1,p2,p3,p4;
   init p1;
   trans
   p1→p2 { effect b=b+1; },
   p2→p3 { effect b=b+1; },
   p3→p4 { sync c?x; },
   p4→p1 { guard x==b; effect b=0, x=0; };
}
```



Příklad konkrétního běhu paralelního programu

State: []; A:[q1, a:0]; B:[p1, b:0, x:0]
0 ⟨0.0⟩: q1 → q2 { effect a = a+1; }
1 ⟨1.0⟩: p1 → p2 { effect b = b+1; }
Command:1

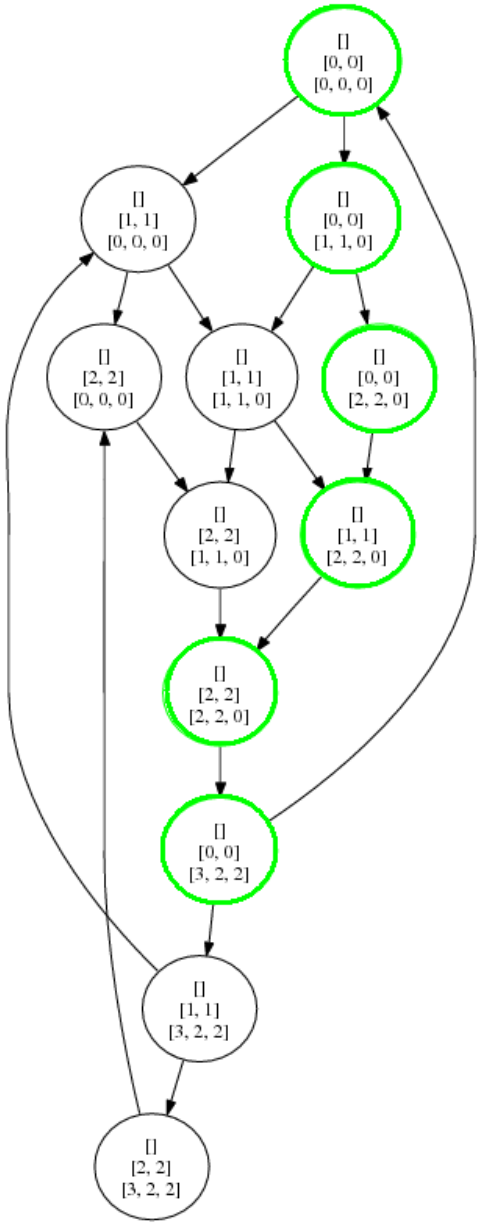
State: []; A:[q1, a:0]; B:[p2, b:1, x:0]
0 ⟨0.0⟩: q1 → q2 { effect a = a+1; }
1 ⟨1.1⟩: p2 → p3 { effect b = b+1; }
Command:1

State: []; A:[q1, a:0]; B:[p3, b:2, x:0]
0 ⟨0.0⟩: q1 → q2 { effect a = a+1; }
Command:0

State: []; A:[q2, a:1]; B:[p3, b:2, x:0]
0 ⟨0.1⟩: q2 → q3 { effect a = a+1; }
Command:0

State: []; A:[q3, a:2]; B:[p3, b:2, x:0]
0 ⟨0.2&1.2⟩: q3 → q1 { sync c!a; effect a = 0; }
p3 → p4 { sync c?x; }
Command:0

State: []; A:[q1, a:0]; B:[p4, b:2, x:2]



Specifikace vlastností

Pozorování

- Specifikace vstup/výstupního chování programu nepokrývá mnoho zajímavých vlastností reaktivních a paralelních programů.

Lepší způsob specifikace vlastností

- Požadavky na vlastnosti každého jednoho běhu systému.
- Systém je množina (nekonečných) běhů.
- Systém je korektní pokud specifikaci vyhovují všechny běhy.

Ověřování modelu

Pozorování

- Pro popis vlastností jednoho běhu potřebujeme automatizovanou rozhodovací proceduru, která ověří, zda daný nekonečný běh systému má požadovanou vlastnost.

Má běh π požadovanou vlastnost?

$$\pi \quad = \quad \overset{a,b}{\bullet} \longrightarrow \overset{b}{\bullet} \longrightarrow \overset{a}{\bullet} \longrightarrow \overset{a,b}{\bullet} \longrightarrow \overset{b}{\bullet} \longrightarrow \overset{b}{\bullet} \cdots$$

Ověřování modelu

Pozorování

- Pro popis vlastností jednoho běhu potřebujeme automatizovanou rozhodovací proceduru, která ověří, zda daný nekonečný běh systému má požadovanou vlastnost.

Má běh π požadovanou vlastnost?

$$\pi \quad = \quad \overset{a,b}{\bullet} \longrightarrow \overset{b}{\bullet} \longrightarrow \overset{a}{\bullet} \longrightarrow \overset{a,b}{\bullet} \longrightarrow \overset{b}{\bullet} \longrightarrow \overset{b}{\bullet} \cdots$$

Ověřování modelu s využitím teorie automatů

- **Büchi automat** rozpoznávající neplatné běhy.
- Synchronně spuštěn jako monitor se zkoumaným systémem.
- Verifikace redukována na problém detekce neplatného běhu.
- **Detekce dosažitelného akceptujícího cyklu v grafu.**

Akcelerace detekce akceptujícího cyklu

Algoritmy pro detekci akceptujících cyklů

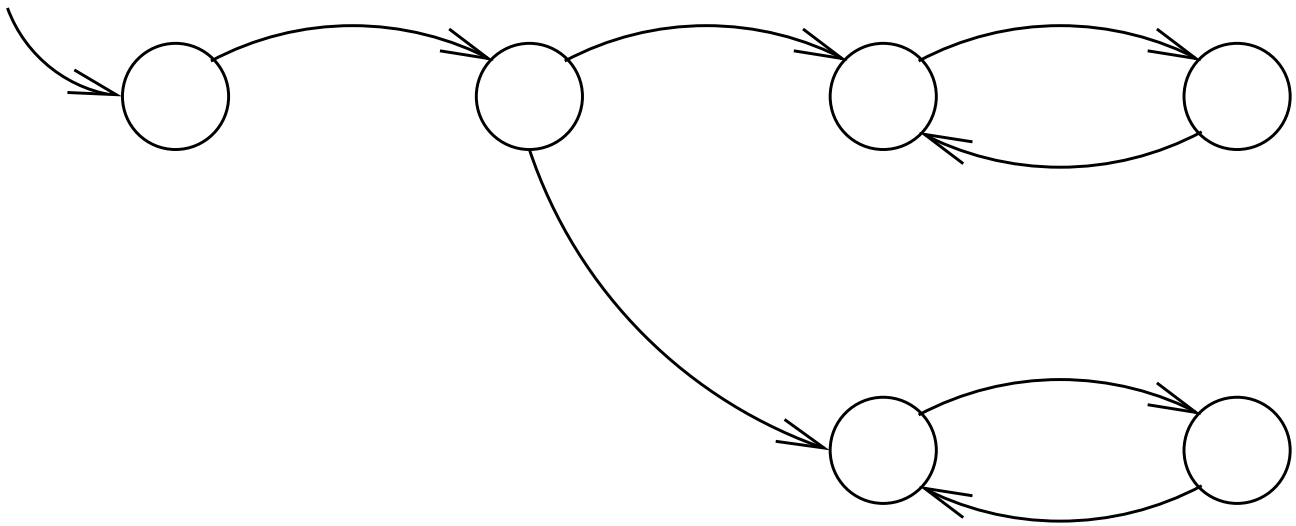
Sekvenční algoritmy

- Rozklad grafu na komponenty
- Vnořené prohledávání do hloubky (Nested DFS)
- DFS-postorder $\Longrightarrow O(|V| + |E|)$

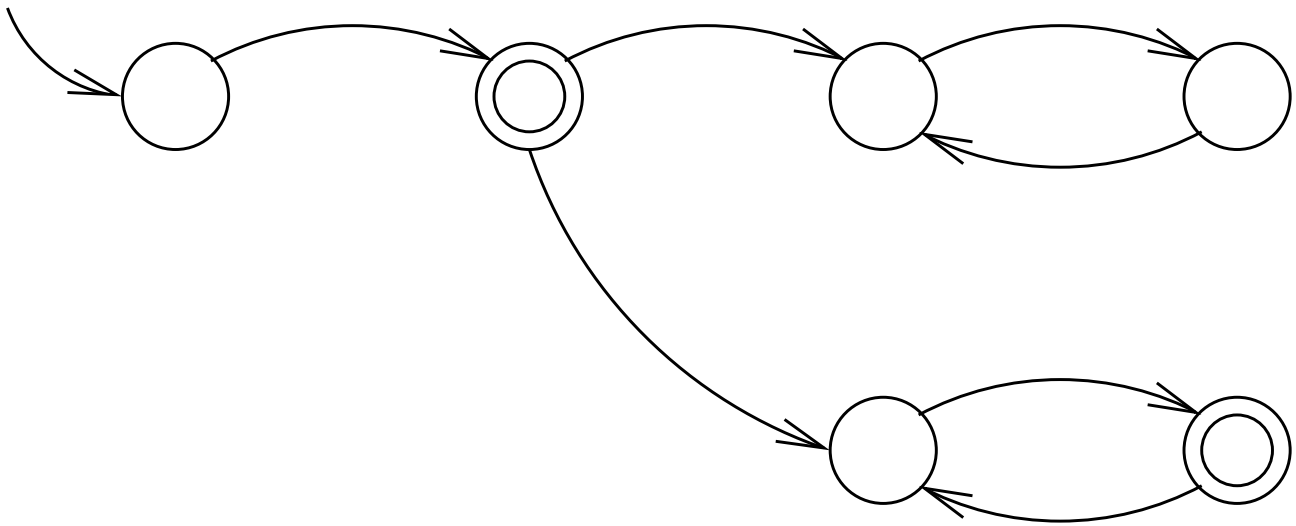
Paralelní algoritmy

- Není známo jak realizovat škálovatelně v čase $O(|V| + |E|)$
- Paralelní algoritmy jsou v obecném případě neoptimální
- Tvořeny z jednoduchých grafových primitiv, které lze dobře paralelizovat:
  - dopředná a zpětná dosažitelnost v grafu
  - topologické třídění
  - propagace hodnot (value iteration)

Algoritmus MAP

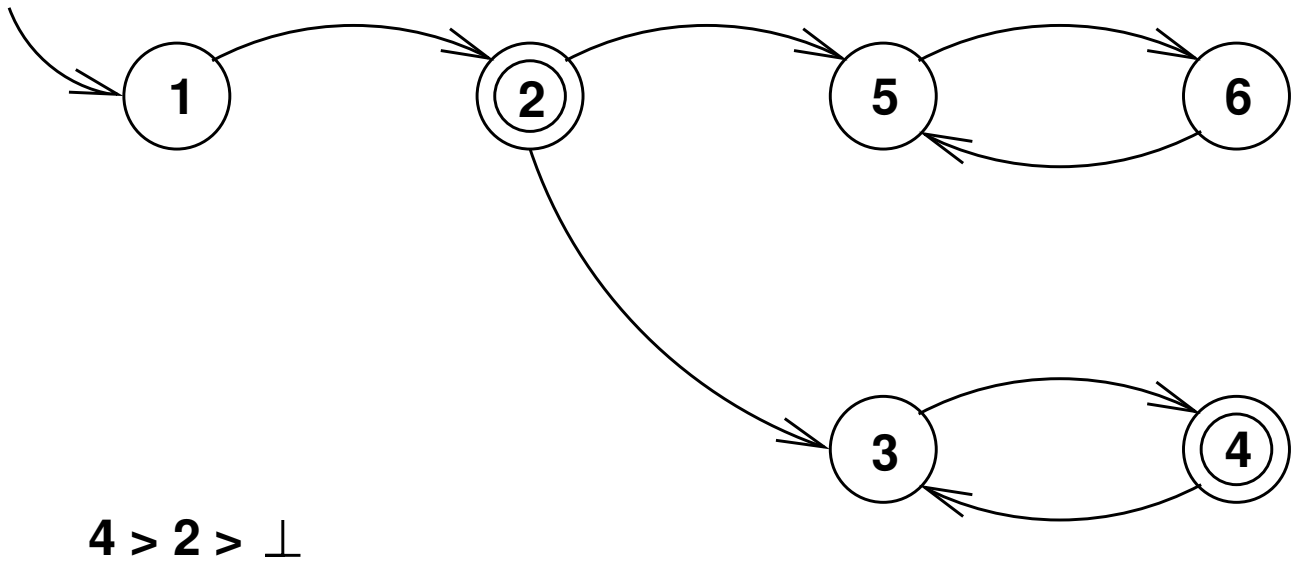


Algoritmus MAP



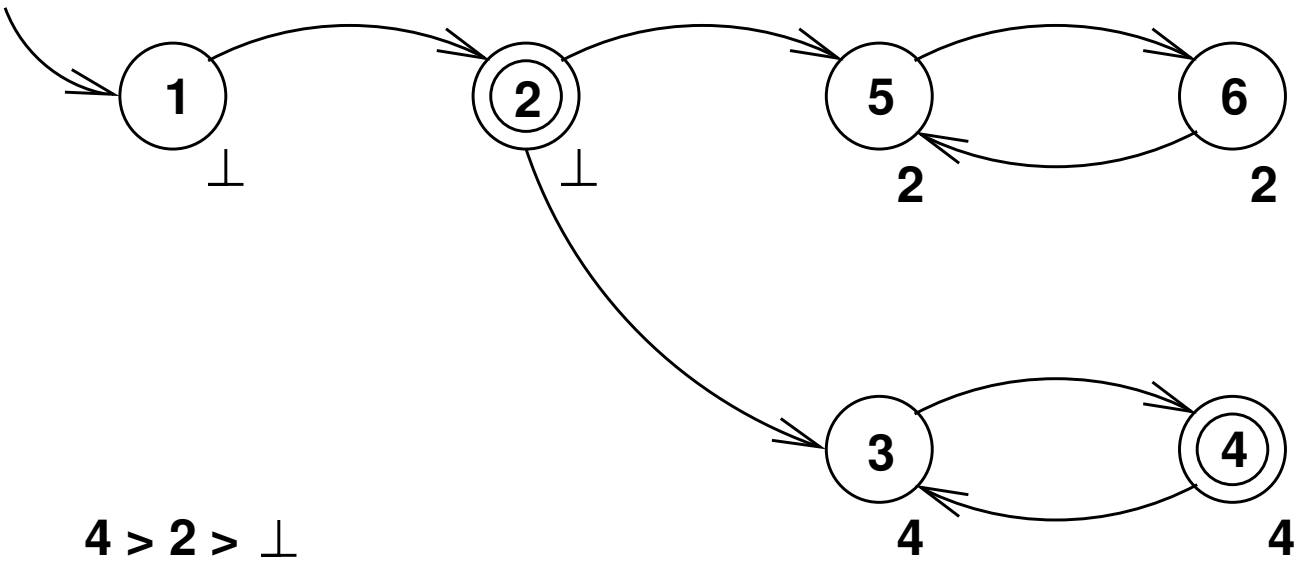
Akceptující stavy, akceptující stavy.

Algoritmus MAP



Uspořádání vrcholů grafu.

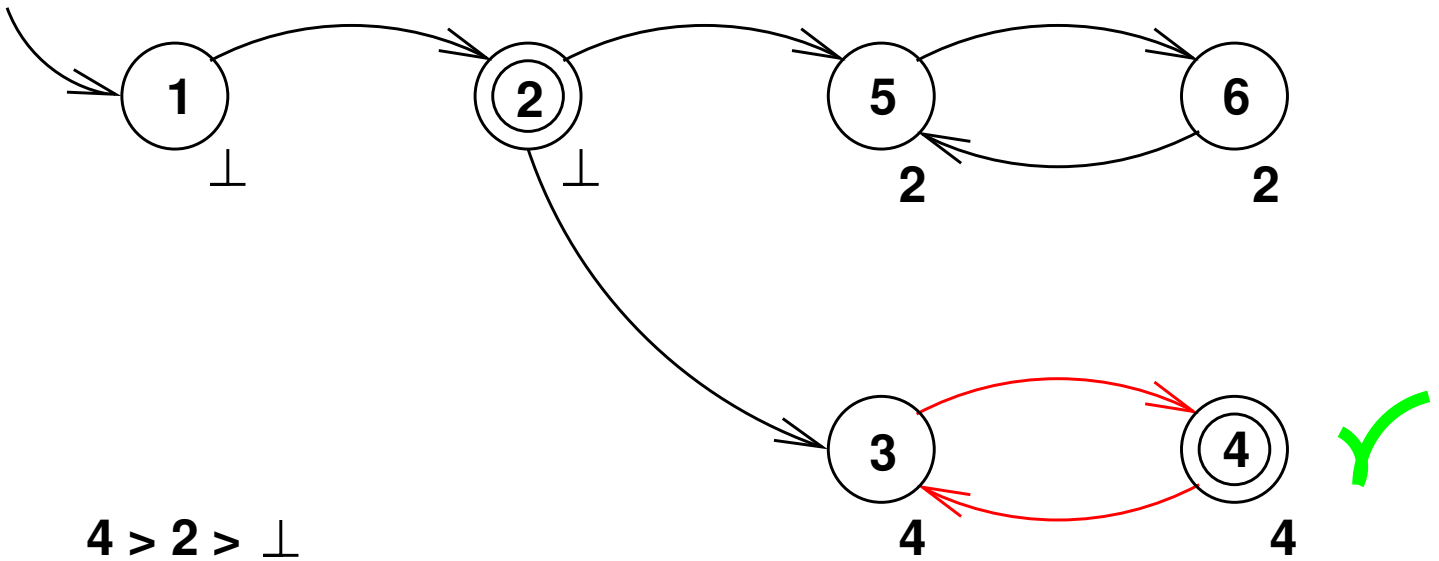
Algoritmus MAP



Maximální Akceptující Předchůdci (MAP)

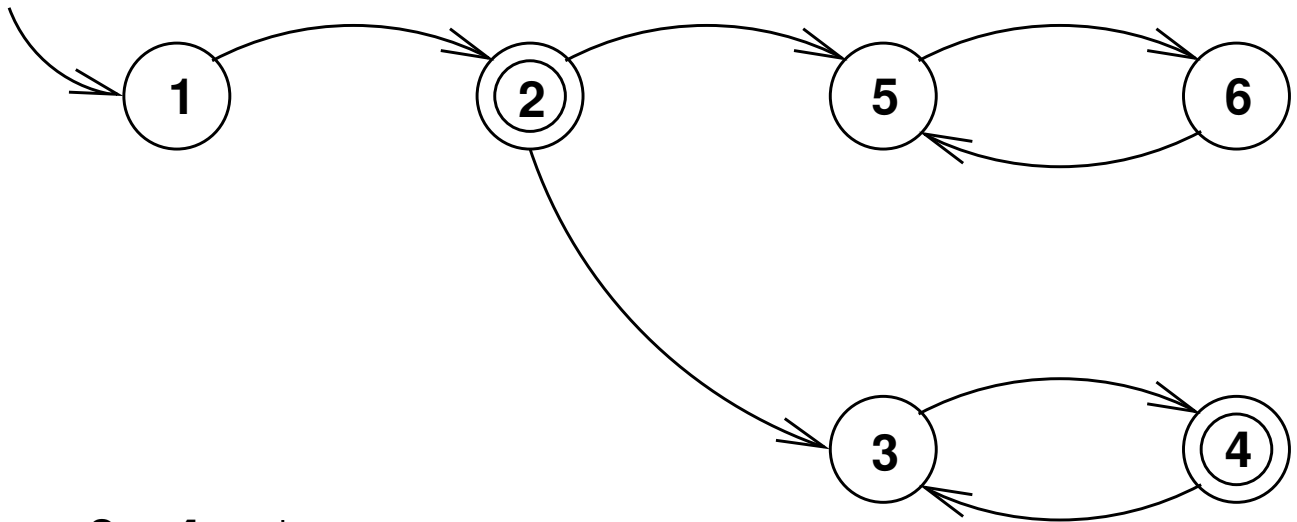
$$map(v) = \max\{\bot, u \mid (u, v) \in E^+ \wedge \mathcal{A}(u)\}$$

Algoritmus MAP



$map(v) = v \quad \Longrightarrow \quad$ akceptující cyklus

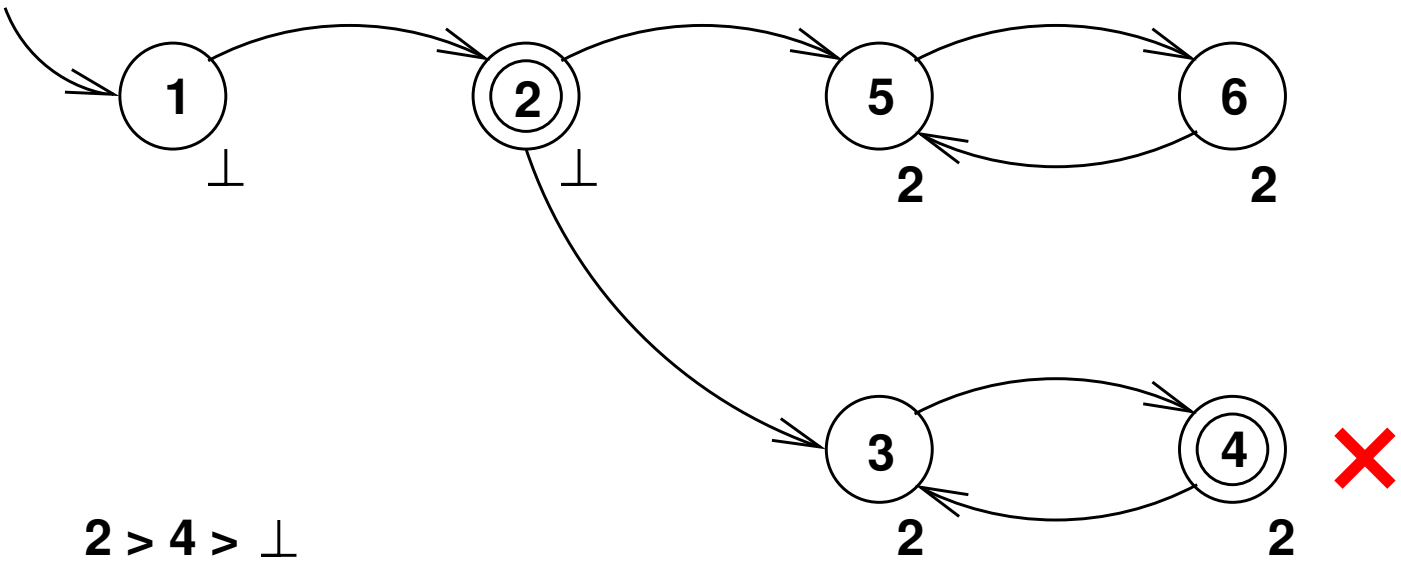
Algoritmus MAP



2 > 4 > $\perp$

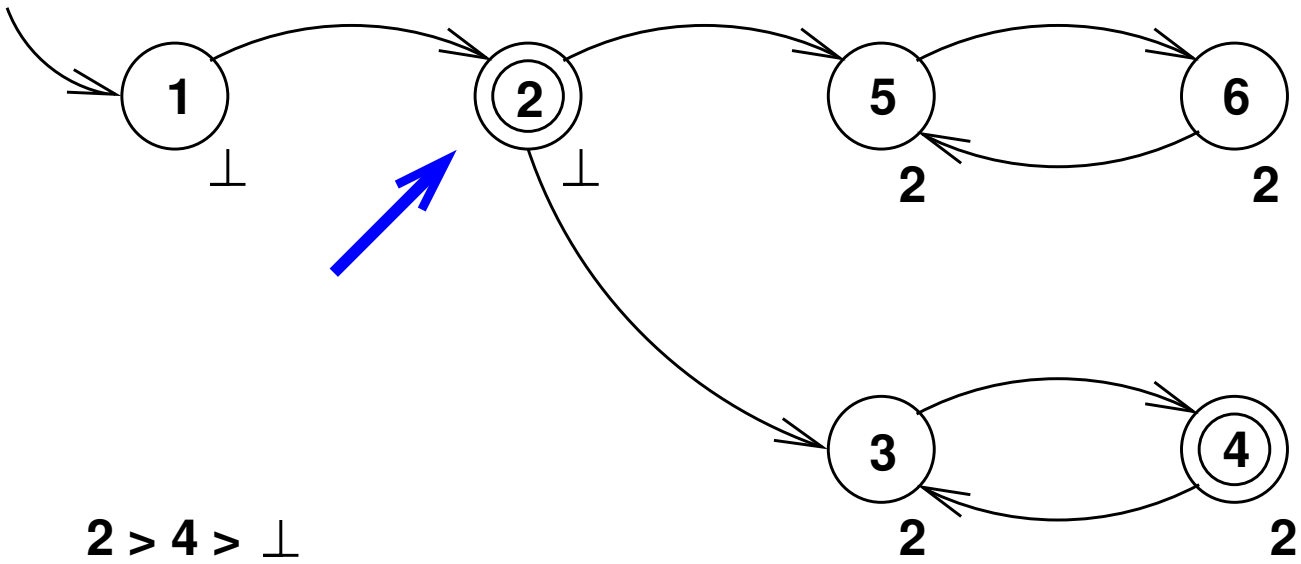
Co když $2 > 4$?

Algoritmus MAP



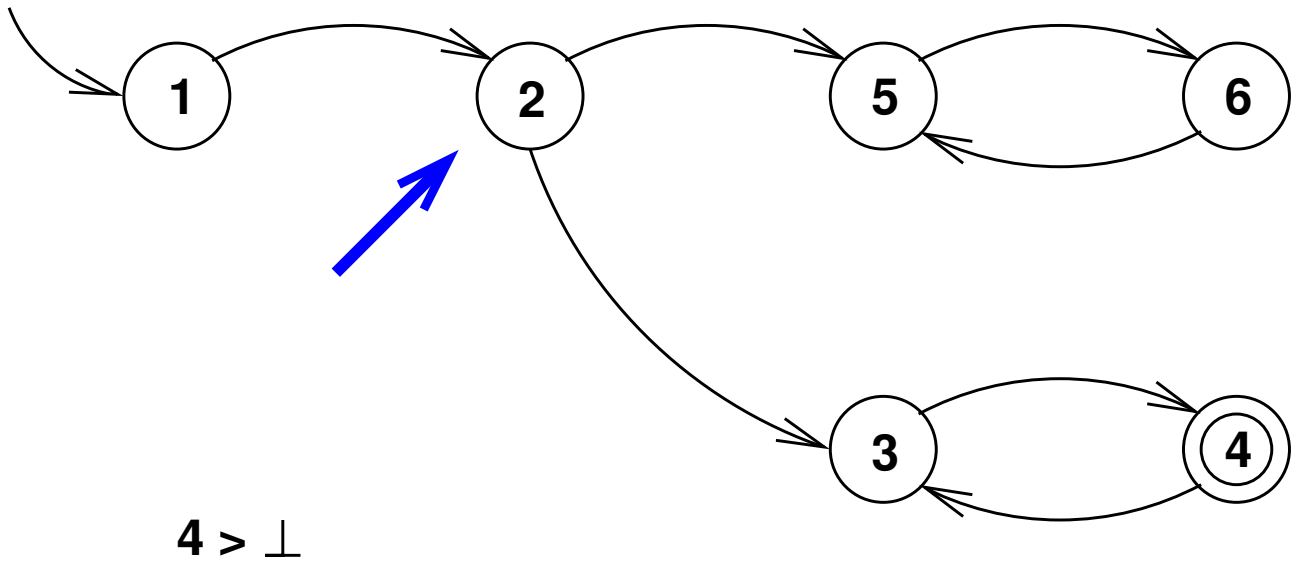
Akceptující cyklus není detekován.

Algoritmus MAP



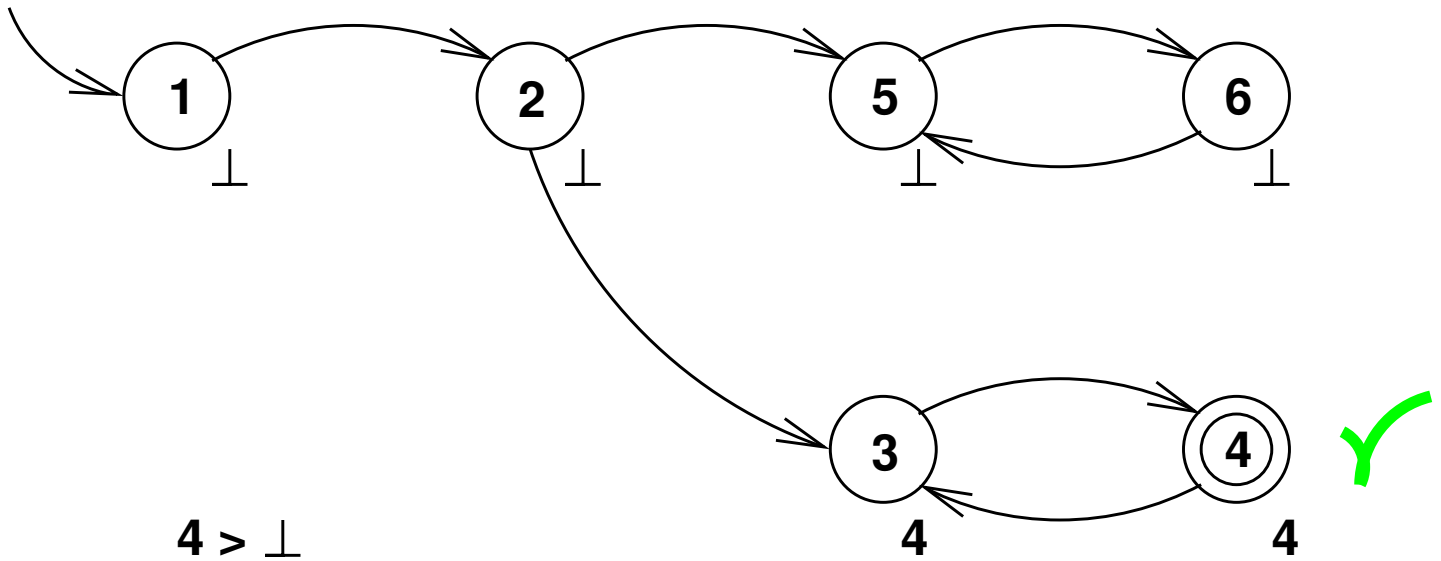
Pokud není nalezen akceptující cyklus, tak maximální akceptující předchůdci nemohou ležet na cyklu.

Algoritmus MAP



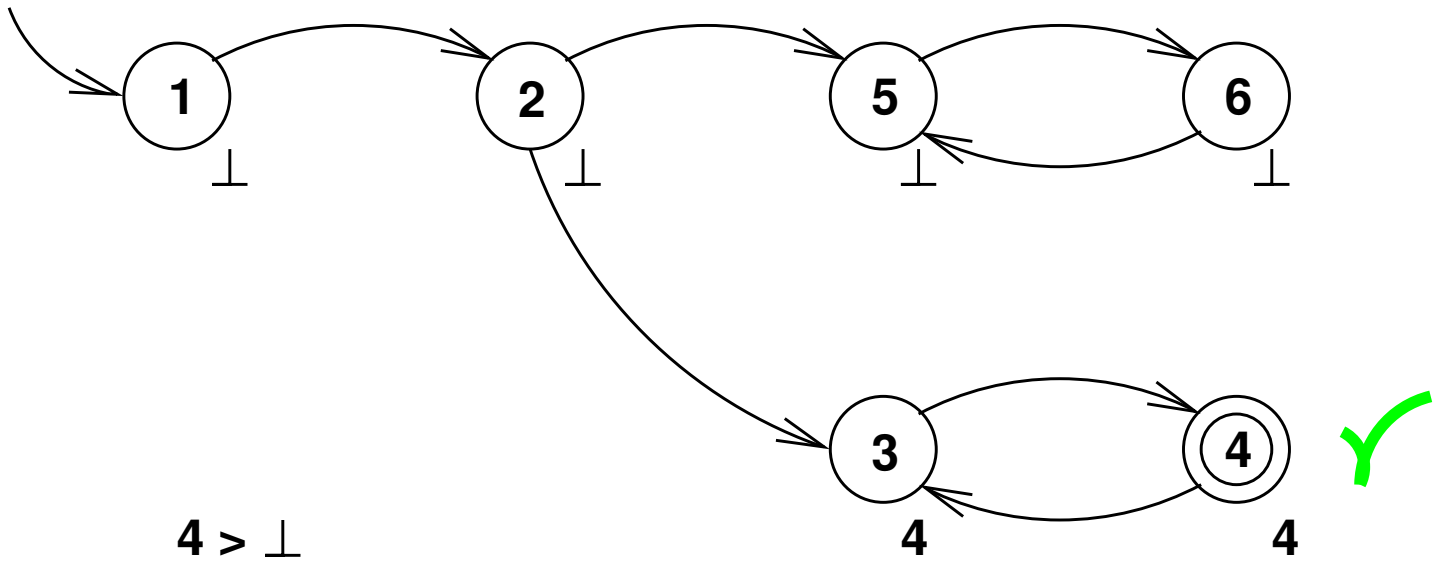
Maximální akceptující vrcholy označeny jako neakceptující.

Algoritmus MAP



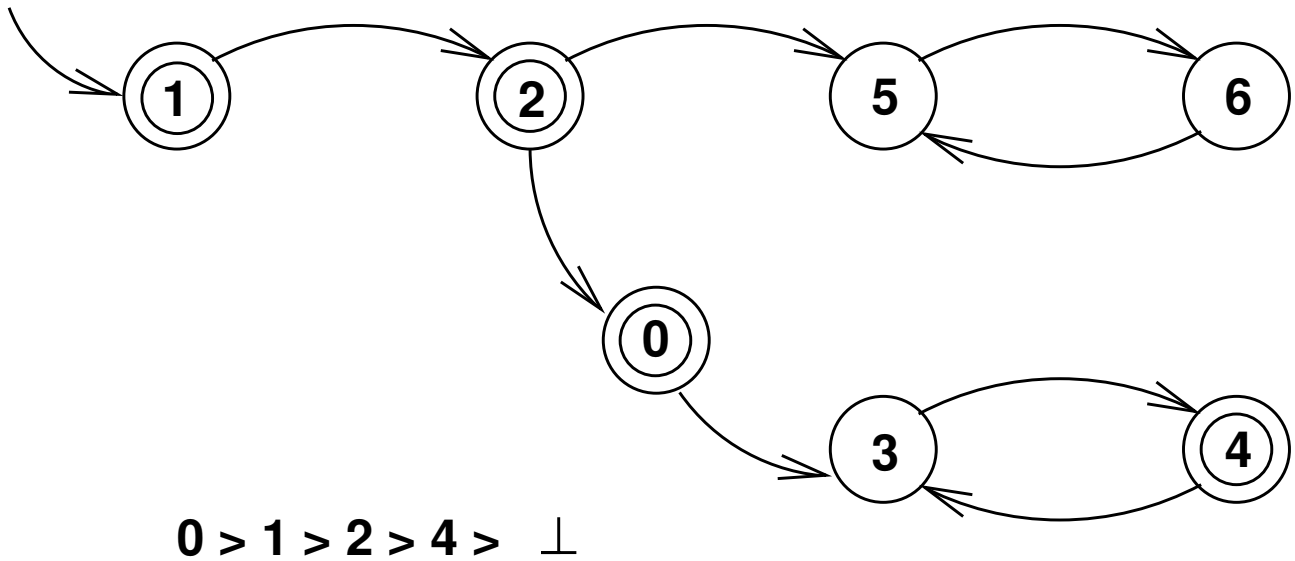
Opakuje se dokud není nalezen akceptující cyklus, nebo se odznačí všechny akceptující vrcholy.

