Parallel Nested Depth First Search

Alfons Laarman

Joint with Jaco van de Pol, Rom Langerak, Michael Weber
Anton Wijs (University of Eindhoven)

Oct 14, 2011
LTL Model Checking

- A buggy run in a system can be viewed as an infinite word
- Absence of bugs: emptiness of some Büchi automaton
- Graph problem: find a reachable accepting state on a cycle
- Basic algorithm: Nested Depth First Search (NDFS)
LTL Model Checking

- A buggy run in a system can be viewed as an infinite word
- Absence of bugs: emptiness of some Büchi automaton
- Graph problem: find a reachable accepting state on a cycle
- Basic algorithm: Nested Depth First Search (NDFS)
LTL Model Checking

- A buggy run in a system can be viewed as an infinite word
- Absence of bugs: emptiness of some Büchi automaton
- Graph problem: find a reachable accepting state on a cycle
- Basic algorithm: Nested Depth First Search (NDFS)
LTL Model Checking

- A buggy run in a system can be viewed as an infinite word
- Absence of bugs: emptiness of some Büchi automaton
- Graph problem: find a reachable accepting state on a cycle
- Basic algorithm: Nested Depth First Search (NDFS)

This talk

- We propose parallel NDFS, scalable
- So far, thought to be impossible
- Focus: algorithm (experiments)
procedure DFSblue(s)
    s.blue := true
    for all t ∈ post(s) do
        if ¬t.blue then DFSblue(t)
    if s ∈ Accepting then
        seed := s
        DFSred(s)

Nested DFS

- Blue search
  - Visits all reachable states
  - Starts Red search on accepting states (seed) in post order

- Red search
  - Finds cycle through seed
  - Visits states at most once
  - Linear time, on-the-fly

- Blue is inherently depth-first
Nested Depth First Search

Procedure $\text{DFSblue}(s)$

- $s$.blue := true
- For all $t \in \text{post}(s)$ do
  - If $\neg t$.blue then $\text{DFSblue}(t)$
- If $s \in \text{Accepting}$ then
  - $\text{seed} := s$
  - $\text{DFSred}(s)$

Procedure $\text{DFSred}(s)$

- $s$.red := true
- For all $t \in \text{post}(s)$ do
  - If $t = \text{seed}$ then $\text{ExitCycle}$
  - If $\neg t$.red then $\text{DFSred}(t)$

**Nested DFS**

- **Blue search**
  - Visits all reachable states
  - Starts Red search on accepting states (seed) in post order
- **Red Search**
  - Finds cycle through seed
  - Visits states at most once
procedure DFSblue(s)
    s.blue := true
    for all t ∈ post(s) do
        if ¬t.blue then DFSblue(t)
    if s ∈ Accepting then
        seed := s
    DFSred(s)

procedure DFSred(s)
    s.red := true
    for all t ∈ post(s) do
        if t = seed then ExitCycle
        if ¬t.red then DFSred(t)

Nested DFS

- Blue search
  - Visits all reachable states
  - Starts Red search on accepting states (seed) in post order
- Red Search
  - Finds cycle through seed
  - Visits states at most once
- Linear time, on-the-fly
- Blue is inherently depth-first
code for worker i

**procedure** DFSblue(s,i)

\[ s.\text{blue}[i] := \text{true} \]

**for all** \( t \in \text{post}(s) \) **do**

\[ \text{if } \neg t.\text{blue}[i] \text{ then } \text{DFSblue}(t,i) \]

**if** \( s \in \text{Accepting} \) **then**

\[ \text{seed}[i] := s \]

\[ \text{DFSred}(s,i) \]

**procedure** DFSred(s,i)

\[ s.\text{red}[i] := \text{true} \]

**for all** \( t \in \text{post}(s) \) **do**

\[ \text{if } t = \text{seed}[i] \text{ then } \text{ExitCycle} \]

\[ \text{if } \neg t.\text{red}[i] \text{ then } \text{DFSred}(t,i) \]

---

**Multi-core Swarmed NDFS**

- **N** workers perform parallel search _independently_

  [G. Holzmann et al.]

