Systematic Development of Programs for Parallel and Cloud Computing: Towards a Framework

NII Lectures Series

Frédéric Loulergue

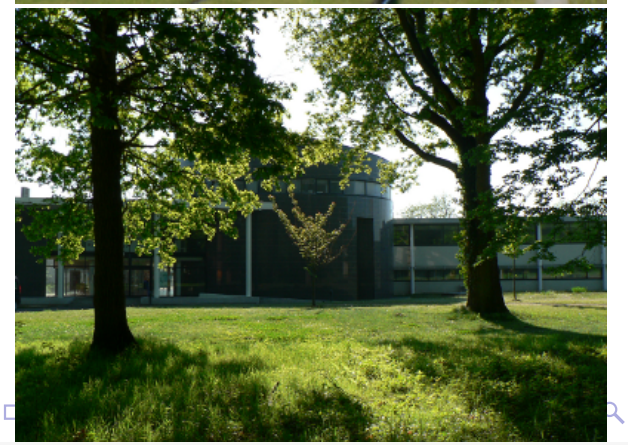
Université d'Orléans – LIFO – PaMDA Team

October-November 2013

Joint Work with (alphabetical order)

- Dr. Wadoud Bousdira (Université d'Orléans)
- Dr. Frédéric Dabrowski (Université d'Orléans)
- Sylvain Dailler (KUT & Université d'Orléans)
- Dr. Kento Emoto (Kyushu University of Technology)
- Pr. Zhenjiang Hu (National Institute of Informatics)
- Dr. Sylvain Jubertie (Université d'Orléans)
- Dr. Hab. Frédéric Gava (Université Paris-Est Créteil)
- Dr. Louis Gesbert (OCamlPro)
- Hideki Hashimoto (The University of Tokyo)
- Joeffrey Legaux (Université d'Orléans)
- Dr. Kiminori Matsuzaki (Kochi University of Technology)
- Dr. Virginia Niculescu (Babes-Bolyai University of Cluj-Napoca)
- Thomas Pinsard (Université d'Orléans)
- Simon Robillard (Université d'Orléans)
- Pr. Masato Takeichi (The University of Tokyo)
- Dr. Julien Tesson (Université Paris-Est Créteil)

Université d'Orléans, LIFO



Outline

Outline

Our Goal

To ease the development of **correct**
and **verified parallel** programs
with **predictable performances**

Our Goal

To ease the development of **correct**
and **verified parallel** programs
with **predictable performances**

using theories and tools to allow
a user to develop an application
by using building blocks and
implementing short programs satisfying
conditions easily or automatically proved

Outline

Parallel Programming

Parallel Programming

Automatic Parallelization

Parallel Programming

Automatic Parallelization

Concurrent & Distributed Programming

Parallel Programming

Automatic Parallelization

Structured Parallelism

- Algorithmic Skeletons
- Bridging Models
- Declarative Parallel Programming
- ...

Concurrent & Distributed Programming

Bridging Model

- Leslie Valiant in his 1990 CACM paper
  "A Bridging Model for Parallel Computation"
  `http://dx.doi.org/10.1145/79173.79181`

  > *The von Neumann model is the connecting bridge that enables programs from the diverse and chaotic world of software to run efficientby on machines from the diverse and chaotic world of hardware*

- Valiant's proposal: Bulk Synchronous Parallelism (BSP)
- Other models: LogP and variants, BSP variants, ...

Parallel Programming – Bridging Models II

Research on BSP

90' by Valiant & McColl

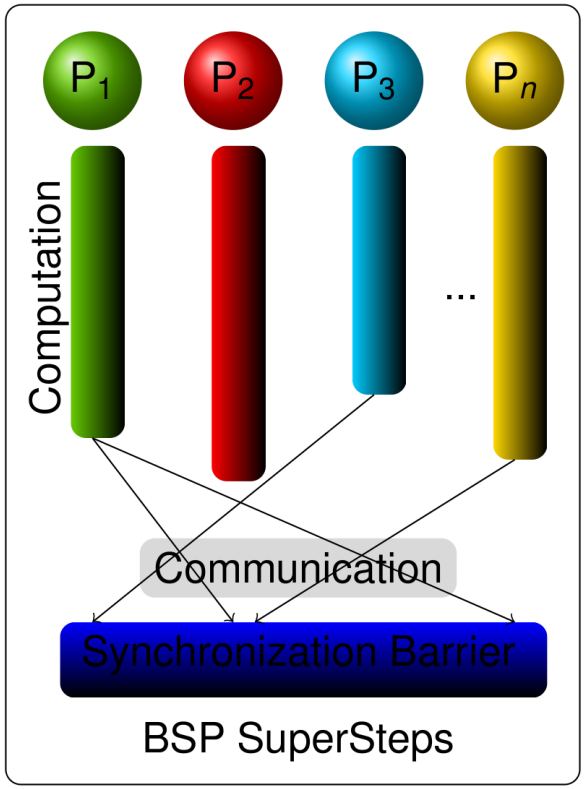
Three models

- abstract architecture
- execution model
- cost model

BSP computer

- p processor / memory pairs *(of speed r)*
- a communication network *(of speed g)*
- a global synchronisation unit *(of speed L)*

Execution model



Cost model

$T(s) = \max_{0 \leq i < p} w_i + h \times g + L$

where $h = \max_{0 \leq i < p} \{h_i^+, h_i^-\}$

Parallel Programming – Bridging Models III

Applications

- scientific computation [3]
- genetic algorithms [4]
- genetic programming [7]
- neural networks [18]
- parallel databases [1]
- parallel constraints solvers [10]
- ...