Algoritmus MAP



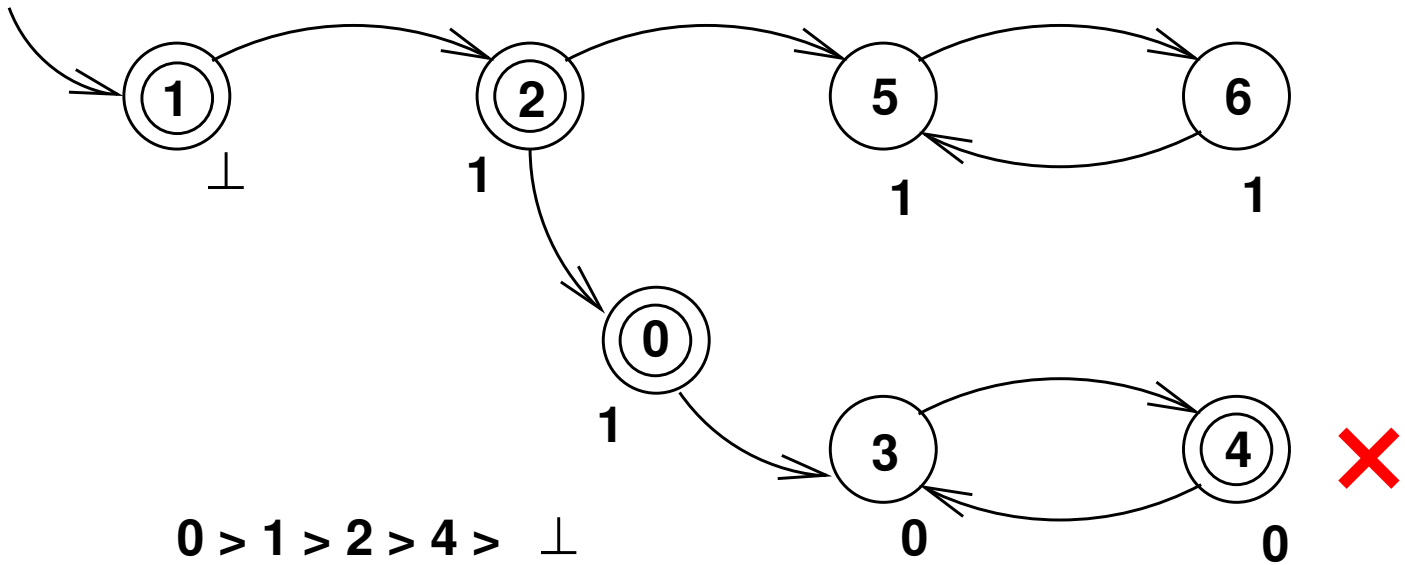
Následné iterace algoritmu mohou být omezeny na podgrafy se stejnou hodnotou funkce MAP.

Algoritmus MAP – podgrafy



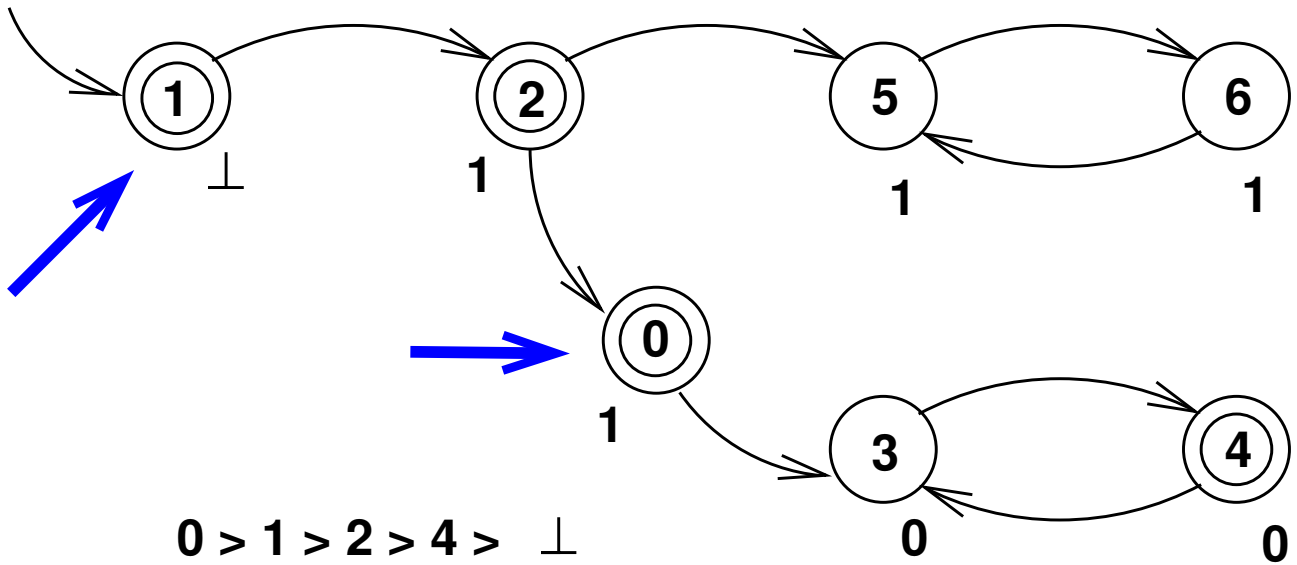
Ukázka dekompozice na podgrafy.

Algoritmus MAP – podgrafy



První iterace.
Nenalezen akceptující cyklus.

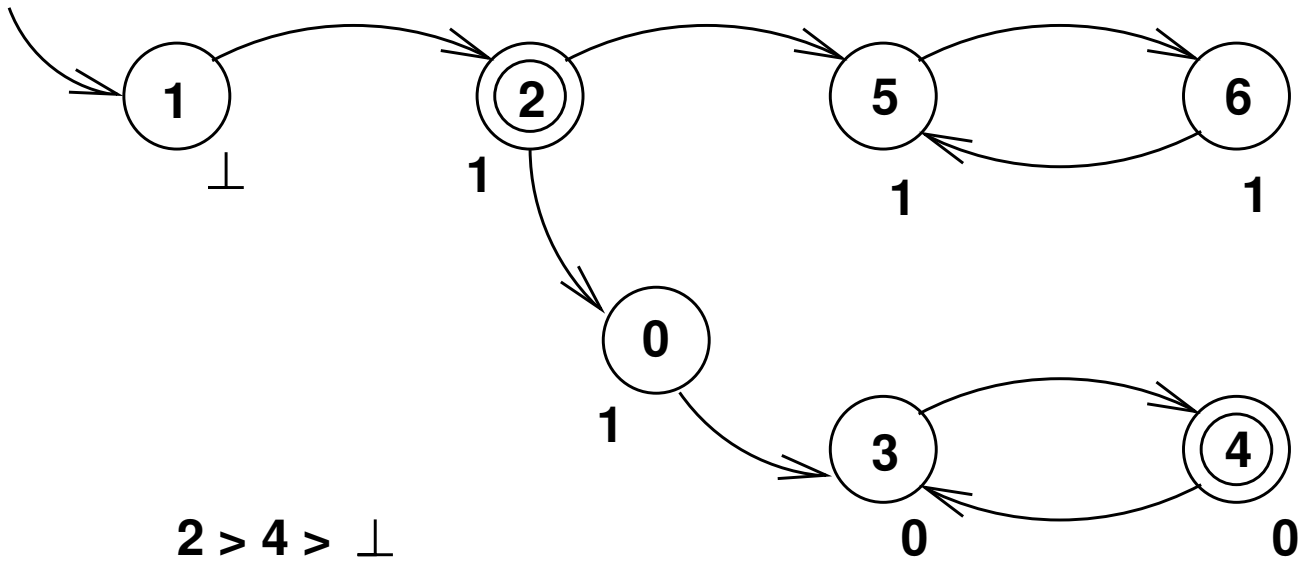
Algoritmus MAP – podgrafy



Vedoucí maximální vrcholy označeny jako neakceptující.

Akceptující vrchol v je vedoucí, pokud $\exists u : map(u) = v$.

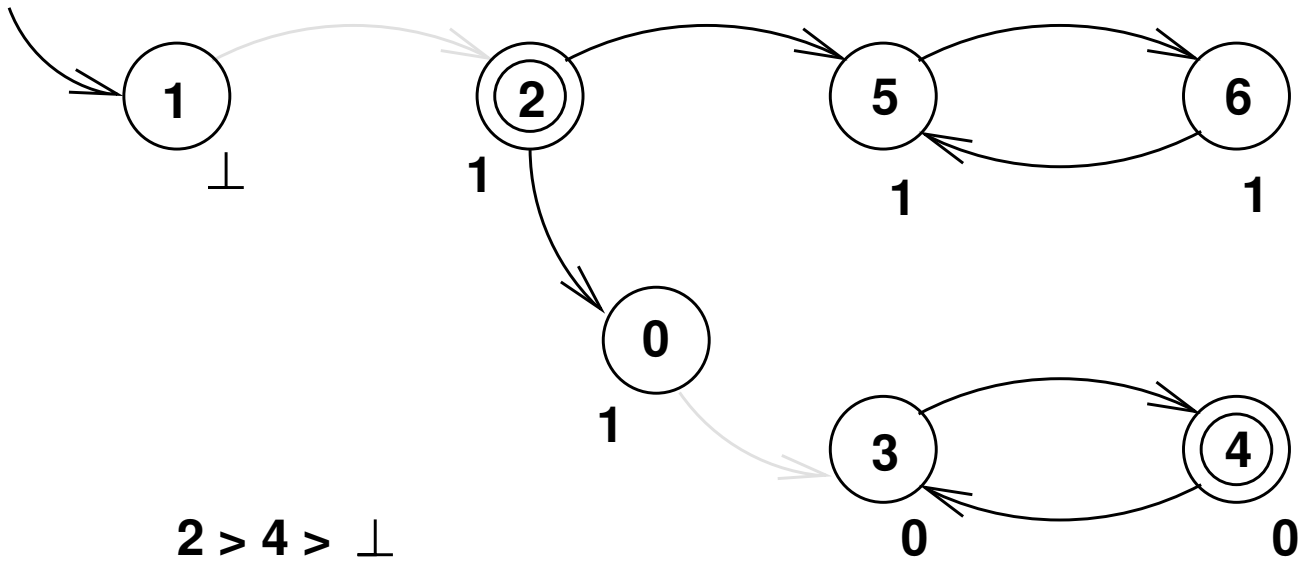
Algoritmus MAP – podgrafy



Vedoucí maximální vrcholy označeny jako neakceptující.

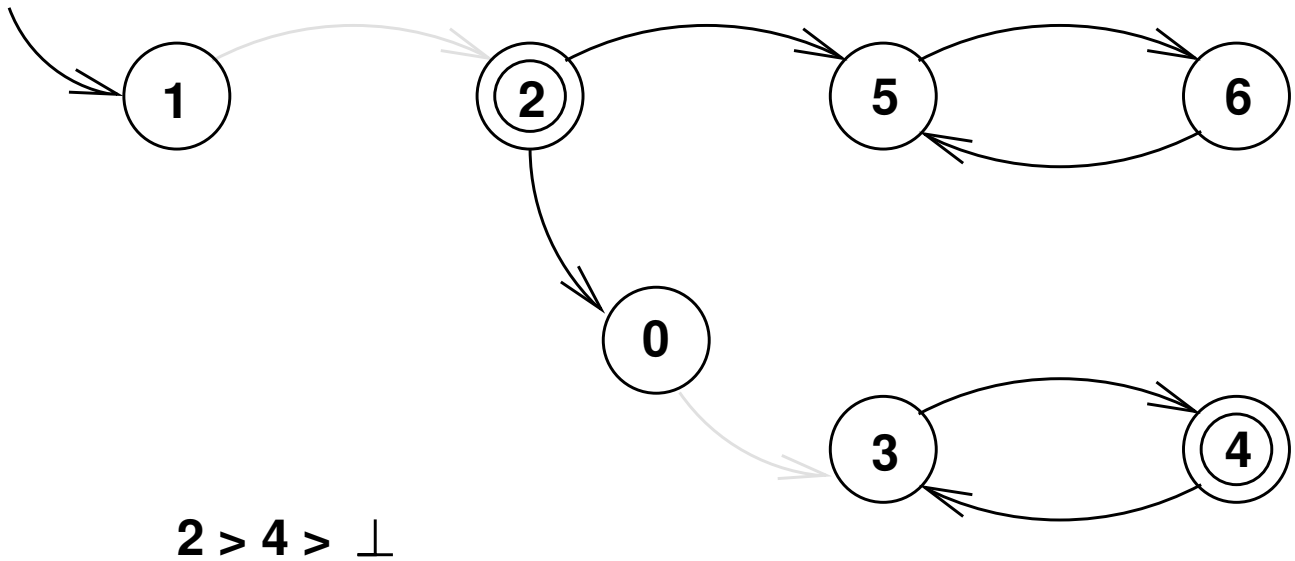
Akceptující vrchol v je vedoucí, pokud $\exists u : map(u) = v$.

Algoritmus MAP – podgrafy



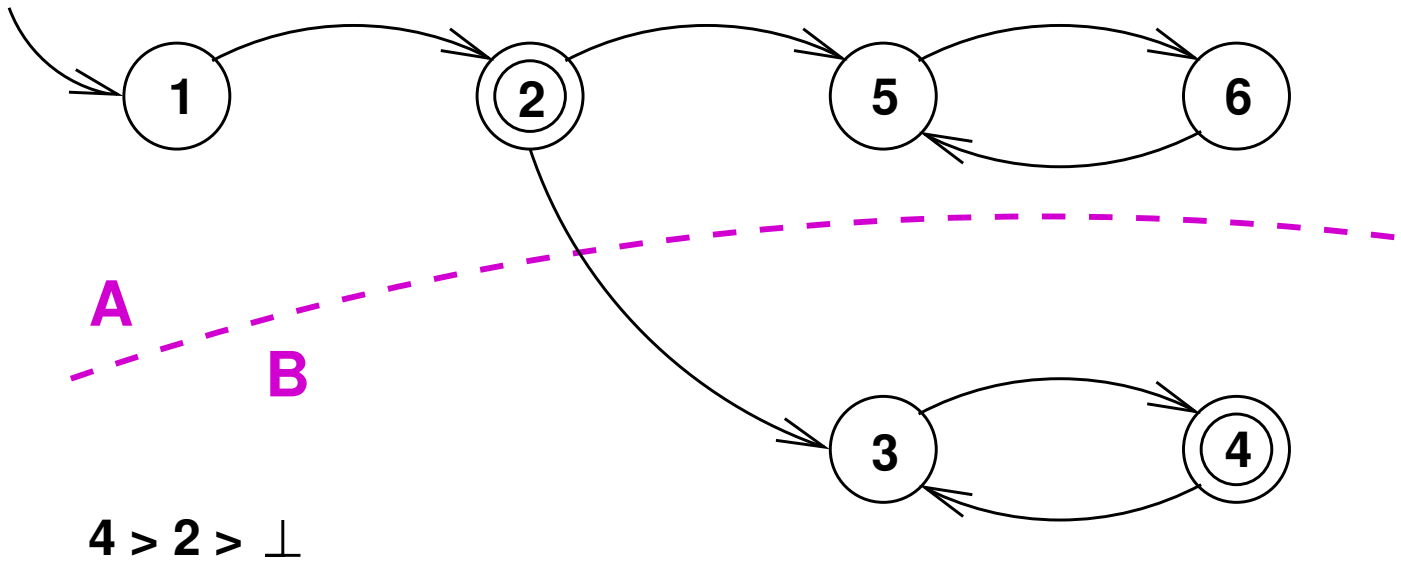
Dekompozice na podgrafy podle hodnot funkce MAP.
V následující iteraci jsou hodnoty uloženy jako oldmap hodnoty.

Algoritmus MAP – podgrafy



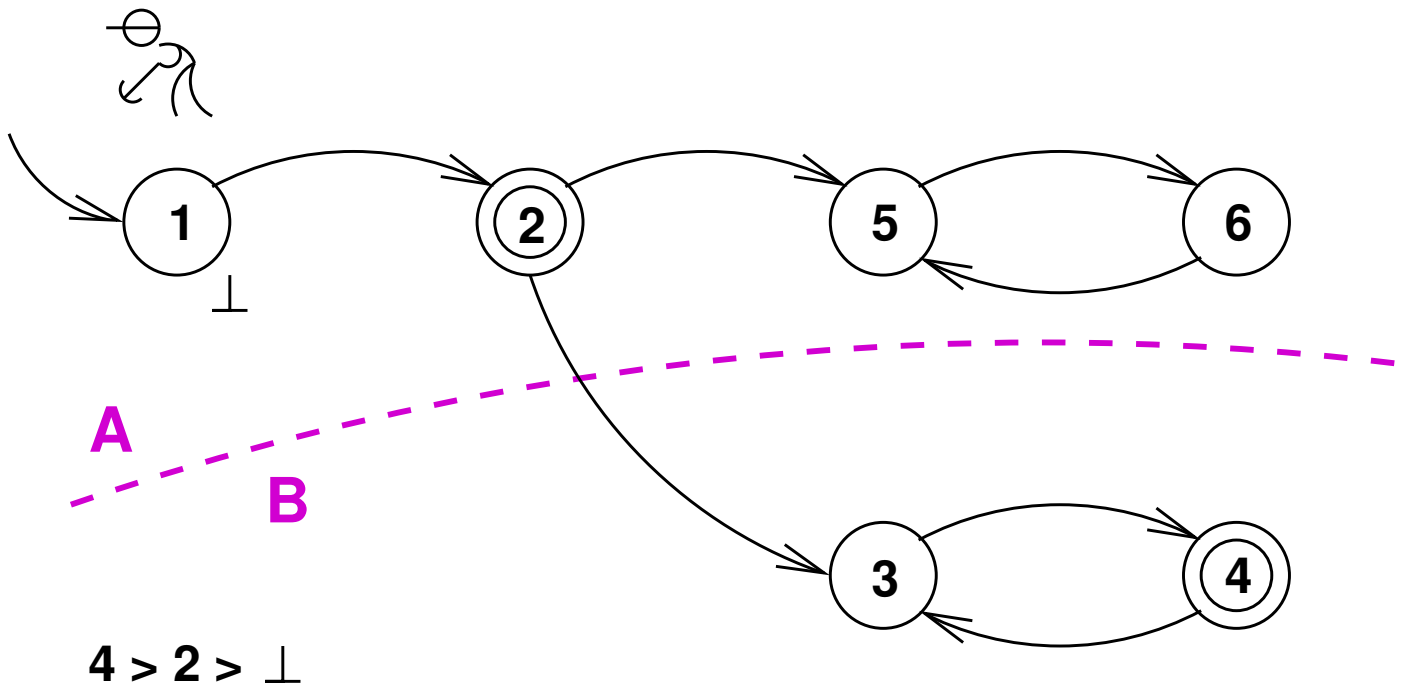
2 > 4 > $\perp$

Výpočet map hodnot – malé množství jader



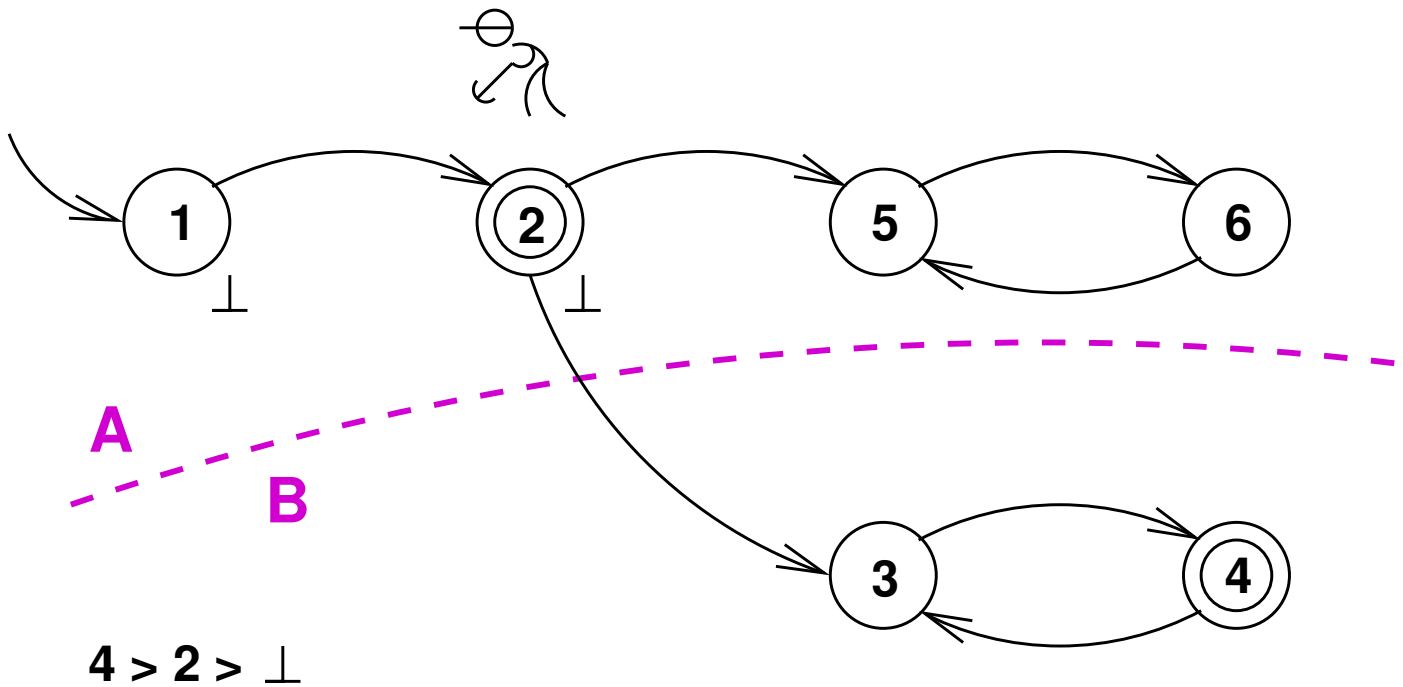
Rozdělení grafu.

Výpočet map hodnot – malé množství jader



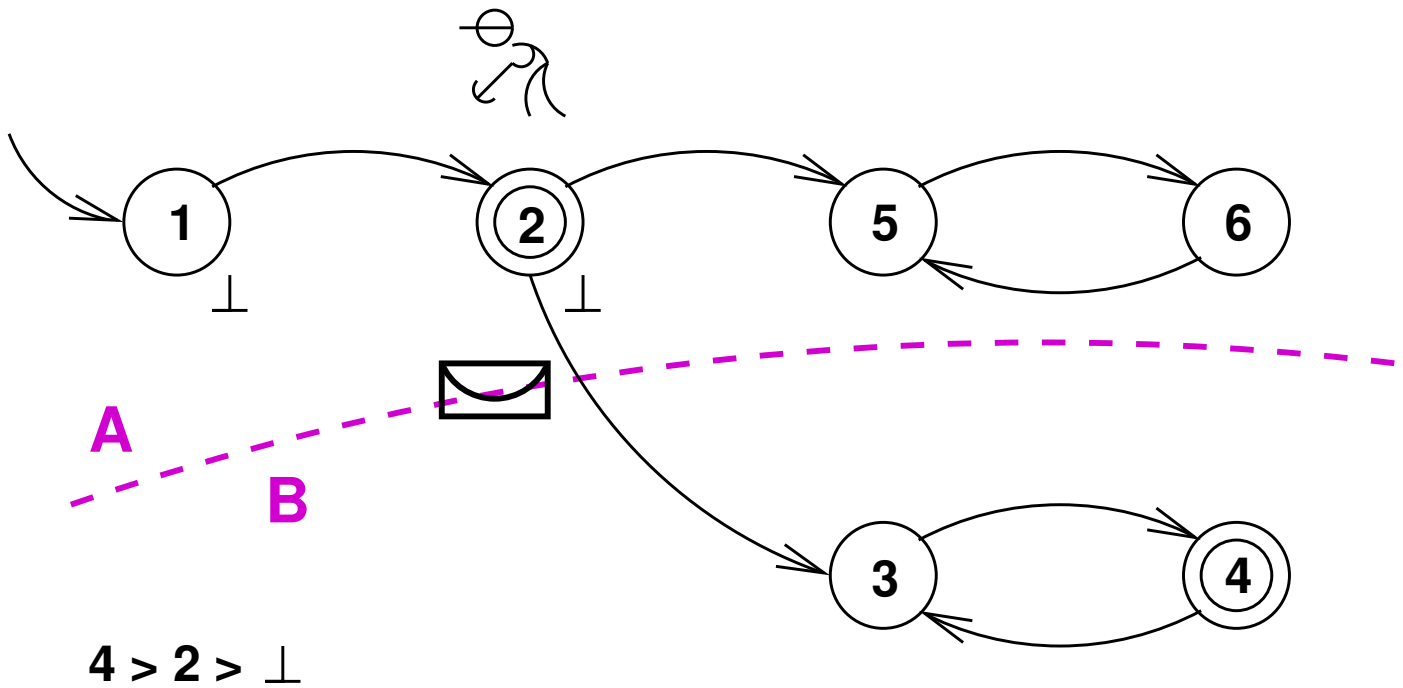
Každé jádro zpracovává svoji část.

Výpočet map hodnot – malé množství jader



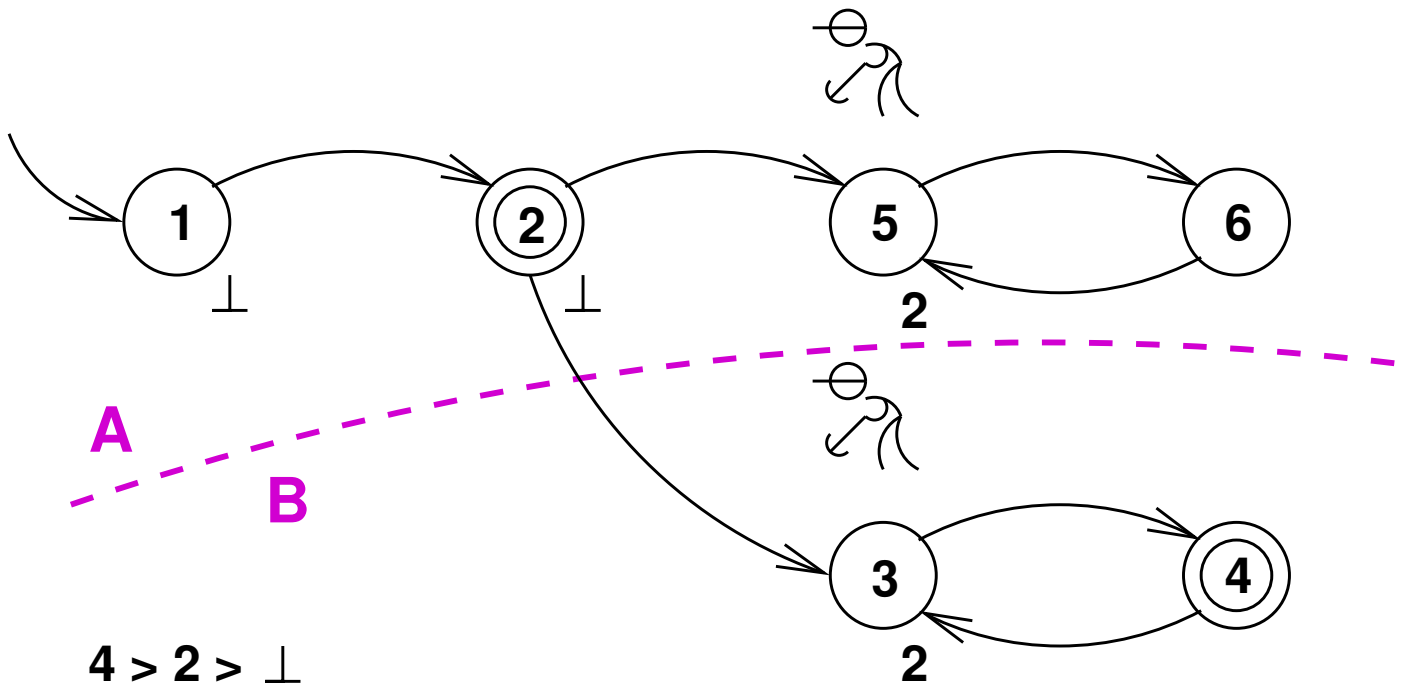
Každé jádro zpracovává svoji část.

Výpočet map hodnot – malé množství jader



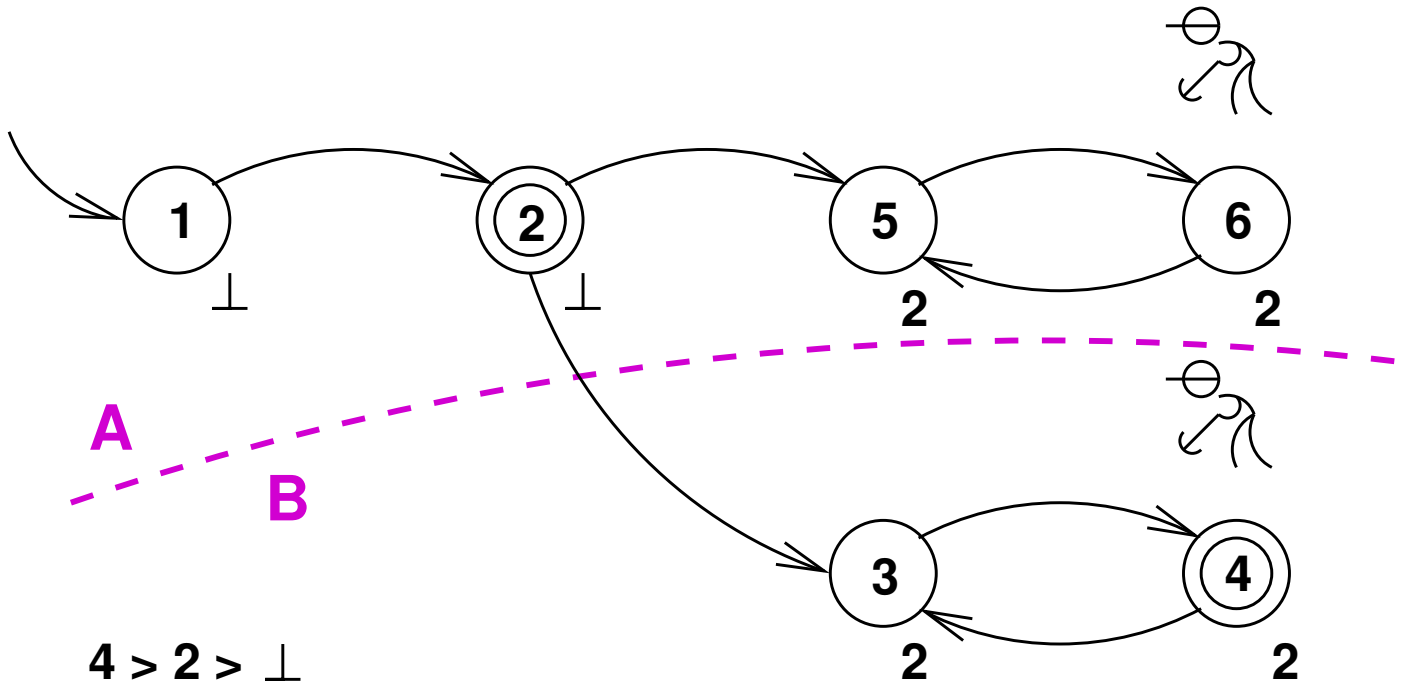
Nelokální vrcholy jsou poslány jádrům, které je mají zpracovat.

Výpočet map hodnot – malé množství jader



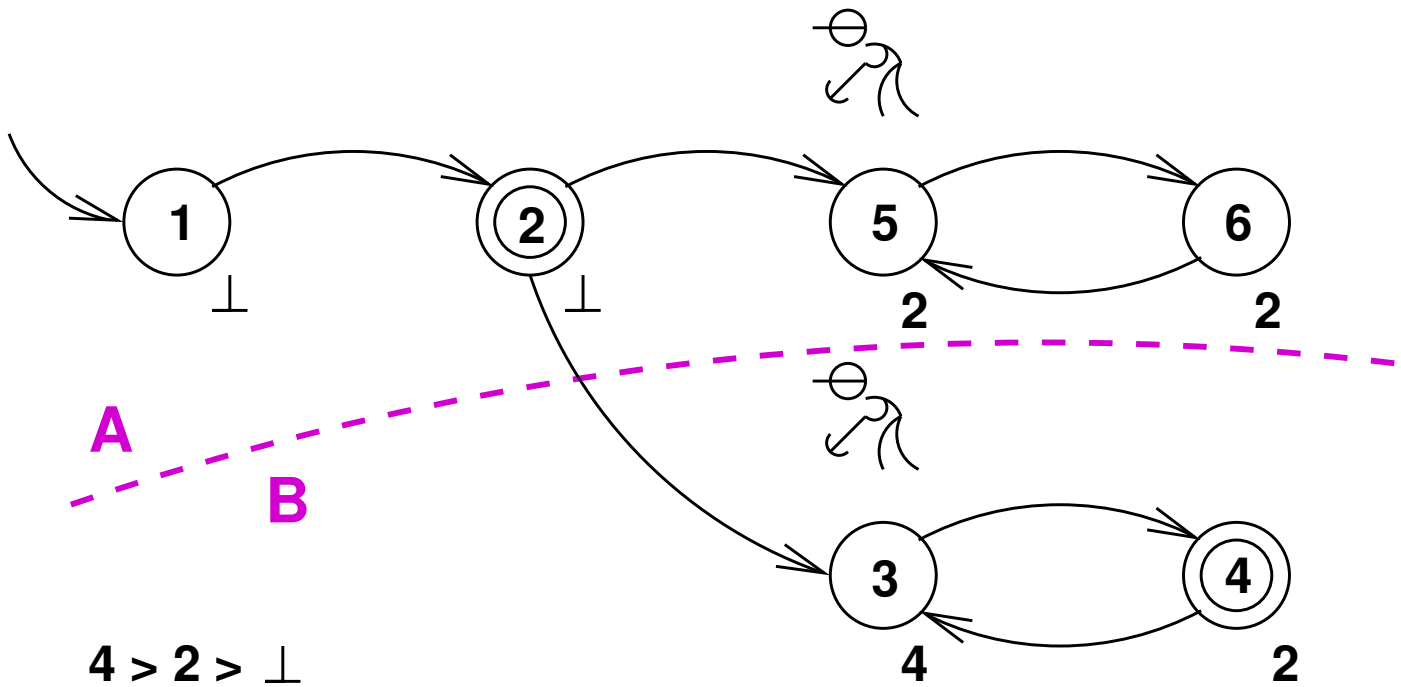
Graf je zpracováván paralelně.

