# CUDA accelerated LTL Model Checking

Jiří Barnat, Luboš Brim, **Milan Češka**, and Tomáš Lamr

Faculty of Informatics, Masaryk University
Brno, Czech Republic

based on [Barnat et al. ICPADS'09] and [Barnat et al. PDMC'09]

Mathematical and Engineering Methods in Computer Science
(MEMICS'09)

13.11, 2009

ParaDiSe
Parallel & Distributed
Systems Laboratory

1. Introduction
   - Motivation and contribution
   - LTL Model Checking
   - Maximal Accepting Predecessor algorithm
2. CUDA Accelerated MAP Algorithm
   - Reformulation of MAP algorithm
   - Compact CSR representation
3. Experimental evaluation
   - Impact of graph reduction
   - The overall run-times and speedup
4. Conclusion and Future Work

ParaDiSe
Parallel & Distributed Systems Laboratory

# Motivation

## Formal verification

- critical system - any mistake may have fatal consequences
- testing is insufficient - can not guarantee correctness
- formal methods can prove or disprove correctness of the system

## Model Checking

- fully automated approach to the formal verification
- state space explosion problem
- possible solution:
  - symbolic representation
  - reduction techniques
  - **platform-dependent verification**

ParaDiSe
Parallel & Distributed Systems Laboratory

# Motivation

## Formal verification

- critical system - any mistake may have fatal consequences
- testing is insufficient - can not guarantee correctness
- formal methods can prove or disprove correctness of the system

## Model Checking

- fully automated approach to the formal verification
- state space explosion problem
- possible solution:
  - symbolic representation
  - reduction techniques
  - **platform-dependent verification**

ParaDiSe
Parallel & Distributed Systems Laboratory

# Platform-dependent verification

**DiVinE**

- Explicit Parallel LTL Model Checker
- Focuses on full utilization of available HW power

**Successful stories on the following parallel platforms**

- Clusters, Grids
  DiVinE Cluster
- Multi-core workstations
  DiVinE Multi-Core

**Many-core architectures**

- Shared-memory setting, many computing cores
- Parallel computing platform of the future?
- Widely accessible due to GP GPU devices
- Not yet covered by DiVinE

ParaDiSe
Parallel & Distributed
Systems Laboratory

# Platform-dependent verification

**DiVinE**

- Explicit Parallel LTL Model Checker
- Focuses on full utilization of available HW power

**Successful stories on the following parallel platforms**

- Clusters, Grids
  DiVinE Cluster
- Multi-core workstations
  DiVinE Multi-Core

**Many-core architectures**

- Shared-memory setting, many computing cores
- Parallel computing platform of the future?
- Widely accessible due to GP GPU devices
- Not yet covered by DiVinE

ParaDiSe Parallel & Distributed Systems Laboratory

# Platform-dependent verification

**DiVinE**

- Explicit Parallel LTL Model Checker
- Focuses on full utilization of available HW power

**Successful stories on the following parallel platforms**

- Clusters, Grids
  DiVinE Cluster
- Multi-core workstations
  DiVinE Multi-Core

**Many-core architectures**

- Shared-memory setting, many computing cores
- Parallel computing platform of the future?
- Widely accessible due to GP GPU devices
- Not yet covered by DiVinE

ParaDiSe
Parallel & Distributed Systems Laboratory

# Platform-dependent verification

**DiVinE**

- Explicit Parallel LTL Model Checker
- Focuses on full utilization of available HW power

**Successful stories on the following parallel platforms**

- Clusters, Grids
  DiVinE Cluster
- Multi-core workstations
  DiVinE Multi-Core

**Many-core architectures**

- Shared-memory setting, many computing cores
- Parallel computing platform of the future?
- Widely accessible due to GP GPU devices
- Not yet covered by DiVinE

ParaDiSe
Parallel & Distributed Systems Laboratory

# Platform-dependent verification

**DiVinE**

- Explicit Parallel LTL Model Checker
- Focuses on full utilization of available HW power

**Successful stories on the following parallel platforms**

- Clusters, Grids
  DiVinE Cluster
- Multi-core workstations
  DiVinE Multi-Core

**Many-core architectures**

- Shared-memory setting, many computing cores
- Parallel computing platform of the future?
- Widely accessible due to GP GPU devices
- Not yet covered by DiVinE

ParaDiSe
Parallel & Distributed Systems Laboratory

# Platform-dependent verification

**DiVinE**

- Explicit Parallel LTL Model Checker
- Focuses on full utilization of available HW power

**Successful stories on the following parallel platforms**

- Clusters, Grids
  DiVinE Cluster
- Multi-core workstations
  DiVinE Multi-Core

**Many-core architectures**

- Shared-memory setting, many computing cores
- Parallel computing platform of the future?
- Widely accessible due to GP GPU devices
- Not yet covered by DiVinE

ParaDiSe
Parallel & Distributed
Systems Laboratory

# Many-core architecture

**NVIDIA CUDA Technology**

- Many-core architecture
- Hundreds of computing cores
- HW support for thousands of computing threads
- Requires quite specific memory-usage pattern
- Incompatible with random memory access (hashing)

**Need for many-core algorithms**

- Straightforward adaptation of multi-core algorithms to many-core architecture is inefficient
- Also the case of parallel LTL Model Checking

ParaDiSe
Parallel & Distributed Systems Laboratory

# Many-core architecture

**NVIDIA CUDA Technology**

- Many-core architecture
- Hundreds of computing cores
- HW support for thousands of computing threads
- Requires quite specific memory-usage pattern
- Incompatible with random memory access (hashing)

**Need for many-core algorithms**

- Straightforward adaptation of multi-core algorithms to many-core architecture is inefficient
- Also the case of parallel LTL Model Checking

ParaDiSe
Parallel & Distributed Systems Laboratory

# Contribution

**Acceleration of model checking process by full utilization of modern massively parallel architectures**

- successful redesign of Maximal Accepting Predecessor algorithm allowing for significant GPU acceleration [Barnat et al. ICPADS'09].

**DiVinE-CUDA**

- tool for CUDA Accelerated LTL Model Checking [Barnat et al. PDMC'09].

**Experimental evaluation**

- significant reduction of runtimes
- identification of the main bottleneck of the designed approach.

**Sketch of possible solutions**

ParaDiSe
Parallel & Distributed Systems Laboratory

# Contribution

**Acceleration of model checking process by full utilization of modern massively parallel architectures**

- successful redesign of Maximal Accepting Predecessor algorithm allowing for significant GPU acceleration [Barnat et al. ICPADS'09].

**DiVinE-CUDA**

- tool for CUDA Accelerated LTL Model Checking [Barnat et al. PDMC'09].

**Experimental evaluation**

- significant reduction of runtimes
- identification of the main bottleneck of the designed approach.

**Sketch of possible solutions**

ParaDiSe
Parallel & Distributed Systems Laboratory

## Contribution

**Acceleration of model checking process by full utilization of modern massively parallel architectures**

- successful redesign of Maximal Accepting Predecessor algorithm allowing for significant GPU acceleration [Barnat et al. ICPADS'09].