Algorithmic Skeletons

- Coined by Murray Cole in
  Algorithmic Skeletons: Structured Management of Parallel Computation, MIT Press, 1989
  `http://homepages.inf.ed.ac.uk/mic/Pubs/skeletonbook.ps.gz`
- Popular skeletons: Google's MapReduce

Skeletal Parallelism

- Skeleton = pattern of a parallel algorithm
  familiar sequential semantics
- Program = composition of skeletons

Algorithmic Skeletons

- Coined by Murray Cole in
  Algorithmic Skeletons: Structured Management of Parallel Computation, MIT Press, 1989
  `http://homepages.inf.ed.ac.uk/mic/Pubs/skeletonbook.ps.gz`
- Popular skeletons: Google's MapReduce

Skeletal Parallelism

- Skeleton = higher-order function implemented in parallel
  familiar sequential semantics
- Program = composition of skeletons

Parallel Programming – Algorithmic Skeletons

Libraries of Algorithmic Skeletons

- For C++: SkeTo[1], OSL[2], Muesli, QUAFF, ...
- For C: eSkel, SKElib
- For Java: Lithium, Muskel, Calcium, ...
- For functional languages:
  - OCaml: OCamlP3L, Parmap
  - Erlang: Skel
  - Haskell: HaskSkel, Edenskeletons

Algorithmic Skeletons Theory

- List homomorphisms for parallel programming (Cole 1993)
- Many further developments in particular in Tokyo

[1] `http://sketo.ipl-lab.org`
[2] `http://traclifo.univ-orleans.fr/OSL`

An Example: Variance

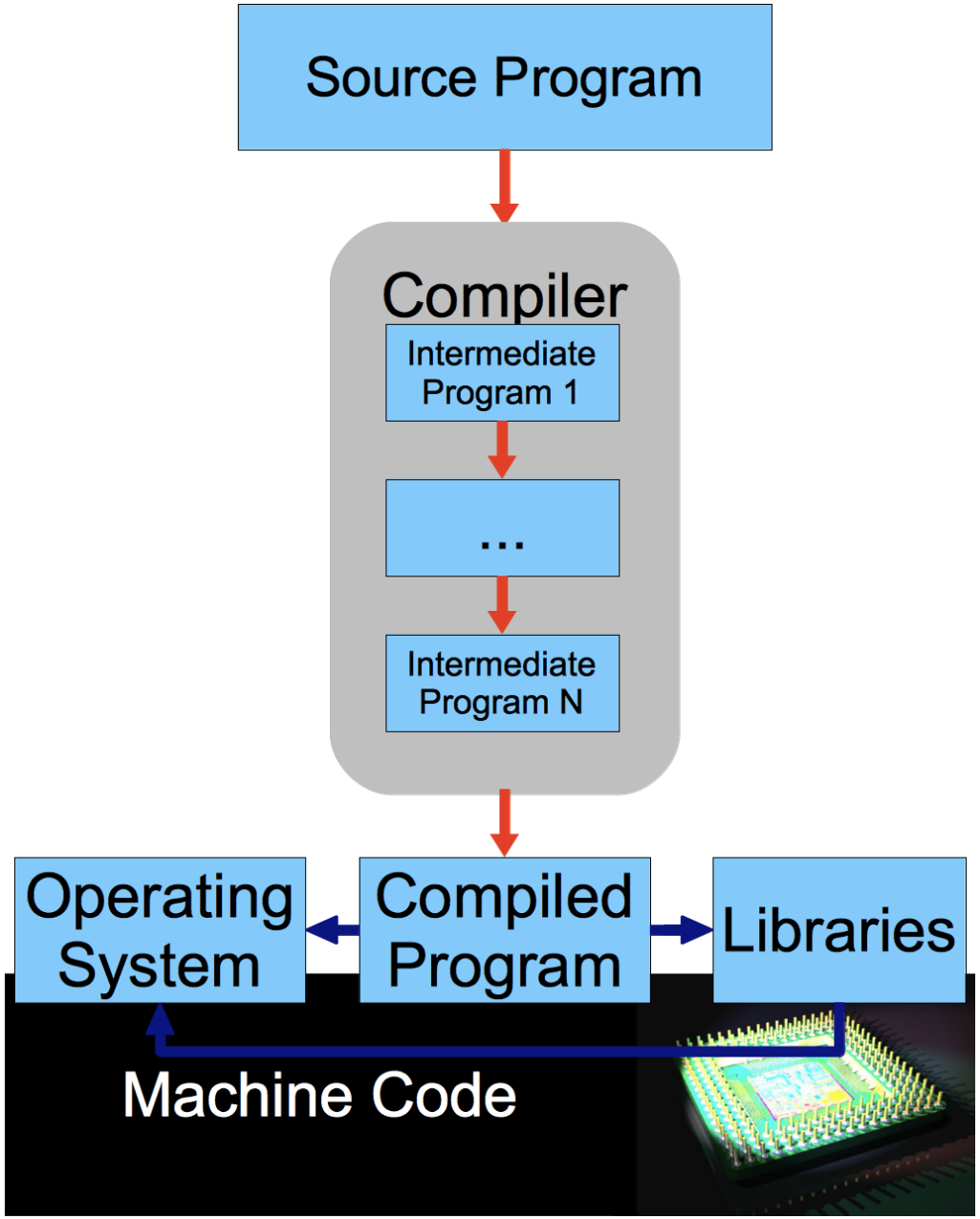
$$\frac{1}{n}\sum_{k=0}^{n-1}(x_k - \frac{1}{n}\sum_{k=0}^{n-1}x_k)^2$$