Výpočet map hodnot – malé množství jader



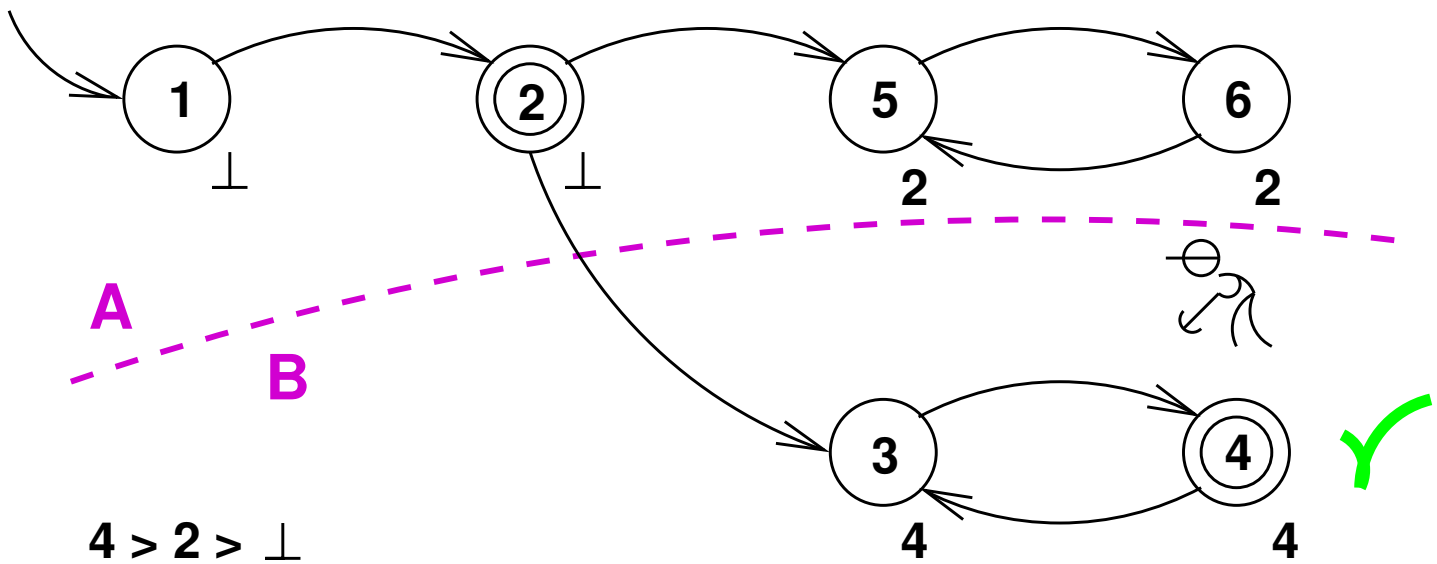
Graf je zpracováván paralelně.

Výpočet map hodnot – malé množství jader



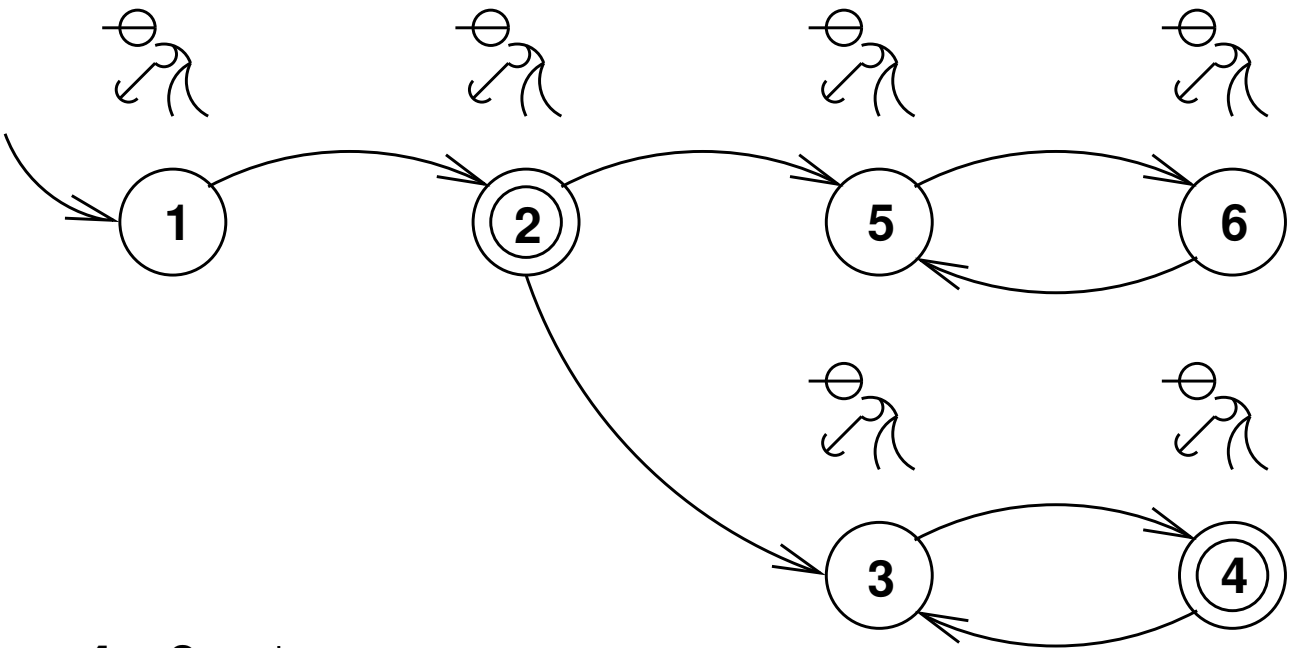
Graf je zpracováván paralelně.

Výpočet map hodnot – malé množství jader



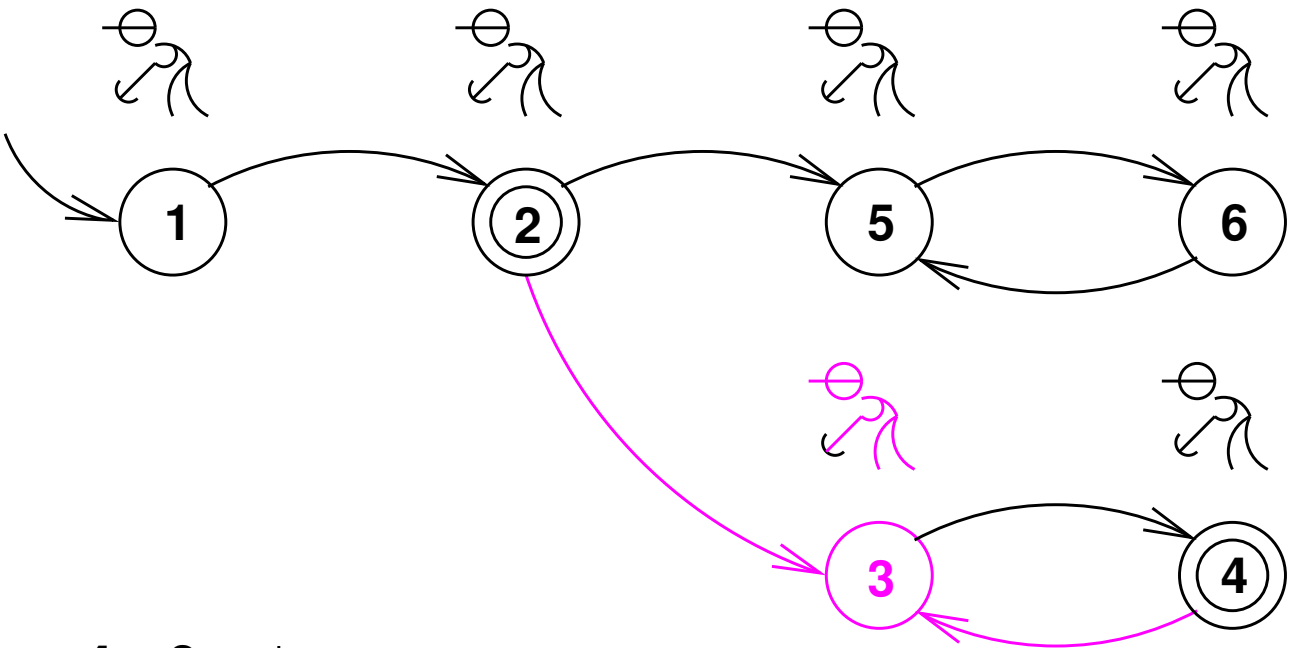
Dokud není nalezen akceptující cyklus.

Výpočet map hodnot – velké množství jader



Každému vrcholu přiřazeno jedno vlákno.

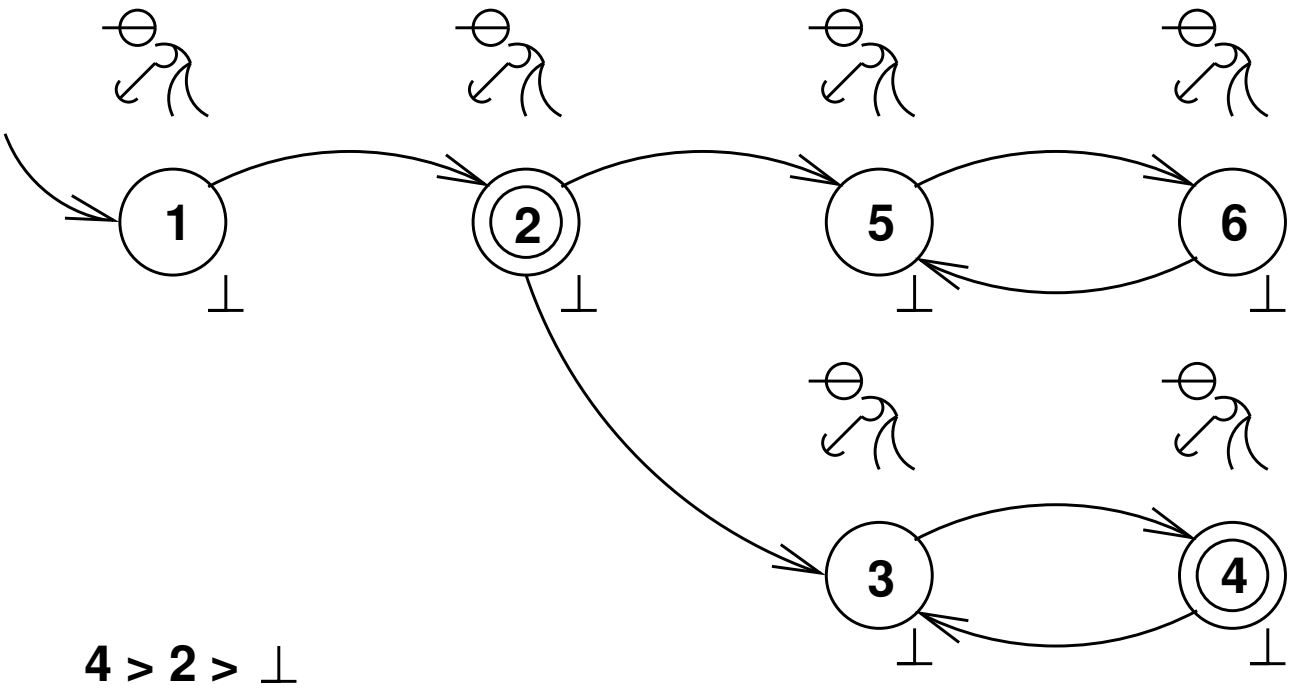
Výpočet map hodnot – velké množství jader



4 > 2 > $\perp$

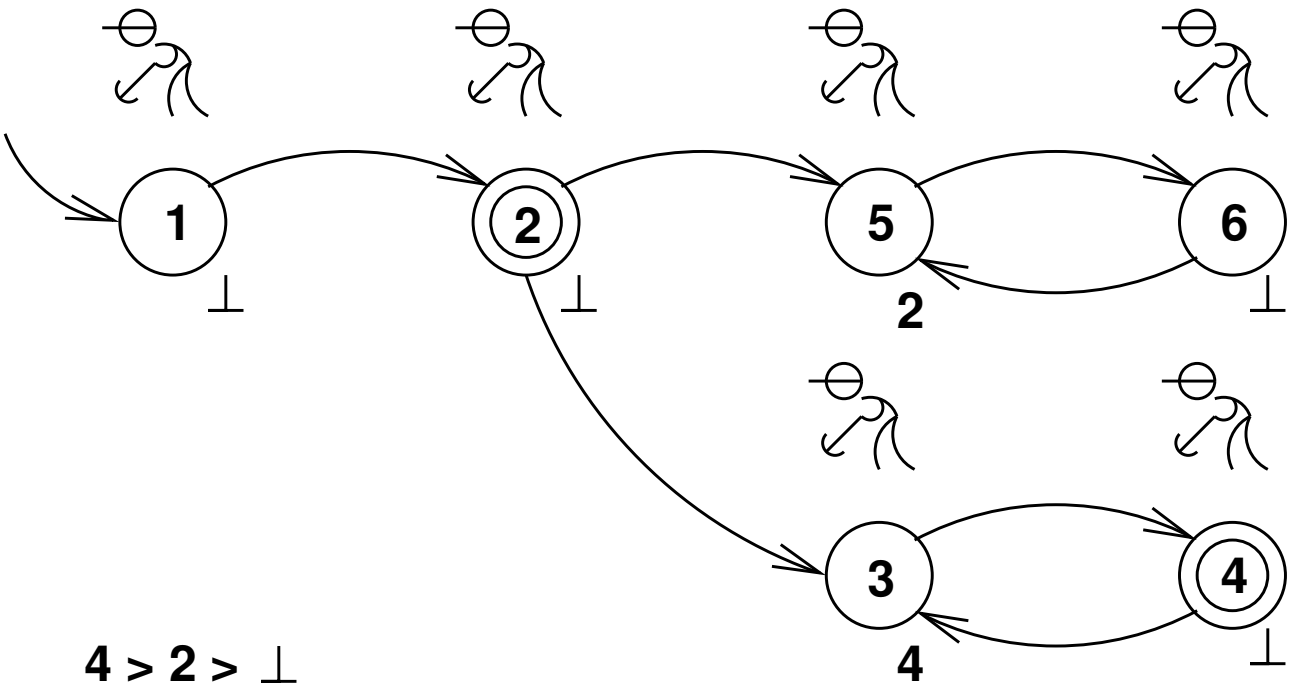
Vlákno zpracovává všechny příchozí hrany.

Výpočet map hodnot – velké množství jader



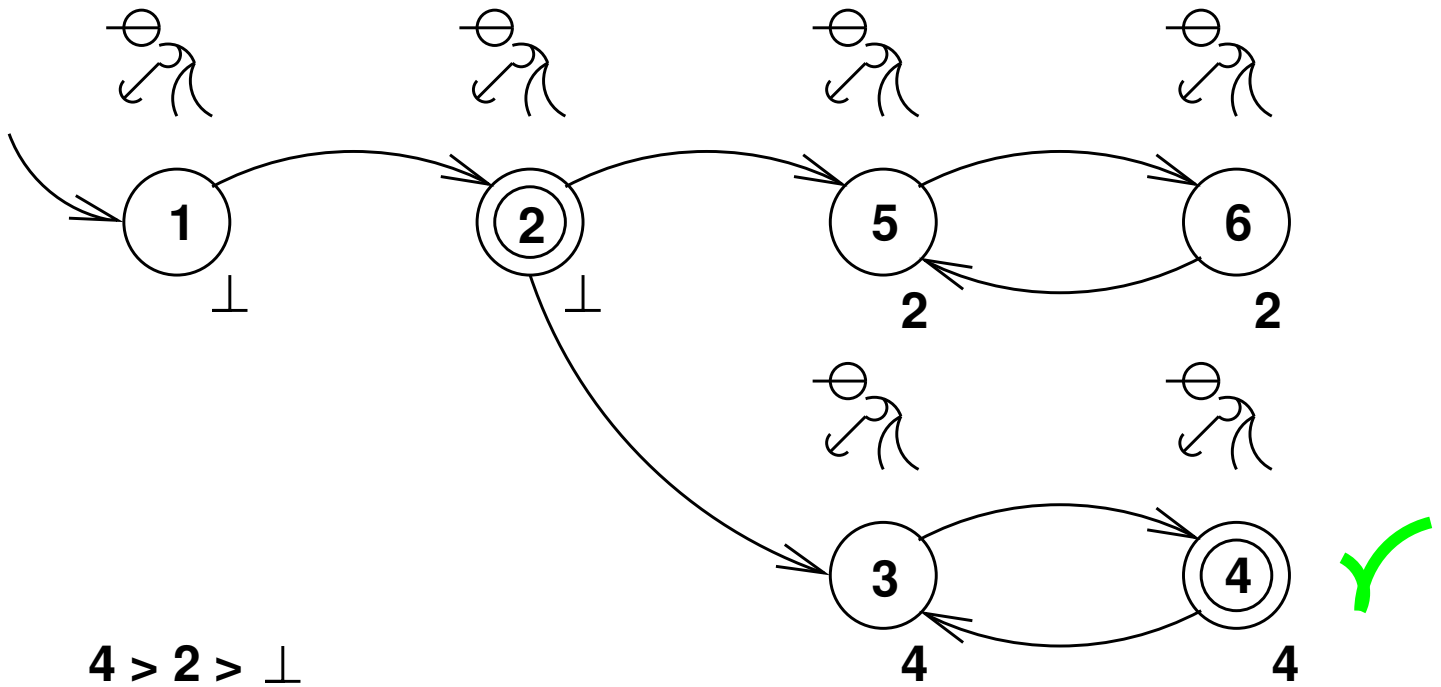
Vlákna pracují souběžně.

Výpočet map hodnot – velké množství jader



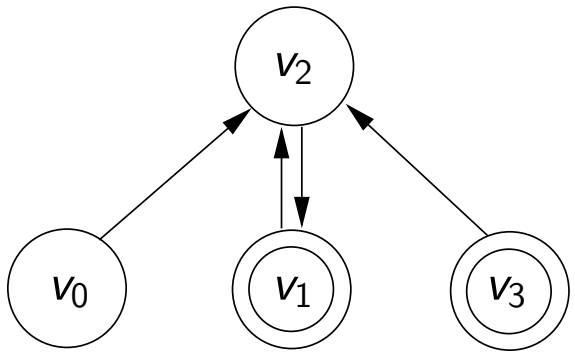
Vlákna pracují souběžně.

Výpočet map hodnot – velké množství jader



Vlákna pracují souběžně.

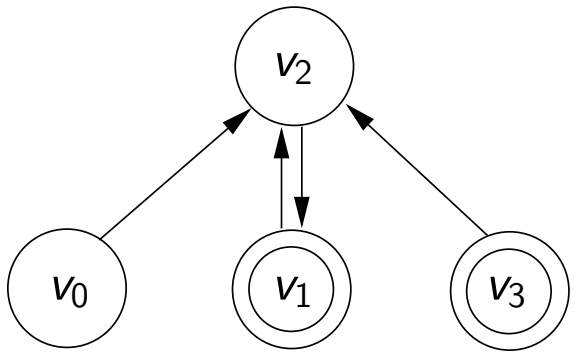
Algoritmus MAP jako součin matice s vektorem



$$
\begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \end{array}
\begin{array}{c}
\begin{array}{cccc} v_0 & v_1 & v_2 & v_3 \end{array} \\
\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}
\end{array}
$$

Matice předchůdců

Algoritmus MAP jako součin matice s vektorem



$$\begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{cccc} v_0 & v_1 & v_2 & v_3 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

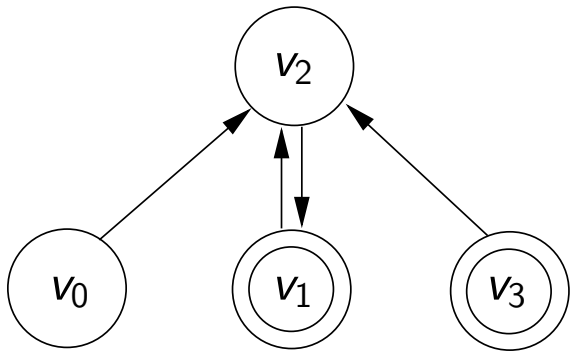
Matice předchůdců

$$\vec{m}$$
$$\bot$$
$$\bot$$
$$\bot$$
$$\bot$$

Přídavná data na vrchol

$\vec{m}$ – map hodnoty

Algoritmus MAP jako součin matice s vektorem



$$\begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{cccc} v_0 & v_1 & v_2 & v_3 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

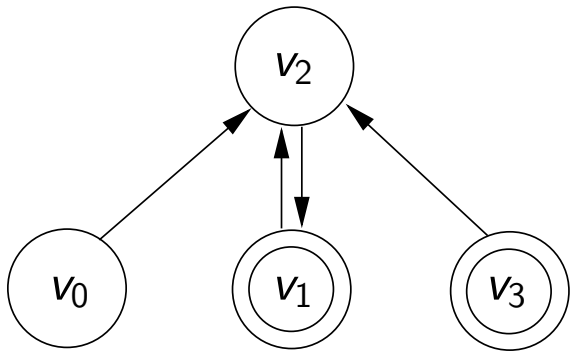
Matice předchůdců

$$\begin{array}{cc} \vec{m} & \vec{o} \\ \bot & \bot \\ \bot & \bot \\ \bot & \bot \\ \bot & \bot \end{array}$$

Přídavná data na vrchol

$\vec{o}$ – oldmap hodnoty

Algoritmus MAP jako součin matice s vektorem



$$\begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{cccc} v_0 & v_1 & v_2 & v_3 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

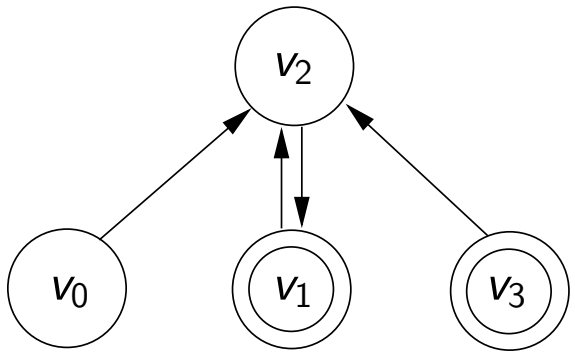
Matice předchůdců

$\vec{m}$	$\vec{o}$	$\vec{\mathcal{A}}$
$\perp$	$\perp$	0
$\perp$	$\perp$	1
$\perp$	$\perp$	0
$\perp$	$\perp$	1

Přídavná data na vrchol

$\vec{\mathcal{A}}$ – akceptující vrcholy

Algoritmus MAP jako součin matice s vektorem



$$\begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{cccc} v_0 & v_1 & v_2 & v_3 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

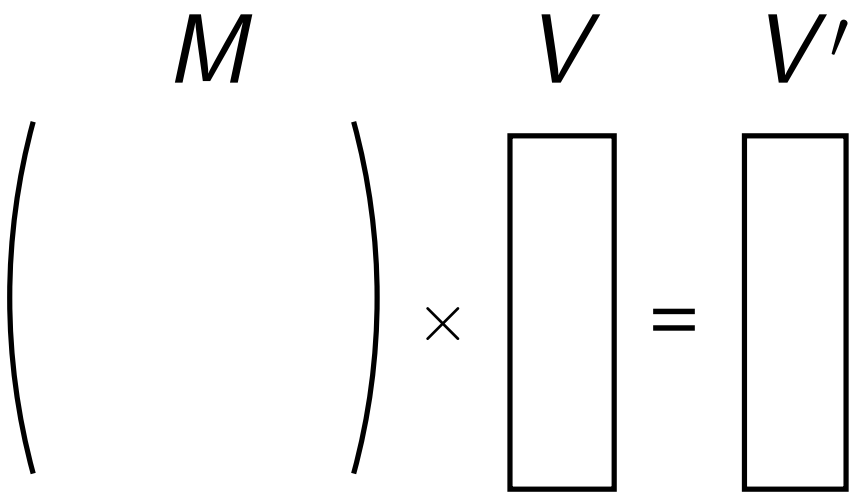
Matice předchůdců

$\vec{m}$	$\vec{o}$	$\vec{A}$	$\vec{r}$
$\perp$	$\perp$	0	0
$\perp$	$\perp$	1	0
$\perp$	$\perp$	0	0
$\perp$	$\perp$	1	0

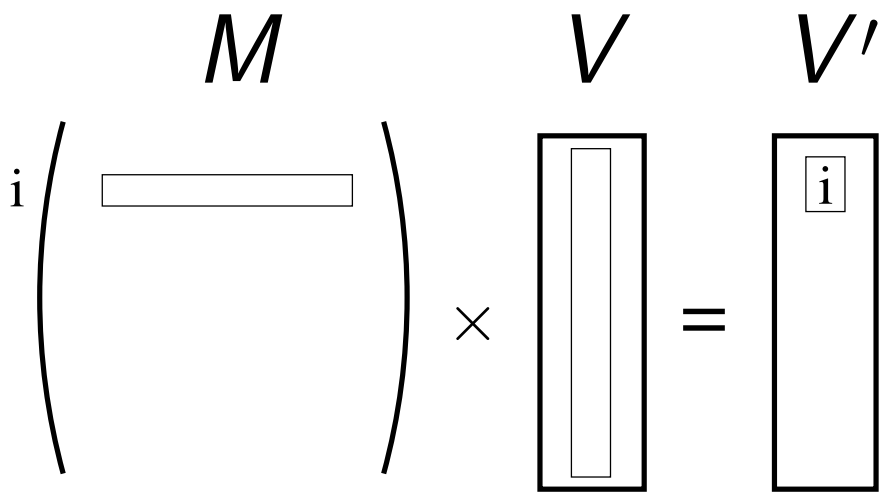
Přídavná data na vrchol

$\vec{r}$ – příznak modifikace

Algoritmus MAP jako součin matice s vektorem

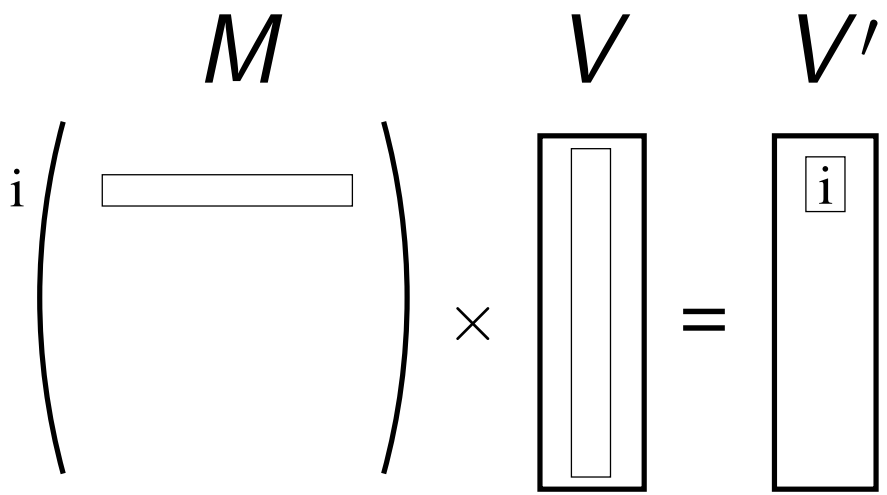


Algoritmus MAP jako součin matice s vektorem



$$V'[i] = \sum_{0 \leq j \leq n} M[i][j] \cdot V[j]$$

Algoritmus MAP jako součin matice s vektorem

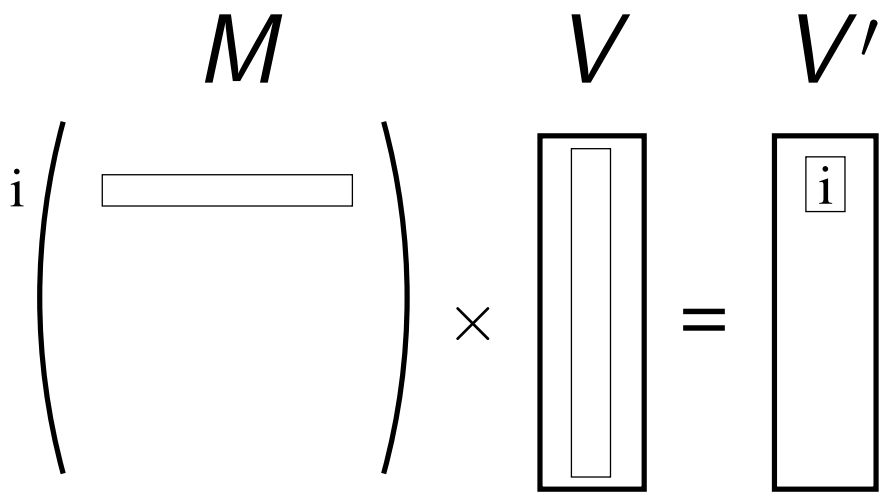


$$V'[i] = \sum_{0 \le j \le n} M[i][j] \cdot V[j]$$

$$V'[i] = \max_{0 \le j \le n} M[i][j] \cdot \text{maxacc(V[j],i)}$$

$$maxacc(u, v) = \begin{cases} \max\{map(u), map(v), u\} & \text{if } \mathcal{A}(u) \\ \max\{map(u), map(v)\} & \text{otherwise.} \end{cases}$$

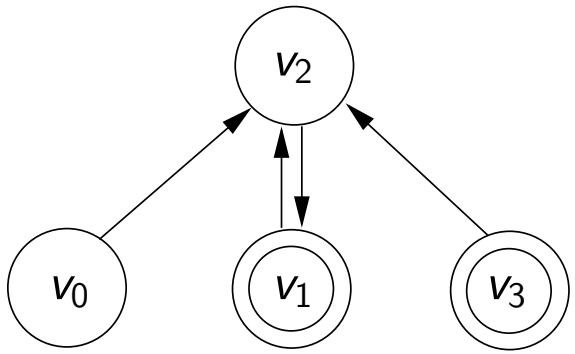
Algoritmus MAP jako součin matice s vektorem



$$
\begin{aligned}
V'[i] &= \sum_{0 \le j \le n} M[i][j] \cdot V[j] \\
V'[i] &= \max_{0 \le j \le n} M[i][j] \cdot \text{maxacc(V[j],i)} \\
V'[i] &= \max_{0 \le j \le n} M[i][j] \cdot \text{maxacc(V[j],i)} \cdot \text{old(V[j],i)}
\end{aligned}
$$

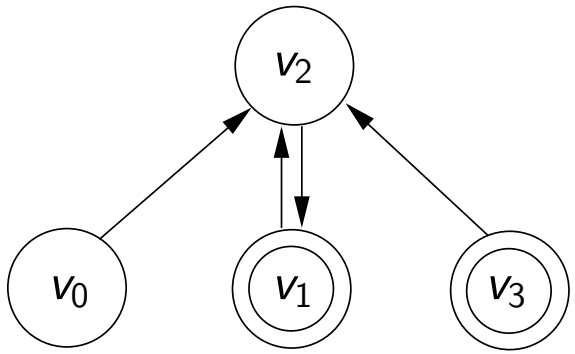
$$
old(u, v) = \begin{cases} 1 & \text{if } oldmap(u) = oldmap(v) \\ \bot & \text{otherwise.} \end{cases}
$$

Algoritmus MAP jako součin matice s vektorem — příklad



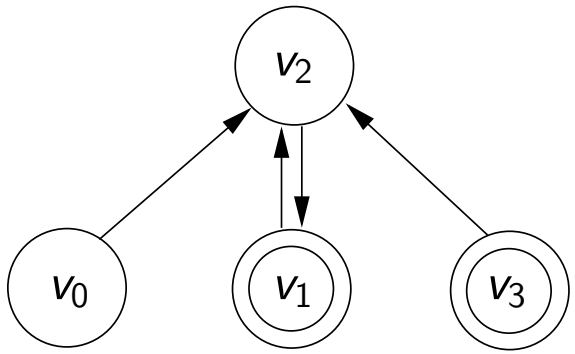
$$
\begin{array}{c} \\ v_2 \end{array}
\begin{array}{c} \begin{array}{cccc} v_0 & v_1 & v_2 & v_3 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}
\times
\begin{array}{cc} \vec{m} & \vec{o} \\ \bot & \bot \\ \bot & \bot \\ \bot & \bot \\ \bot & \bot \end{array}
\begin{array}{c} \vec{\mathcal{A}} \\ 0 \\ 1 \\ 0 \\ 1 \end{array}
=
\begin{array}{cc} \vec{m} & \vec{r} \\ \bot & 0 \\ \bot & 0 \\ \mathbf{3} & \mathbf{1} \\ \bot & 0 \end{array}
$$

Algoritmus MAP jako součin matice s vektorem — příklad



$$
\begin{array}{c} \\ v_1 \end{array}
\begin{array}{c} \begin{array}{cccc} v_0 & v_1 & v_2 & v_3 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}
\times
\begin{array}{ccc} \vec{m} & \vec{o} & \vec{\mathcal{A}} \\ \bot & \bot & 0 \\ \bot & \bot & 1 \\ \mathbf{3} & \bot & 0 \\ \bot & \bot & 1 \end{array}
=
\begin{array}{cc} \vec{m} & \vec{r} \\ \bot & 0 \\ \mathbf{3} & \mathbf{1} \\ 3 & \mathbf{0} \\ \bot & 0 \end{array}
$$

Algoritmus MAP jako součin matice s vektorem — příklad

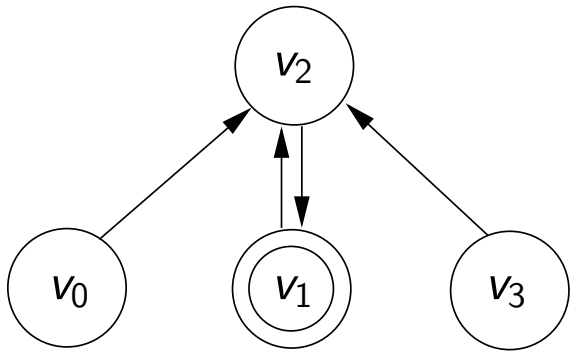


$$\begin{array}{c} \begin{matrix} v_0 & v_1 & v_2 & v_3 \end{matrix} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \times \begin{array}{ccc} \vec{m} & \vec{o} & \vec{\mathcal{A}} \\ \bot & \bot & 0 \\ \mathbf{3} & \bot & 1 \\ 3 & \bot & 0 \\ \bot & \bot & 1 \end{array} = \begin{array}{cc} \vec{m} & \vec{r} \\ \bot & 0 \\ 3 & \mathbf{0} \\ 3 & 0 \\ \bot & 0 \end{array}$$

Pevný bod.

Akceptující cyklus nenalezen.