**DiVinE-CUDA**

- tool for CUDA Accelerated LTL Model Checking [Barnat et al. PDMC'09].

**Experimental evaluation**

- significant reduction of runtimes
- identification of the main bottleneck of the designed approach.

ParaDiSe Parallel & Distributed Systems Laboratory

**Sketch of possible solutions**

# Contribution

**Acceleration of model checking process by full utilization of modern massively parallel architectures**

- successful redesign of Maximal Accepting Predecessor algorithm allowing for significant GPU acceleration [Barnat et al. ICPADS'09].

**DiVinE-CUDA**

- tool for CUDA Accelerated LTL Model Checking [Barnat et al. PDMC'09].

**Experimental evaluation**

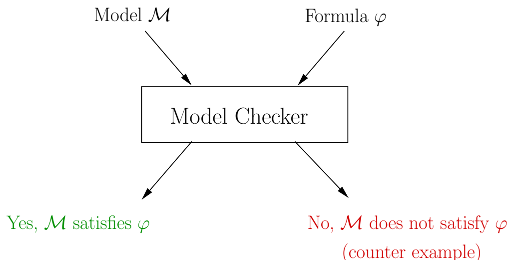
- significant reduction of runtimes
- identification of the main bottleneck of the designed approach.

**Sketch of possible solutions**

ParaDiSe
Parallel & Distributed Systems Laboratory

# LTL Model Checking

- inspected system → suitable model $M$
- required property of the system → formula $\varphi$ in Linear Temporal Logic (LTL)
- problem : **Does the model $M$ satisfy the formula $\varphi$ ?**

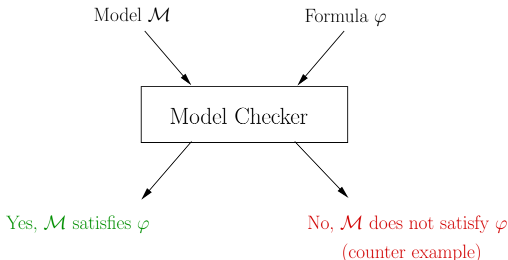



- solution: automata based approach
  - reduction on **accepting cycle detection**

ParaDiSe Parallel & Distributed Systems Laboratory

# LTL Model Checking

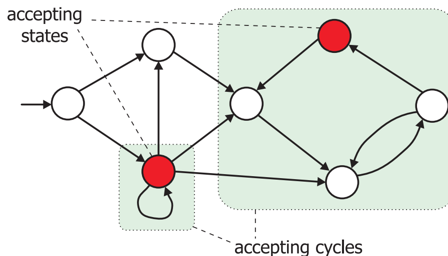
- inspected system → suitable model $M$
- required property of the system → formula $\varphi$ in Linear Temporal Logic (LTL)
- problem : **Does the model $M$ satisfy the formula $\varphi$ ?**




- solution: automata based approach
  - reduction on **accepting cycle detection**

ParaDiSe Parallel & Distributed Systems Laboratory

# Algorithms for accepting cycle detection



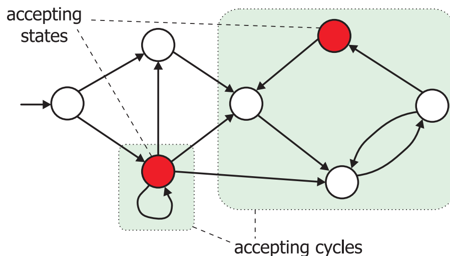

Optimal time complexity but hard to parallelize:

- Nested DFS
- Tarjan's SCC decomposition

Unoptimal time complexity but easy to parallelize:

- OWCTY
- **Maximal accepting predecessor (MAP)**

# Algorithms for accepting cycle detection




Optimal time complexity but hard to parallelize:

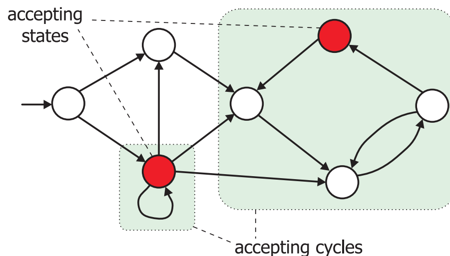
- Nested DFS
- Tarjan's SCC decomposition

Unoptimal time complexity but easy to parallelize:

- OWCTY
- **Maximal accepting predecessor (MAP)**

ParaDiSe
Parallel & Distributed
Systems Laboratory

# Algorithms for accepting cycle detection




Optimal time complexity but hard to parallelize:

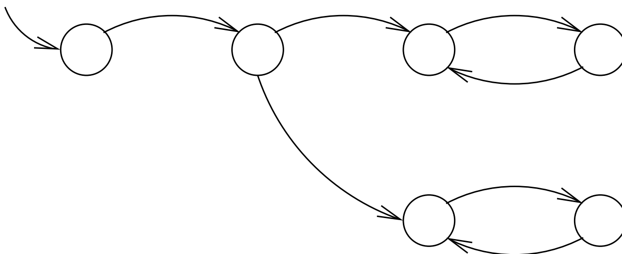
- Nested DFS
- Tarjan's SCC decomposition

Unoptimal time complexity but easy to parallelize:

- OWCTY
- **Maximal accepting predecessor (MAP)**

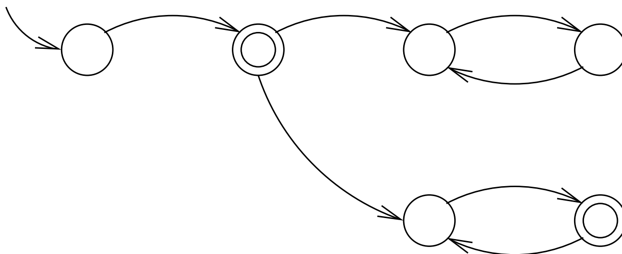
ParaDiSe
Parallel & Distributed Systems Laboratory

# Algorithm MAP



Graph corresponding to the state space.

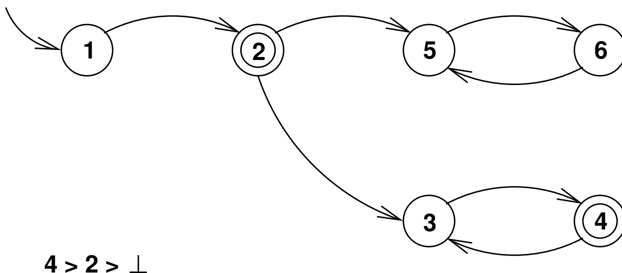
# Algorithm MAP



Accepting vertices, accepting cycle.