Variance as an OSL Program

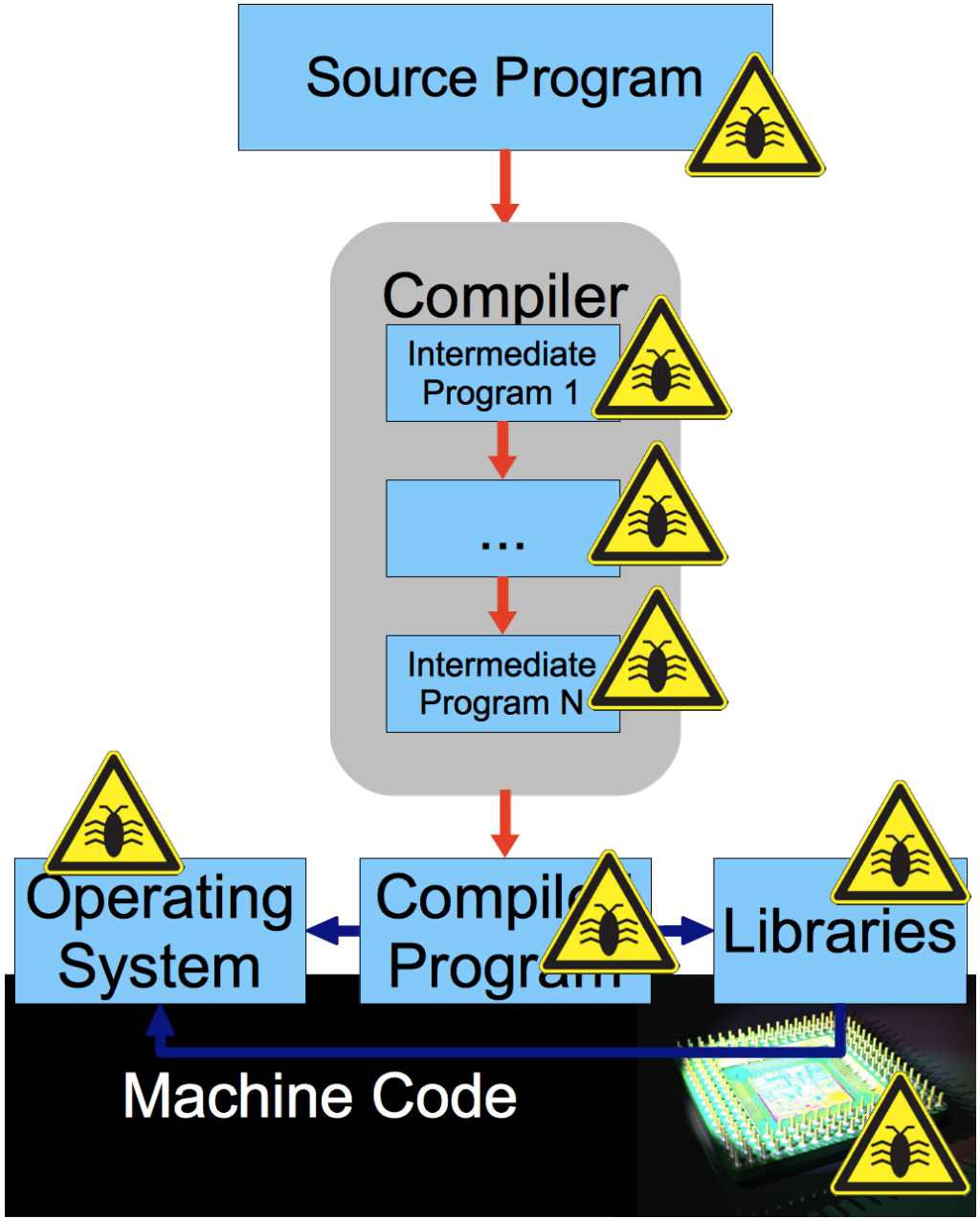
```
double avg = reduce(plus<double>(), x) / x.getSize();
double variance =
  reduce(plus<double>(),
         map(bind(minus<double>(),avg, _2), x));
```

Outline

Program Correctness?



Program Correctness?



Program Verification I

Verification Methods

Some properties:

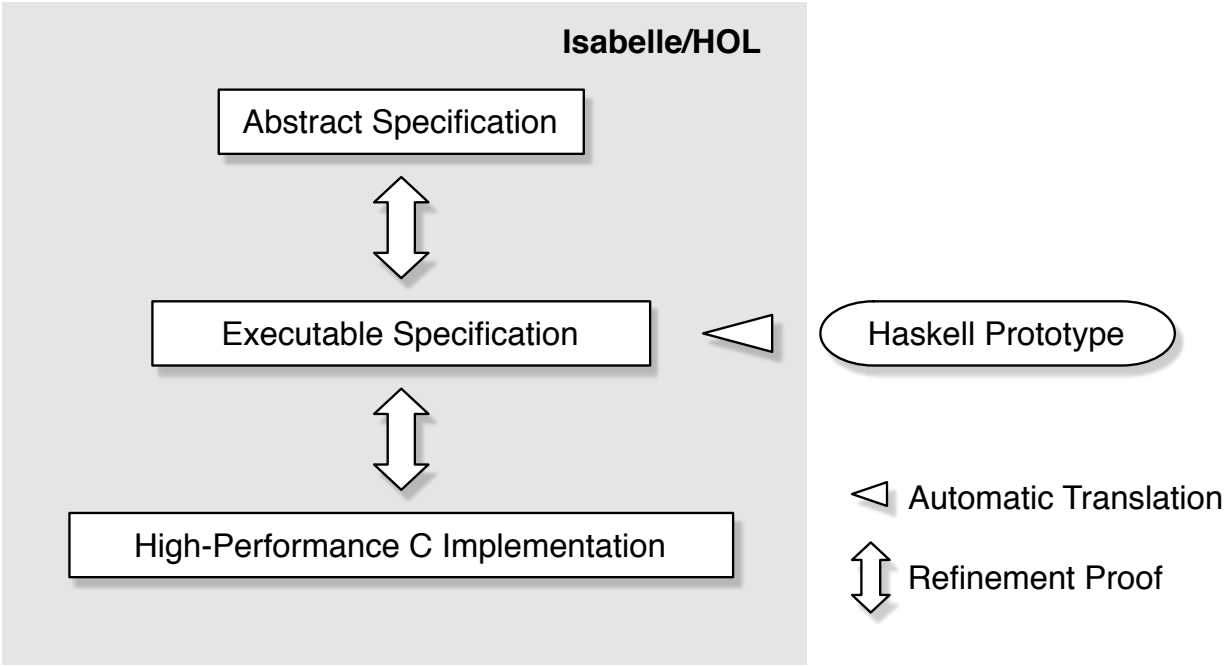
- Model checking
- Static analysis in particular abstract interpretation

Functional correctness $\{P\}\ c\ \{Q\}$:

- interactive provers:
  - Coq
  - Isabelle/HOL
- deductive program verification:
  verification condition generator + (SMT) solvers
  - Why3
  - Boogie

Program Verification II

The seL4 operating system kernel — Klein et al. [12]

