Practical Probability
Applying pGCL to Lattice Scheduling

David Cock

25 July 2013
In this talk, I present a **verified lattice scheduler**, that eliminates leakage via a shared cache, while guaranteeing non-starvation. In addition, this work:

- Applies our existing pGCL package for Isaelle.
- Presents a multilevel probabilistic refinement proof.
- Integrates with the seL4 proof.
Outline

- Lattice Scheduling
  - The Probabilistic Scheduler
    - Refinement
  - Lottery Scheduling
    - Data Refinement
- seL4 Integration
- Non-Leakage
- Summary
Consider a system with two classification tags: A and B. Information tagged with A may only be seen by an agent cleared to see A, likewise for B.
• Consider a system with two classification tags: A and B. Information tagged with A may only be seen by an agent cleared to see A, likewise for B.

• Any output from an agent clear for A is tagged A, likewise for B.
Consider a system with two classification tags: A and B. Information tagged with A may only be seen by an agent cleared to see A, likewise for B.

Any output from an agent clear for A is tagged A, likewise for B.

There are four possible clearances: A, B, A and B, and nothing. These are **domains**.
Consider a system with two classification tags: A and B. Information tagged with A may only be seen by an agent cleared to see A, likewise for B.

Any output from an agent clear for A is tagged A, likewise for B.

There are four possible clearances: A, B, A and B, and nothing. These are **domains**.

The who-may-talk-to-whom order is a lattice:

```
{A, B}  ↗
\(\uparrow\)  ↗
\{A\}  \(\uparrow\)  \{B\}
\{\}  ↖  \{\}  ↖
```

Practical Probability
Applying pGCL to Lattice Scheduling
Copyright NICTA 2013
• For brevity, label the domains and then forget the sets.
• For brevity, label the domains and then forget the sets.
• Enforcing rules for explicit communication in such a system is a well-studied problem.
- For brevity, label the domains and then forget the sets.
- Enforcing rules for explicit communication in such a system is a well-studied problem.
- **Implicit** communication is harder.
• For brevity, label the domains and then forget the sets.
• Enforcing rules for explicit communication in such a system is a well-studied problem.
• **Implicit** communication is harder.
• We’re specifically concerned with covert channels due to sharing hardware.
Even if two domains are unable to communicate, they leave detectable traces in the machine state.
Even if two domains are unable to communicate, they leave detectable traces in the machine state. For example, 2 cannot read 3’s cache lines, but it can infer where they are, by timing its own memory accesses.
The Cache Channel

How do we mitigate this channel?
The Cache Channel

How do we mitigate this channel?

- We could flush the cache everytime.
The Cache Channel

How do we mitigate this channel?

- We could flush the cache everytime ... expensive!
The Cache Channel

How do we mitigate this channel?

- We could flush the cache everytime … expensive!
- We don’t need to flush when transitioning up.
The Cache Channel

How do we mitigate this channel?

- We could flush the cache everytime ... expensive!
- We don’t need to flush when transitioning up.
- Transition up as long as possible...
The Cache Channel

How do we mitigate this channel?

- We could flush the cache everytime ... expensive!
- We don’t need to flush when transitioning up.
- Transition up as long as possible... then flush and start again.

This is **Lattice Scheduling**
The schedule relation $S$, is a subset of the upward transitions.

This schedule is incomplete: There's no way to leave 3.

We must add downward transitions, but how?
- The schedule relation $S$, is a subset of the up transitions.
- The schedule relation $S$, is a subset of the up transitions.
- The schedule relation $S$, is a subset of the up transitions.
- This schedule is incomplete: There’s no way to leave 3.
• The schedule relation $S$, is a subset of the up transitions.
• This schedule is incomplete: There’s no way to leave 3.
• We must add downward transitions, but how?
- Designate a downgrader, \( \perp \).
- Designate a downgrader, $\bot$.
- The downgrader clears the cache.
• Designate a downgrader, \(\perp\).
• The downgrader clears the cache.