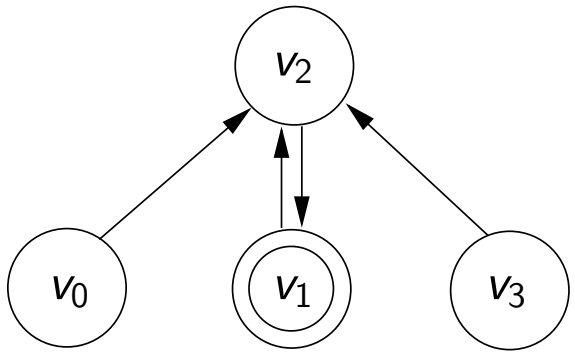
Algoritmus MAP jako součin matice s vektorem — příklad



m	o	$\mathcal{A}$		m	o	$\mathcal{A}$
$\perp$	$\perp$	0		$\perp$	$\perp$	0
3	$\perp$	1	$\Longrightarrow$	3	$\perp$	1
3	$\perp$	0		3	$\perp$	0
$\perp$	$\perp$	1		$\perp$	$\perp$	**0**

Vrchol v_3 označen jako neakceptující.

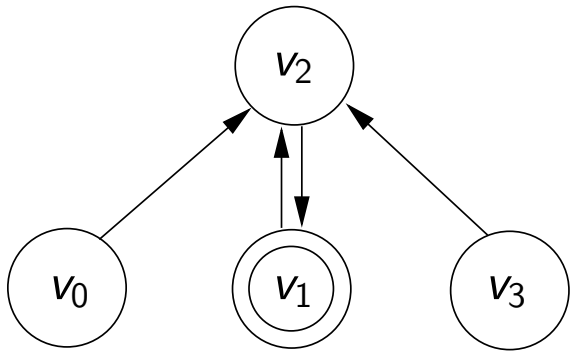
Algoritmus MAP jako součin matice s vektorem — příklad



m	o	$\mathcal{A}$		m	o	$\mathcal{A}$
$\perp$	$\perp$	0		$\perp$	$\perp$	0
3	$\perp$	1	$\Longrightarrow$	$\perp$	**3**	1
3	$\perp$	0		$\perp$	**3**	0
$\perp$	$\perp$	0		$\perp$	$\perp$	0

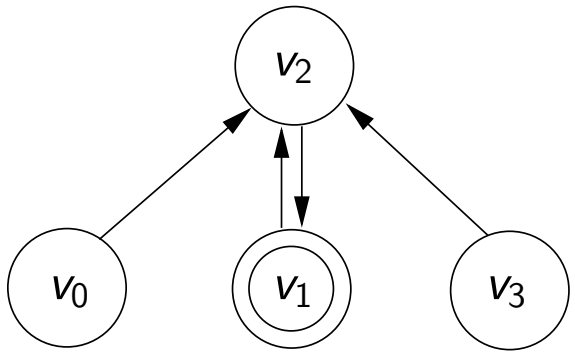
Rozdělení na podgrafy.

Algoritmus MAP jako součin matice s vektorem — příklad



$$
\begin{array}{c} \\ v_2 \end{array}
\begin{array}{c} \begin{array}{cccc} v_0 & v_1 & v_2 & v_3 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}
\times
\begin{array}{c} \begin{array}{ccc} \vec{m} & \vec{o} & \vec{\mathcal{A}} \end{array} \\ \begin{pmatrix} \bot & \bot & 0 \\ \bot & 3 & 1 \\ \bot & 3 & 0 \\ \bot & \bot & 0 \end{pmatrix} \end{array}
=
\begin{array}{c} \begin{array}{cc} \vec{m} & \vec{r} \end{array} \\ \begin{pmatrix} \bot & 0 \\ \bot & 0 \\ \mathbf{1} & \mathbf{1} \\ \bot & 0 \end{pmatrix} \end{array}
$$

Algoritmus MAP jako součin matice s vektorem — příklad



$$
\begin{array}{c} \\ v_1 \end{array}
\begin{array}{c} \begin{array}{cccc} v_0 & v_1 & v_2 & v_3 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}
\times
\begin{array}{c} \begin{array}{ccc} \vec{m} & \vec{o} & \vec{\mathcal{A}} \end{array} \\ \begin{pmatrix} \bot & \bot & 0 \\ \bot & 3 & 1 \\ 1 & 3 & 0 \\ \bot & \bot & 0 \end{pmatrix} \end{array}
=
\begin{array}{c} \begin{array}{cc} \vec{m} & \vec{r} \end{array} \\ \begin{pmatrix} \bot & 0 \\ \mathbf{1} & \mathbf{1} \\ 1 & \mathbf{0} \\ \bot & 0 \end{pmatrix} \end{array}
$$

Nalezen akceptující cyklus $map(v1) = v1$

Reprezentace a výpočet grafu

Reprezentace grafu

Generování stavového prostoru

- Graf je zadán implicitně zdrojovým kódem programu.
- Explicitní (maticovou) reprezentaci grafu, je třeba vypočítat.
- Orientovaný, **řídký** graf.
- Zpětné hrany nelze vypočítat (kvůli příkazu přiřazení).

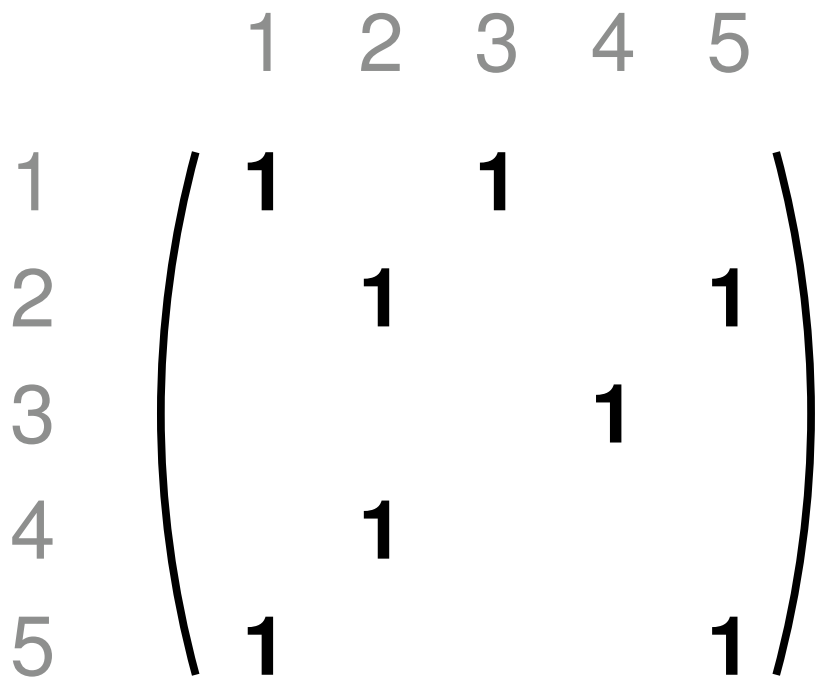
Reprezentace grafu

- Matice souslednosti – prostorově náročná reprezentace.
- Matice obsahuje pouze hodnoty 0 a 1.
- Zjednodušený **Compressed Sparse Row** (CSR) formát.

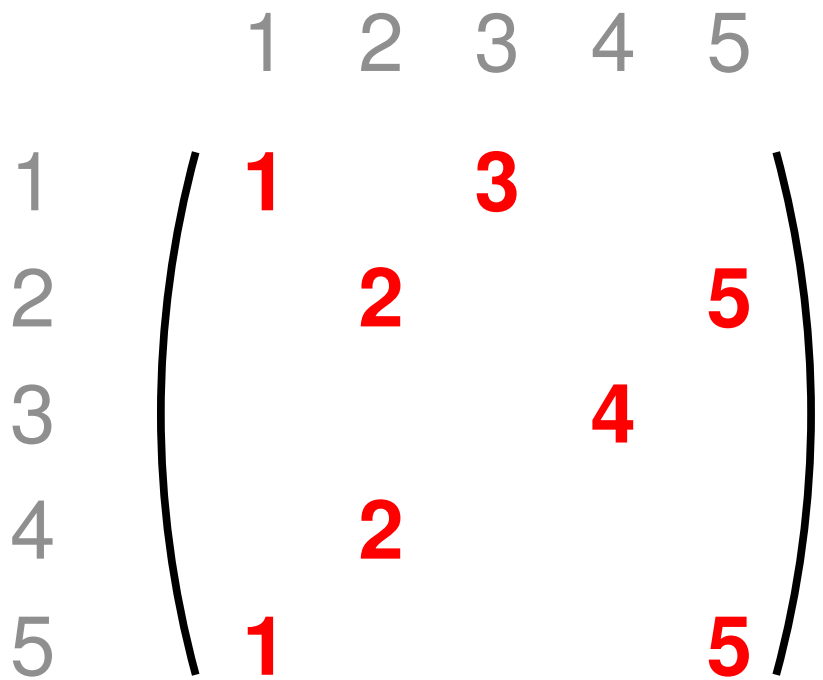
CSR reprezentace

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{pmatrix}
\mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\
\mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1}
\end{pmatrix}
\end{array}
$$

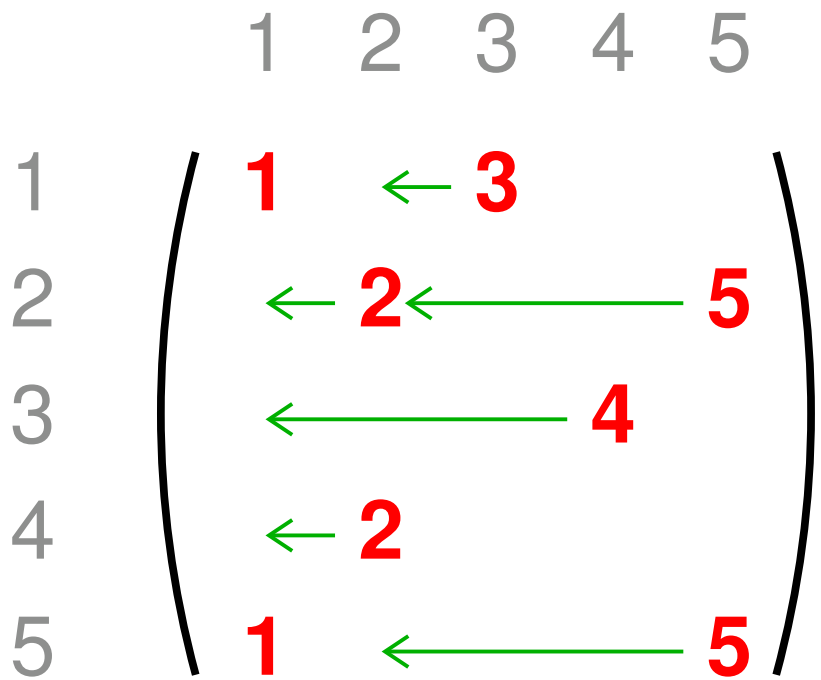
CSR reprezentace



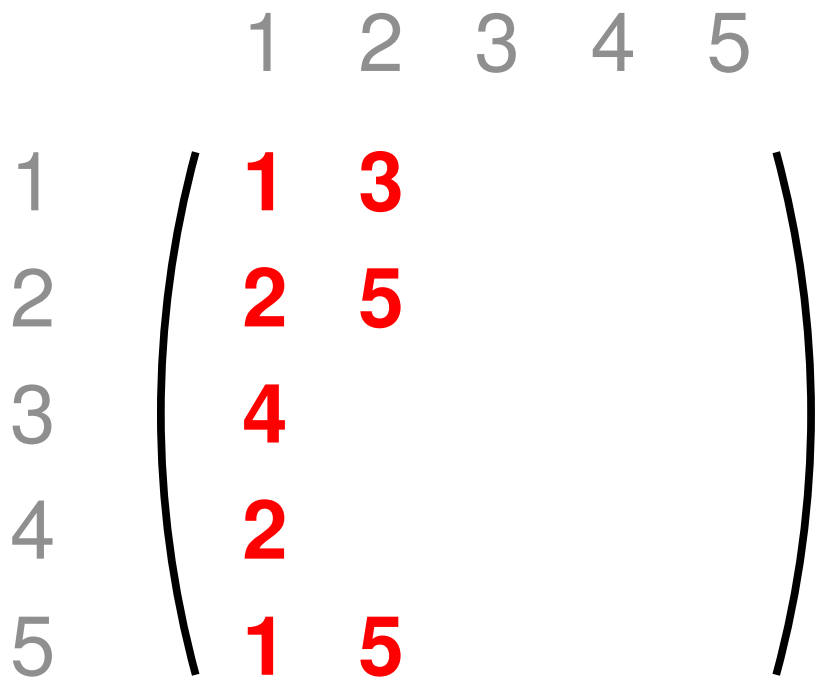
CSR reprezentace



CSR reprezentace



CSR reprezentace



CSR reprezentace

1 3

2 5

4

2

1 5

CSR reprezentace

Mc=(1 3)

2 5

4

2

1 5

CSR reprezentace

4

2

1 5

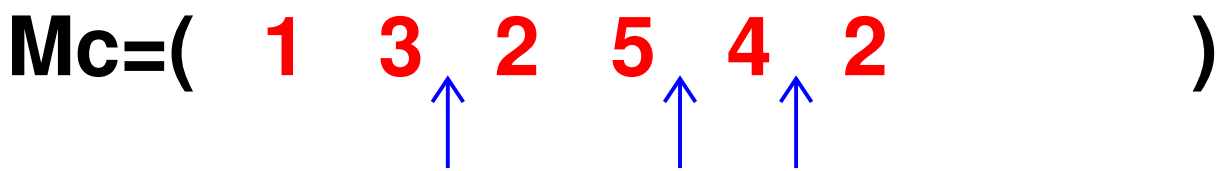
CSR reprezentace

Mc=(1 3 2 5 4)

2

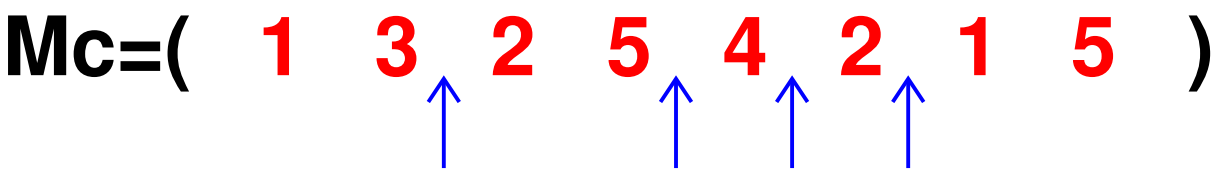
1 5

CSR reprezentace

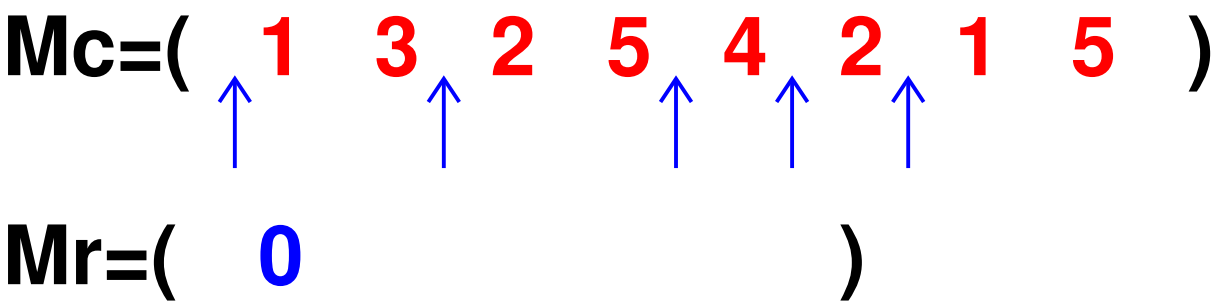


1 5

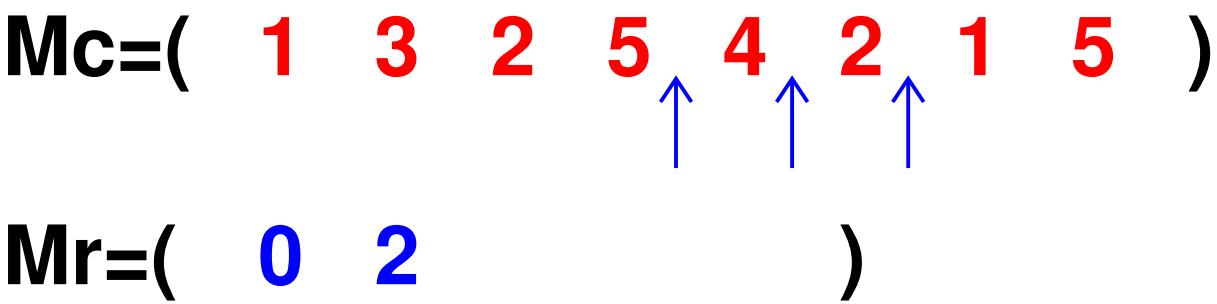
CSR reprezentace



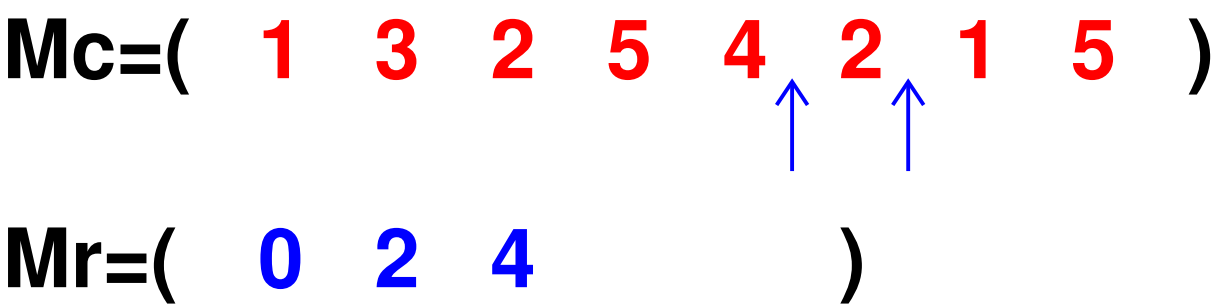
CSR reprezentace



CSR reprezentace



CSR reprezentace



CSR reprezentace

Mc=(1 3 2 5 4 2 1 5)

Mr=(0 2 4 5 6)

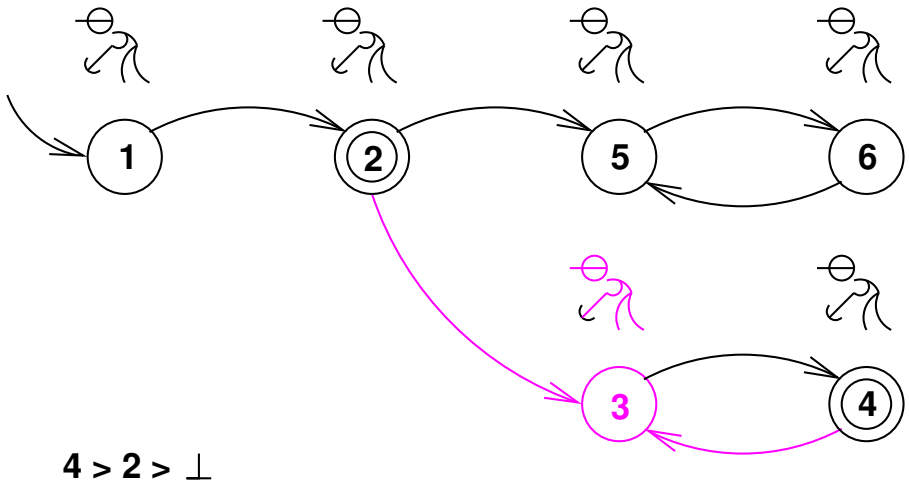
CSR reprezentace

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Mc=(1 3 2 5 4 2 1 5)
Mr=(0 2 4 5 6)

CSR reprezentace a algoritmus MAP

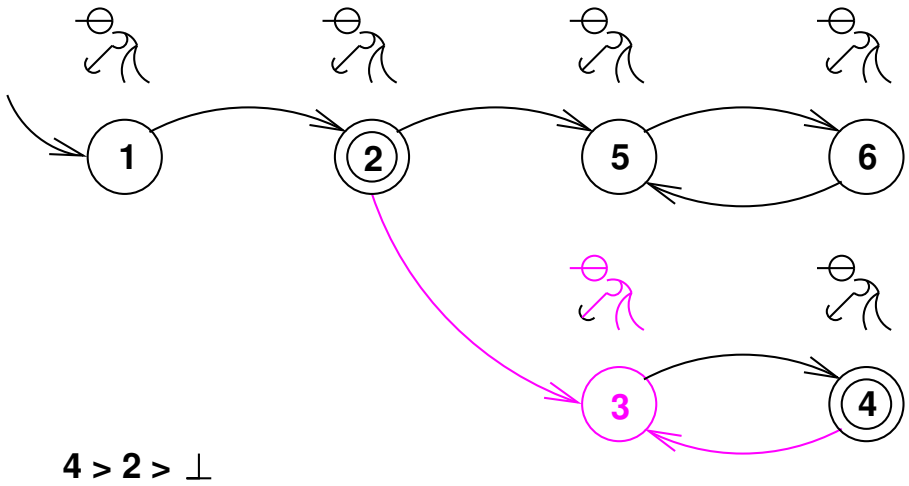
MAP vynucuje enumeraci předchůdců



CSR a MAP problém

- Enumerace předchůdců v CSR formátu znamená prohledání celé CSR reprezentace grafu.
- Pozorování: Enumerace následníků je v CSR reprezentaci snadná (následnící vrcholu uloženy v jednom řádku matice).

MAP vynucuje enumeraci předchůdců

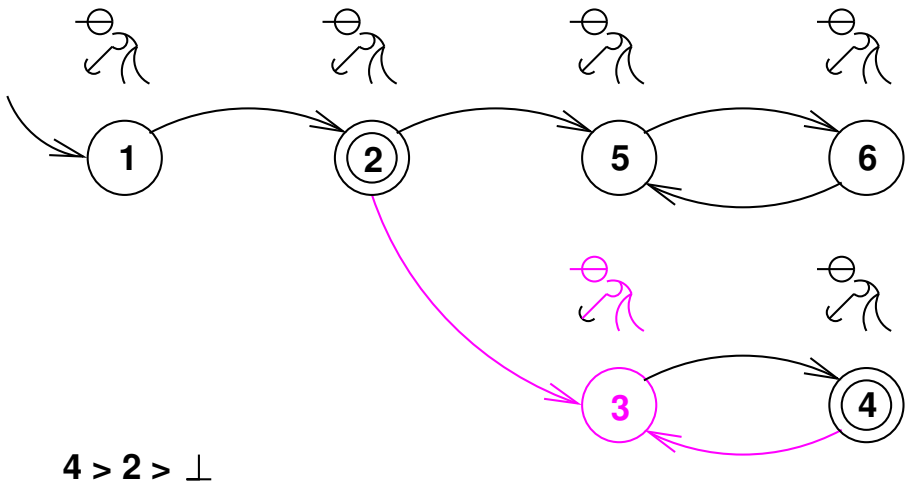


CSR a MAP řešení #1

- Vlákna místo "stahování" hodnot ostatních vrcholů směrem k sobě, "nahrávají" své hodnoty k sousedním vrcholům.
- Problém souběžného zápisu na cílových vrcholech.

CSR reprezentace a algoritmus MAP

MAP vynucuje enumeraci předchůdců



CSR a MAP řešení #2

- Transpozice grafu je zbytečná práce.
- Je možné transponovat "algoritmus":
  MAP $\longrightarrow$ MAS (Maximální akceptující následník.)
- MAS vyžaduje efektivní enumeraci následníků.

Výpočet CSR reprezentace

Pozorování

- CSR reprezentaci je třeba vypočítat.
- Část výpočtu není akcelerována pomocí GP GPU.
- Negativní dopad na celkové zrychlení dosažené použitím technologie CUDA.

Paralelizace výpočtu CSR reprezentace

- CSR je možné počítat paralelním CPU algoritmem.
- Problém unikátního číslování vrcholů.
- Přečíslování pomocí CUDA kernelu: $(CPU\_ID, N) \rightarrow N$

Časná detekce akceptujícího cyklu

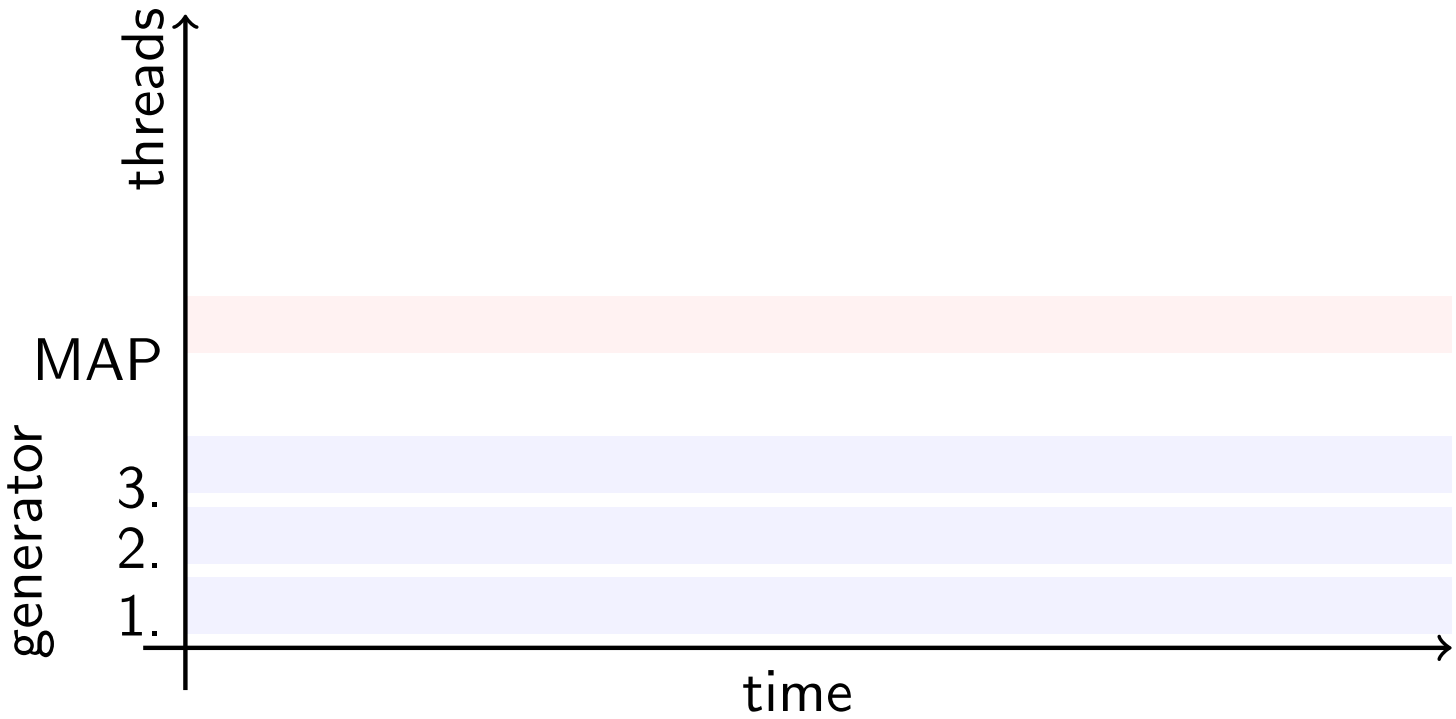
Časná detekce akceptujícího cyklu (On-The-Fly verifikace)

- **Schopnost detekce akceptujícího cyklu bez nutnosti generování kompletního grafu stavového prostoru.**
- Detekce akceptujícího cyklu v neúplné reprezentaci grafu.

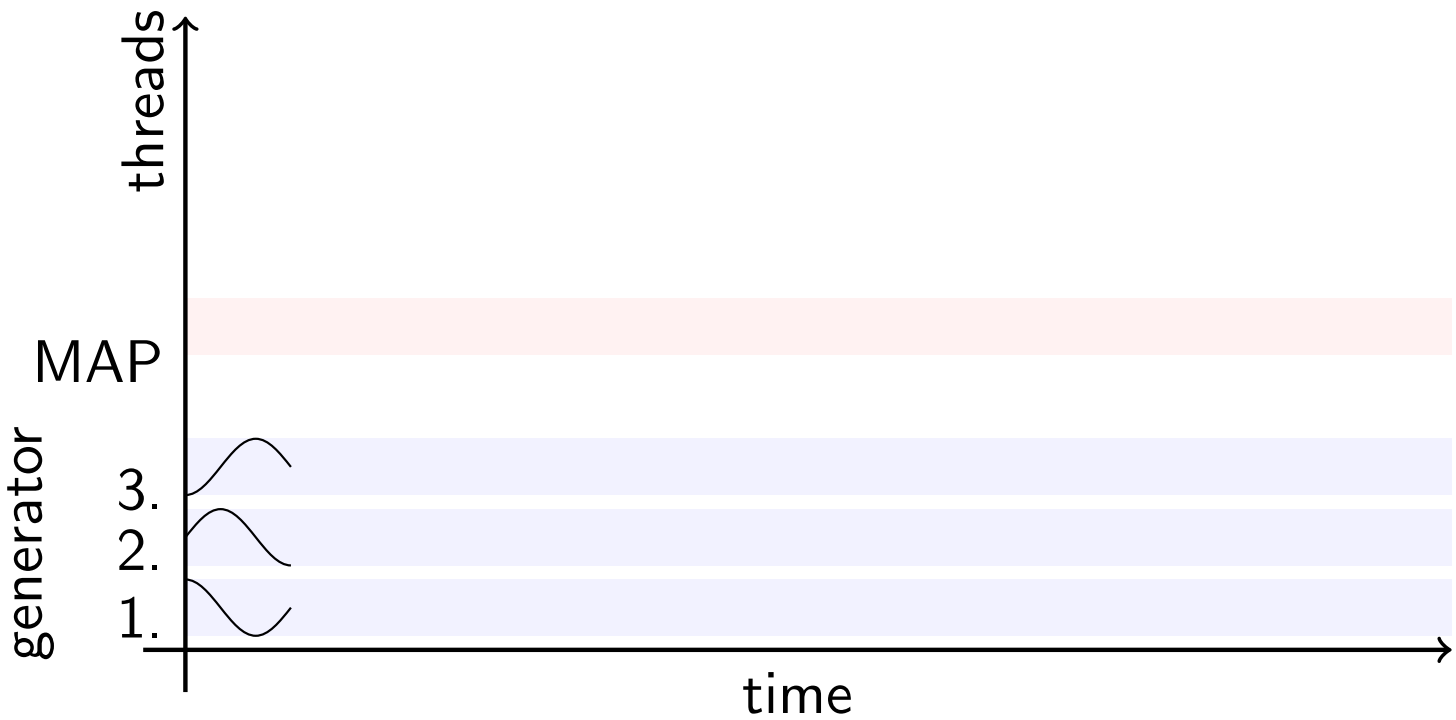
Pozorování

- Akceptující cyklus je možné detekovat stejným GPU algoritmem i v neúplně vygenerovaném grafu.
- Souběžné generování CSR reprezentace na CPU a detekce akceptujícího cyklu na GPU.