---
Swarmed Multi-core Nested Depth First Search

code for worker $i$

**procedure** DFSblue($s,i$)

  $s$.blue[$i$] := true

  for all $t \in \text{post}(s)$ do
    if $\neg t$.blue[$i$] then DFSblue($t,i$)
  
  if $s \in \text{Accepting}$ then
    seed[$i$] := $s$
    DFSred($s,i$)

**procedure** DFSred($s,i$)

  $s$.red[$i$] := true

  for all $t \in \text{post}(s)$ do
    if $t = \text{seed}[i]$ then ExitCycle
    if $\neg t$.red[$i$] then DFSred($t,i$)

---

**Multi-core Swarmed NDFS**

- $N$ workers perform parallel search *independently*  
  [G. Holzmann et al.]

- Multi-core: store visited states in a shared hash table  
  [FMCAD 2010, SPIN 2011]
code for worker $i$

**procedure** DFSblue($s,i$)

\[ s\.blue[i] := \text{true} \]

**for all** $t \in \text{post}(s)$ **do**

\[ \text{if } \neg t\.blue[i] \text{ then } \text{DFSblue}(t,i) \]

\[ \text{if } s \in \text{Accepting} \] then

\[ \text{seed}[i] := s \]

\[ \text{DFSred}(s,i) \]

**procedure** DFSred($s,i$)

\[ s\.red[i] := \text{true} \]

**for all** $t \in \text{post}(s)$ **do**

\[ \text{if } t = \text{seed}[i] \text{ then } \text{ExitCycle} \]

\[ \text{if } \neg t\.red[i] \text{ then } \text{DFSred}(t,i) \]

---

**Multi-core Swarmed NDFS**

- $N$ workers perform parallel search *independently*
  [G. Holzmann et al.]

- Multi-core: store visited states in a shared hash table
  [FMCAD 2010, SPIN 2011]

- Scales well in the presence of accepting cycles (bugs)
- Otherwise, all workers traverse the whole graph
Speedup of Swarmed NDFS (1 versus 16 cores)

Alternatives

- Swarm verification with NDFS
  - Effective, only for bug finding

[BEEM database]
Approaches to Parallel LTL Model Checking

Speedup of Swarmed NDFS (1 versus 16 cores)

Alternatives

- Swarm verification with NDFS
  - Effective, only for bug finding
- Dual-core NDFS [Holzmann]
  - Red search on 2nd CPU
  - Speedup of at most factor 2

[BEEM database]
Approaches to Parallel LTL Model Checking

Speedup of Swarmed NDFS (1 versus 16 cores)

Alternatives

- Swarm verification with NDFS
  - Effective, only for bug finding
- Dual-core NDFS [Holzmann]
  - Red search on 2nd CPU
  - Speedup of at most factor 2
- Red Search as parallel reachability
  - Speedup still $\leq 2$: $|G| + |G|/N$

[BEEM database]
Approaches to Parallel LTL Model Checking

Speedup of Swarmed NDFS (1 versus 16 cores)

![Graph showing speedup comparison between 1 core and 16 cores.]

**Alternatives**

- **Swarm verification with NDFS**
  - Effective, only for bug finding
- **Dual-core NDFS [Holzmann]**
  - Red search on 2nd CPU
  - Speedup of at most factor 2
- **Red Search as parallel reachability**
  - Speedup still \( \leq 2: |G| + |G|/N \)
- **Can one do better?**
  - Post-order is P-Complete, so
  - DFS not efficiently parallelizable

[BEEM database]
Approaches to Parallel LTL Model Checking

Speedup of Swarmed NDFS (1 versus 16 cores)

![Graph showing speedup of Swarmed NDFS](image)

Alternatives

- **Swarm verification with NDFS**
  - Effective, only for bug finding
- **Dual-core NDFS [Holzmann]**
  - Red search on 2nd CPU
  - Speedup of at most factor 2
- **Red Search as parallel reachability**
  - Speedup still $\leq 2$: $|G| + |G|/N$
- **Can one do better?**
  - Post-order is P-Complete, so
  - DFS not efficiently parallelizable
- **Breadth-first based:**
  - OWCTY, MAP [Brno]
  - Not linear ($|G| \cdot h$), not on-the-fly

[BEEM database]
New NDFS with Cyan and Pink  

\[ \text{à la Schwoon/Esparza} \]

\[ s.bc: \text{ white } \rightarrow \text{ cyan } \rightarrow \text{ blue} \]
\[ s.rc: \text{ white } \rightarrow \text{ pink } \rightarrow \text{ red} \]