**Lemma (Downgrading)**

*If \(S\) allows a downward transition, it is to the downgrader, \(\perp\):*

\[
(c, n) \in S \quad \text{clearance } c \not\subseteq \text{clearance } n
\]

\[
n = \perp
\]
The Lattice Scheduler

We’ll verify a scheduler written in pGCL, an imperative, probabilistic language:

\[
\text{record stateA = current\_domain :: dom\_id} \\
\text{scheduleS =} \\
\text{c is current\_domain in} \\
\text{current\_domain} \in (\lambda_. \{ n. (c, n) \in S\})
\]

This program selects a new domain nondeterministically from among those with a valid transition from the current.
• We want to **refine** this to a realistic implementation.
- We want to **refine** this to a realistic implementation.
- The refinement may produce any trace permitted here.
- We want to **refine** this to a realistic implementation.
- The refinement may produce any trace permitted here.
- For example: \( \bot, 2, \bot, 2, \ldots \).
• We want to *refine* this to a realistic implementation.
• The refinement may produce any trace permitted here.
• For example: \( \bot, 2, \bot, 2, \ldots \).
• The specification permits starvation.
We want to **refine** this to a realistic implementation.
The refinement may produce any trace permitted here.
For example: $\bot, 2, \bot, 2, \ldots$.
The specification permits starvation.
Randomisation gives us a neat solution.
Random Transitions

- Assign a probability to each transition such that $T(c, n) > 0$ only if $(c, n) \in S$.
- Outgoing probabilities sum to 1 (or less).
- The previous trace now has probability 0!
A Probabilistic Scheduler

scheduleT =
    c is current_domain in
current_domain :∈ (λ_. {⊥, 1, 2, 3} at (λ_ n. T (c, n)))
A Probabilistic Scheduler

\[
\text{schedule}_T =
\begin{align*}
c \text{ is current\_domain} &\in 
\text{current\_domain} :\in \left( \lambda \_ . \{ \bot, 1, 2, 3 \} \text{ at } (\lambda \_ n. \ T (c, n)) \right)
\end{align*}
\]

Lemma (Non-starvation)

*Taking at least 8 steps from any initial domain, we reach any final domain with non-zero probability:*

\[
\forall s. \ 0 < \wp \text{ schedule}_T^{8+n} (\text{in\_dom } d_f) \ s
\]

Note that predicates (expectations) in pGCL are real-valued.
What about downgrading, does it still hold? We show this using refinement, but first some notes on pGCL:

- pGCL generalises Boolean logic with real values: True is 1, False is 0.
- Entailment ($\vdash$) generalises ($\models$), which is really just $\leq$:
  - False $\rightarrow$ True $0 \leq 1$
  - False $\vdash \lambda x.\text{False}$
  - True $\vdash \lambda x.\text{True}$
- Predicates are lifted to expectations: $\llbracket P \rrbracket = \lambda x.\text{if } P x \text{ then } 1 \text{ else } 0$
- We reason about weakest-preexpectations $Q \models \wp \text{prog } R$
What about downgrading, does it still hold? We show this using refinement, but first some notes on pGCL:

- pGCL generalises Boolean logic with real values: True is 1, False is 0.
Refinement in pGCL

What about downgrading, does it still hold? We show this using refinement, but first some notes on pGCL:

- pGCL generalises Boolean logic with real values: True is 1, False is 0.
- Entailment ($\vdash$) generalises ($\models$), which is really just $\leq$:
  
  $$\text{False} \rightarrow \text{True} \quad 0 \leq 1$$
  
  $$\lambda x. \text{False} \vdash \lambda x. \text{True} \quad \lambda x. 0 \models \lambda x. 1$$
Refinement in pGCL

What about downgrading, does it still hold? We show this using refinement, but first some notes on pGCL:

- pGCL generalises Boolean logic with real values: True is 1, False is 0.
- Entailment (\(\vdash\)) generalises (\(\models\)), which is really just \(\leq\):

\[
\text{False } \rightarrow \text{ True} \quad \quad 0 \leq 1 \\
\lambda x. \text{False } \vdash \lambda x. \text{True} \quad \lambda x. 0 \models \lambda x. 1
\]

- Predicates are lifted to **expectations**:

\[
\langle P \rangle = \lambda x. \text{if } P \ x \text{ then } 1 \text{ else } 0
\]
Refinement in pGCL

What about downgrading, does it still hold? We show this using refinement, but first some notes on pGCL:

- pGCL generalises Boolean logic with real values: True is 1, False is 0.
- Entailment ($\vdash$) generalises ($\models$), which is really just $\leq$:
  
  $$\text{False} \rightarrow \text{True} \quad 0 \leq 1$$
  $$\lambda x. \text{False} \vdash \lambda x. \text{True} \quad \lambda x. 0 \models \lambda x. 1$$

- Predicates are lifted to **expectations**:
  
  $$\langle P \rangle = \lambda x. \text{if } P x \text{ then } 1 \text{ else } 0$$

- We reason about **weakest-preexpectations**:
  
  $$Q \models \text{wp prog } R$$
Refinement in pGCL

pGCL refinement has the usual properties:
Refinement in pGCL

pGCL refinement has the usual properties:

**Definition**

Program $b$ *refines* program $a$, written $a \sqsubseteq b$, exactly when all expectation-entailments on $a$ also hold on $b$:

\[
\begin{align*}
  P \models \text{wp} a Q \\
  \implies \quad P \models \text{wp} b Q
\end{align*}
\]
Refinement in pGCL

pGCL refinement has the usual properties:

**Definition**
Program $b$ refines program $a$, written $a \sqsubseteq b$, exactly when all expectation-entailments on $a$ also hold on $b$:

\[
P \models \text{wp } a \ Q \\
\Rightarrow \\
P \models \text{wp } b \ Q
\]

**Lemma**

The transition scheduler refines the lattice scheduler:

\[
scheduleS \sqsubseteq scheduleT
\]
First Refinement

**Downgrading** \(\Rightarrow\) schedule\(_S\) \(\subseteq\) schedule\(_T\)

- Downgrading is preserved by refinement,
First Refinement

- Downgrading is preserved by refinement, and therefore holds for scheduleT.
First Refinement

- Downgrading is preserved by refinement, and therefore holds for scheduleT.
- Non-starvation holds **only** for scheduleT.
Outline

- Lattice Scheduling
- The Probabilistic Scheduler
  - Refinement
- Lottery Scheduling
  - Data Refinement
- seL4 Integration
- Non-Leakage
- Summary
Our scheduler is so far very abstract. The next step is to implement the randomisation. We use a lottery:

- We only need a uniform random choice from $\mathbb{Z}_{32}$.
- Each option is assigned some number, $x$, of tickets.
- The chance of winning is $\frac{x}{2^{32}}$.
- We need to assume that the lottery relation holds:

$$T(c, n) = 2^{-32} \parallel \{ t. \text{lottery (domains s c)} t = n \}$$

- Different state spaces: need more than simple refinement.
The Lottery Scheduler

record domain = lottery :: 32 word ⇒ dom_id
record stateC = current_domain :: dom_id
domains :: dom_id ⇒ domain

scheduleM t = do c ← gets current_domain
dl ← gets domains
let n = lottery (dl c) t in
modify (λs. s\{current_domain := n\})
  od

scheduleC = t from (λs. UNIV) at 2^{-32} in
Exec (scheduleM t)
Data Refinement

Definition (Probabilistic Data Refinement)