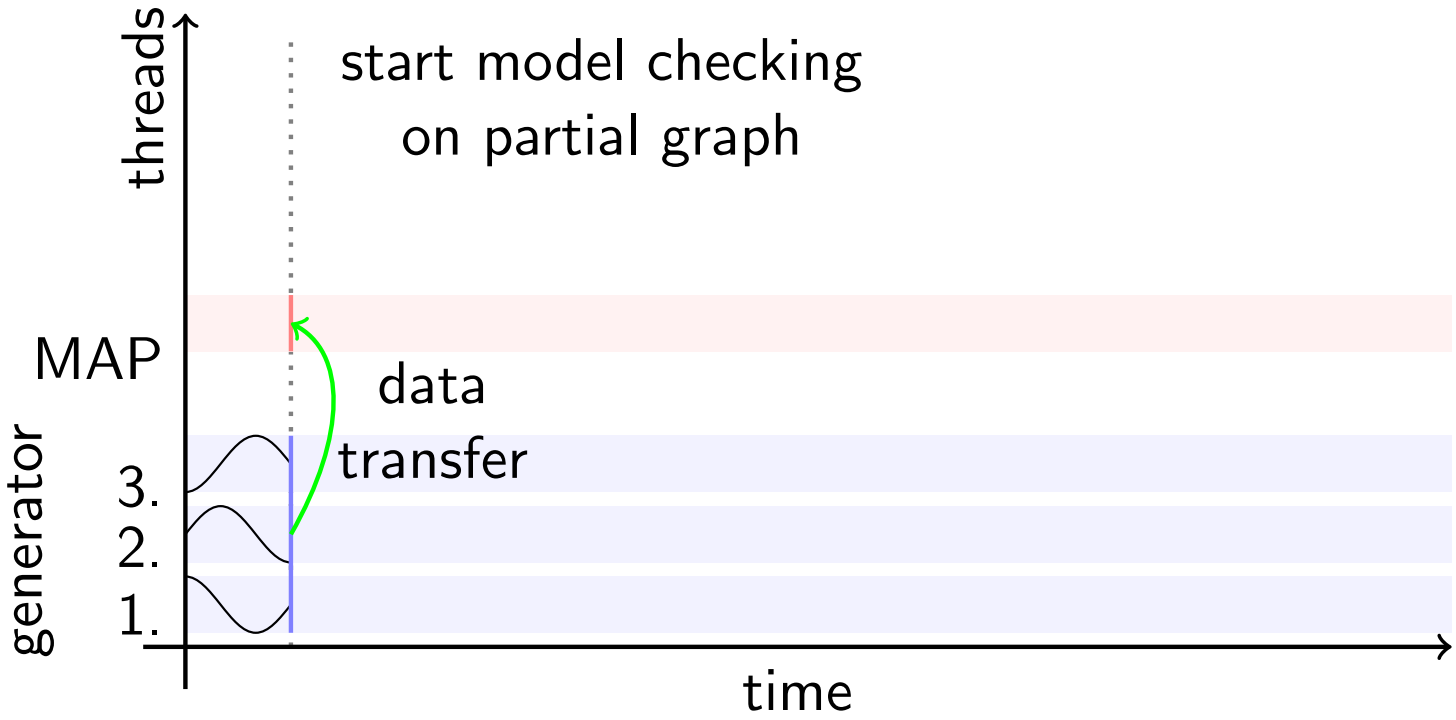
Myšlenka on-the-fly verifikace s využitím GPGPU



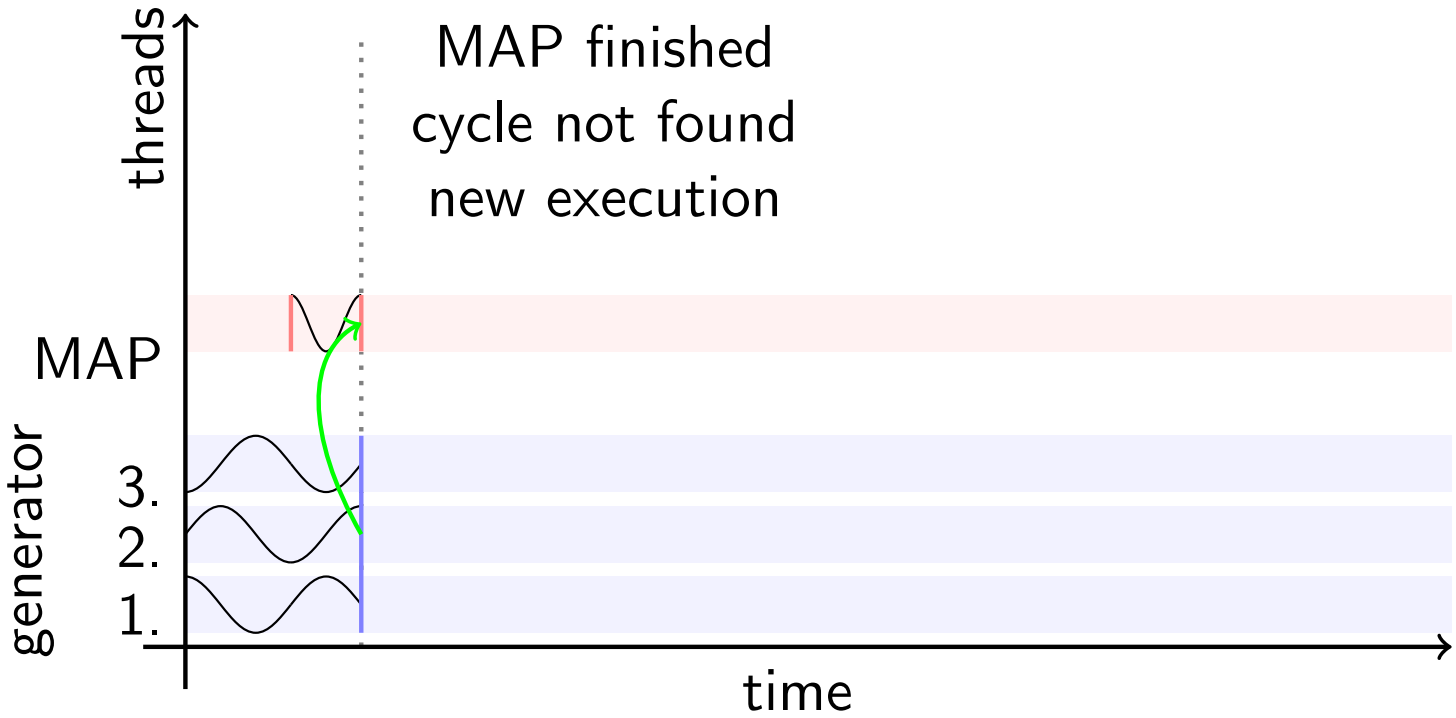
Myšlenka on-the-fly verifikace s využitím GPGPU



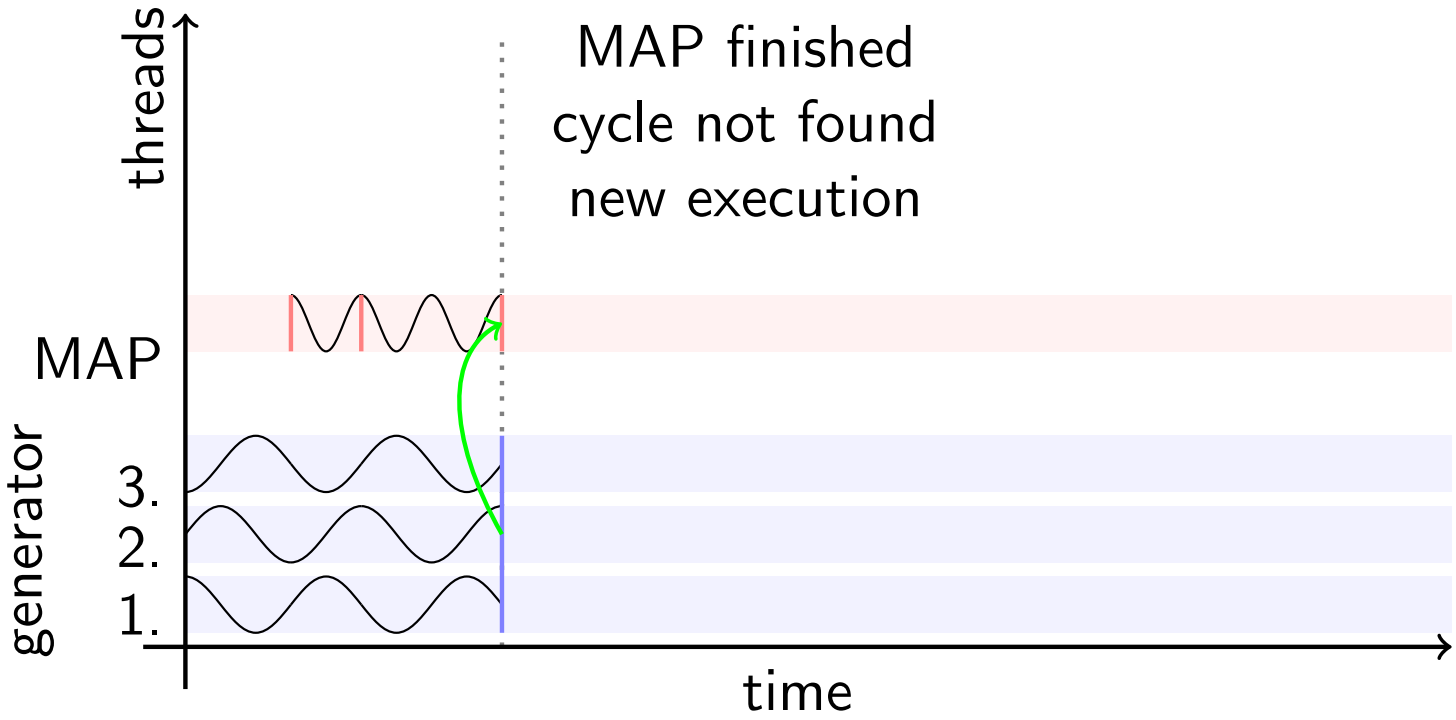
Myšlenka on-the-fly verifikace s využitím GPGPU



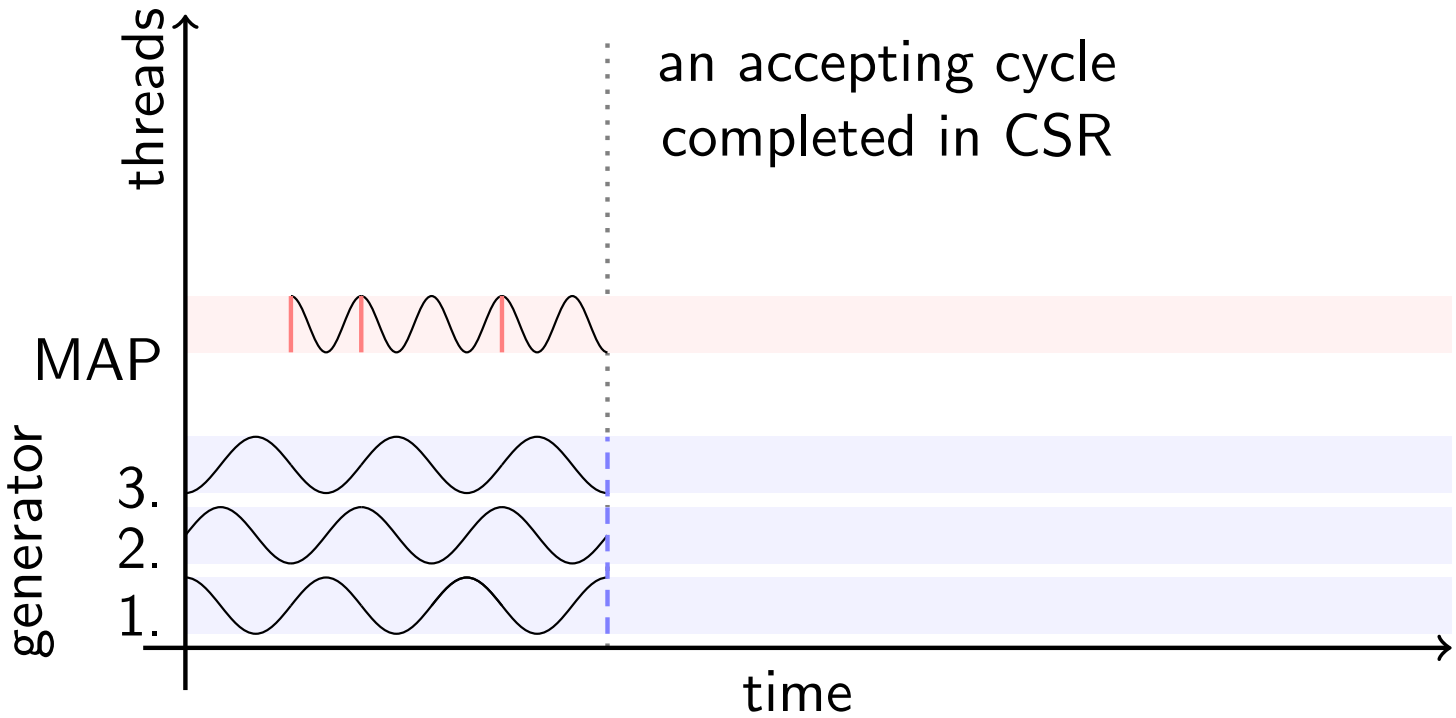
Myšlenka on-the-fly verifikace s využitím GPGPU



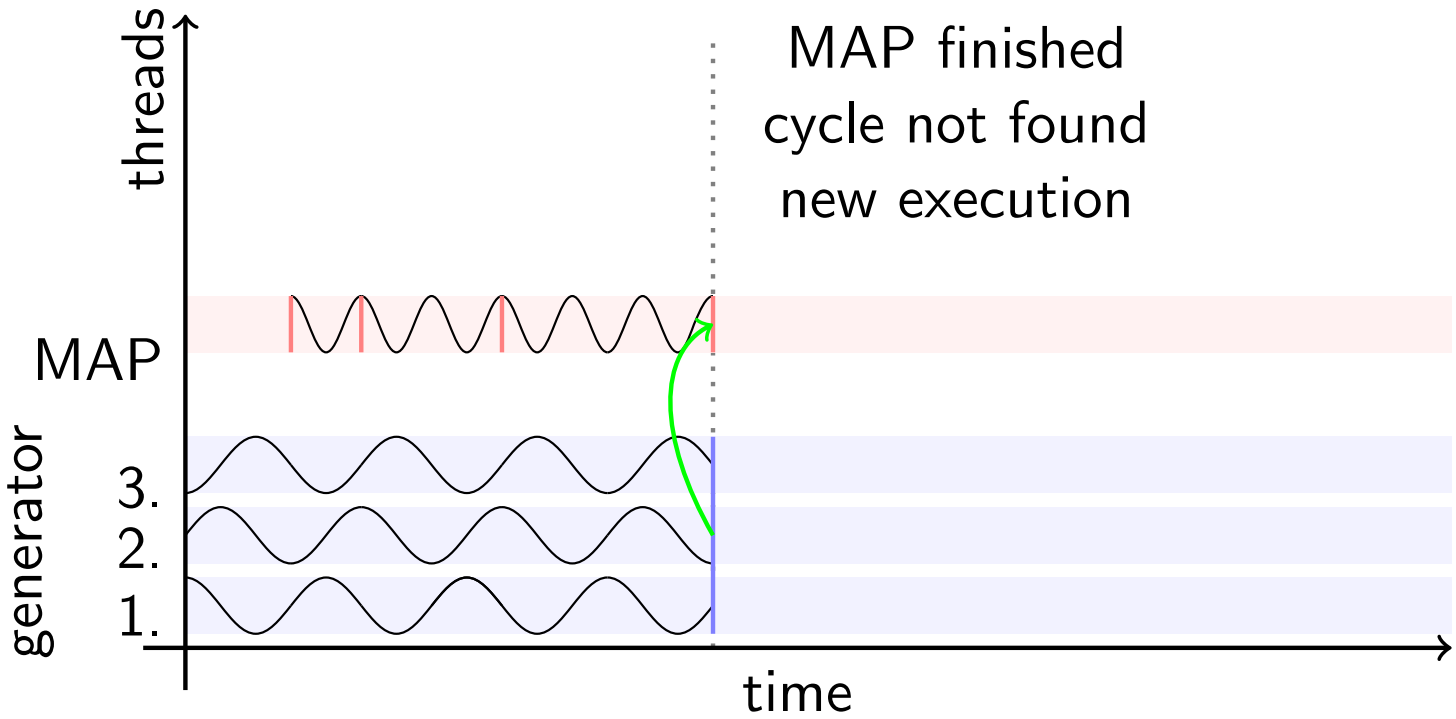
Myšlenka on-the-fly verifikace s využitím GPGPU



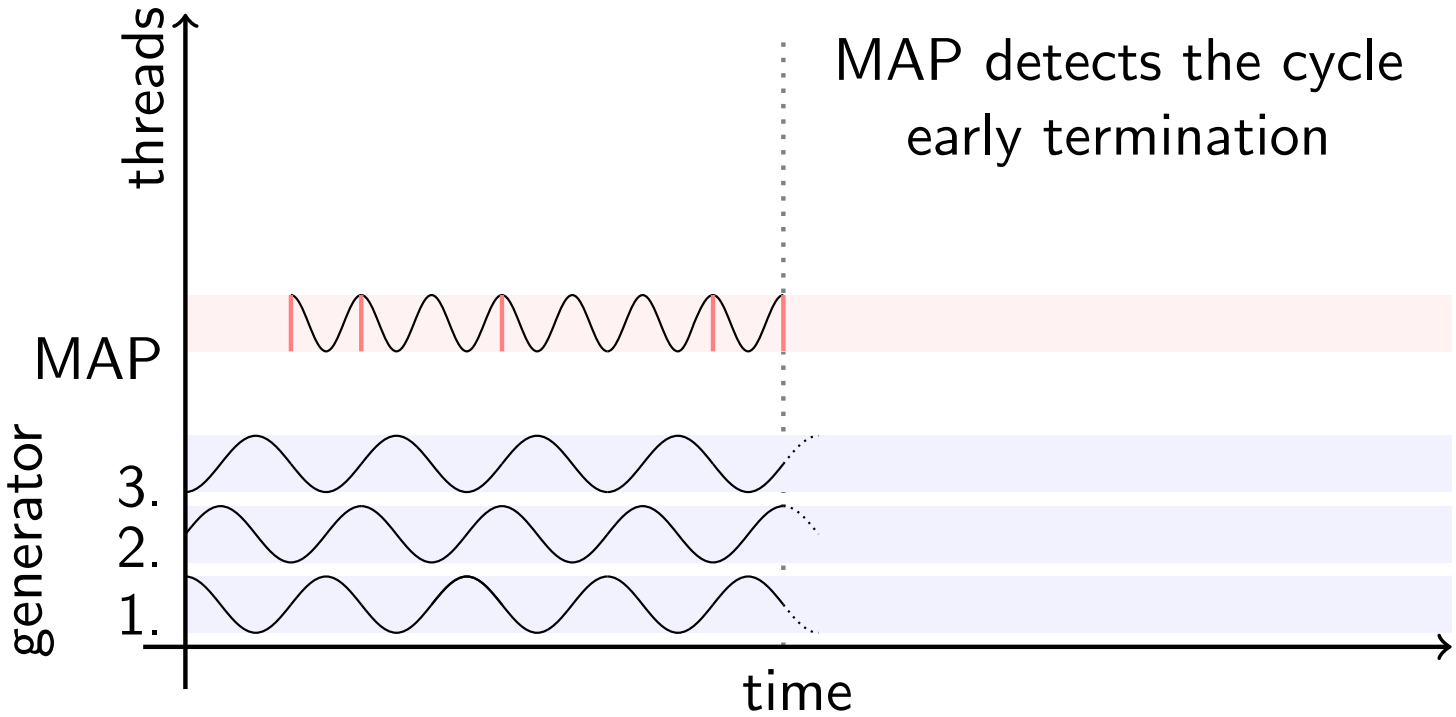
Myšlenka on-the-fly verifikace s využitím GPGPU



Myšlenka on-the-fly verifikace s využitím GPGPU



Myšlenka on-the-fly verifikace s využitím GPGPU



Využití více GP GPU karet

Paměťové nároky na GPU

Paměťová náročnost reprezentace

- 12B na vrchol (4B CSR + 8B MAP *map*,*oldmap*,$\mathcal{A}$,*r*)
- 4B na hranu (CSR)

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

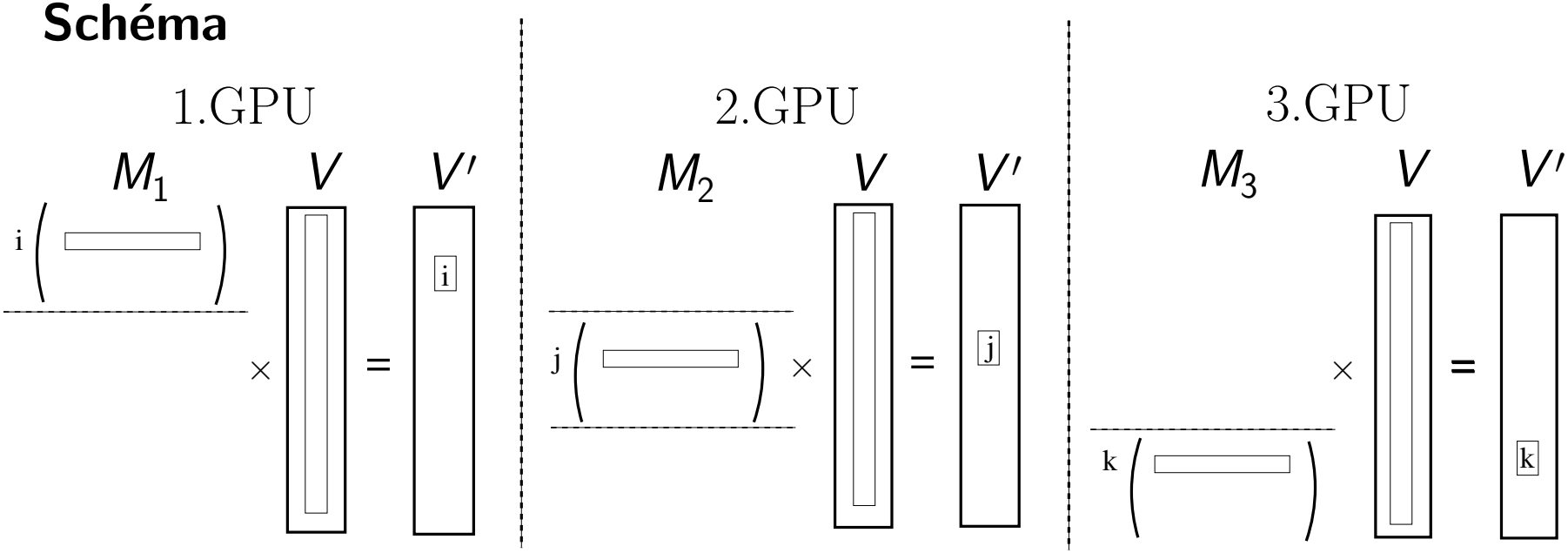
Mc=(1 3 2 5 4 2 1 5)
Mr=(0 2 4 5 6)

1 GB GPU karta

- 30 miliónů vrcholů, 150 miliónů hran (outdegree 5)
- 50 miliónů vrcholů, 100 miliónů hran (outdegree 2)

Distribuce dat mezi více GPU karet – verze 1

Schéma

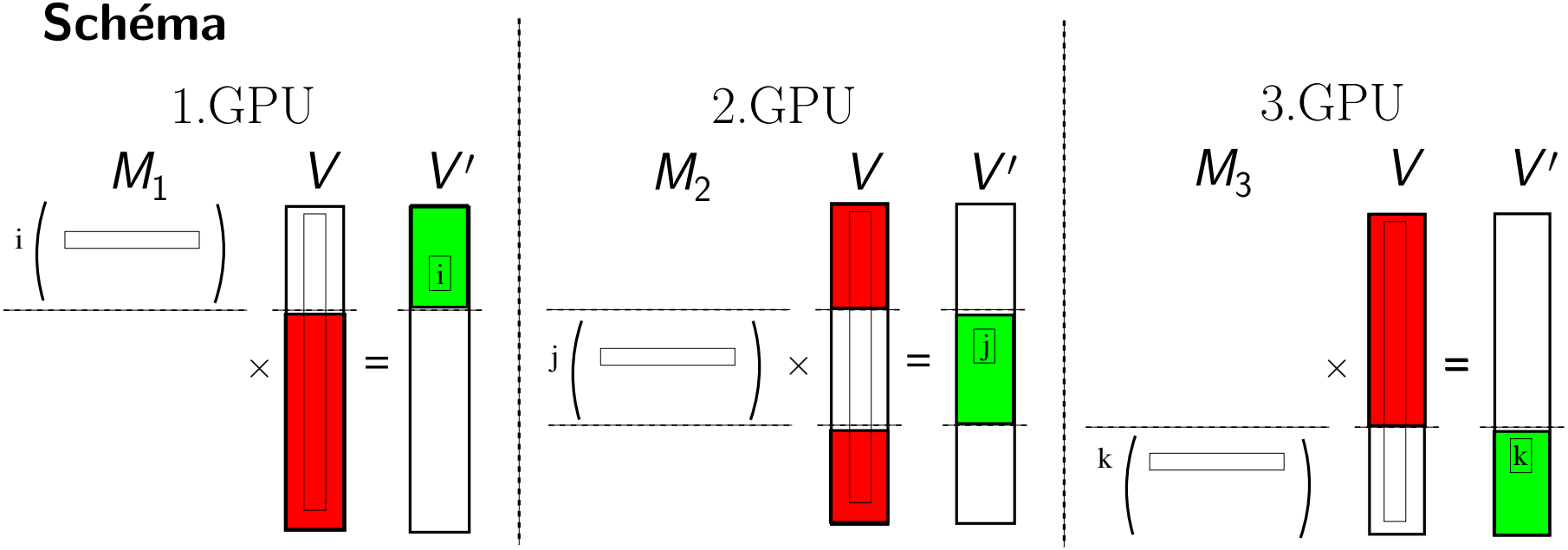


Výhody/Nevýhody

- Pouze částečná distribuce.
- Jednu iteraci (jedno násobení) je možné kompletně spočítat bez nutnosti další komunikace mezi GPU zařízeními.

Distribuce dat mezi více GPU karet – verze 1

Schéma

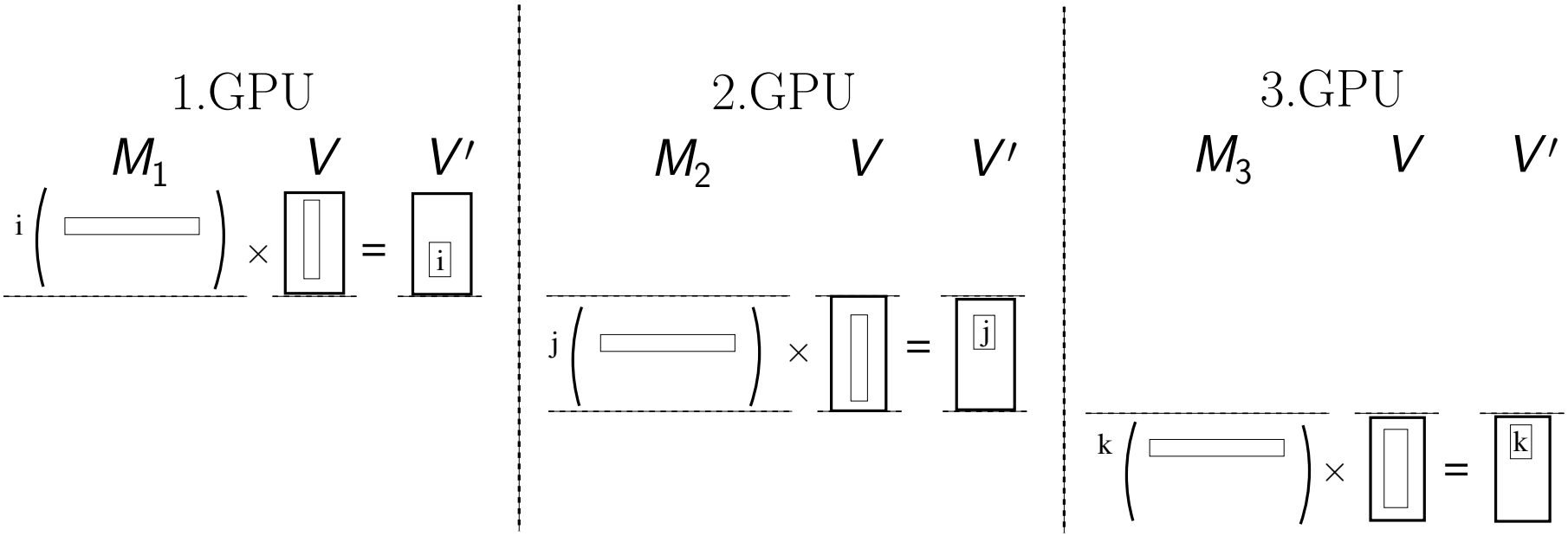


Výhody/Nevýhody

- Pouze částečná distribuce.
- Jednu iteraci (jedno násobení) je možné kompletně spočítat bez nutnosti další komunikace mezi GPU zařízeními.
- Další iterace vyžadují výměnu dat mezi GPU zařízeními.

Distribuce dat mezi více GPU karet – verze 2

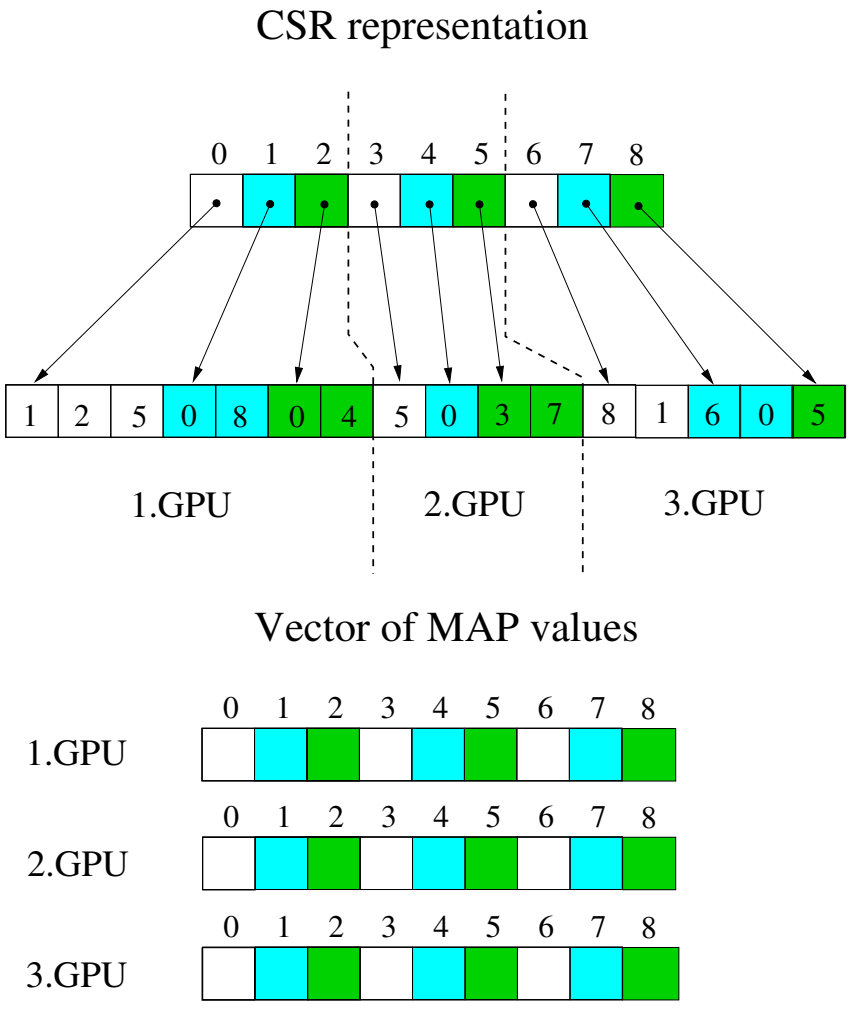
Schéma



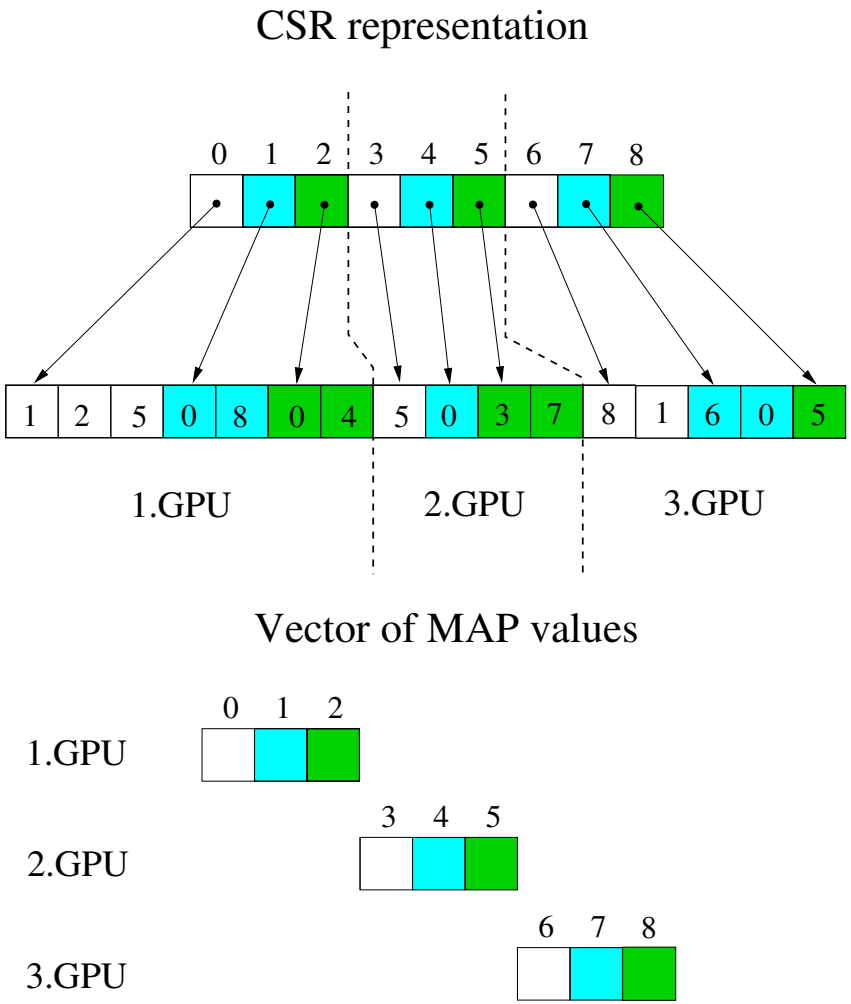
Výhody/Nevýhody

- Kompletní distribuce dat.
- Ani jednu iteraci nelze dokončit bez nutnosti další komunikace mezi GPU zařízeními.