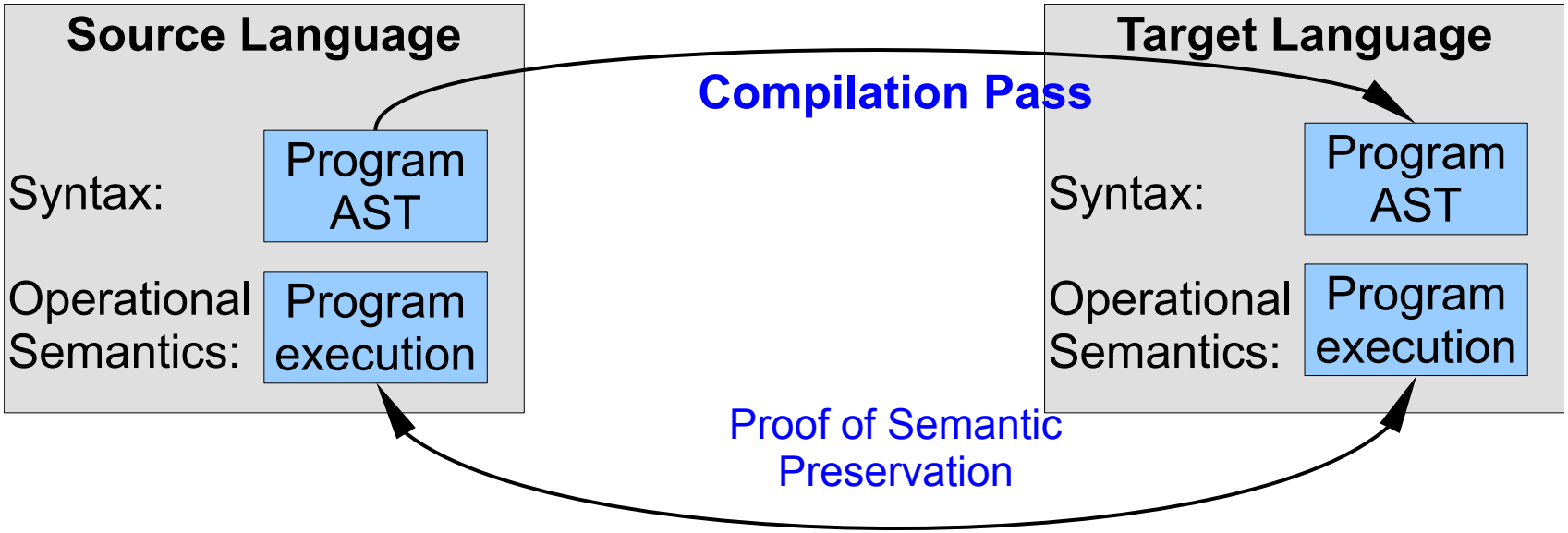


from [12]

- Proof assistant: **Isabelle/HOL**
- Memory model: Sequential Consistency

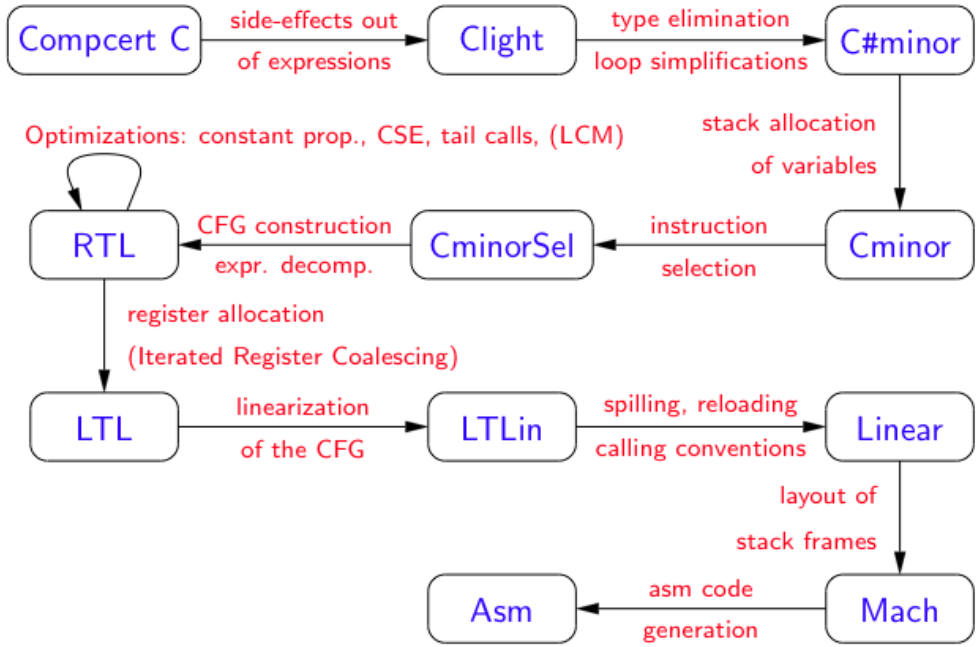
Program Verification III

Verified compilation



Program Verification IV

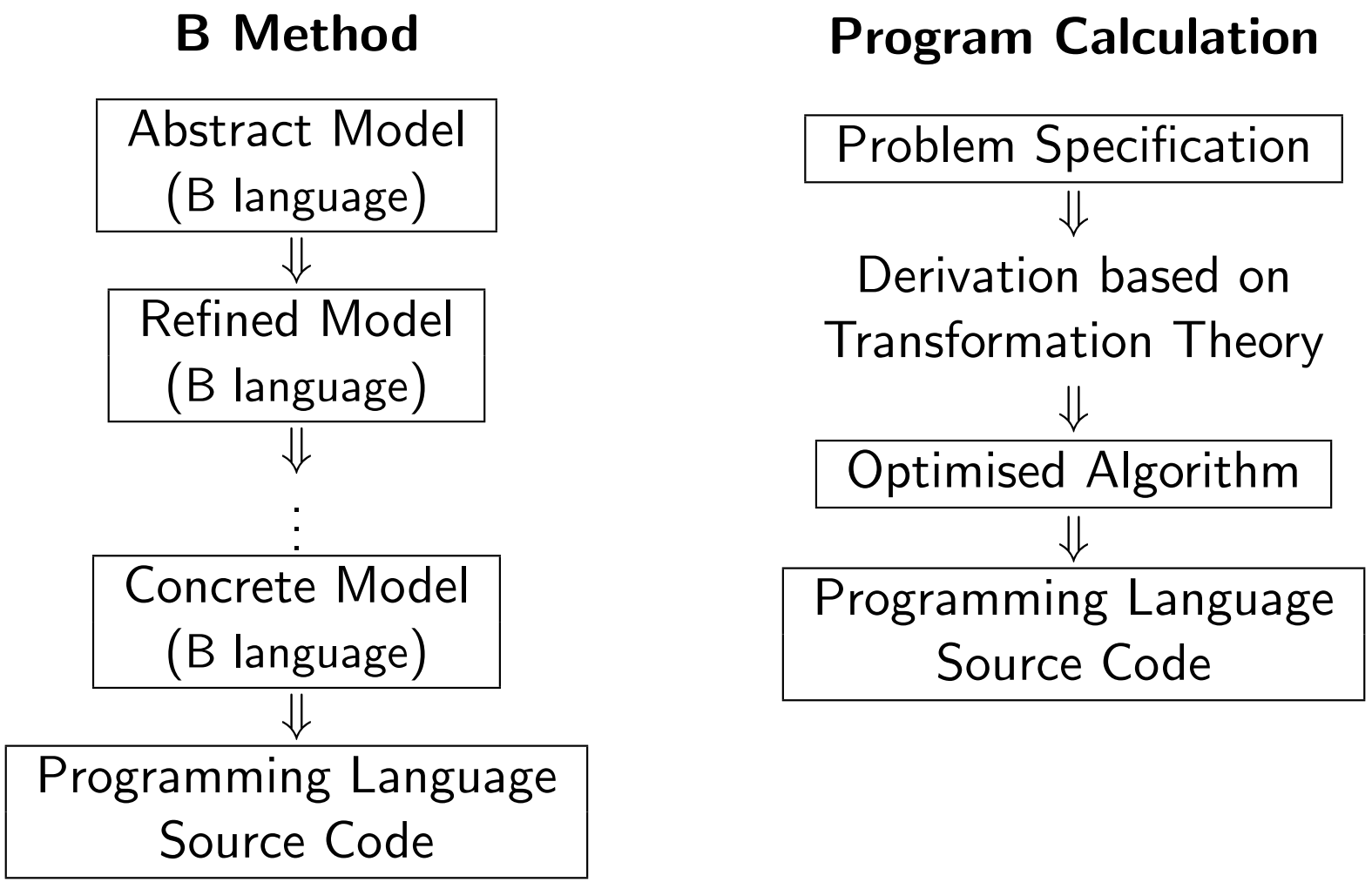
The Compcert [13] and CompcertTSO Compilers [19]