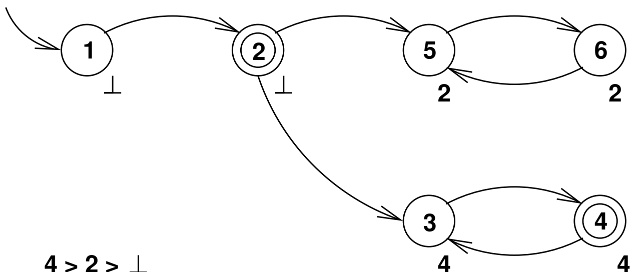
ParaDiSe
Parallel & Distributed
Systems Laboratory

# Algorithm MAP




Vertex ordering.

# Algorithm MAP

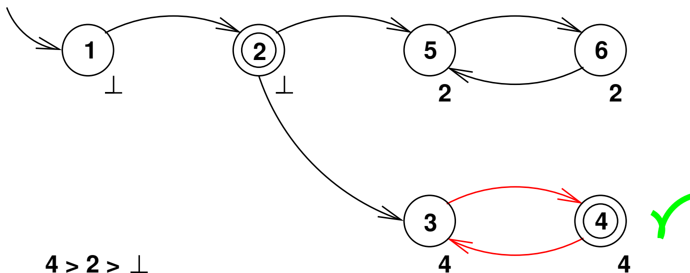



Maximal Accepting Predecessor (MAP)

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

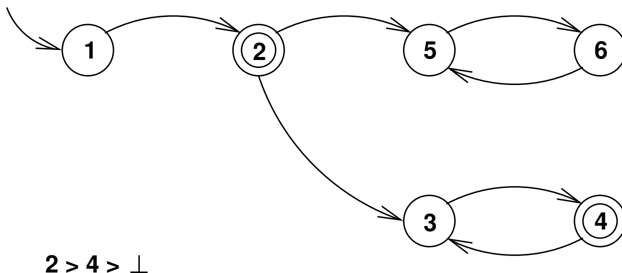
ParaDiSe Parallel & Distributed Systems Laboratory

# Algorithm MAP




$map(v) = v \implies$ accepting cycle

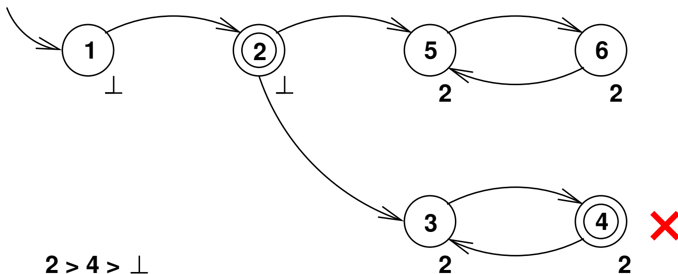
# Algorithm MAP




What if $2 > 4$ ?

ParaDiSe
Parallel & Distributed
Systems Laboratory

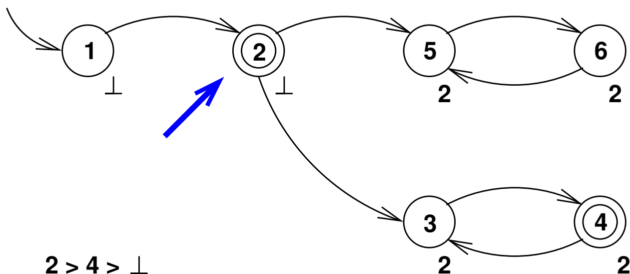
# Algorithm MAP




Accepting cycle undetected.

ParaDiSe
Parallel & Distributed
Systems Laboratory

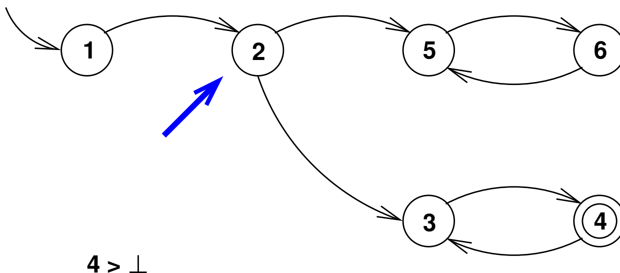
# Algorithm MAP




If no accepting cycle is found, then maximal accepting vertices cannot be part of an accepting cycle.

ParaDiSe Parallel & Distributed Systems Laboratory

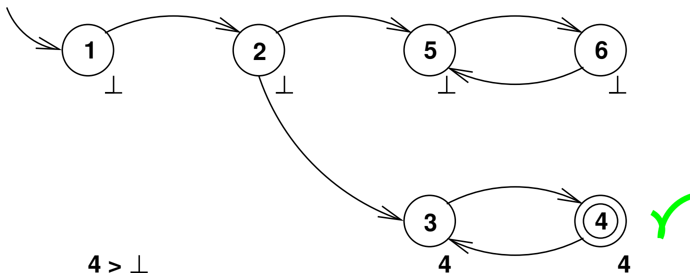
# Algorithm MAP




Maximal accepting vertices marked as non-accepting.

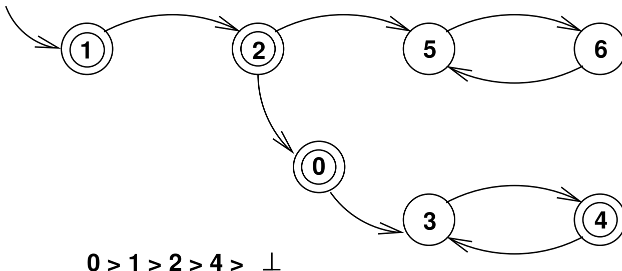
ParaDiSe
Parallel & Distributed
Systems Laboratory

# Algorithm MAP



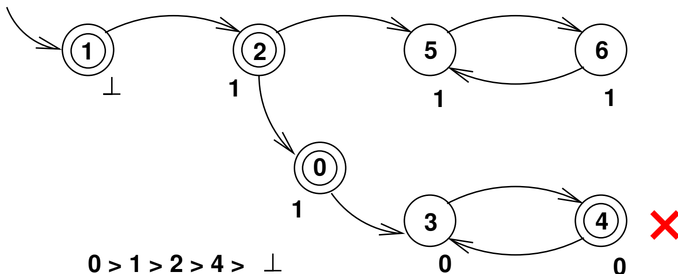

Repeat until accepting cycle is found or there are no accepting vertices.

# Algorithm MAP - partitioning into subgraphs




ParaDiSe
Parallel & Distributed
Systems Laboratory

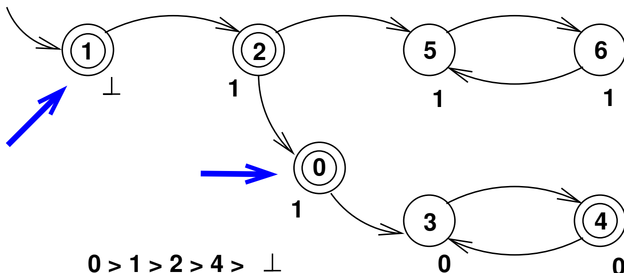
# Algorithm MAP - partitioning into subgraphs




Accepting cycle undetected.