**procedure DFSblue(s)**

\[ s.bc := \text{ cyan} \]
\[ \text{for all } t \in \text{ post}(s) \text{ do} \]
\[ \quad \text{if } t.bc = \text{ white } \text{ then } \text{ DFSblue}(t) \]
\[ \text{if } s \in \text{ Acc } \text{ then } \text{ DFSred}(s) \]
\[ s.bc := \text{ blue} \]

**procedure DFSred(s)**

\[ s.rc := \text{ pink} \]
\[ \text{for all } t \in \text{ post}(s) \text{ do} \]
\[ \quad \text{if } t.bc = \text{ cyan } \text{ then } \text{ ExitCycle} \]
\[ \quad \text{if } t.rc = \text{ white } \text{ then } \text{ DFSred}(t) \]
\[ s.rc := \text{ red} \]
Parallel NDFS: share the red color (first try)

s.color[i] : white $\rightarrow$ cyan $\rightarrow$ blue
s.pink[i], s.red : Boolean

procedure DFSblue(s,i) pruned by shared red color
s.color[i] := cyan
for all $t \in \text{post}(s)$ do
    if $t.color[i]=$white and $\neg t.red$ then DFSblue(t,i)
if s $\in$ Acc then DFSred(s,i)
s.color[i] := blue

procedure DFSred(s,i) pruned by shared red color
s.pink[i] := true
for all $t \in \text{post}(s)$ do
    if $t.color[i]=$cyan then ExitCycle
    if $\neg t.pink[i]$ and $\neg t.red$ then DFSred(t,i)
s.red := true
s.color[i] : white $\rightarrow$ cyan $\rightarrow$ blue
s.pink[i], s.red : Boolean

**procedure** DFSblue(s,i) pruned by shared red color

s.color[i] := cyan
for all t ∈ post(s) do
    if t.color[i]=white and $\neg$ t.red then DFSblue(t,i)
if s ∈ Acc then DFSred(s,i)
s.color[i] := blue

**procedure** DFSred(s,i) pruned by shared red color

s.pink[i] := true
for all t ∈ post(s) do
    if t.color[i]=cyan then ExitCycle
    if $\neg$t.pink[i] and $\neg$t.red then DFSred(t,i)
s.red := true (unfortunately incorrect)
Example: what is the meaning of red? (2 workers)
Example: what is the meaning of red? (2 workers)
Example: what is the meaning of red? (2 workers)
Example: what is the meaning of red? (2 workers)
Example: what is the meaning of red? (2 workers)
Example: what is the meaning of red? (2 workers)
Example: what is the meaning of red? (2 workers)
Example: what is the meaning of red? (2 workers)
Example: what is the meaning of red? (2 workers)

Accepting states on cycles get red:

No problem: path pink $\rightarrow$ cyan
Example: what is the meaning of red? (2 workers)
Example: what is the meaning of red? (2 workers)
Example: what is the meaning of red? (2 workers)
Example: what is the meaning of red? (2 workers)
Example: what is the meaning of red? (2 workers)
Example: what is the meaning of red? (2 workers)

Accepting states on cycles get red:
Example: what is the meaning of red? (2 workers)

All accepting cycles contain red:

Accepting states on cycles get red:
Example: what is the meaning of red? (2 workers)

All accepting cycles contain red:

Accepting states on cycles get red:

No problem: path pink→cyan
Synchronisation is necessary: third worker strikes!

Workers 1,2 proceed as before
Synchronisation is necessary: third worker strikes!

Workers 1,2 proceed as before

Worker 3 starts Red search in 1, 0
Synchronisation is necessary: third worker strikes!