from `http://compcert.inria.fr`

- Proof assistant: **Coq**
- Transformation passes are proved to preserve semantics
- The compiler is **extracted** from the formal development
- CompcertTSO: a first step towards parallelism

Program Refinement & Program Calculation

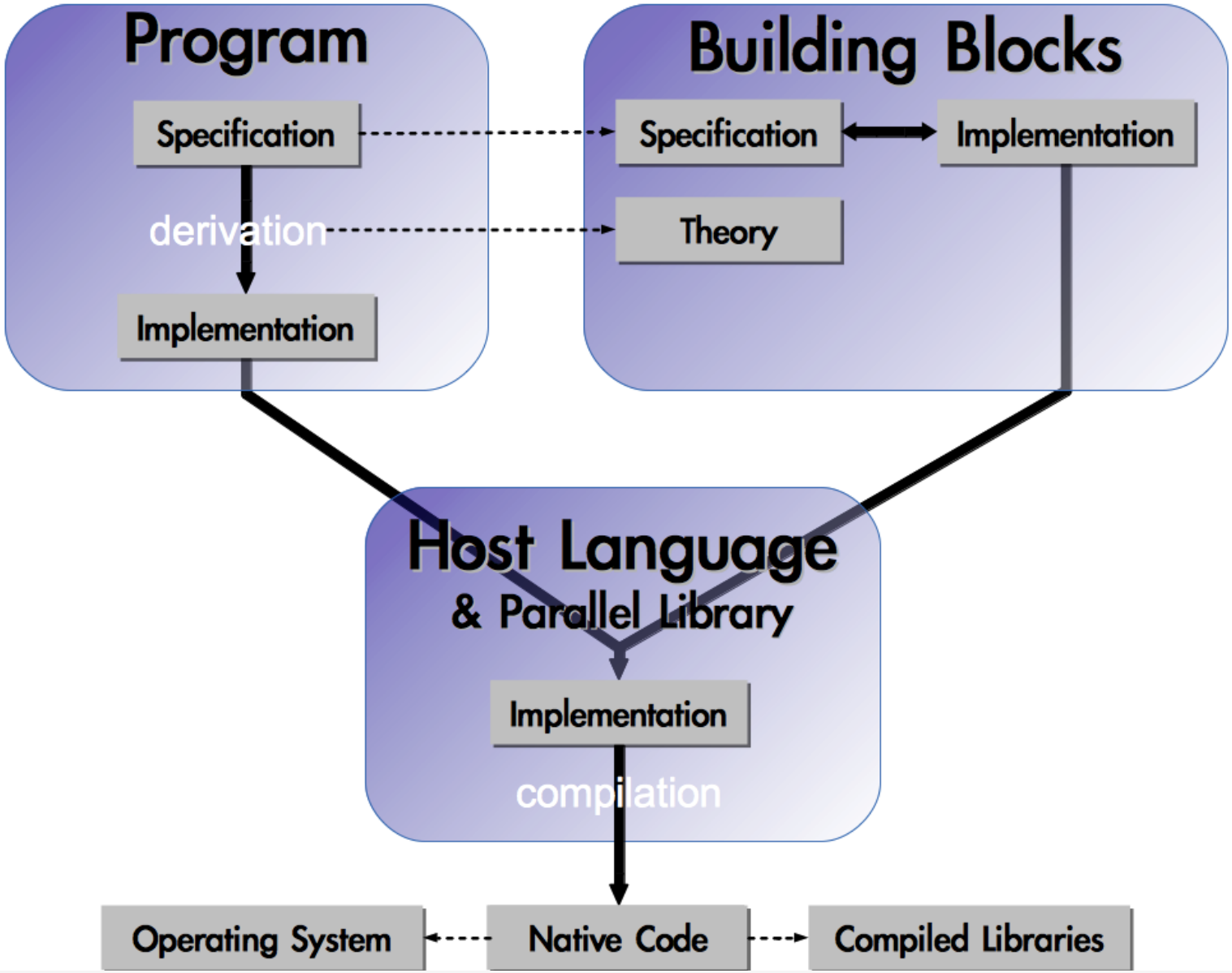


Verification of Parallel Programs?