ParaDiSe Parallel & Distributed Systems Laboratory

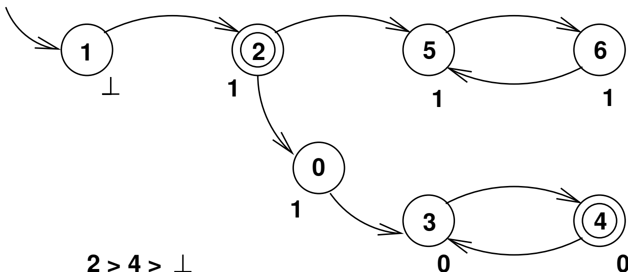
# Algorithm MAP - partitioning into subgraphs




If no accepting cycle is found, then maximal accepting vertices cannot be part of an accepting cycle.

ParaDiSe
Parallel & Distributed Systems Laboratory

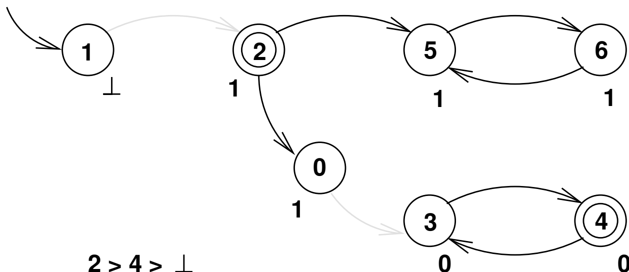
# Algorithm MAP - partitioning into subgraphs




Maximal accepting vertices marked as non-accepting.

ParaDiSe
Parallel & Distributed
Systems Laboratory

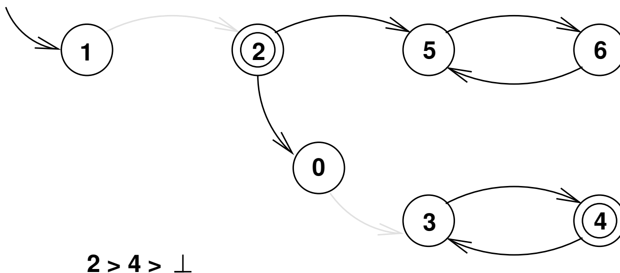
# Algorithm MAP - partitioning into subgraphs




Partitioning into subgraphs according to the *map* values.

ParaDiSe
Parallel & Distributed
Systems Laboratory

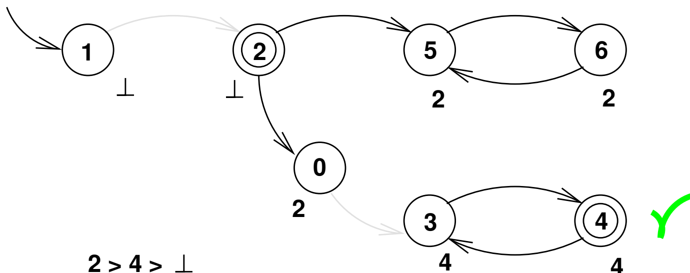
# Algorithm MAP - partitioning into subgraphs




No cycle of the original graph maps to multiple partitions.

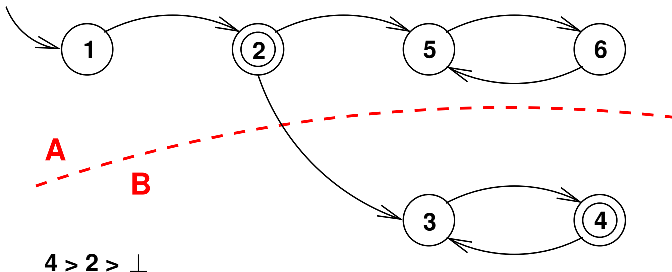
ParaDiSe Parallel & Distributed Systems Laboratory

# Algorithm MAP - partitioning into subgraphs



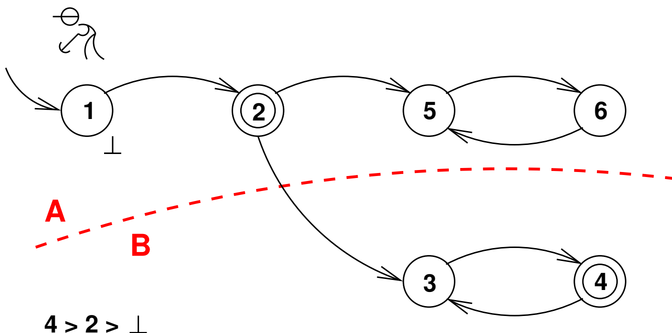

Earlier detection of accepting cycle.

# Computing values of map – multi-cores




Graph partitioning.

# Computing values of map – multi-cores



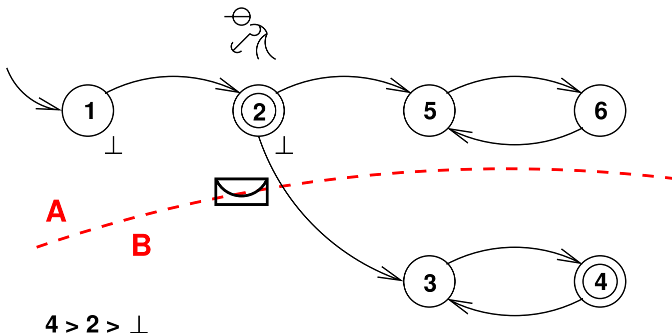

Each core processes own vertices.

# Computing values of map – multi-cores



Each core processes own vertices.

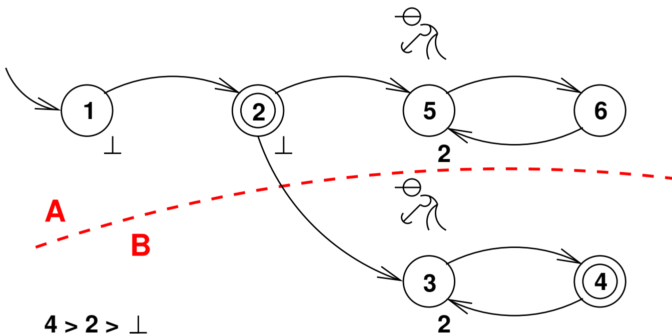
# Computing values of map – multi-cores




Non local vertices are sent to the owners.

ParaDiSe Parallel & Distributed Systems Laboratory

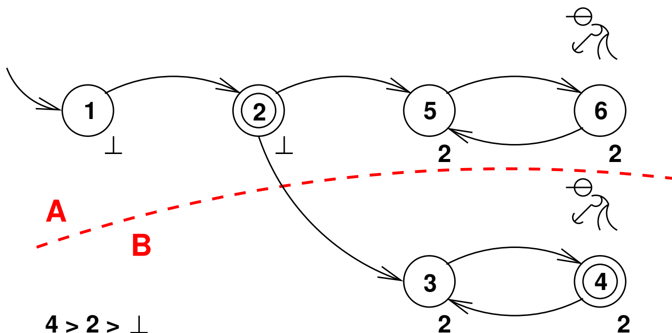
# Computing values of map – multi-cores




Parallel processing of vertices.