Workers 1,2 proceed as before

Worker 3 starts Red search in 1, 0
No cycle will be detected!
procedure DFSblue(s,i)
    s.color[i] := cyan
    for all t ∈ post(s) do
        if t.color[i]=white and ¬t.red then DFSblue(t,i)
    if s ∈ Acc then DFSred(s,i)
    s.color[i] := blue

procedure DFSred(s,i)
    s.pink[i] := true
    for all t ∈ post(s) do
        if t.color[i]=cyan then ExitCycle
        if ¬t.pink[i] and ¬t.red then DFSred(t,i)
    pink[i] := false
    if s ∈ Acc then await ∀j : ¬s.pink[j]
    s.red := true
Optimization 1: Early detection and $2N+1+\log(N)$ bits

procedure DFSblue(s,i)
    s.color[i] := cyan
    for all t ∈ post(s) do
        if t.color[i]=cyan and s or t ∈ Acc then ExitCycle
        if t.color[i]=white and ¬t.red then DFSblue(t,i)
    if s ∈ Acc then s.count++; DFSred(s,i)
s.color[i] := blue

procedure DFSred(s,i)
    s.color[i] := pink
    for all t ∈ post(s) do
        if t.color[i]=cyan then ExitCycle
        if t.color[i]≠pink and ¬t.red then DFSred(t,i)
    if s ∈ Acc then s.count--; await s.count=0
    s.red := true
Optimization 1: Early detection and $2N + 1 + \log(N)$ bits

\begin{verbatim}
procedure DFSblue(s, i)
    s.color[i] := cyan
    for all t ∈ post(s) do
        if t.color[i] = cyan and s or t ∈ Acc then ExitCycle
        if t.color[i] = white and ¬t.red then DFSblue(t, i)
    if s ∈ Acc then s.count++;
    DFSred(s, i)
    s.color[i] := blue

procedure DFSred(s, i)
    s.color[i] := pink
    for all t ∈ post(s) do
        if t.color[i] = cyan then ExitCycle
        if t.color[i] ≠ pink and ¬t.red then DFSred(t, i)
    if s ∈ Acc then s.count--; await s.count=0
    s.red := true
\end{verbatim}
**Optimization 1: Early detection and \(2N+1+\log(N)\) bits**

**procedure** DFSblue(s,i)

\[
\text{s.color}[i] := \text{cyan} \\
\text{for all } t \in \text{post}(s) \text{ do} \\
\quad \text{if } t.\text{color}[i]=\text{cyan} \text{ and } s \text{ or } t \in \text{Acc} \text{ then ExitCycle} \\
\quad \text{if } t.\text{color}[i]=\text{white} \text{ and } \neg t.\text{red} \text{ then DFSblue}(t,i) \\
\text{if } s \in \text{Acc} \text{ then } s.\text{count}++; \text{DFSred}(s,i) \\
\text{s.color}[i] := \text{blue}
\]

**procedure** DFSred(s,i)

\[
\text{s.color}[i] := \text{pink} \\
\text{for all } t \in \text{post}(s) \text{ do} \\
\quad \text{if } t.\text{color}[i]=\text{cyan} \text{ then ExitCycle} \\
\quad \text{if } t.\text{color}[i]\neq \text{pink} \text{ and } \neg t.\text{red} \text{ then DFSred}(t,i) \\
\text{if } s \in \text{Acc} \text{ then } s.\text{count}--; \text{await } s.\text{count}=0 \\
\text{s.red} := \text{true}
\]
Optimization 2: Sprinkle red paint

procedure DFSblue(s,i)
    s.color[i] := cyan
    all_successors_red := true
    for all t ∈ post(s) do
        if t.color[i]=cyan and s or t ∈ Acc then ExitCycle
        if t.color[i]=white and ¬t.red then DFSblue(t,i)
        if ¬t.red then all_successors_red := false
    if all_successors_red then s.red := true
else if s ∈ Acc then s.count++; DFSred(s,i)
    s.color[i] := blue

procedure DFSred(s,i)
    s.color[i] := pink
    for all t ∈ post(s) do
        if t.color[i]=cyan then ExitCycle
        if t.color[i]≠pink and ¬t.red then DFSred(t,i)
    if s ∈ Acc then s.count--; await s.count=0
    s.red := true
Optimization 2: Sprinkle red paint

\[
\text{procedure } \text{DFSblue}(s,i) \\
\quad \text{s.color}[i] := \text{cyan} \\
\quad \text{all.successors.red} := \text{true} \\
\quad \text{for all } t \in \text{post}(s) \text{ do} \\
\quad \quad \text{if } t.\text{color}[i] = \text{cyan} \text{ and } s \text{ or } t \in \text{Acc} \text{ then ExitCycle} \\
\quad \quad \text{if } t.\text{color}[i] = \text{white} \text{ and } \neg t.\text{red} \text{ then DFSblue}(t,i) \\
\quad \quad \text{if } \neg t.\text{red} \text{ then all.successors.red} := \text{false} \\
\quad \text{if all.successors.red then } s.\text{red} := \text{true} \\
\quad \text{else if } s \in \text{Acc} \text{ then } s.\text{count}++; \text{ DFSred}(s,i) \\
\quad \text{s.color}[i] := \text{blue} \\
\]