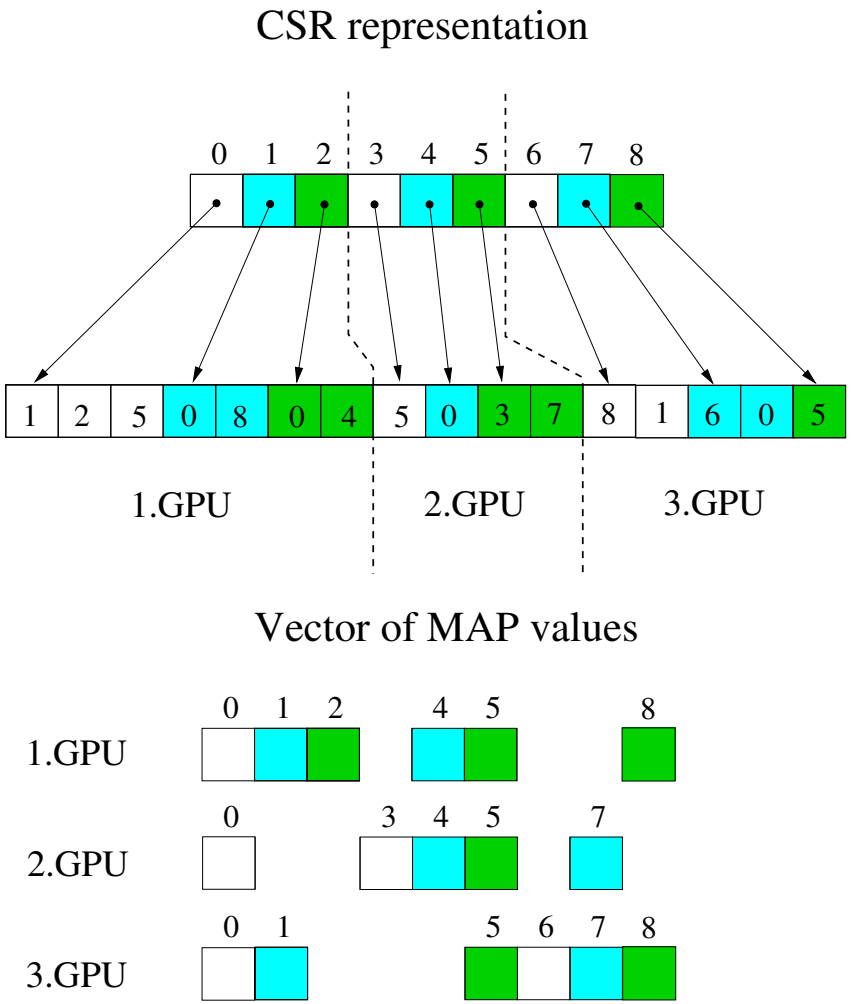
Přístup k nelokálním vrcholům



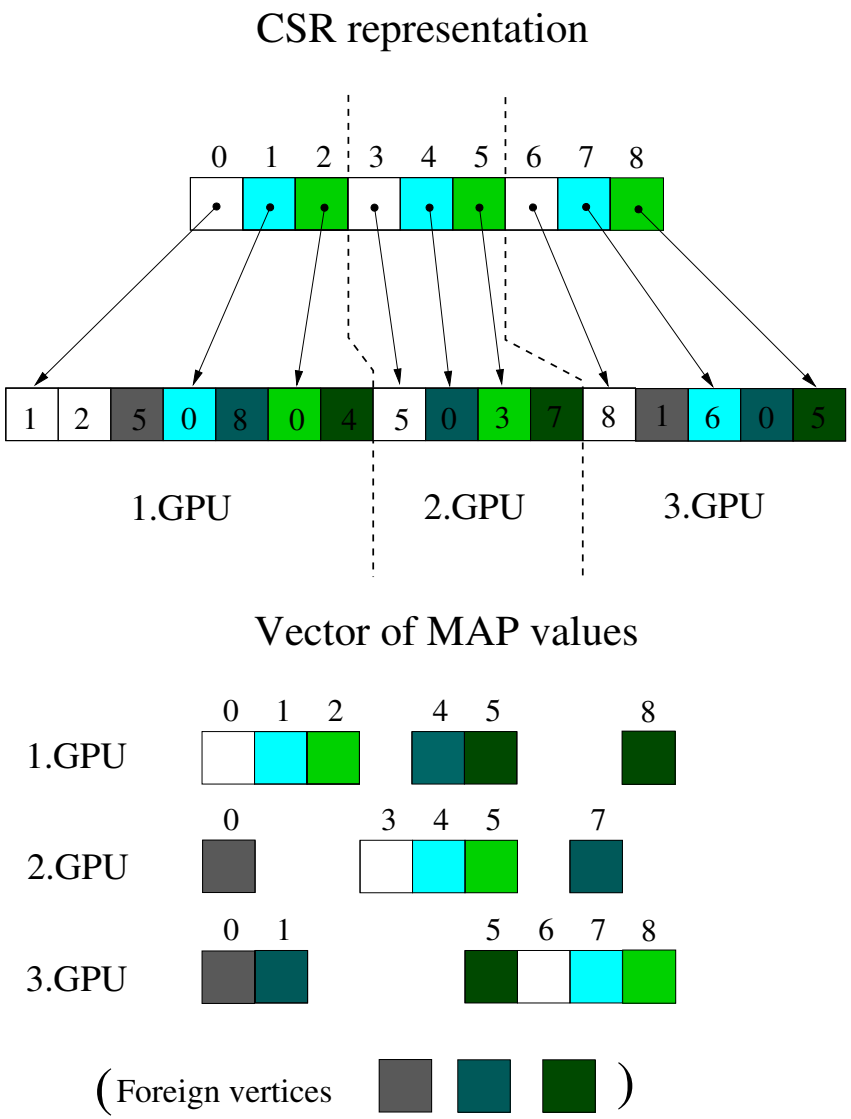
Přístup k nelokálním vrcholům



Přístup k nelokálním vrcholům



Přístup k nelokálním vrcholům



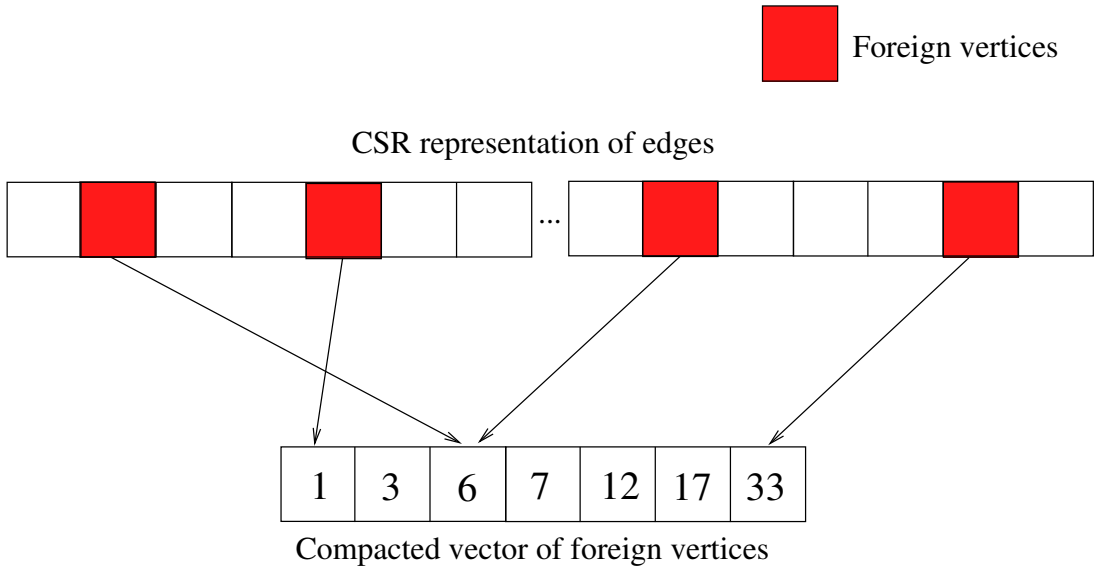
Kompakce nelokálních vrcholů

Pozorování

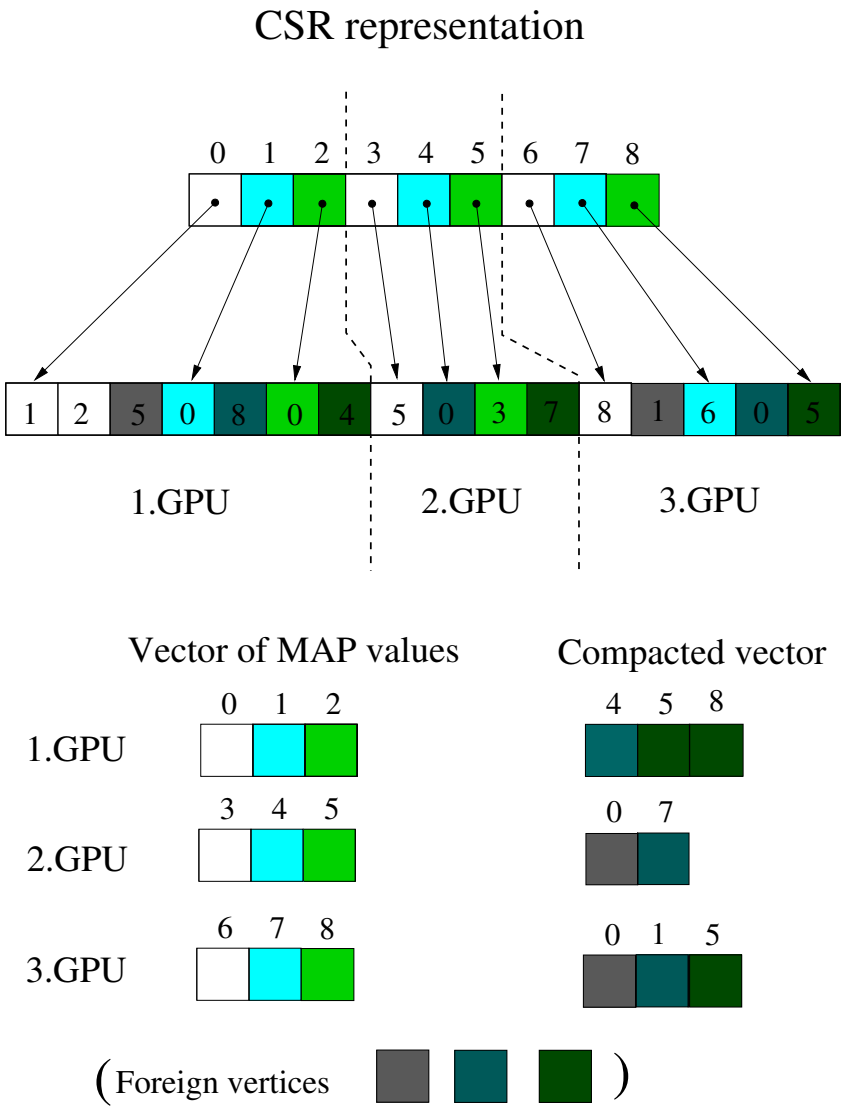
- Výměna všech nelokálních hodnot je drahá.
- Výměna pouze požadovaných nelokálních hodnot je obtížná, neboť hodnoty jsou prostorově nespojité.

Navržené řešení

- Každé zařízení používá kompaktní vektor nelokálních vrcholů.
- CSR reprezentace modifikována odpovídajícím způsobem.



Kompakce nelokálních vrcholů



Příprava kompaktních vektorů nelokálních hodnot

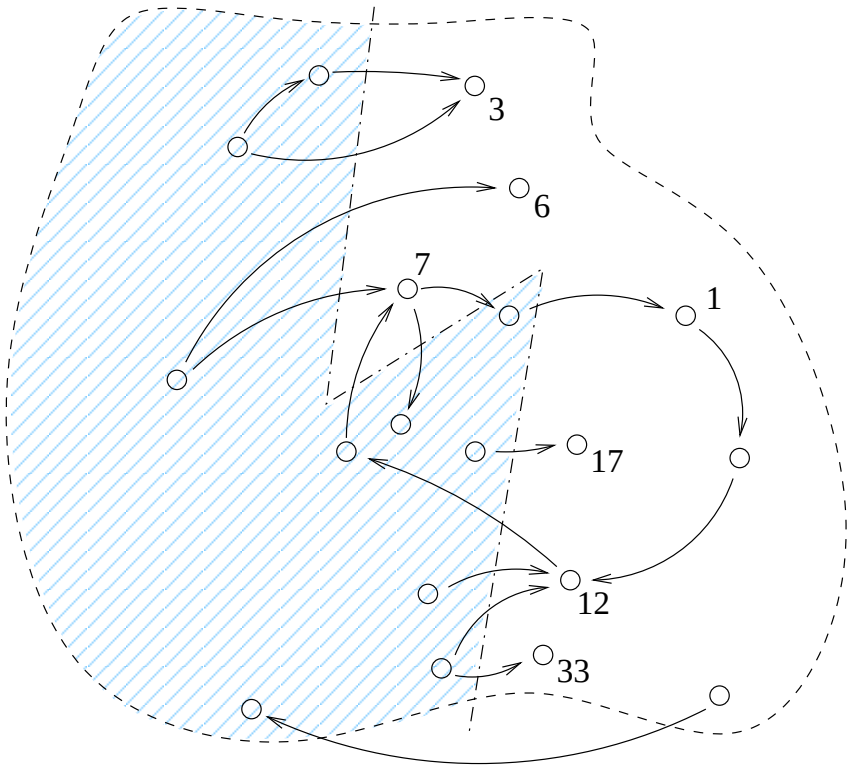
Omezení

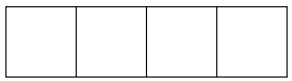
- Modifikace CSR reprezentace je výpočetně náročná, **nutno akcelerovat pomocí GPU**.
- Některé nelokální vrcholy se v CSR opakují!
- Nelze použít `reduce` (protorově náročné) ani `AtomicCAS` (časově náročné).

Řešení

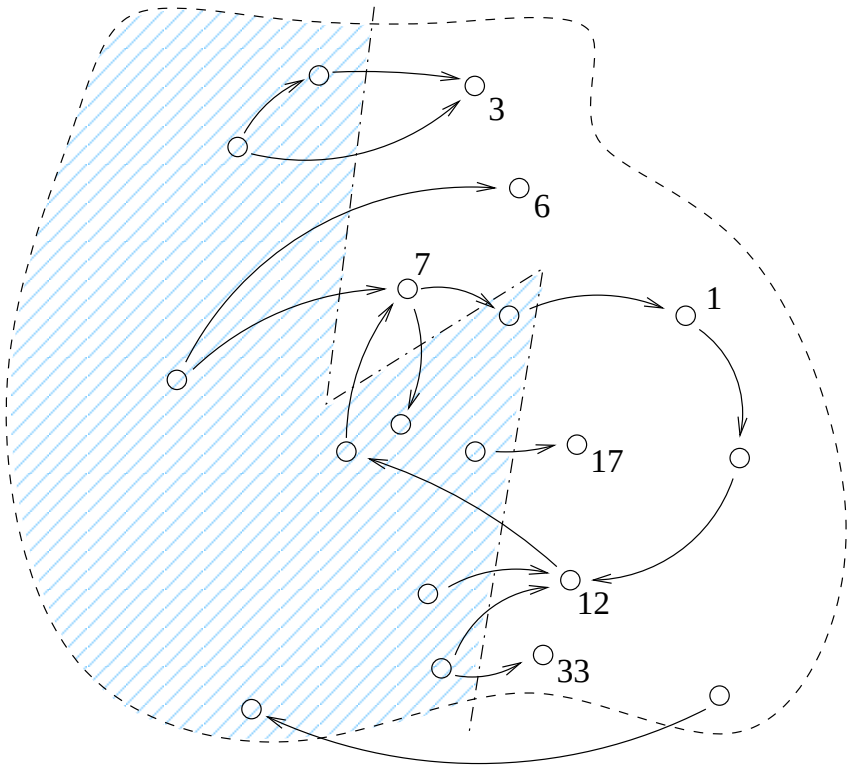
- Kompaktní vektor (téměř) lze vytvořit opakovaným hašování do dynamicky rostoucího pole. (CUDA)
- Kompaktní vektor se utřídí a ořeže (CPU).
- Aktualizuje se CSR (binární prohledávání, CUDA).

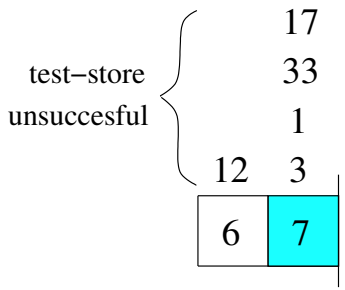
Příprava kompaktních vektorů nelokálních hodnot – demo



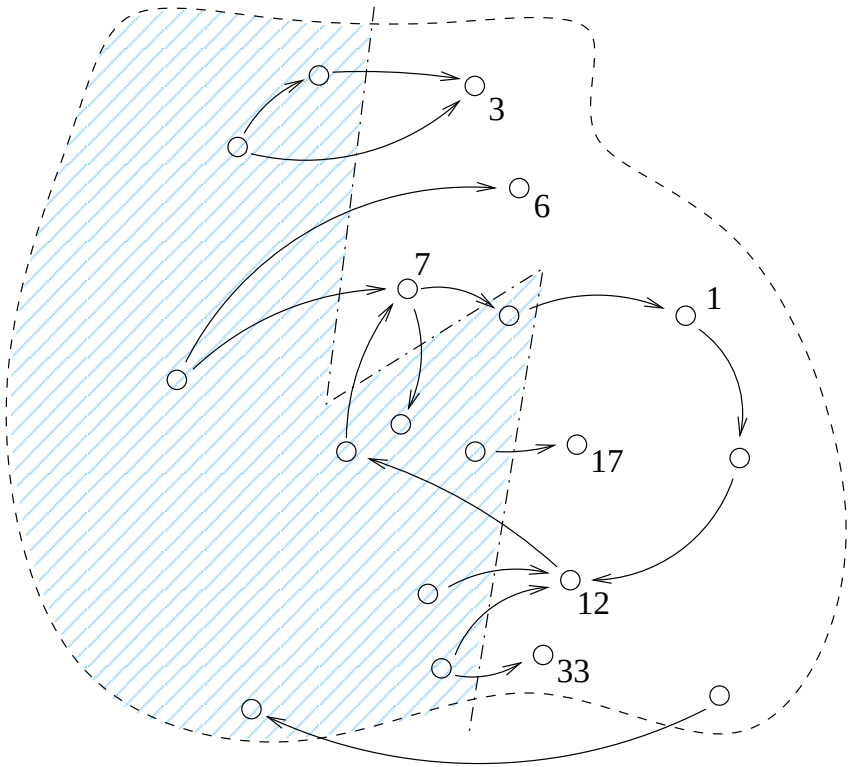


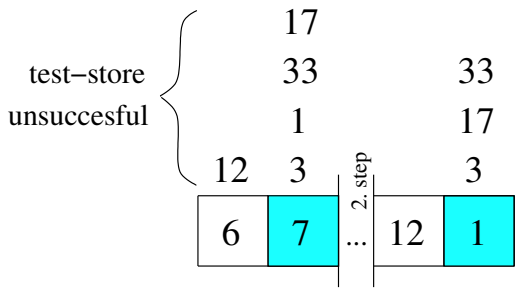
Příprava kompaktních vektorů nelokálních hodnot – demo



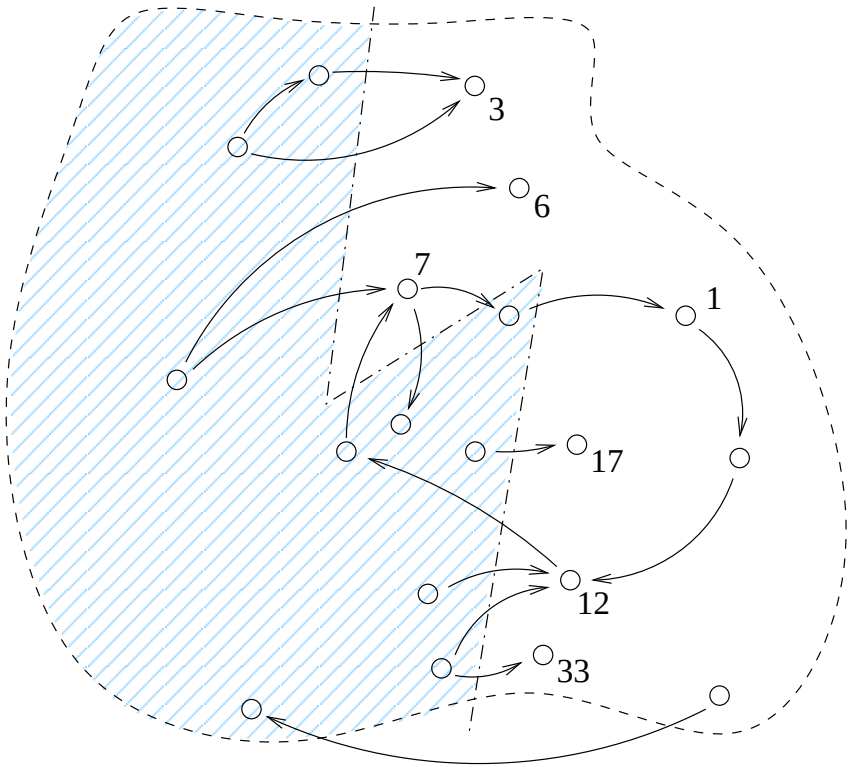


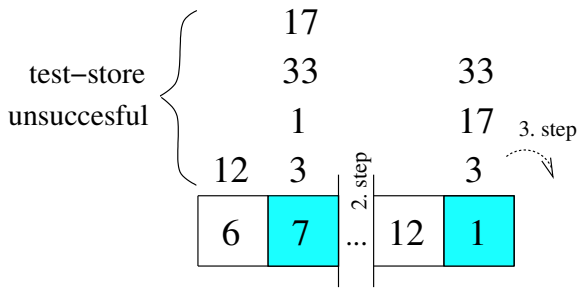
Příprava kompaktních vektorů nelokálních hodnot – demo



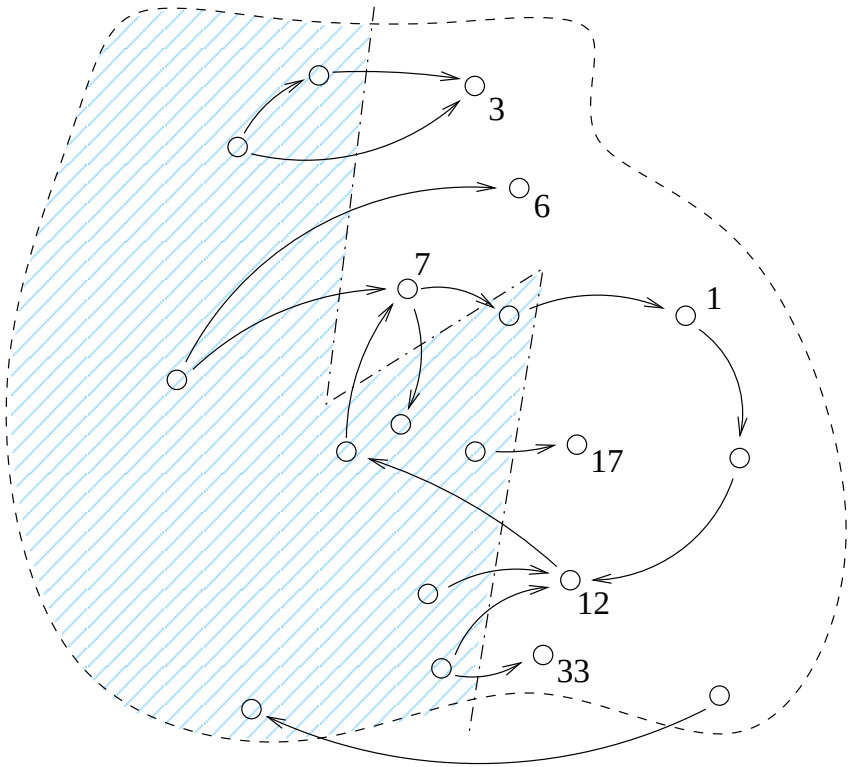


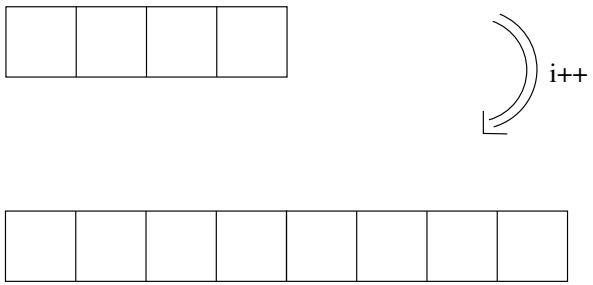
Příprava kompaktních vektorů nelokálních hodnot – demo



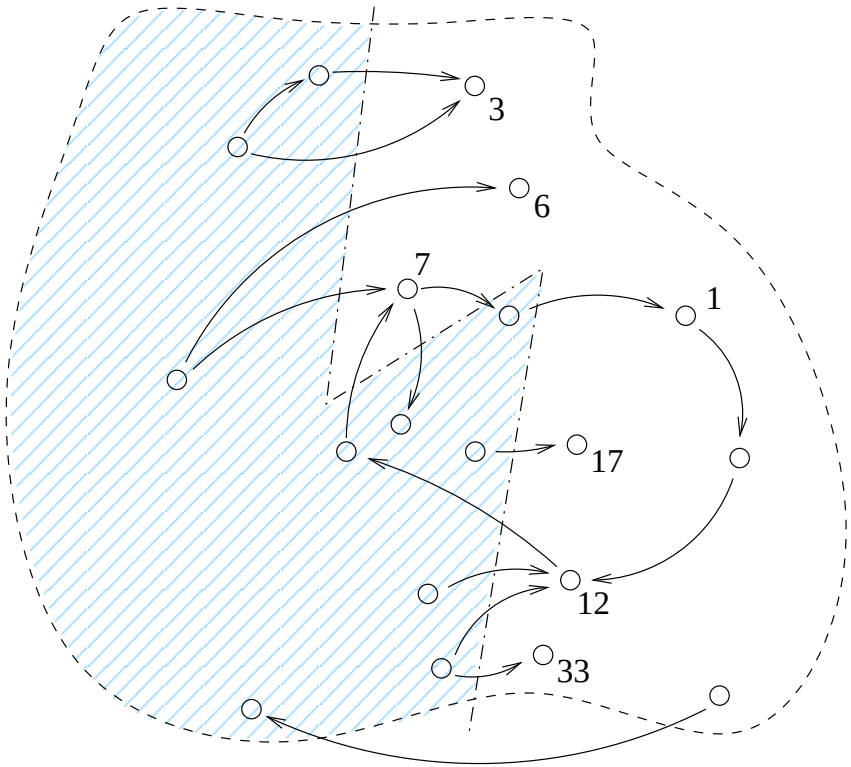


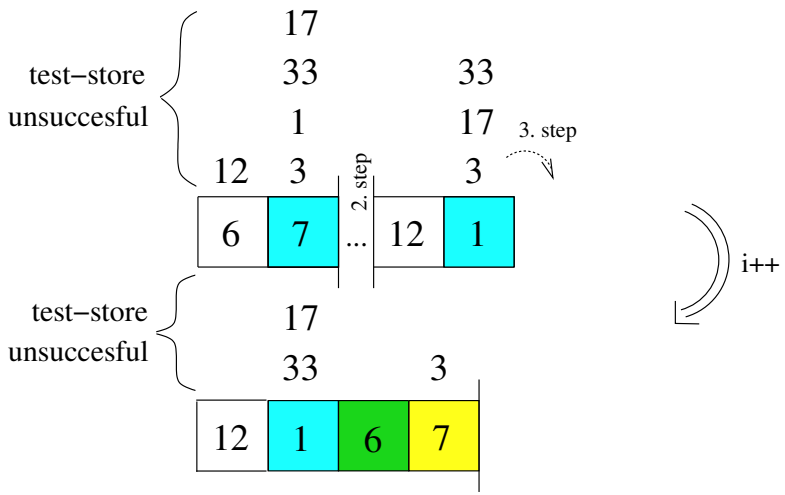
Příprava kompaktních vektorů nelokálních hodnot – demo



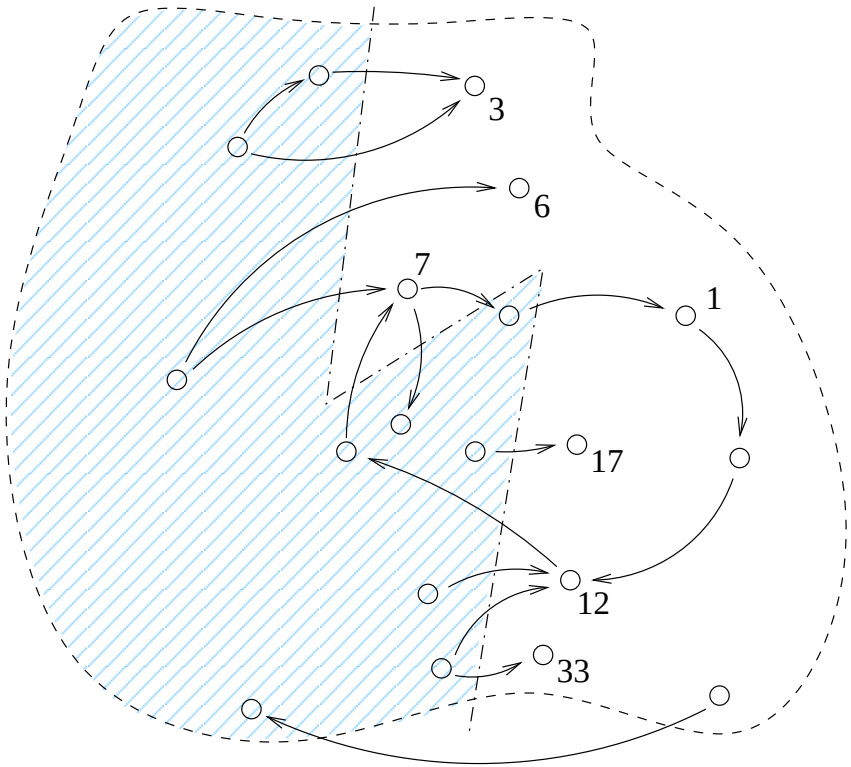


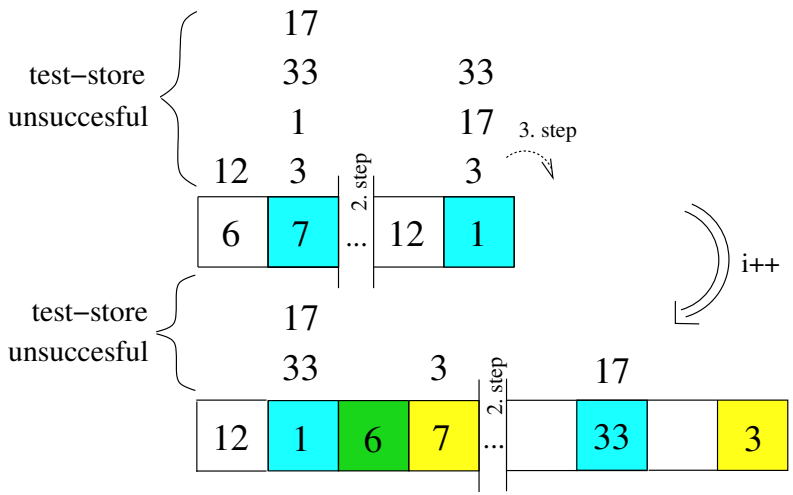
Příprava kompaktních vektorů nelokálních hodnot – demo



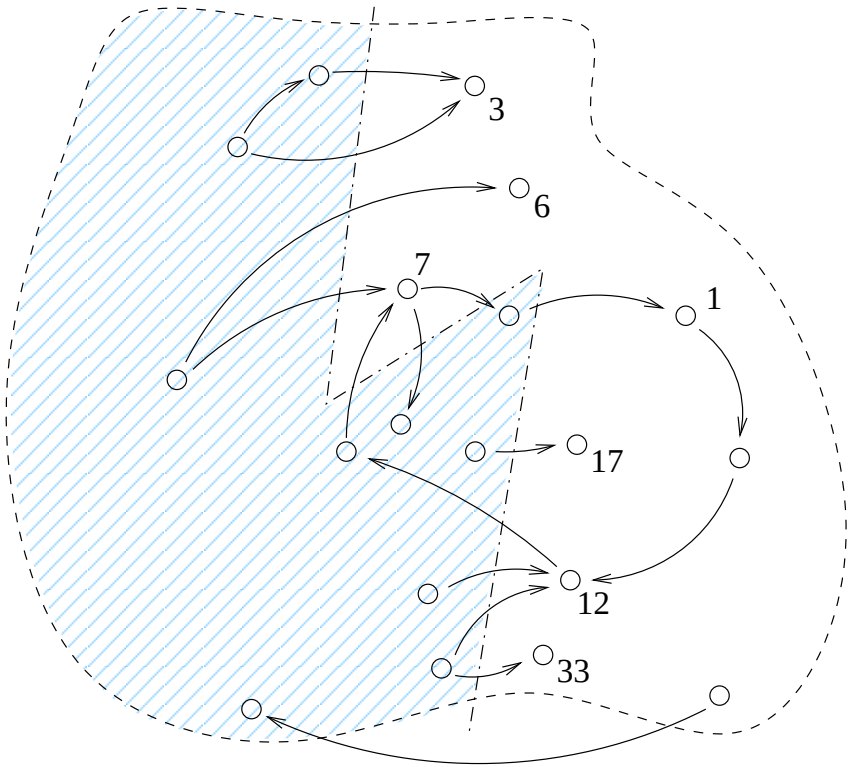


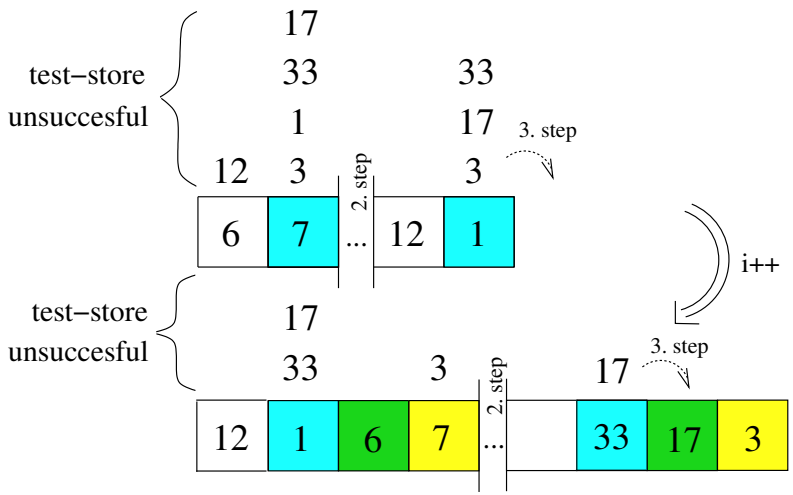
Příprava kompaktních vektorů nelokálních hodnot – demo



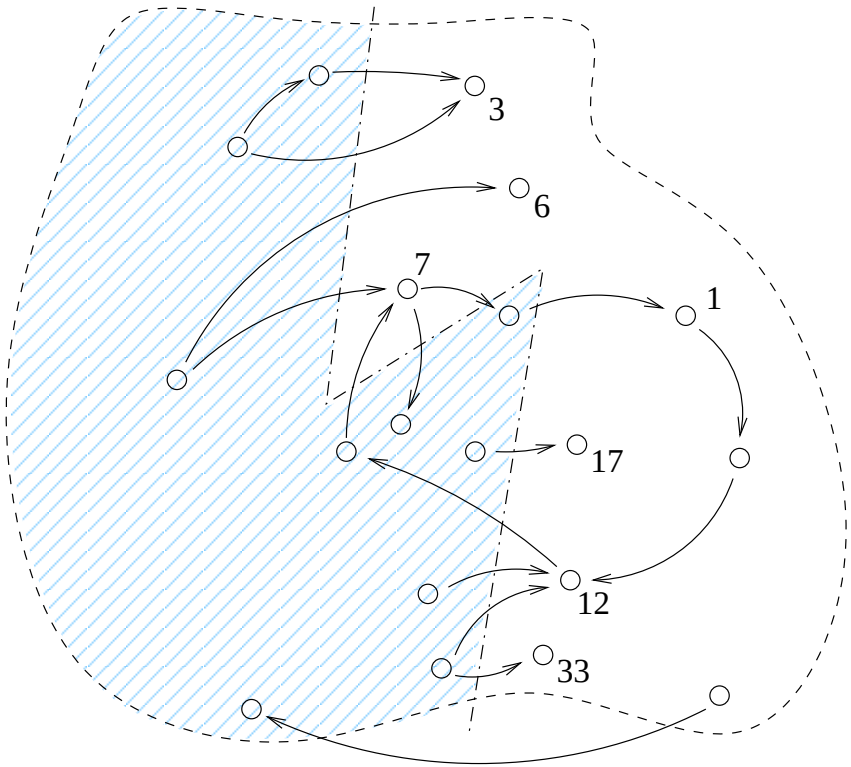


Příprava kompaktních vektorů nelokálních hodnot – demo



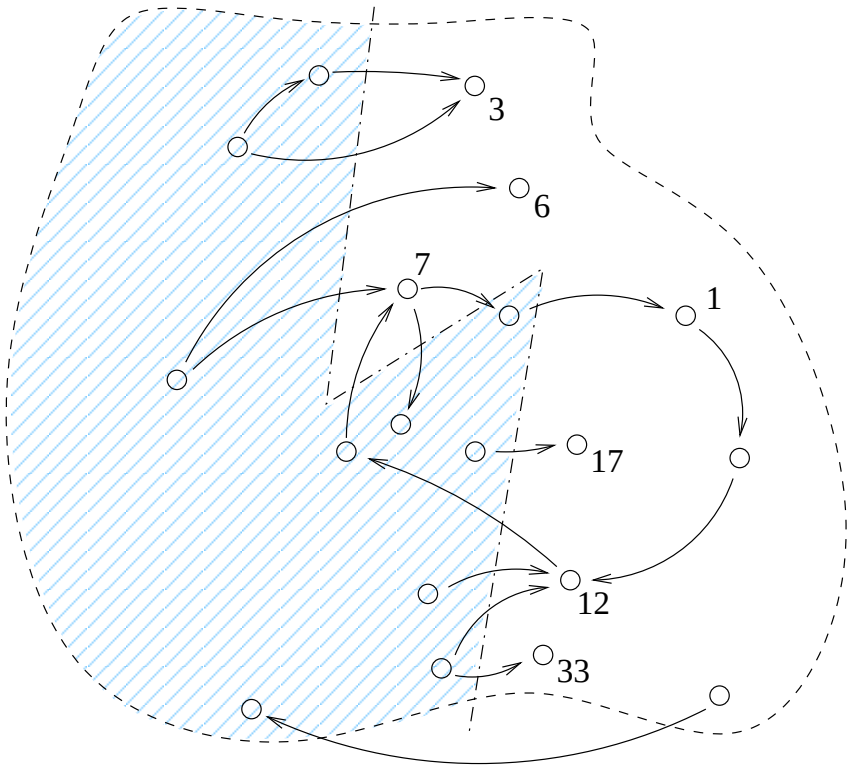


Příprava kompaktních vektorů nelokálních hodnot – demo



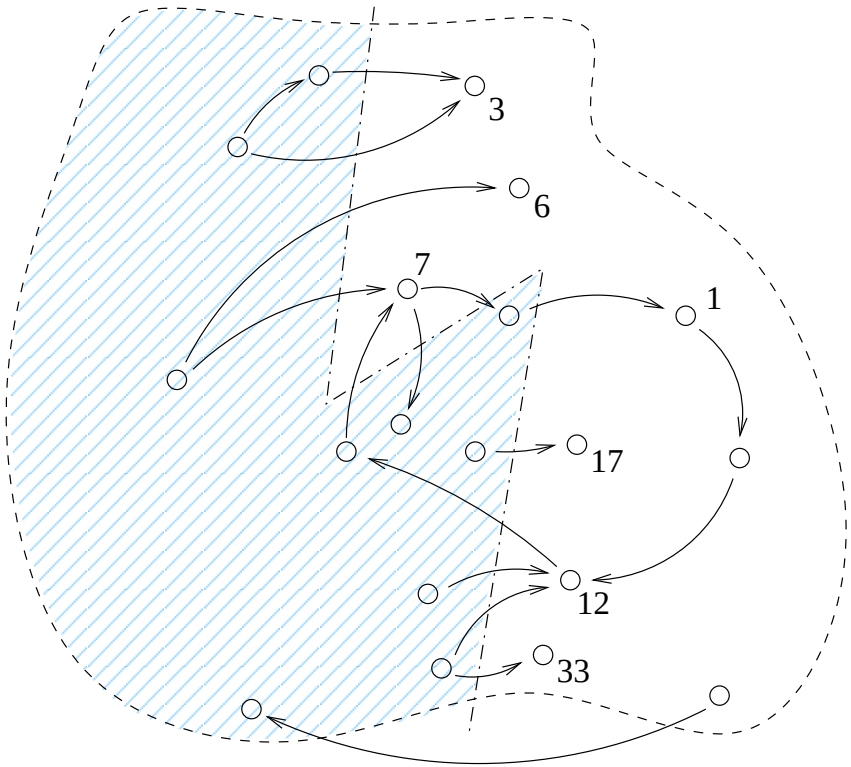
	1	3	6	7	12	17	33

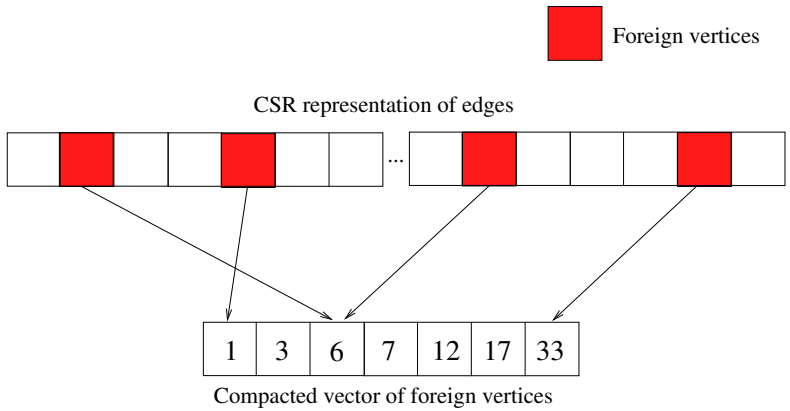
Příprava kompaktních vektorů nelokálních hodnot – demo



1	3	6	7	12	17	33

Příprava kompaktních vektorů nelokálních hodnot – demo





Poznámka o odolnosti vůči znovu-provedení

Pozorování

- Výpočet hodnot funkce MAP/MAS končí v okamžiku dosažení pevného bodu.