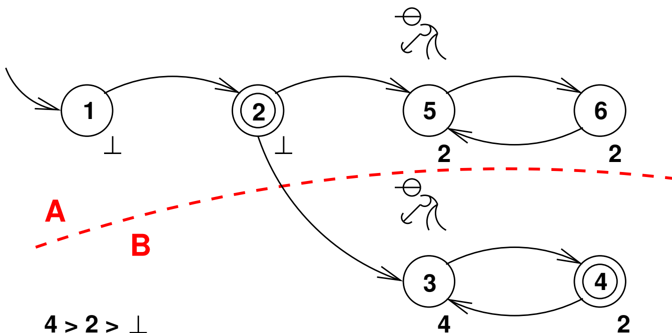
ParaDiSe
Parallel & Distributed
Systems Laboratory

# Computing values of map – multi-cores



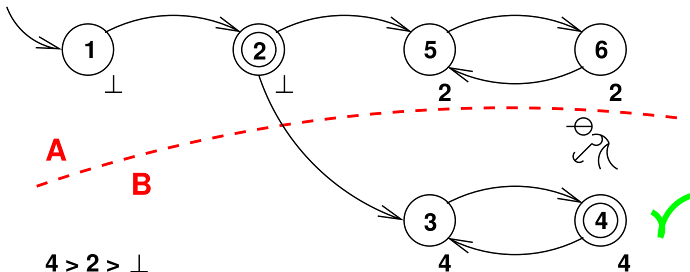

Parallel processing of vertices.

# Computing values of map – multi-cores



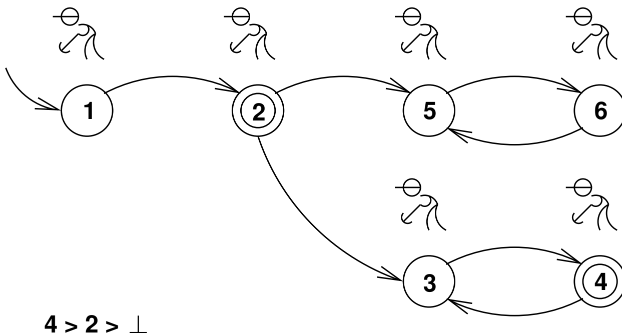

Parallel processing of vertices.

# Computing values of map – multi-cores




Accepting cycle found.

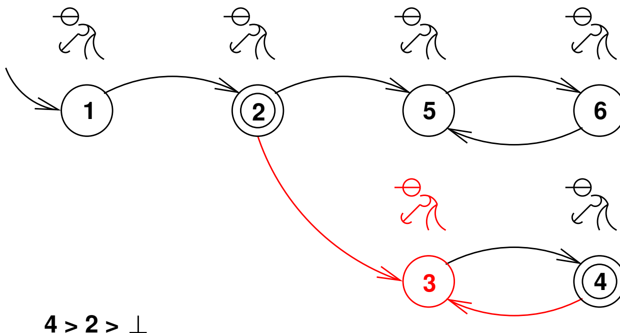
# Computing values of map – many-cores




One thread per vertex.

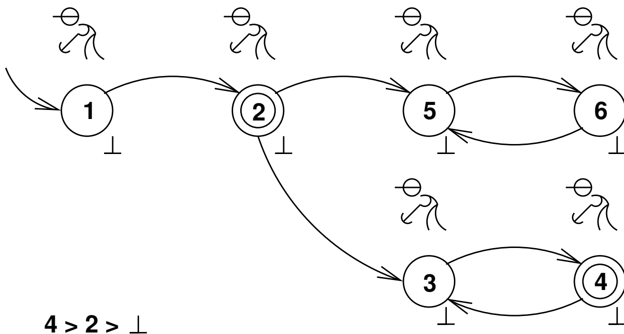
ParaDiSe Parallel & Distributed Systems Laboratory

# Computing values of map – many-cores



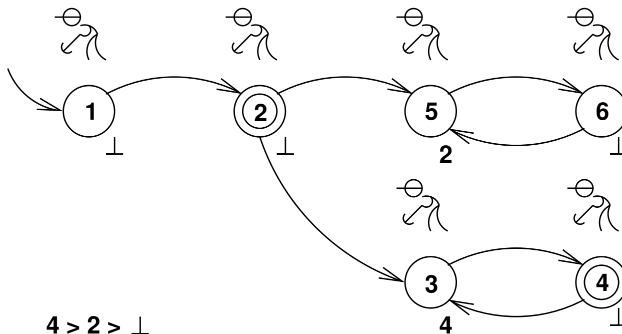

Each thread processes all incoming edges.

# Computing values of map – many-cores




Threads proceed simultaneously.

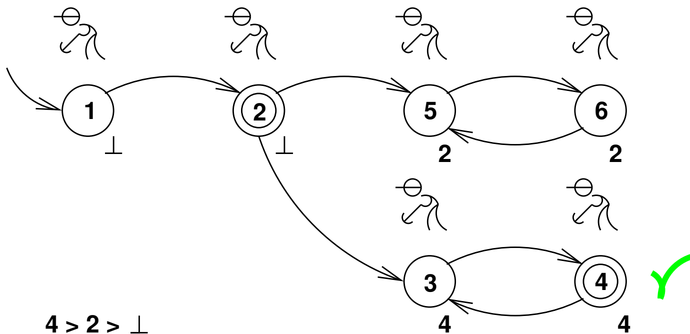
# Computing values of map – many-cores




Threads proceed simultaneously.

ParaDiSe Parallel & Distributed Systems Laboratory

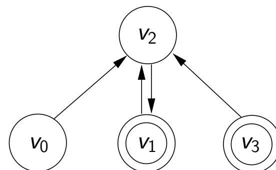
# Computing values of map – many-cores




Accepting cycle found.

ParaDiSe
Parallel & Distributed Systems Laboratory

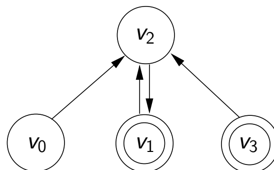
# MAP Algorithm as Matrix-Vector Product




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

**Matrix of predecessors**

# MAP Algorithm as Matrix-Vector Product




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

**Matrix of predecessors**

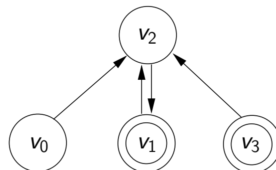
$$\begin{array}{c} \vec{m} \\ \bot \\ \bot \\ \bot \\ \bot \end{array}$$

**Additional data per vertex**

$\vec{m}$ – map values

ParaDiSe Parallel & Distributed Systems Laboratory

# MAP Algorithm as Matrix-Vector Product




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

**Matrix of predecessors**

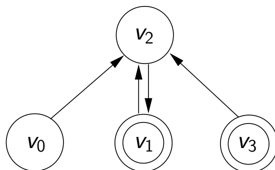
| $\vec{m}$ | $\vec{o}$ |
|---|---|
| $\perp$ | $\perp$ |
| $\perp$ | $\perp$ |
| $\perp$ | $\perp$ |
| $\perp$ | $\perp$ |