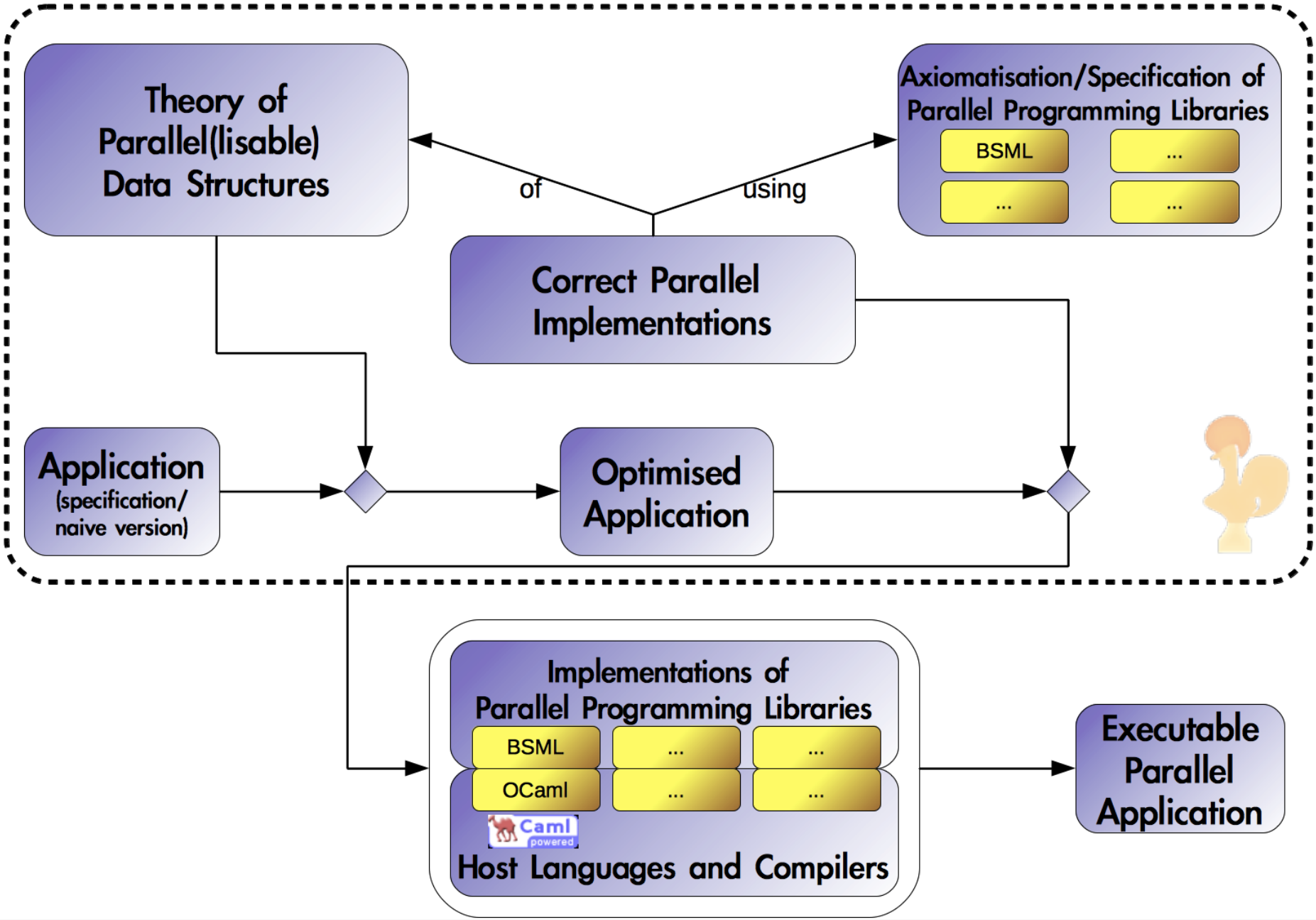
- Functional correctness:
  - concurrent extension of C
  - BSP-Why
- Some properties:
  - MPI in TLA+
  - ...

Outline

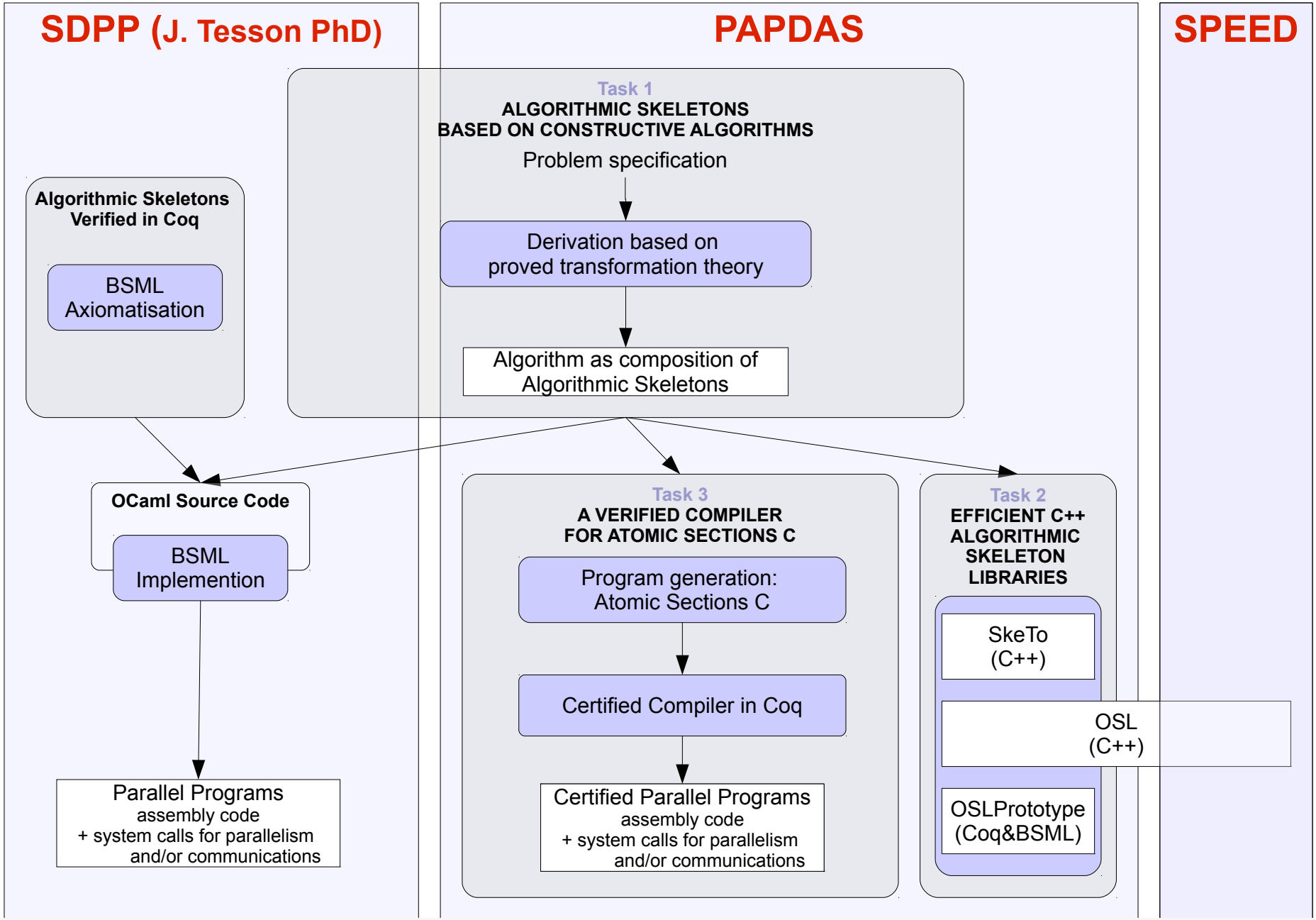
SyDPaCC – Systematic Development of Programs for Parallel and Cloud Computing



SyDPaCC: Program Development



Past and Ongoing Projects



Outline

Outline

The Coq Proof Assistant I

ACM SIGPLAN Software Award 2013

The Coq proof assistant provides a rich environment for interactive development of machine-checked formal reasoning. Coq is having a profound impact on research on programming languages and systems [...] It has been widely adopted as a research tool by the programming language research community [...] Last but not least, these successes have helped to spark a wave of widespread interest in dependent type theory, the richly expressive core logic on which Coq is based.