- Hodnoty MAP/MAS monotónně rostou.
- Znovu-provedení aktualizací nepokazí korektnost. (Maximum z hodnot a jejich maxima je jejich maximum.)

Důsledek

- Pořadí provedení výpočtů maxim může být libovolné, po dostatečně dlouhém opakování se vždy dosáhne pevného bodu.
- Různé strategie pro výměny dat mezi jednotlivými GPU.

Použití jiných algoritmů

Algoritmus OWCTY

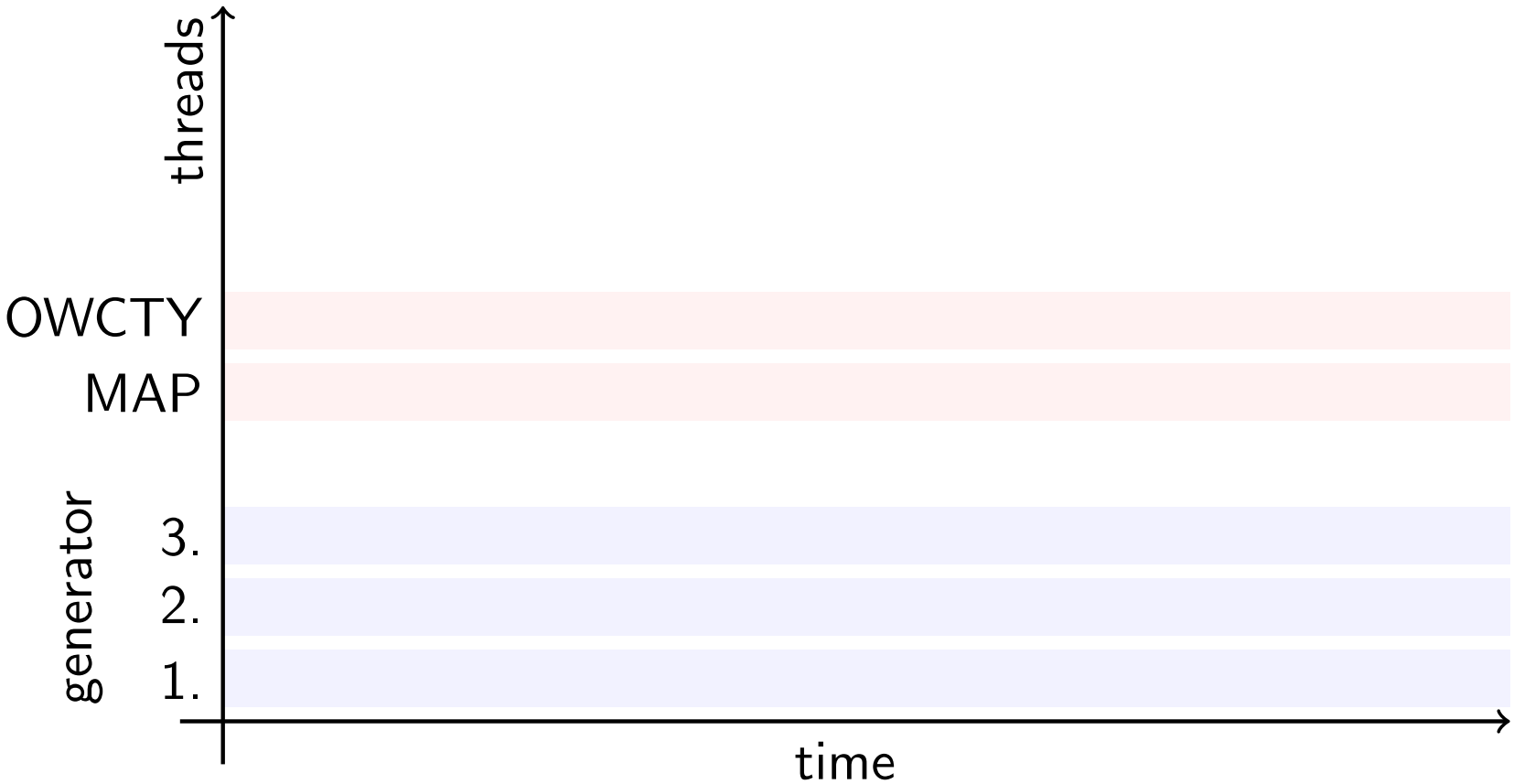
Princip algoritmu

- Alternace topologického třídění a výpočtu relace dosažitelnosti.

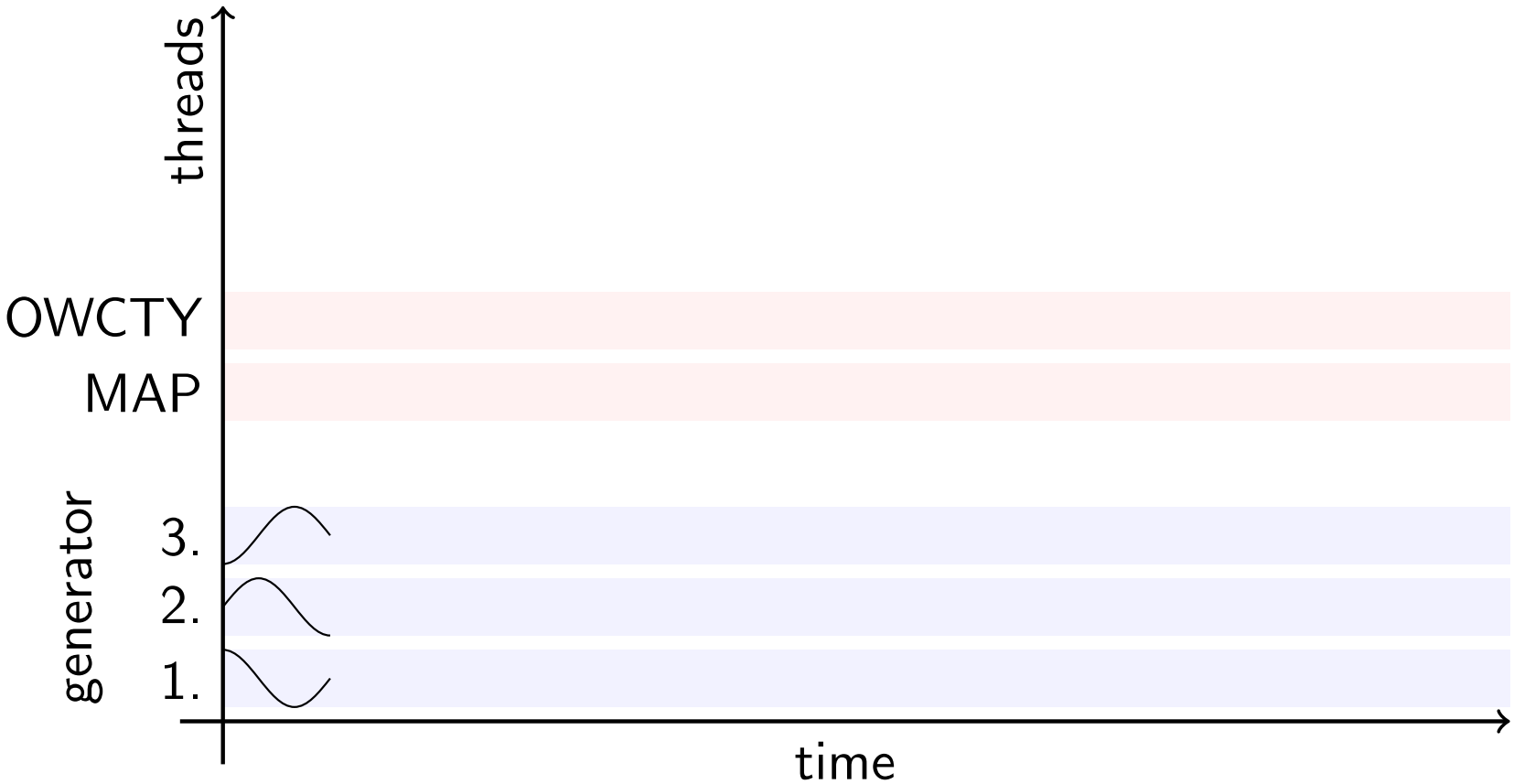
Princip GPU akcelerace

- Akcelerace identickým způsobem jako výpočet funkce MAP.
- Operace převedeny na problém násobení matice vektorem, ve vzorečku pro násobení se použijí vhodné matematické operace.
- Paměťové nároky algoritmu: 4B na vrchol + CSR.

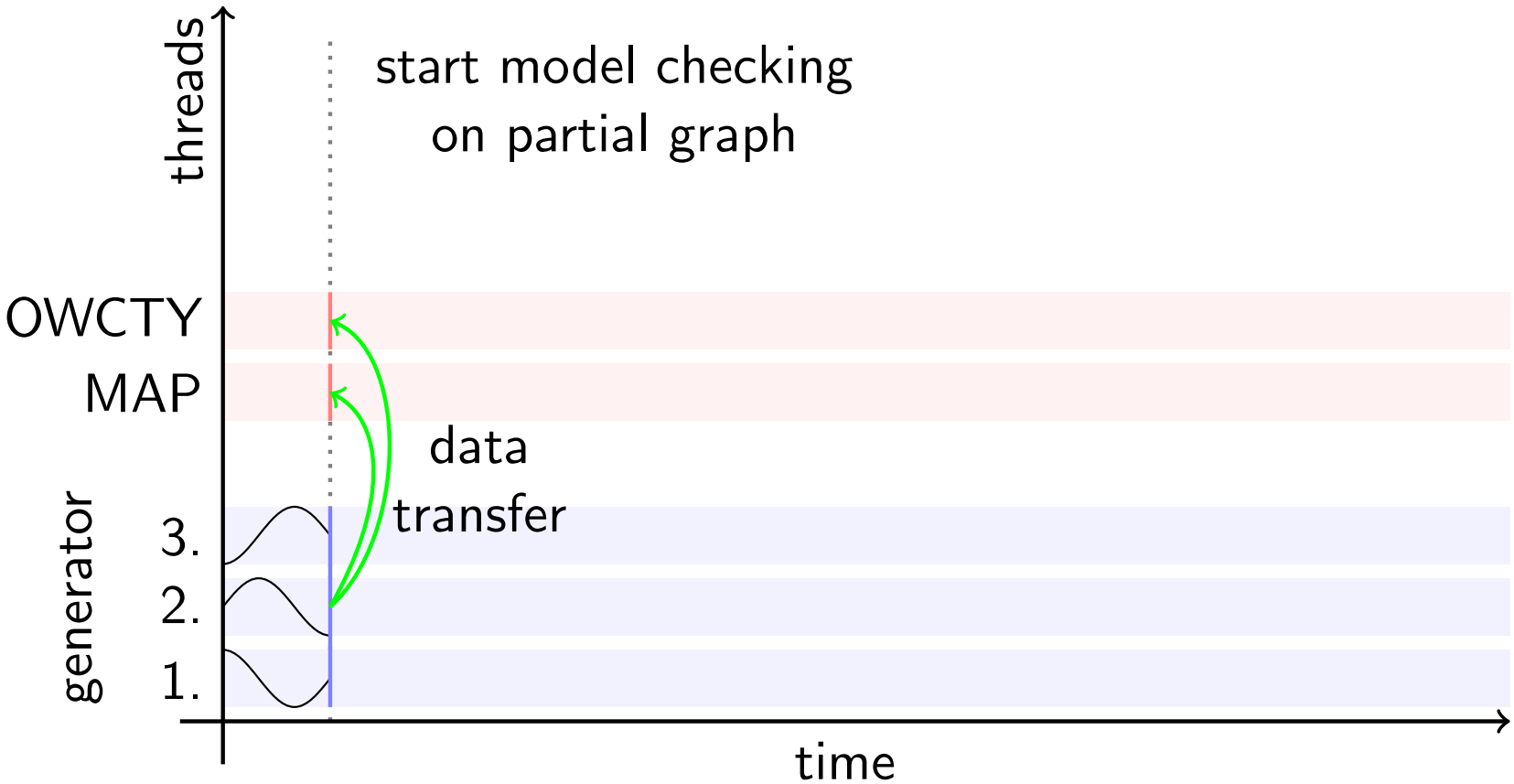
Kombinace algoritmů



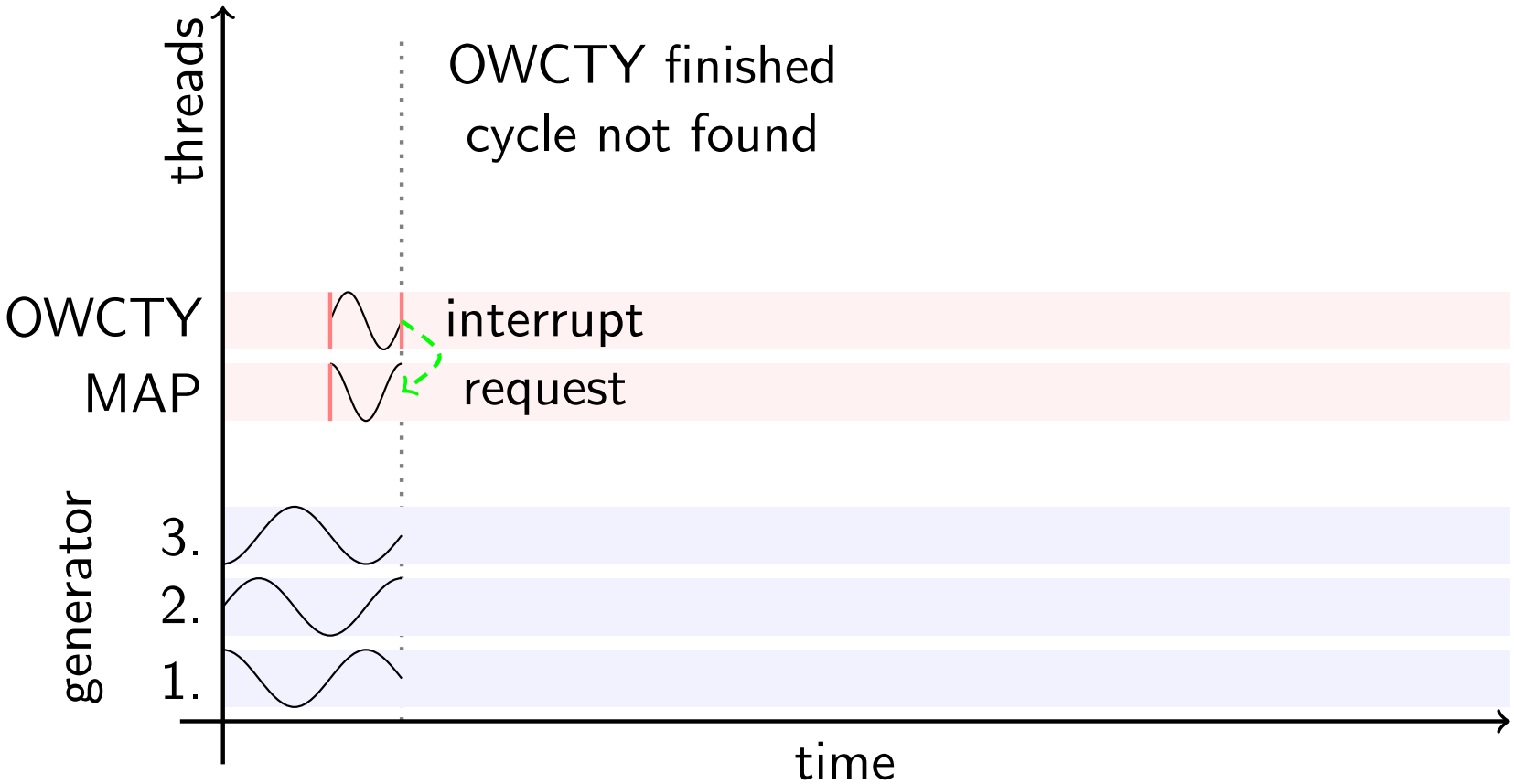
Kombinace algoritmů



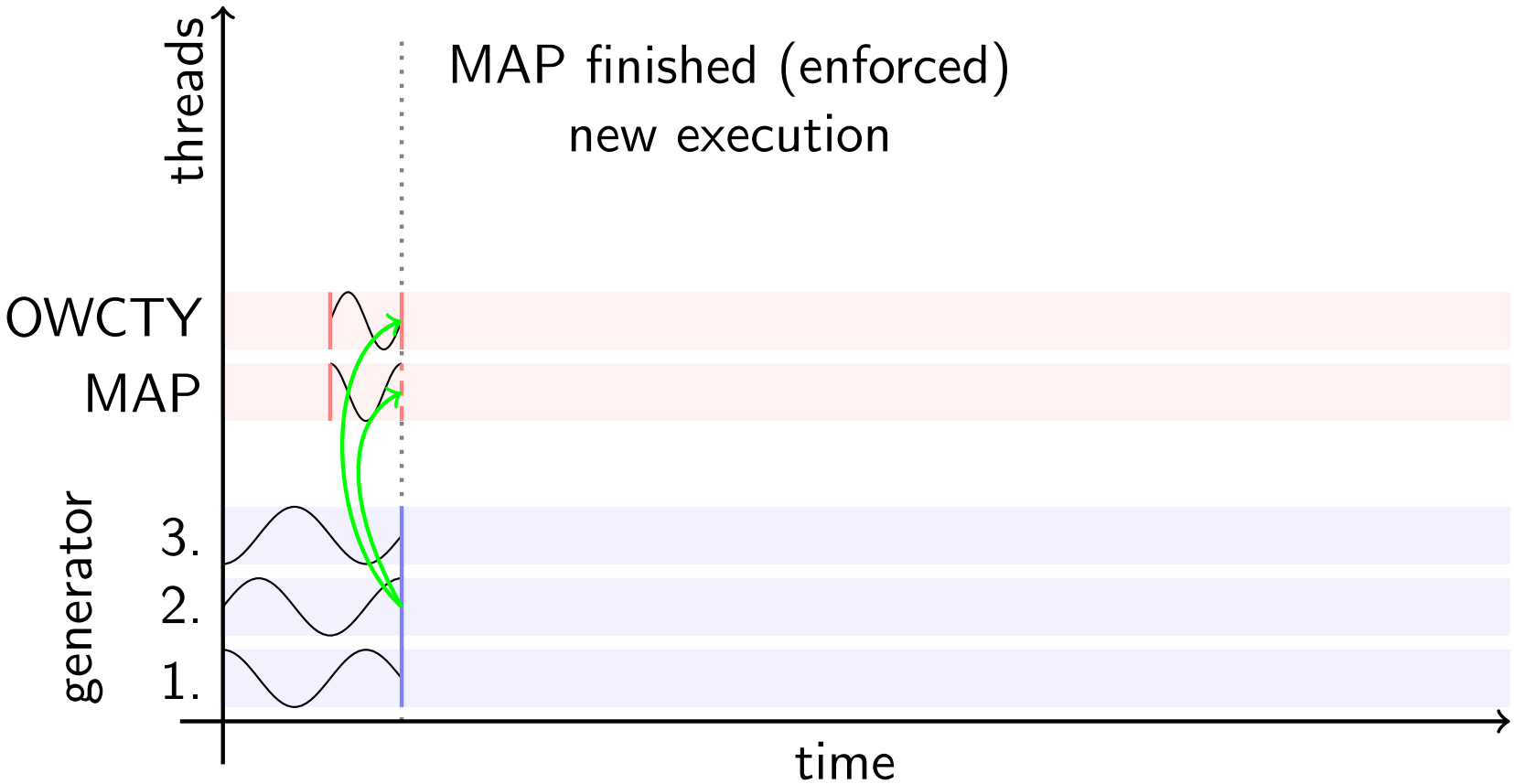
Kombinace algoritmů



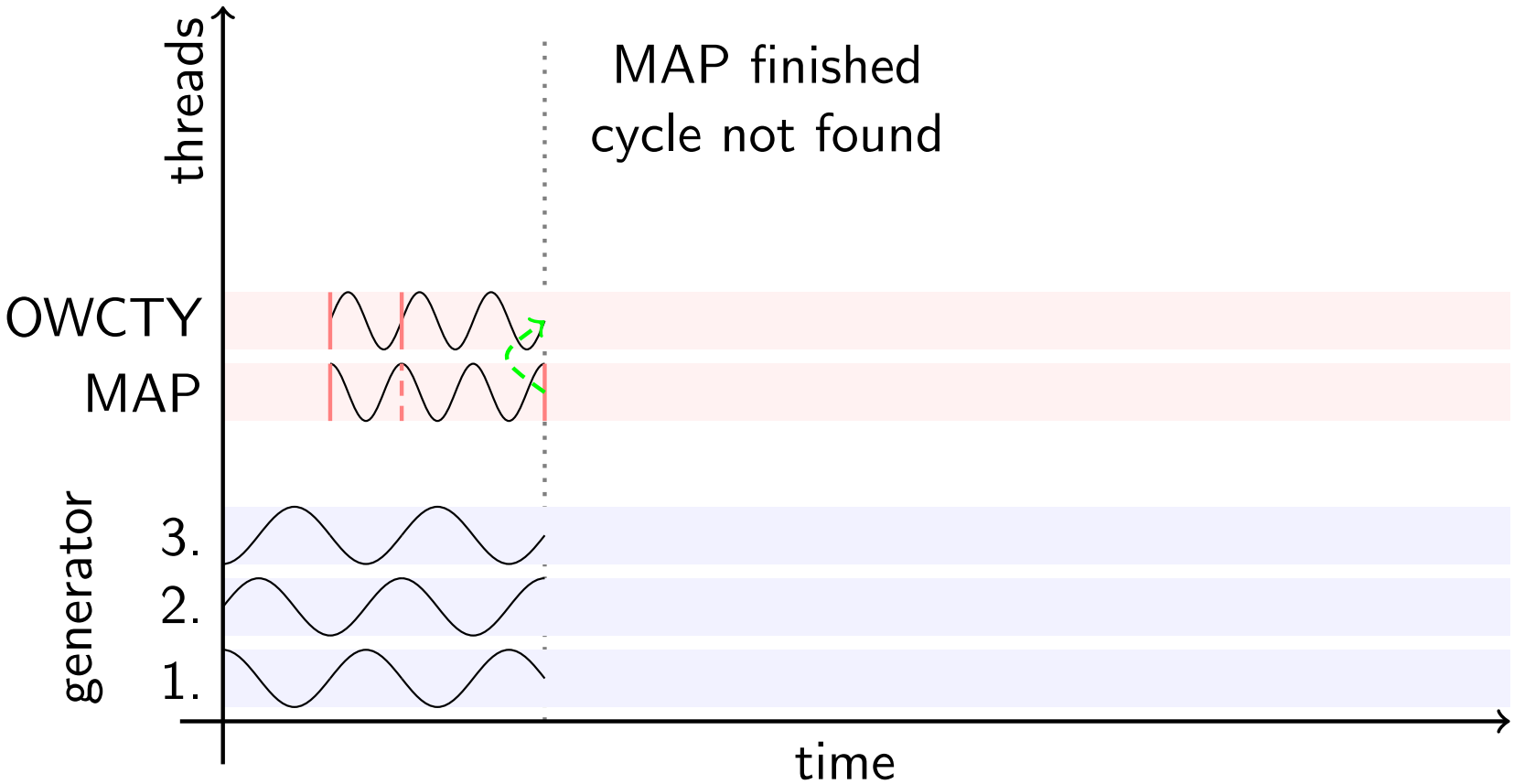
Kombinace algoritmů



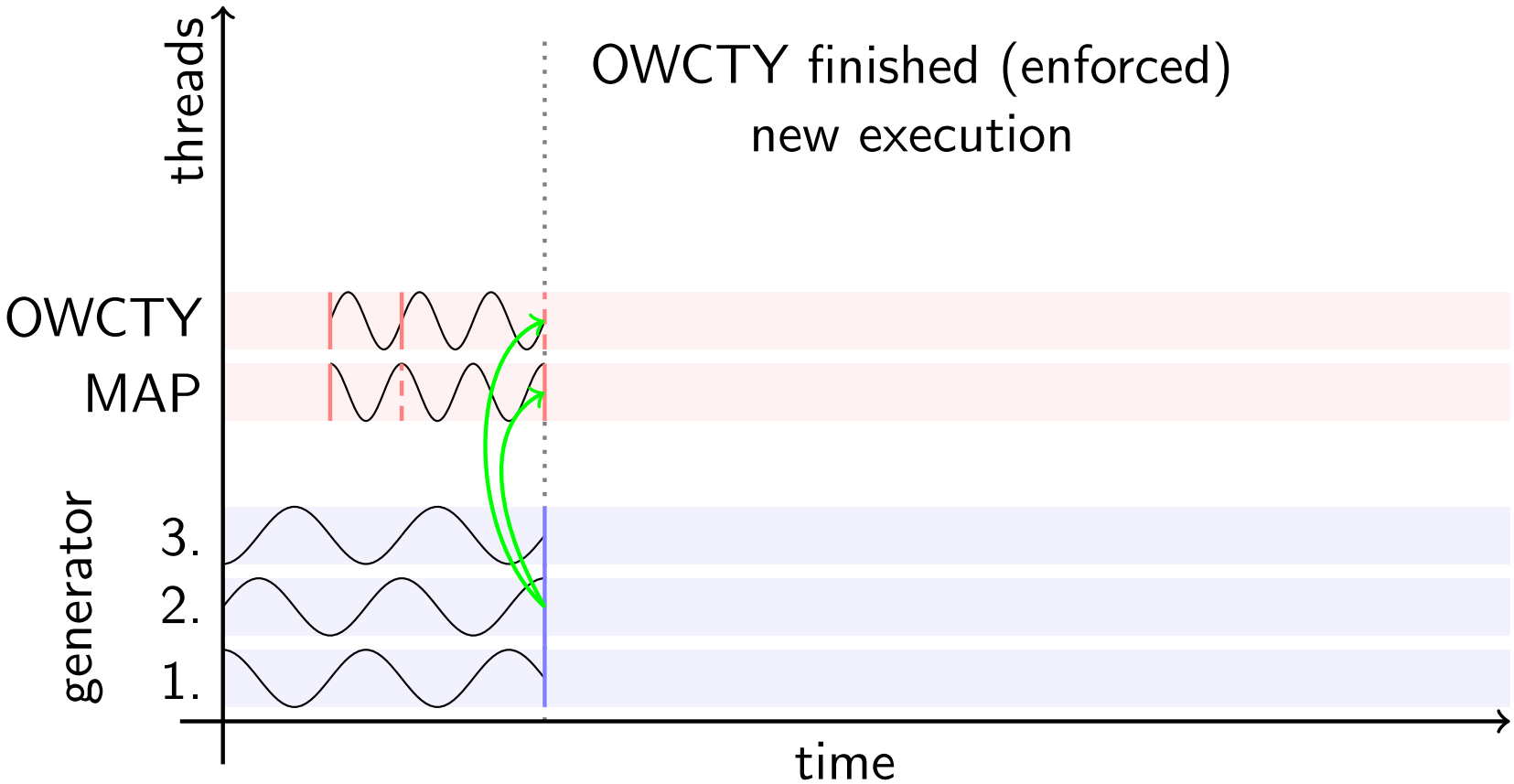
Kombinace algoritmů



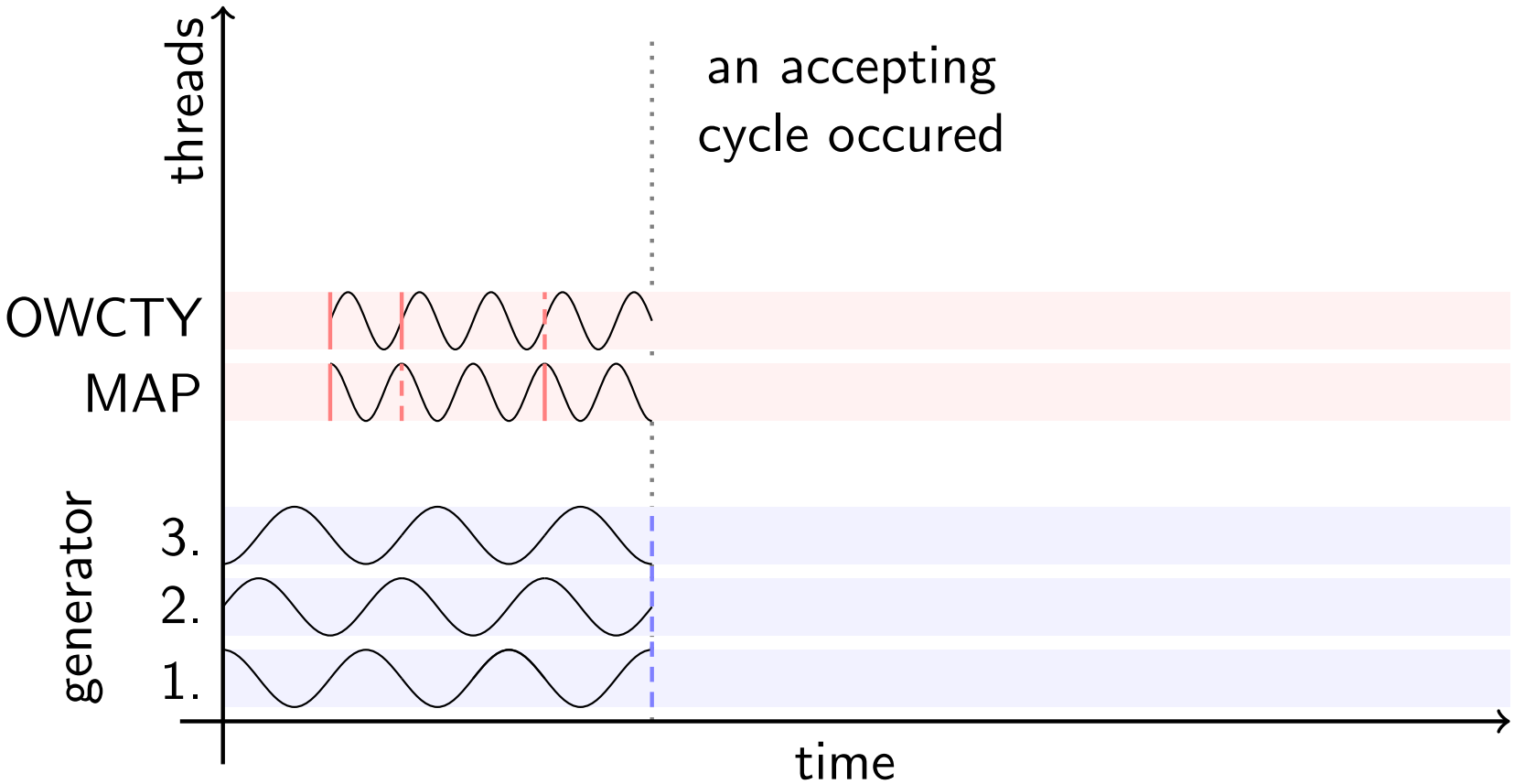
Kombinace algoritmů



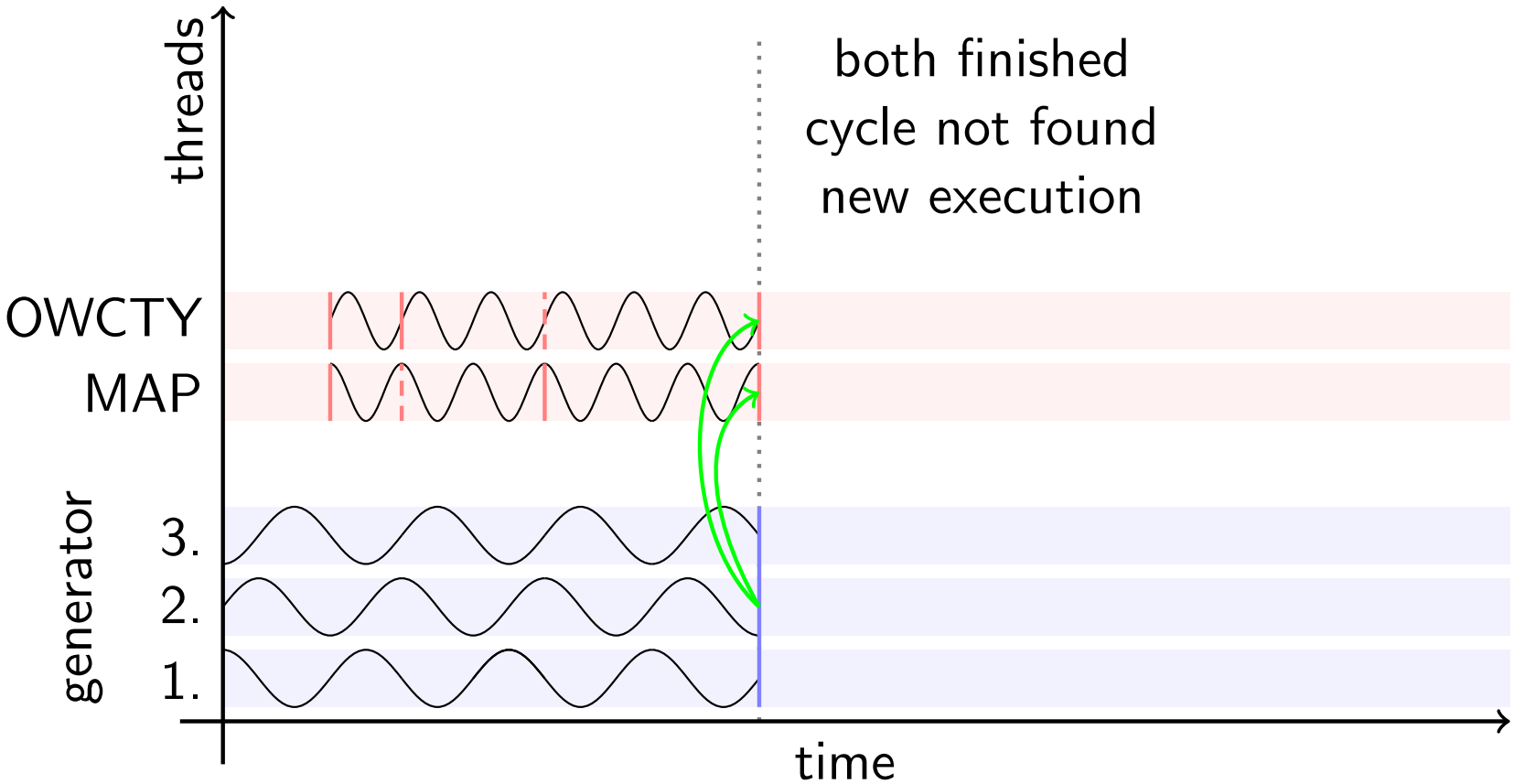
Kombinace algoritmů



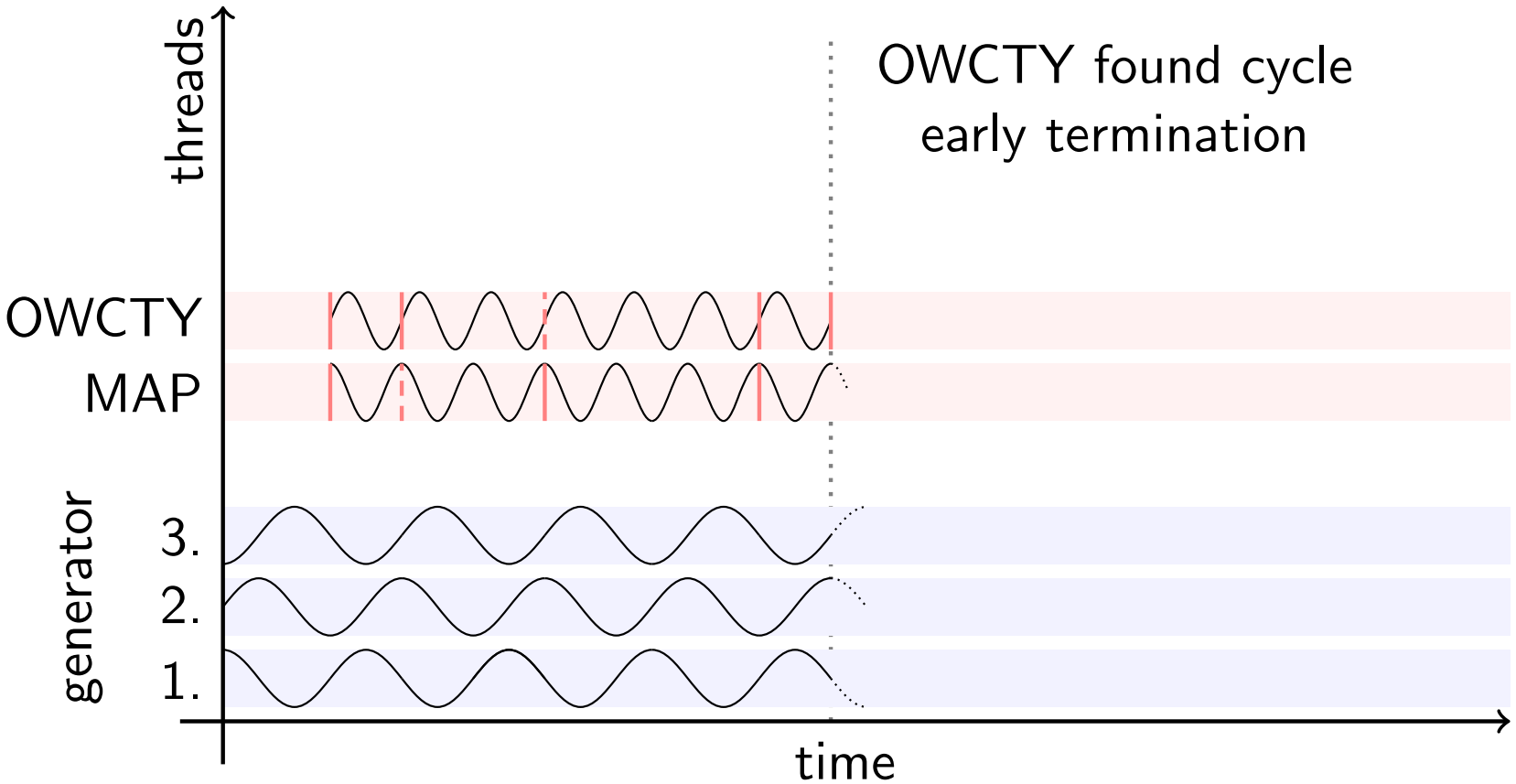
Kombinace algoritmů



Kombinace algoritmů



Kombinace algoritmů



Experimentální měření

Experimentální prostředí

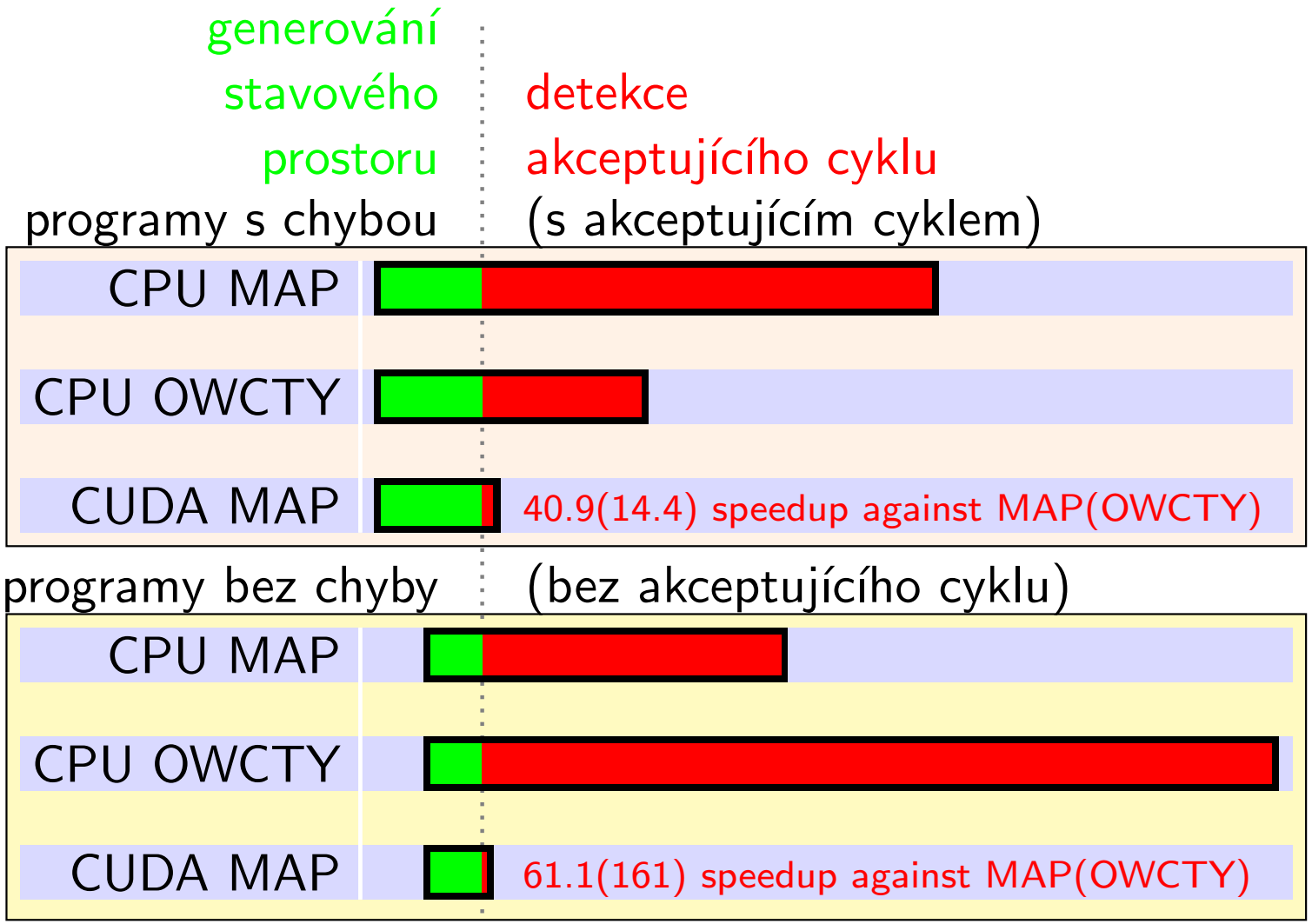
Stanice

- Linux OS
- AMD Phenom(tm) II X4 940 Processor @ 3GHz
- 8 GB DDR2 @ 1066 MHz RAM
- 2x NVIDIA GeForce GTX 280 s 1GB RAM

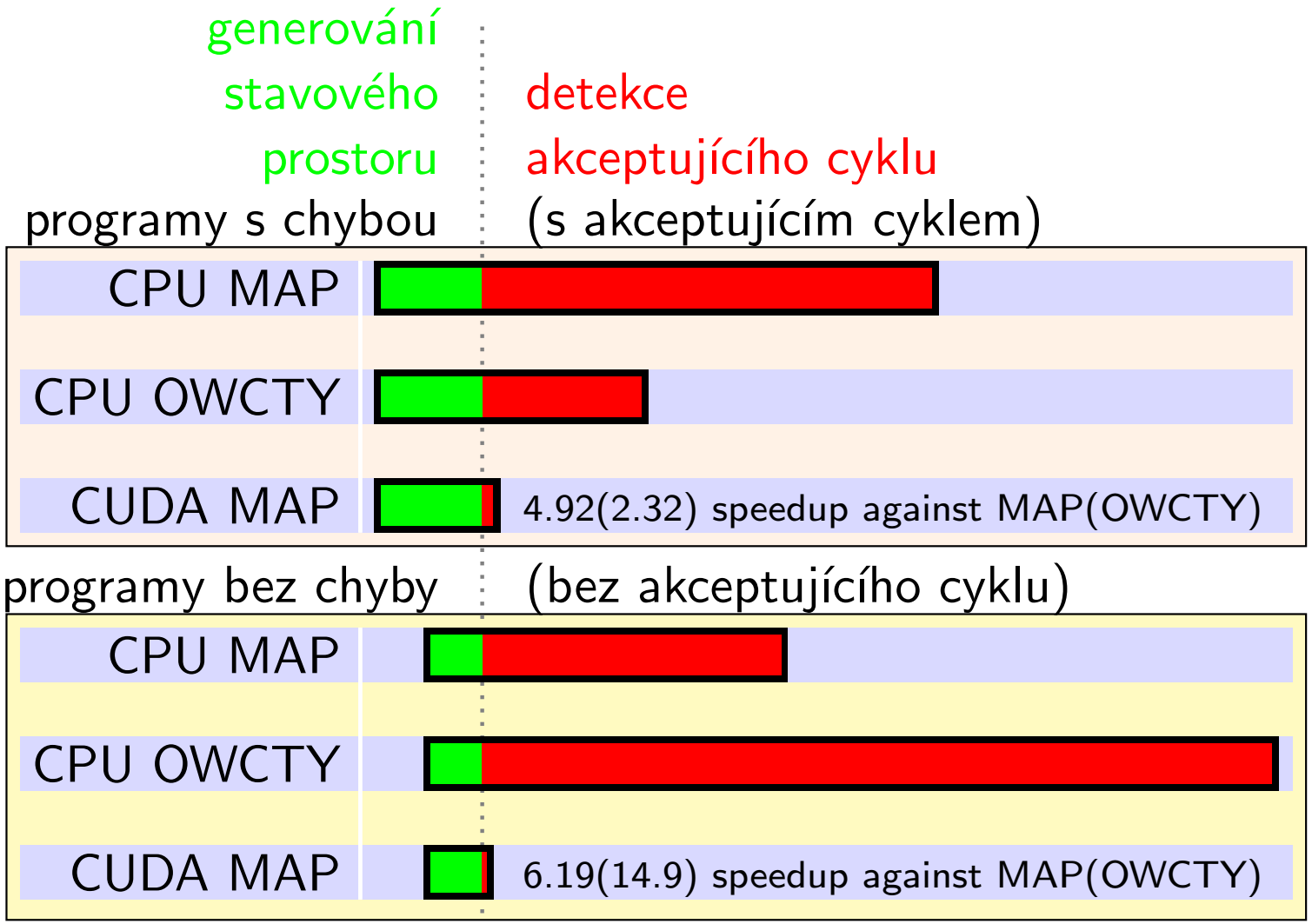
V nových experimentech

- GPU GTX480 with 1.5GB VRAM

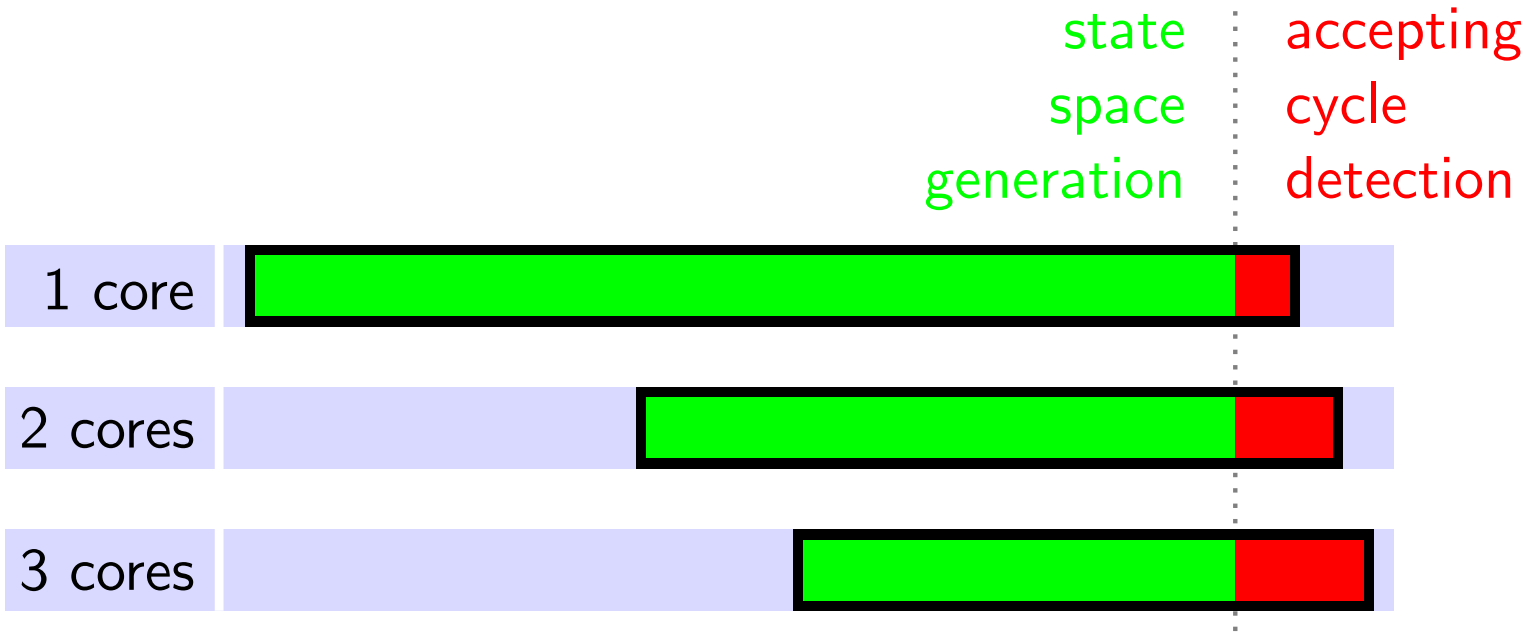
Akcelerace algoritmů na GPU (CPU nepoužívá CSR)



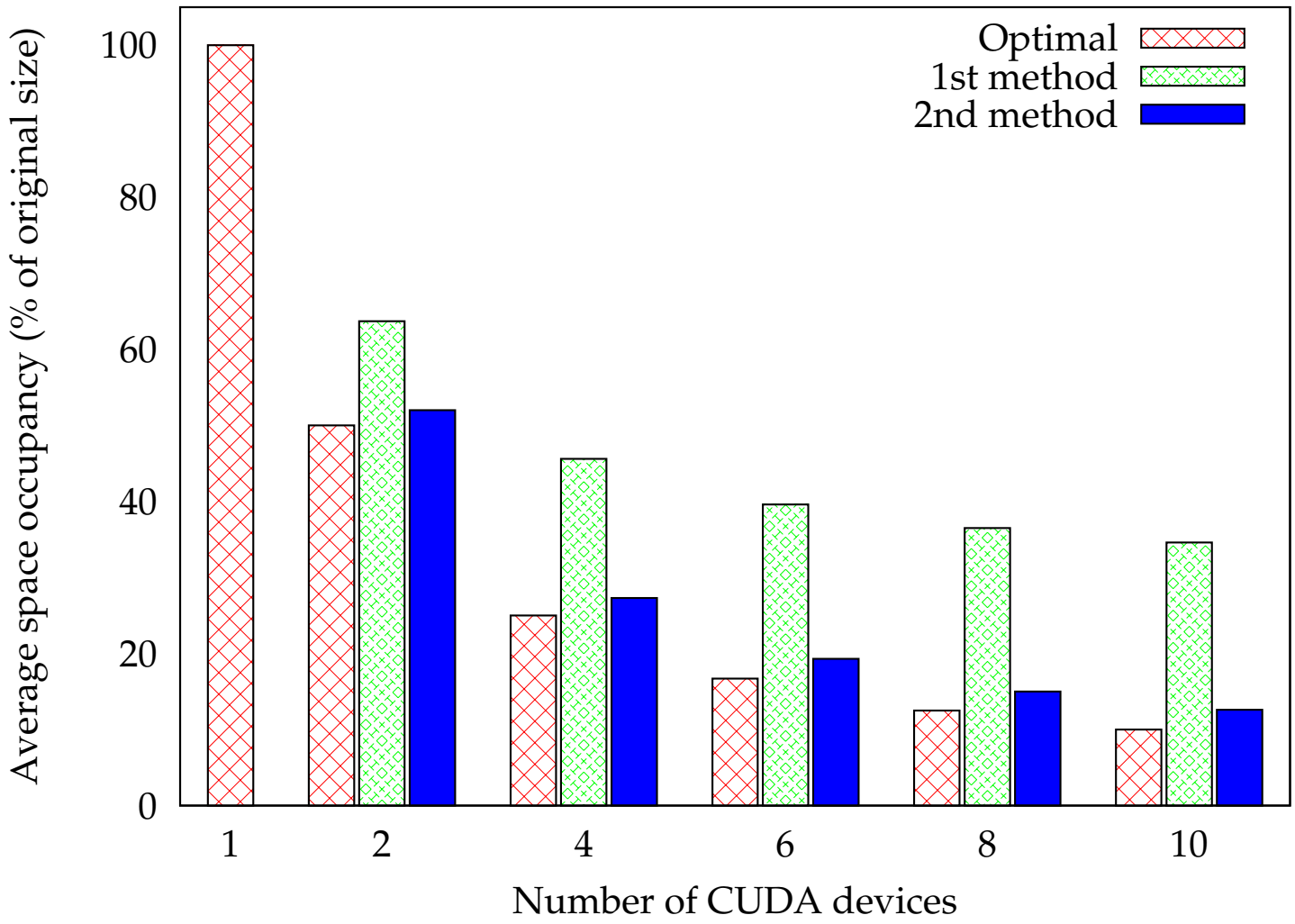