Program $b$, on state type $\sigma$, refines program $a$, state $\tau$, given precondition $G : \sigma \rightarrow \text{Bool}$ and under projection $\theta : \sigma \rightarrow \tau$, written $a \sqsubseteq_{G,\theta} b$, exactly when any expectation entailment on $a$ implies the same for $b$, on the projected state and with a guarded pre-expectation:

$$P \models \text{wp } a \ Q$$

$$\llbracket G \rrbracket \& \& (P \circ \theta) \models \text{wp } b \ (Q \circ \theta)$$

$$a \& \& b = \max (a + b - 1) \ 0$$
Correspondence

Definition (Probabilistic Correspondence)

Programs $a$ and $b$ are said to be in probabilistic correspondence, $\text{pcorres}\ G\ a\ b$, given condition $G$ and under projection $\theta$ if, for any post-expectation $Q$, the guarded pre-expectations coincide:

$$\langle G\rangle \land (\text{wp}\ a\ Q \circ \theta) = \langle G\rangle \land \text{wp}\ b\ (Q \circ \theta)$$
Correspondence

Definition (Probabilistic Correspondence)

Programs $a$ and $b$ are said to be in *probabilistic correspondence*, $\text{pcorres} \theta G a b$, given condition $G$ and under projection $\theta$ if, for any post-expectation $Q$, the guarded pre-expectations coincide:

$$\langle G \rangle \land \left( \text{wp } a \right) Q \circ \theta = \langle G \rangle \land \text{wp } b \left( Q \circ \theta \right)$$

Lemma

*The specifications schedule$_T$ and schedule$_C$ correspond given condition LR and under projection $\phi$:*

$$\text{pcorres } \phi \text{ LR schedule}_T \text{ schedule}_C$$
Second Refinement

\[
\text{DOWNGRADING} \quad \rightarrow \quad \text{scheduleS} \\
\text{NON-STARVATION} \quad \rightarrow \quad \text{scheduleT}
\]
Second Refinement

- The double arrow represents correspondence.

Downgrading $\rightarrow$ scheduleS

Non-starvation $\rightarrow$ scheduleT

$\phi$, LR $\rightarrow$ scheduleC

Non-leakage

Summary
Second Refinement

- The double arrow represents correspondence.
- Correspondence composes with refinement.

\[ \text{DOWNGRADING} \rightarrow \text{schedule}_S \]
\[ \text{NON-STARVATION} \rightarrow \text{schedule}_T \]
\[ \phi, LR \]
\[ \text{schedule}_C \]
Second Refinement

The double arrow represents correspondence.
Correspondence composes with refinement.
Downgrading and non-starvation are preserved.
Outline

- Lattice Scheduling
- The Probabilistic Scheduler
  - Refinement
- Lottery Scheduling
  - Data Refinement
- seL4 Integration
- Non-Leakage
- Summary
The Nondeterministic State Monad

- The seL4 specification is written using a **nondeterministic state monad**.
- We can embed this cleanly into pGCL.
The seL4 specification is written using a **nondeterministic state monad**.

- We can embed this cleanly into pGCL.
- In fact, we just used it: scheduleM and Exec.
The seL4 specification is written using a nondeterministic state monad.

We can embed this cleanly into pGCL.

In fact, we just used it: scheduleM and Exec.

L4.verified used a particular notion of nondeterministic correspondence.
The seL4 specification is written using a **nondeterministic state monad**.

- We can embed this cleanly into pGCL.
- In fact, we just used it: scheduleM and Exec.
- L4.verified used a particular notion of nondeterministic correspondence.
- We know how to lift these results, probabilistically:
Lemma (Lifting Correspondence)

Given correspondence between \( M \) and \( M' \) : with

\[
\text{corres}_{\text{underlying}} \{(s, s') \mid s = \phi s'\} \quad \text{True rrel } G (G \circ \phi) M M'
\]

and standard side-conditions:

\[
\text{no}_\text{fail} G M \quad \text{empty}_\text{fail} M \quad \text{empty}_\text{fail} M'
\]

and that \( M \) is deterministic on the image of the projection,

\[
\forall s. \exists (r, s'). M (\phi s) = \{(\text{False}, (r, s'))\}
\]

then we have probabilistic correspondence:

\[
\text{pcorres} \phi (G \circ \phi) (\text{Exec } M) (\text{Exec } M')
\]
Lemma