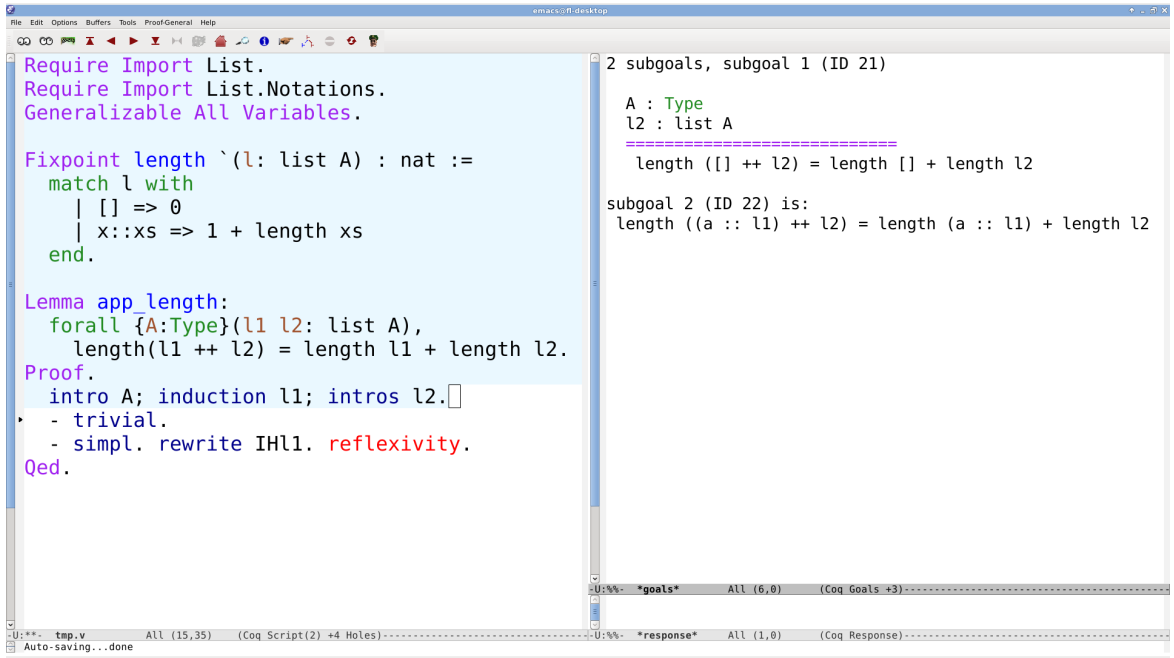
[...] The Coq team continues to develop the system, bringing significant improvements in expressiveness and usability with each new release.

In short, Coq is playing an essential role in our transition to a new era of formal assurance in mathematics, semantics, and program verification.

The Coq Proof Assistant II

Foundations

- Calculus of inductive constructions
- Curry-Howard correspondance



Natural Deduction

$$(v)\frac{A \in \Gamma}{\Gamma \vdash A}$$

$$(I)\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}$$

$$(a)\frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B}$$

Simply Typed λ-Calculus

$$(V)\frac{x : A \in \Gamma}{\Gamma \vdash x : A}$$

$$(L)\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash (\lambda x{:}A.e) : A \to B}$$

$$(A)\frac{\Gamma \vdash e : A \to B \quad \Gamma \vdash e' : A}{\Gamma \vdash (e\ e') : B}$$

The Coq Proof Assistant IV

Curry-Howard Isomomorphism

For all formula there exists a proof of this formula in natural deduction if and only if there exists a λ-term that has this formula as type.

- Theorem statement $\Leftrightarrow$ Type
- Proof $\Leftrightarrow$ Program

Coq in practice

- Functional programming language
- Rich type system: allow to express logical properties
- Language for building proofs (ie proof terms)
- Program extraction

The Coq Proof Assistant V

The Lecture

- Functional programming in Coq
- Writing specifications
- Proving correctness
- Program extraction

Outline

Parallel Programming in Coq I

High-level Building Blocks

- Candidates: algorithmic skeletons
- How to implement them?

- How to prove the correctness of the implementation?

Parallel Programming in Coq I

High-level Building Blocks

- Candidates: algorithmic skeletons
- How to implement them?
  with **Bulk Synchronous Parallel ML** (BSML)
  a dialect/library for OCaml
- How to prove the correctness of the implementation?
  using the **Coq** Proof Assistant

The Bulk Synchronous Parallel ML Approach

- an efficient functional programming language with formal semantics and easy reasoning about the performance of programs (strict evaluation):

  ML **(Objective Caml flavor)**

- a restricted model of parallelism with no deadlock, very limited cases of non-determinism, a simple cost model:

  Bulk Synchronous Parallelism

The result is:

Bulk Synchronous Parallel ML (BSML)

Parallel Programming in Coq III

Bulk Synchronous Parallel ML

Design principles:

- Small set of parallel primitives
- Universal for bulk synchronous parallelism
- Global view of programs
- Simple formal semantics

BSML:

a sequential functional language

\+ a parallel data structure (non nestable)

\+ parallel operations on this data structure

Papers and software

- `http://traclifo.univ-orleans.fr/BSML`

Parallel Programming in Coq IV

The Lecture

- Tutorial on programming with BSML
- BSML in Coq
- Reasoning about BSML programs in Coq
- Applications and experiments

Based on [21, 20]