\[
\text{procedure } \text{DFSred}(s,i) \\
\quad \text{s.color}[i] := \text{pink} \\
\quad \text{for all } t \in \text{post}(s) \text{ do} \\
\quad \quad \text{if } t.\text{color}[i] = \text{cyan} \text{ then ExitCycle} \\
\quad \quad \text{if } t.\text{color}[i] \neq \text{pink} \text{ and } \neg t.\text{red} \text{ then DFSred}(t,i) \\
\quad \quad \text{if } s \in \text{Acc} \text{ then } s.\text{count}--; \text{ await } s.\text{count}=0 \\
\quad \text{s.red} := \text{true} \\
\]
Optimization 2: Sprinkle red paint

**procedure** DFSblue(s,i)

\[
\begin{align*}
\text{s.color}[i] & := \text{cyan} \\
\text{all_successors_red} & := \text{true} \\
\text{for all } t & \in \text{post}(s) \text{ do} \\
\quad & \text{if } t.\text{color}[i] = \text{cyan} \text{ and } s \text{ or } t \in \text{Acc} \text{ then } \text{ExitCycle} \\
\quad & \text{if } t.\text{color}[i] = \text{white} \text{ and } \neg t.\text{red} \text{ then } \text{DFSblue}(t,i) \\
\quad & \text{if } \neg t.\text{red} \text{ then } \text{all_successors_red} := \text{false} \\
\text{if } \text{all_successors_red} & \text{ then } s.\text{red} := \text{true} \\
\text{else if } s & \in \text{Acc} \text{ then } s.\text{count}++; \text{ DFSred}(s,i) \\
\text{s.color}[i] & := \text{blue}
\end{align*}
\]

**procedure** DFSred(s,i)

\[
\begin{align*}
\text{s.color}[i] & := \text{pink} \\
\text{for all } t & \in \text{post}(s) \text{ do} \\
\quad & \text{if } t.\text{color}[i] = \text{cyan} \text{ then } \text{ExitCycle} \\
\quad & \text{if } t.\text{color}[i] \neq \text{pink} \text{ and } \neg t.\text{red} \text{ then } \text{DFSred}(t,i) \\
\text{if } s & \in \text{Acc} \text{ then } s.\text{count}--; \text{ await } s.\text{count}=0 \\
\text{s.red} & := \text{true}
\end{align*}
\]
Swarmed NDFS versus Parallel NDFS

Swarmed NDFS (1 versus 16-core)

Parallel NDFS (1 versus 16-core)
OWCTY and Swarmed NDFS versus Parallel NDFS

Swarmed versus Parallel NDFS
(both 16 cores)

OWCTY versus Parallel NDFS
(both 16 cores)
**Recent developments**

- Next talk: parallelizes blue search
- PDMC’11: Variations on Multi-Core Nested-Depth Search
  - Experimental results for both parallel NDFS algorithms
  - A combination of both approaches
  - Investigation of the effects of random search
Conclusions

- We have proposed a parallel NDFS algorithm
- It is linear in the input size and on-the-fly
- It scales well for a certain set of inputs
- Without accepting states, all workers still visit whole graph

Availability

The benchmarks were done with LTSmin, DiVinE and BEEM
The implementation is available (open source) at:
http://fmt.cs.utwente.nl/tools/ltsmin/
See also: CAV’10, FMCAD’10, NFM’11, SPIN’11, PDMC’11
Conclusions:

- We have proposed a parallel NDFS algorithm
- It is linear in the input size and on-the-fly
- It scales well for a certain set of inputs
- Without accepting states, all workers still visit whole graph

Availability:

- The benchmarks were done with LTSmin, DiVinE and BEEM
- The implementation is available (open source) at: [http://fmt.cs.utwente.nl/tools/ltsmin/](http://fmt.cs.utwente.nl/tools/ltsmin/)
- See also: CAV’10, FMCAD’10, NFM’11, SPIN’11, PDMC’11