If the kernel preserves the lottery relation,

\[ \{ LR \} \ stepKernel \ \{ \lambda_\cdot \ LR \} \]

and the current domain,

\[ \{ \lambda s. \ CD \ s = d \} \ stepKernel \ \{ \lambda \_ \ s. \ CD \ s = d \} \]

and is total,

\[ no\_fail \top \ stepKernel \ \emptyset\_fail \ stepKernel \]

then with the concrete scheduler, it refines the transition scheduler:

\[ \text{scheduleT} \subseteq_{LR,\phi} \ stepKernel;;\text{scheduleC} \]
Composed Refinement

DOWNGRADING → \text{scheduleS}

\text{NON-STARVATION} → \text{scheduleT}

\text{callKernelD} = \text{stepKernel; ; scheduleC} \leftarrow \text{scheduleC}

\text{callKernelH}

\text{callKernelC}
Outline

- Lattice Scheduling
- The Probabilistic Scheduler
  - Refinement
- Lottery Scheduling
  - Data Refinement
- seL4 Integration
- Non-Leakage
- Summary
Ultimately, we want to know that our scheduler eliminates leakage via the cache. We append a machine model:

```
record (sh, pr) machine = private :: dom_id ⇒ pr
shared :: sh
```

- A private state per domain.
- A shared state between domains (the cache).
- Domains are underspecified, but may only update their own private state and the shared state.
• Propagating taint takes at least 2 steps.
• A single-step policy isn’t enough.
Lemma (Non-leakage)

If the clearance of domain \( h \) is not entirely contained within that of domain \( l \),

\[
\text{clearance } h \not\subseteq \text{clearance } l
\]

then any function of the state after execution, which depends only on elements within \( l \)'s clearance,

\[
Q \circ \text{mask } l
\]

is invariant under modifications to \( h \)'s private state (as represented by replace):

\[
\wp ((\text{runDom};;\text{scheduleT})^n (Q \circ \text{mask})) = (\wp ((\text{runDom};;\text{scheduleT})^n (Q \circ \text{mask}))) \circ (\text{replace } h p)
\]
We’ve have non-leakage for the probabilistic scheduler (scheduleT), and it is preserved by refinement.

We now have all 3 properties for the concrete implementation.
Outline

- Lattice Scheduling
- The Probabilistic Scheduler
  - Refinement
- Lottery Scheduling
  - Data Refinement
- seL4 Integration
- Non-Leakage
- Summary
What The Message?

- Probabilistic programs need not be harder to verify than traditional ones.
- Good tool support now exists—pGCL for Isabelle available from:
  Will also be submitted to AFP.
- Some problems in security are unavoidably probabilistic.
- Probabilistic results can compose well with large existing proofs.
• Probabilistic programs need not be harder to verify than traditional ones.
What The Message?

- Probabilistic programs need not be harder to verify than traditional ones.
- Good tool support now exists
What The Message?

- Probabilistic programs need not be harder to verify than traditional ones.
- Good tool support now exists — pGCL for Isabelle available from:
  http://www.cse.unsw.edu.au/~davec/pGCL/
  Will also to be submitted to AFP.
What The Message?

- Probabilistic programs need not be harder to verify than traditional ones.
- Good tool support now exists — pGCL for Isabelle available from: http://www.cse.unsw.edu.au/~davec/pGCL/
  Will also to be submitted to AFP.
- Some problems in security are **unavoidably** probabilistic.
What The Message?

- Probabilistic programs need not be harder to verify than traditional ones.
- Good tool support now exists — pGCL for Isabelle available from:
  http://www.cse.unsw.edu.au/~davec/pGCL/
  Will also to be submitted to AFP.
- Some problems in security are **unavoidably** probabilistic.
- Probabilistic results can compose well with large existing proofs.
Questions?