Akcelerace algoritmů na GPU (CPU nepoužívá CSR)



Dopad použití paralelní konstrukce CSR na MAP

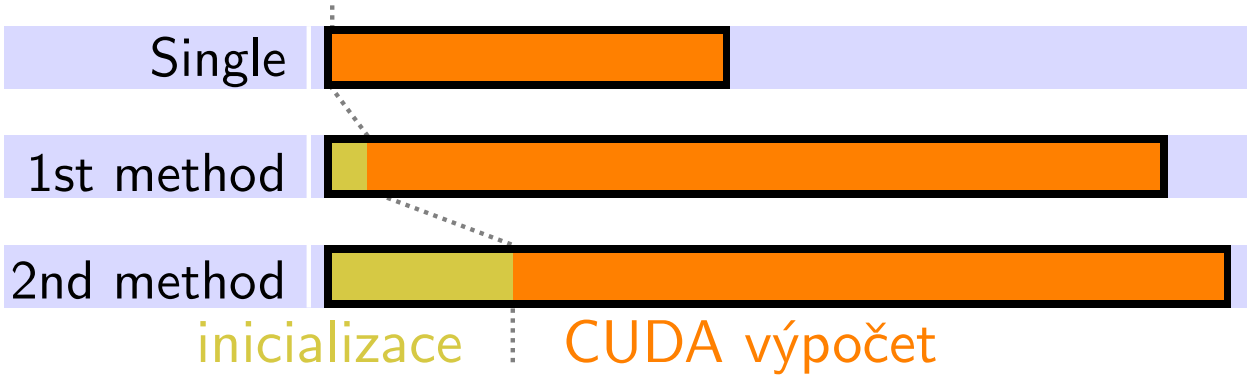


Využití více CUDA zařízení – redukce paměti

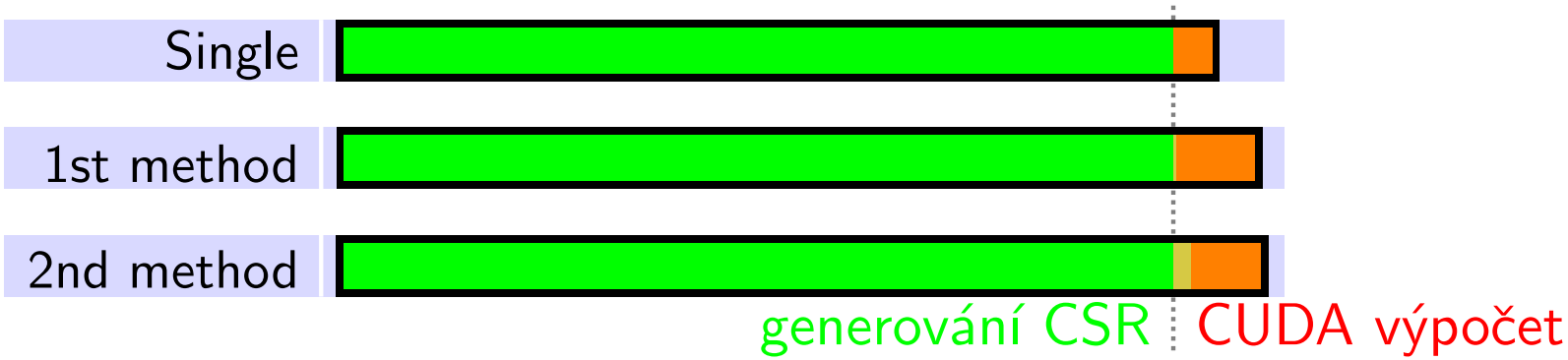


Využití více CUDA zařízení – zpomalení

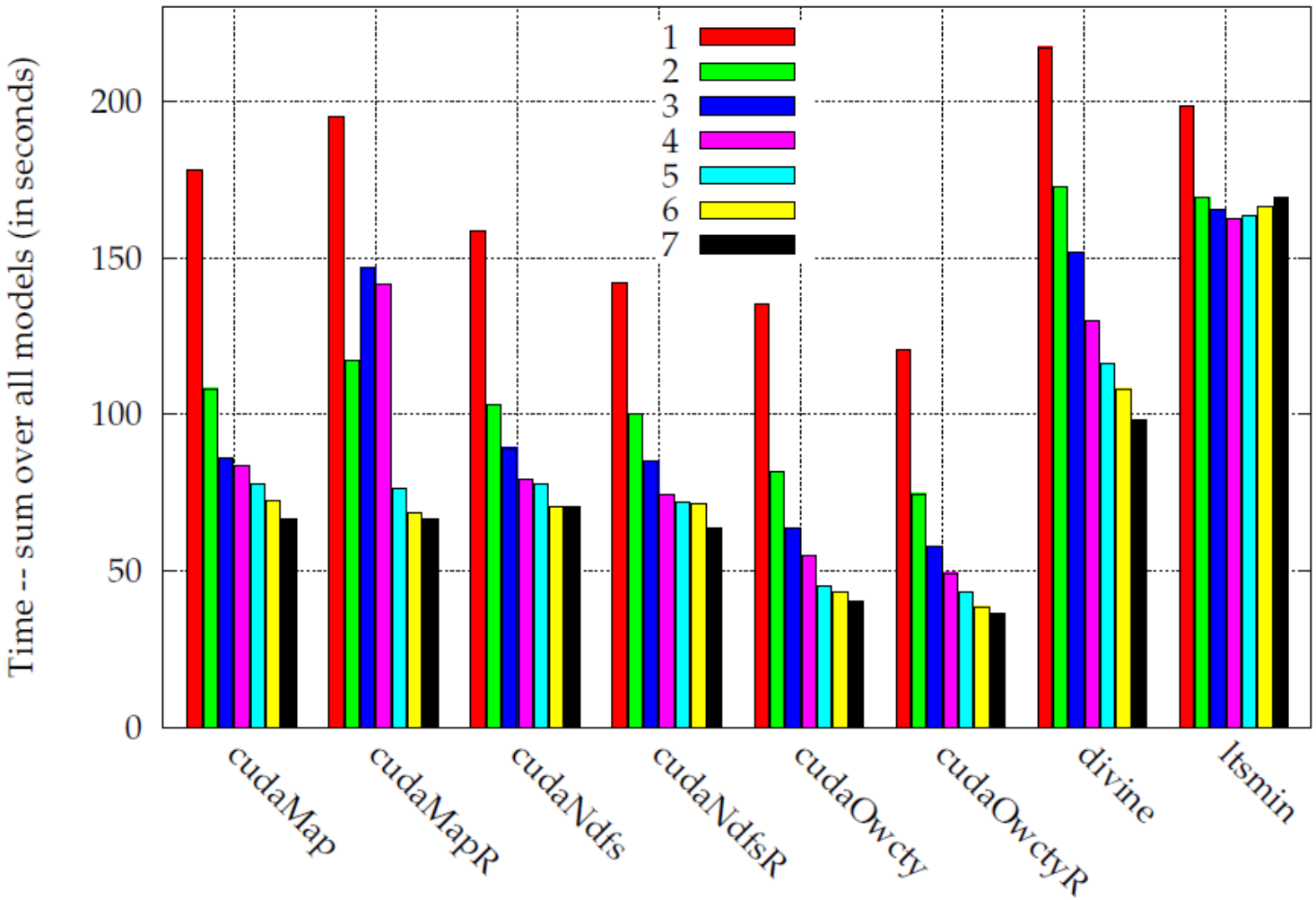
Zpomalení



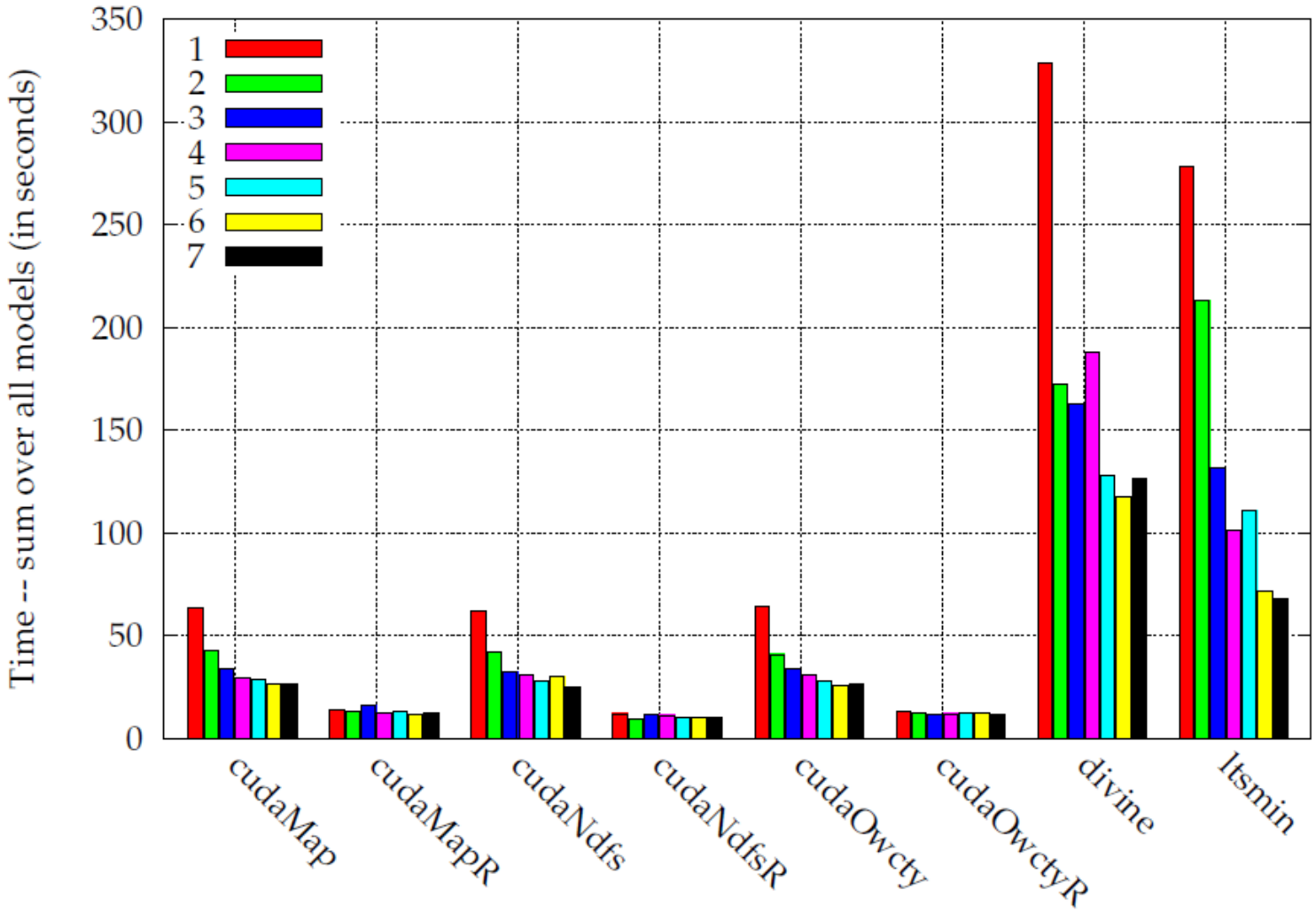
Dopad na celkovou proceduru verifikace



Porovnání se state-of-the-art (bez akceptujícího cyklu)



Porovnání se state-of-the-art (s akceptujícím cyklem)



Shrnutí

Závěrem

V několika větách ...

- Akcelerace samotných algoritmů a grafových primitiv je opodstatněná, nicméně v kontextu celé procedury je diskutabilní, zejména kvůli nákladnému generování grafu.
- Akcelerace algoritmů v celočíselné aritmetice.
- Dosaženo lepších časů s neoptimálními paralelními algoritmy na GPU, než s optimálními sekvenční algoritmy pracujícími na CPU.

Kudy dál?

- Jak akcelerovat generování stavového prostoru?
- Je to vůbec možné?

Bibliografie

- J. Barnat and P. Bauch and L. Brim and M. Ceska: **Designing Fast LTL Model Checking Algorithms for Many-Core GPUs**, Journal PDC, 2012.

- J. Barnat and P. Bauch and L. Brim and M. Ceska: **Employing Multiple CUDA Devices to Accelerate LTL Model Checking** 16th International Conference on Parallel and Distributed Systems (ICPADS 2010), IEEE Computer Society, 2010, 259-266.

- J. Barnat and L. Brim and M. Ceska and T. Lamr: **CUDA accelerated LTL Model Checking** 15th International Conference on Parallel and Distributed Systems (ICPADS 2009), IEEE Computer Society, 2009, 34-41.

- J. Barnat and L. Brim and M. Ceska: **DiVinE-CUDA: A Tool for GPU Accelerated LTL Model Checking** Electronic Proceedings in Theoretical Computer Science (PDMC 2009), volume 14, 2009, 107-111.