**Additional data per vertex**

$\vec{o}$ – oldmap values

ParaDiSe Parallel & Distributed Systems Laboratory

# MAP Algorithm as Matrix-Vector Product




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

**Matrix of predecessors**

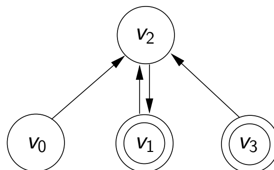
| $\vec{m}$ | $\vec{o}$ | $\vec{\mathcal{A}}$ |
|---|---|---|
| $\perp$ | $\perp$ | 0 |
| $\perp$ | $\perp$ | 1 |
| $\perp$ | $\perp$ | 0 |
| $\perp$ | $\perp$ | 1 |

**Additional data per vertex**

$\vec{\mathcal{A}}$ – is_accepting

ParaDiSe
Parallel & Distributed Systems Laboratory

# MAP Algorithm as Matrix-Vector Product




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

**Matrix of predecessors**

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

**Additional data per vertex**

$\vec{r}$ – has been modified

ParaDiSe Parallel & Distributed Systems Laboratory

# MAP Algorithm as Matrix-Vector Product

$$M \times V = V'$$

ParaDiSe
Parallel & Distributed
Systems Laboratory

# MAP Algorithm as Matrix-Vector Product

$$M \times V = V'$$

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

ParaDiSe Parallel & Distributed Systems Laboratory

# MAP Algorithm as Matrix-Vector Product

$$M \times V = V'$$

i

i

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

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

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

ParaDiSe
Parallel & Distributed
Systems Laboratory

# MAP Algorithm as Matrix-Vector Product

$$M \times V = V'$$

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

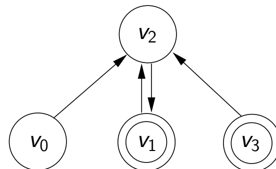
$$V'[i] = \max_{0 \leq j \leq n} M[i][j] \cdot \mathrm{maxacc(V[j],i)}$$

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

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

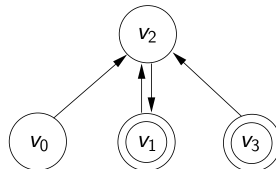
ParaDiSe Parallel & Distributed Systems Laboratory

# MAP Algorithm as Matrix-Vector Product – example




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

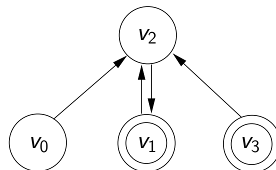
# MAP Algorithm as Matrix-Vector Product – example




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

ParaDiSe Parallel & Distributed Systems Laboratory

# MAP Algorithm as Matrix-Vector Product – example




$$\begin{array}{cccc} v_0 & v_1 & v_2 & v_3 \end{array}$$

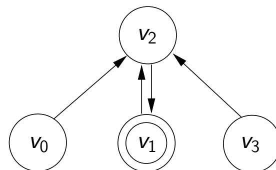
$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \times \begin{array}{ccc} \vec{m} & \vec{o} & \vec{A} \\ \bot & \bot & 0 \\ \mathbf{3} & \bot & 1 \\ 3 & \bot & 0 \\ \bot & \bot & 1 \end{array} = \begin{array}{cc} \vec{m} & \vec{r} \\ \bot & 0 \\ 3 & \mathbf{0} \\ 3 & 0 \\ \bot & 0 \end{array}$$

**Fix point.**

**No accepting cycles found**

ParaDiSe Parallel & Distributed Systems Laboratory

# MAP Algorithm as Matrix-Vector Product – example

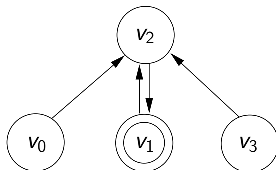



| $m$ | $o$ | $\mathcal{A}$ | | $m$ | $o$ | $\mathcal{A}$ |
|---|---|---|---|---|---|---|
| $\perp$ | $\perp$ | 0 | | $\perp$ | $\perp$ | 0 |
| 3 | $\perp$ | 1 | $\Longrightarrow$ | 3 | $\perp$ | 1 |
| 3 | $\perp$ | 0 | | 3 | $\perp$ | 0 |
| $\perp$ | $\perp$ | 1 | | $\perp$ | $\perp$ | **0** |

**Mark $v_3$ as non-accepting**

ParaDiSe Parallel & Distributed Systems Laboratory

# MAP Algorithm as Matrix-Vector Product – example

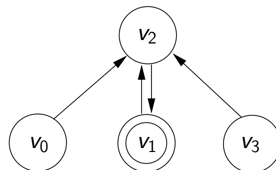



| $m$ | $o$ | $\mathcal{A}$ | | $m$ | $o$ | $\mathcal{A}$ |
|---|---|---|---|---|---|---|
| $\bot$ | $\bot$ | 0 | | $\bot$ | $\bot$ | 0 |
| 3 | $\bot$ | 1 | $\Longrightarrow$ | $\bot$ | **3** | 1 |
| 3 | $\bot$ | 0 | | $\bot$ | **3** | 0 |
| $\bot$ | $\bot$ | 0 | | $\bot$ | $\bot$ | 0 |

**Partitioning into subgraphs**

ParaDiSe Parallel & Distributed Systems Laboratory

# MAP Algorithm as Matrix-Vector Product – example




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

ParaDiSe
Parallel & Distributed
Systems Laboratory

# MAP Algorithm as Matrix-Vector Product – example

$v_2$ $v_0$ $v_1$ $v_3$

$$v_1 \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \times \begin{matrix} \vec{m} & \vec{o} & \vec{\mathcal{A}} \\ \bot & \bot & 0 \\ \bot & 3 & 1 \\ 1 & 3 & 0 \\ \bot & \bot & 0 \end{matrix} = \begin{matrix} \vec{m} & \vec{r} \\ \bot & 0 \\ \mathbf{1} & \mathbf{1} \\ 1 & \mathbf{0} \\ \bot & 0 \end{matrix}$$

(matrix columns: $v_0\ v_1\ v_2\ v_3$)

**Accepting cycles found** $map(v1) = v1$

ParaDiSe Parallel & Distributed Systems Laboratory

# Compact representation

**GPU memory is expensive**

- Size of the GPU memory is limited
- If a thread accesses too much memory, it blocks other threads

**Matrix of predecessors is sparse**

- The matrix is made of 1 and 0 only
- Handling full matrices is memory inefficient
- Compact Sparse Row (CSR) representation

ParaDiSe
Parallel & Distributed
Systems Laboratory

# Compact representation

**GPU memory is expensive**

- Size of the GPU memory is limited
- If a thread accesses too much memory, it blocks other threads

**Matrix of predecessors is sparse**

- The matrix is made of 1 and 0 only
- Handling full matrices is memory inefficient
- Compact Sparse Row (CSR) representation

ParaDiSe
Parallel & Distributed
Systems Laboratory

# CSR representation of sparse matrices

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

ParaDiSe
Parallel & Distributed Systems Laboratory

# CSR representation of sparse matrices

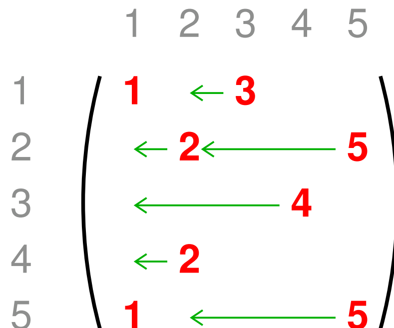
|   | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | **1** |   | **1** |   |   |
| 2 |   | **1** |   |   | **1** |
| 3 |   |   |   | **1** |   |
| 4 |   | **1** |   |   |   |
| 5 | **1** |   |   |   | **1** |

ParaDiSe
Parallel & Distributed Systems Laboratory

# CSR representation of sparse matrices

|   | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | **1** |  | **3** |  |  |
| 2 |  | **2** |  |  | **5** |
| 3 |  |  |  | **4** |  |
| 4 |  | **2** |  |  |  |
| 5 | **1** |  |  |  | **5** |

# CSR representation of sparse matrices




ParaDiSe
Parallel & Distributed Systems Laboratory

# CSR representation of sparse matrices

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | **1** | **3** | | | |
| 2 | **2** | **5** | | | |
| 3 | **4** | | | | |
| 4 | **2** | | | | |
| 5 | **1** | **5** | | | |

ParaDiSe
Parallel & Distributed Systems Laboratory

# CSR representation of sparse matrices

| | |
|---|---|
| 1 | 3 |
| 2 | 5 |
| 4 | |
| 2 | |
| 1 | 5 |

ParaDiSe
Parallel & Distributed Systems Laboratory

# CSR representation of sparse matrices

Mc=( 1 3 )

2 5

4

2

1 5

ParaDiSe
Parallel & Distributed
Systems Laboratory

# CSR representation of sparse matrices

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

**4**

**2**

**1 5**

# CSR representation of sparse matrices

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

**2**

**1 5**

ParaDiSe
Parallel & Distributed
Systems Laboratory

# CSR representation of sparse matrices

Mc=( 1 3 2 5 4 2 )

↑ ↑ ↑

1 5

# CSR representation of sparse matrices

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

ParaDiSe Parallel & Distributed Systems Laboratory

# CSR representation of sparse matrices

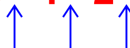
Mc=( 1 3 2 5 4 2 1 5 )

Mr=( 0 )

ParaDiSe
Parallel & Distributed Systems Laboratory

# CSR representation of sparse matrices

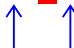
Mc=( 1 3 2 5 4 2 1 5 )



Mr=( 0 2 )

# CSR representation of sparse matrices

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



**Mr=( 0 2 4 )**

# CSR representation of sparse matrices

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

**Mr=( 0 2 4 5 6 )**

ParaDiSe
Parallel & Distributed Systems Laboratory

# CSR representation of sparse matrices

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

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

ParaDiSe
Parallel & Distributed Systems Laboratory

# Memory limitations

## GPU Memory consumption

- Data stored on GPU
  - 12B per vertex (4B due to CSR + 8B due to *map,oldmap*,$\mathcal{A}$,*r*)
  - 4B per edge (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 of GPU memory
  - 30 millions of vertices, 150 millions of edges (avg. outdegree 5)
  - 50 millions of vertices, 100 millions of edges (avg. outdegree 2)

ParaDiSe
Parallel & Distributed Systems Laboratory

# Memory limitations

## GPU Memory consumption

- Data stored on GPU
  - 12B per vertex (4B due to CSR + 8B due to *map,oldmap*,$\mathcal{A}$,*r*)
  - 4B per edge (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 of GPU memory
  - 30 millions of vertices, 150 millions of edges (avg. outdegree 5)
  - 50 millions of vertices, 100 millions of edges (avg. outdegree 2)

ParaDiSe Parallel & Distributed Systems Laboratory

# Getting the Matrix of Predecessors

**Matrix of predecessors**

- Must be computed from implicit definition of the graph
- Requires state space generation
- Stored directly in CSR representation
- CSR representation is constructed on-the-fly
- Explicit matrix representation is never stored

ParaDiSe
Parallel & Distributed
Systems Laboratory

# Matrix of predecessors

**Problem**

- Successive discovery of predecessors in state space generation
- “Random” writes to the matrix
- Difficult with CSR representation

**Graph transposition**

- Preserves accepting cycles
- Predecessors are successors in the original graph
- Successors of the original graph are easily generated
- Successive updates to CSR representation

ParaDiSe
Parallel & Distributed Systems Laboratory

# Matrix of predecessors

**Problem**

- Successive discovery of predecessors in state space generation
- “Random” writes to the matrix
- Difficult with CSR representation

**Graph transposition**

- Preserves accepting cycles
- Predecessors are successors in the original graph
- Successors of the original graph are easily generated
- Successive updates to CSR representation

ParaDiSe
Parallel & Distributed Systems Laboratory

# Other important features of DiVinE-CUDA

**Reduction of the graph stored on GPU**

- Exploits automata-based approach to LTL MC
- By analyzes of property automaton some parts of the graph may be known to contain no accepting cycles
- Non-accepting parts are omitted in CSR representation

**On-the-fly verification**

- CPU and GPU can execute concurrently
- GPU performs MAP algorithm on partially constructed graph
- Accepting cycle is detected prior full state space exploration

ParaDiSe
Parallel & Distributed Systems Laboratory

# Other important features of DiVinE-CUDA

**Reduction of the graph stored on GPU**

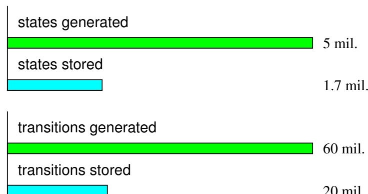
- Exploits automata-based approach to LTL MC
- By analyzes of property automaton some parts of the graph may be known to contain no accepting cycles
- Non-accepting parts are omitted in CSR representation

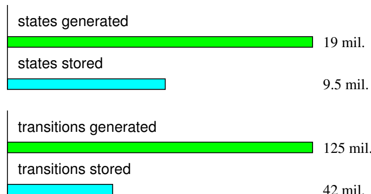
**On-the-fly verification**

- CPU and GPU can execute concurrently
- GPU performs MAP algorithm on partially constructed graph
- Accepting cycle is detected prior full state space exploration

ParaDiSe
Parallel & Distributed Systems Laboratory

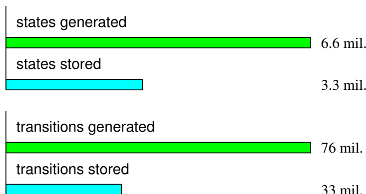
# Impact of graph reduction – Valid properties.

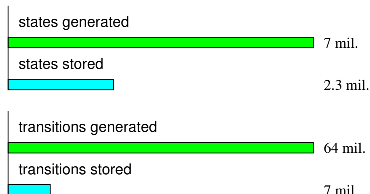


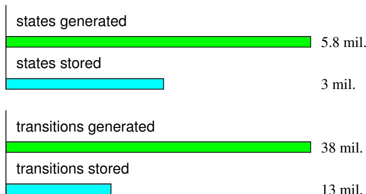


ParaDiSe Parallel & Distributed Systems Laboratory

# Impact of graph reduction – Invalid properties.



ParaDiSe
Parallel & Distributed Systems Laboratory

# Comparison to parallel algorithms running on single core

| Model | acc. cycle | CUDA MAP CSR time | CUDA MAP CUDA time | CUDA MAP total time | CPU MAP 1st iter. time | CPU MAP other iter. time | CPU MAP total time | CPU MAP # iter. | CPU OWCTY reach. time | CPU OWCTY total time |
|---|---|---|---|---|---|---|---|---|---|---|
| elevator 1 | N | 27 | 7 | 34 | 44 | 56 | 100 | 16 | 24 | 41 |
| leader | N | 88 | 1 | 90 | 97 | 600 | 697 | 17 | 90 | 297 |
| peterson 1 | N | 107 | 6 | 113 | 175 | 270 | 445 | 16 | 110 | 188 |
| anderson | N | 32 | 7 | 39 | 64 | 51 | 115 | 5 | 33 | 113 |
| elevator 2 | Y | 34 | 1 | 35 | 50 | – | 50 | 1 | 41 | 177 |
| phils | Y | 47 | 1 | 47 | 295 | 102 | 397 | 5 | 180 | 576 |
| peterson 2 | Y | 26 | 5 | 31 | 173 | – | 173 | 1 | 114 | 404 |
| bakery | Y | 25 | 1 | 26 | 240 | – | 240 | 1 | 219 | 907 |

| | CUDA MAP | CPU MAP | CPU OWCTY |
|---|---|---|---|
| Total time: | 386 + 29 = 415 | 2 173 | 2 730 |
| Speedup: | | 5.24 | 6.51 |

ParaDiSe Parallel & Distributed Systems Laboratory

# Many-cores vs. Multi-cores (predicted, optimal speedup)

| | CUDA MAP | | | CPU MAP | | CPU OWCTY | |
|---|---|---|---|---|---|---|---|
| | *CSR time* | *CUDA time* | *total time* | | | | |
| **1 core** | | | | | | | |
| Total time: | 386 | + 29 = | 415 | Total time: | 2 173 | Total time: | 2 730 |
| | | | | Speedup: | 5.24 | Speedup: | 6.51 |
| **2 cores** | | | | | | | |
| Total time: | 193 | + 29 = | 222 | Total time: | 1087 | Total time: | 1365 |
| | | | | Speedup: | 4.87 | Speedup: | 6.15 |
| **4 cores** | | | | | | | |
| Total time: | 97 | + 29 = | 126 | Total time: | 544 | Total time: | 683 |
| | | | | Speedup: | 4.32 | Speedup: | 5.42 |
| **8 cores** | | | | | | | |
| Total time: | 49 | + 29 = | 78 | Total time: | 272 | Total time: | 342 |
| | | | | Speedup: | 3.48 | Speedup: | 4.38 |

ParaDiSe
Parallel & Distributed Systems Laboratory

# Conclusion

**Acceleration of model checking process by full utilization of modern massively parallel architectures**

- successful redesign of Maximal Accepting Predecessor algorithm allowing for significant GPU acceleration.

**DiVinE-CUDA**

- tool for CUDA Accelerated LTL Model Checking.

**Experimental evaluation**

- significant reduction of runtimes
- identification of the main bottleneck of the designed approach.

**Pros and Cons**

+ On liveness LTL properties outperforms multi-core solutions
– On safety properties does not help
– Can handle verification instances of limited size

ParaDiSe
Parallel & Distributed Systems Laboratory

# Conclusion

**Acceleration of model checking process by full utilization of modern massively parallel architectures**

- successful redesign of Maximal Accepting Predecessor algorithm allowing for significant GPU acceleration.

**DiVinE-CUDA**

- tool for CUDA Accelerated LTL Model Checking.

**Experimental evaluation**

- significant reduction of runtimes
- identification of the main bottleneck of the designed approach.

**Pros and Cons**

+ On liveness LTL properties outperforms multi-core solutions
− On safety properties does not help
− Can handle verification instances of limited size

ParaDiSe
Parallel & Distributed Systems Laboratory

# Conclusion

**Acceleration of model checking process by full utilization of modern massively parallel architectures**

- successful redesign of Maximal Accepting Predecessor algorithm allowing for significant GPU acceleration.

**DiVinE-CUDA**

- tool for CUDA Accelerated LTL Model Checking.

**Experimental evaluation**

- significant reduction of runtimes
- identification of the main bottleneck of the designed approach.

**Pros and Cons**

+ On liveness LTL properties outperforms multi-core solutions
– On safety properties does not help
– Can handle verification instances of limited size

ParaDiSe
Parallel & Distributed Systems Laboratory

# Conclusion

**Acceleration of model checking process by full utilization of modern massively parallel architectures**

- successful redesign of Maximal Accepting Predecessor algorithm allowing for significant GPU acceleration.

**DiVinE-CUDA**

- tool for CUDA Accelerated LTL Model Checking.

**Experimental evaluation**

- significant reduction of runtimes
- identification of the main bottleneck of the designed approach.

**Pros and Cons**

+ On liveness LTL properties outperforms multi-core solutions
– On safety properties does not help
– Can handle verification instances of limited size

ParaDiSe Parallel & Distributed Systems Laboratory

# Future Work

**Multi-core preparation of CSR representation**

- Ready, will be part of the next release

**Ideas to overcome GPU memory limitations**

- Partition the matrix of predecessors
- Employ multiple GPU devices

**Open problems**

- How to CUDA-accelerate state space generation?
- How to CUDA-accelerate CSR representation preparation?

ParaDiSe
Parallel & Distributed Systems Laboratory

# Future Work

**Multi-core preparation of CSR representation**

- Ready, will be part of the next release

**Ideas to overcome GPU memory limitations**

- Partition the matrix of predecessors
- Employ multiple GPU devices

**Open problems**

- How to CUDA-accelerate state space generation?
- How to CUDA-accelerate CSR representation preparation?

ParaDiSe
Parallel & Distributed Systems Laboratory

# Future Work

**Multi-core preparation of CSR representation**

- Ready, will be part of the next release

**Ideas to overcome GPU memory limitations**

- Partition the matrix of predecessors
- Employ multiple GPU devices

**Open problems**

- How to CUDA-accelerate state space generation?
- How to CUDA-accelerate CSR representation preparation?

ParaDiSe
Parallel & Distributed Systems Laboratory

# The End

**Thank you four your attention**

ParaDiSe
Parallel & Distributed
Systems Laboratory