Outline

Constructive Algorithms in Coq I

$maximum = hd \circ sort.$

if law :
$\forall$ f,
f (if a then b else c)
= if a then f b else f c

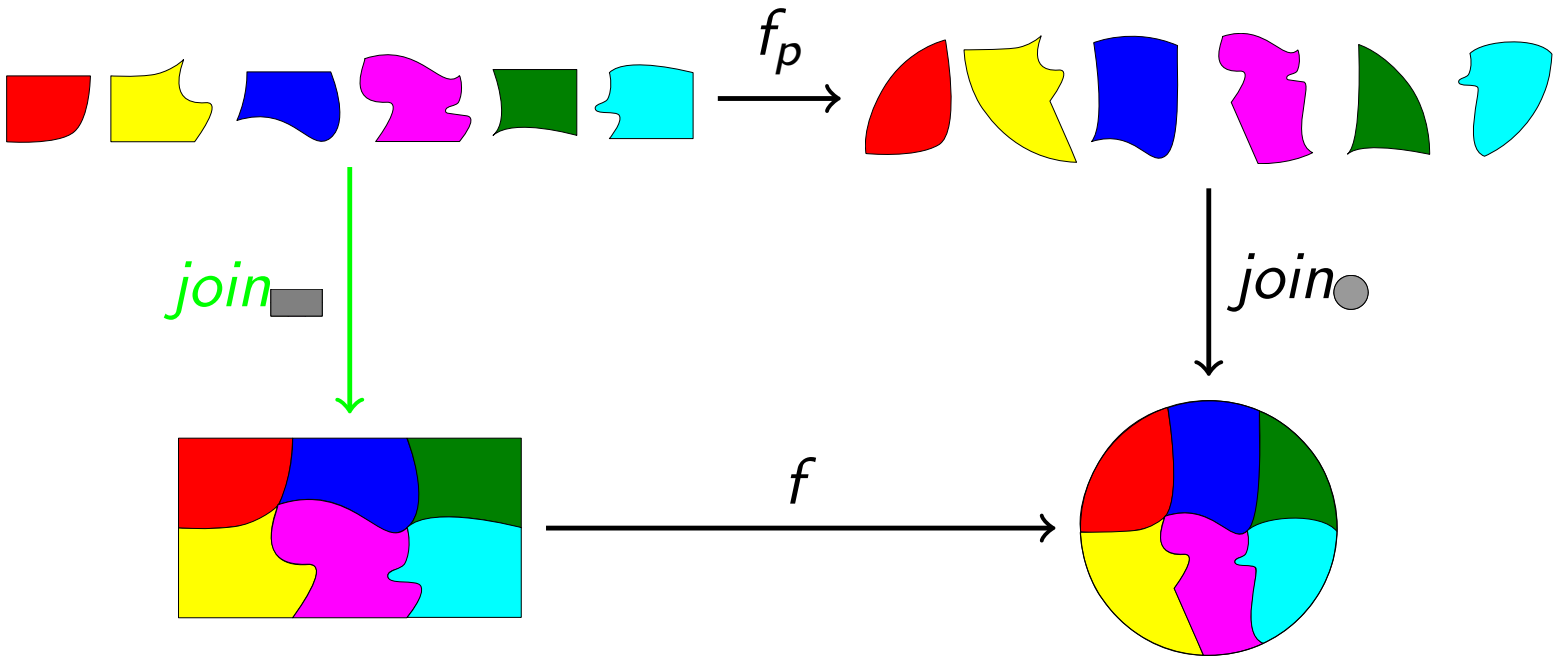
$$
\begin{array}{ll}
 & maximum\ (a :: x) \\
= & \quad \{\ \text{def. of maximum}\ \} \\
 & hd\ (sort\ (a :: x)) \\
= & \quad \{\ \text{def. of sort}\ \} \\
 & hd\ (\text{if } a > hd(sort\ x) \text{ then } a :: sort\ x \\
 & \qquad \text{else } hd(sort\ x) :: insert\ a\ (tail(sort\ x))) \\
= & \quad \{\ \text{by if law}\ \} \\
 & \text{if } a > hd(sort\ x) \text{ then } hd(a : sort\ x) \\
 & \text{else } hd(hd(sort\ x) :: insert\ a\ (tail(sort\ x))) \\
= & \quad \{\ \text{def. of hd}\ \} \\
 & \text{if } a > hd(sort\ x) \text{ then } a \text{ else } hd(sort\ x) \\
= & \quad \{\ \text{def. of maximum}\ \} \\
 & \text{if } a > maximum\ x \text{ then } a \text{ else } maximum\ x \\
= & \quad \{\ \text{define } x \uparrow y = \text{if } x > y \text{ then } x \text{ else } y\ \} \\
 & a \uparrow (maximum\ x)
\end{array}
$$

Constructive Algorithms in Coq II

```
Lemma maximum_over_list :
    ∀ a x d,
      maximum d (a::x) = if a ?> (maximum a x) then a else maximum a x.
  Proof.
    Begin.
    LHS
    = { by def maximum}
       (hd d (sort (a::x)) ).
    = { unfold_sort }
       (hd d (
        if a ?> (hd a (sort x)) then a::(sort x)
         else (hd a (sort x))::(insert a (tail (sort x))))).
    ={ rewrite (if_law _ (hd d))}
       (if a ?> hd a (sort x) then hd d (a :: (sort x))
        else hd d (hd a (sort x) :: insert a (tail (sort x)))) .
    ={ by def hd ; simpl_if }
       (if a ?> hd a (sort x) then a else hd a (sort x)) .
    ={by def maximum}
       (if a ?> (maximum a x) then a else maximum a x).
    [].
  Qed.
```

Constructive Algorithms in Coq III

Correct Parallelisation



Constructive Algorithms in Coq IV

The Lecture

- Program calculation in Coq
- Examples
- Correct parallelisation
- Application to BSML skeletons/applications

Based on [21, 20]

Outline

Powerlists in Coq I

Powerlists — J. Misra

- singleton powerlist
- p, q powerlists of the same length:
  - *tie*: $p|q$ concatenation of p and q,
  - *zip*: $p \bowtie q$ alternating items from p and q

Powerlist laws

- L0 (identical base cases): $\langle x \rangle | \langle y \rangle = \langle x \rangle \bowtie \langle y \rangle$
- L1 (dual deconstruction): P a non-singleton powerlist there exist r, s, u, v such that:
  $(a) \quad P = r \mid s$
  $(b) \quad P = u \bowtie v$
- L2 (unique deconstruction):
  $(a) \quad (\langle x \rangle = \langle y \rangle) \equiv (x = y)$
  $(b) \quad (p \mid q = u|v) \equiv (p = u \wedge q = v)$
  $(c) \quad (p \bowtie q = u \bowtie v) \equiv (p = u \wedge q = v)$
- L3 (commutativity):
  $(p \mid q) \bowtie (u \mid v) = (p \bowtie u) \mid (q \bowtie v)$

Powerlists in Coq III

The Lecture

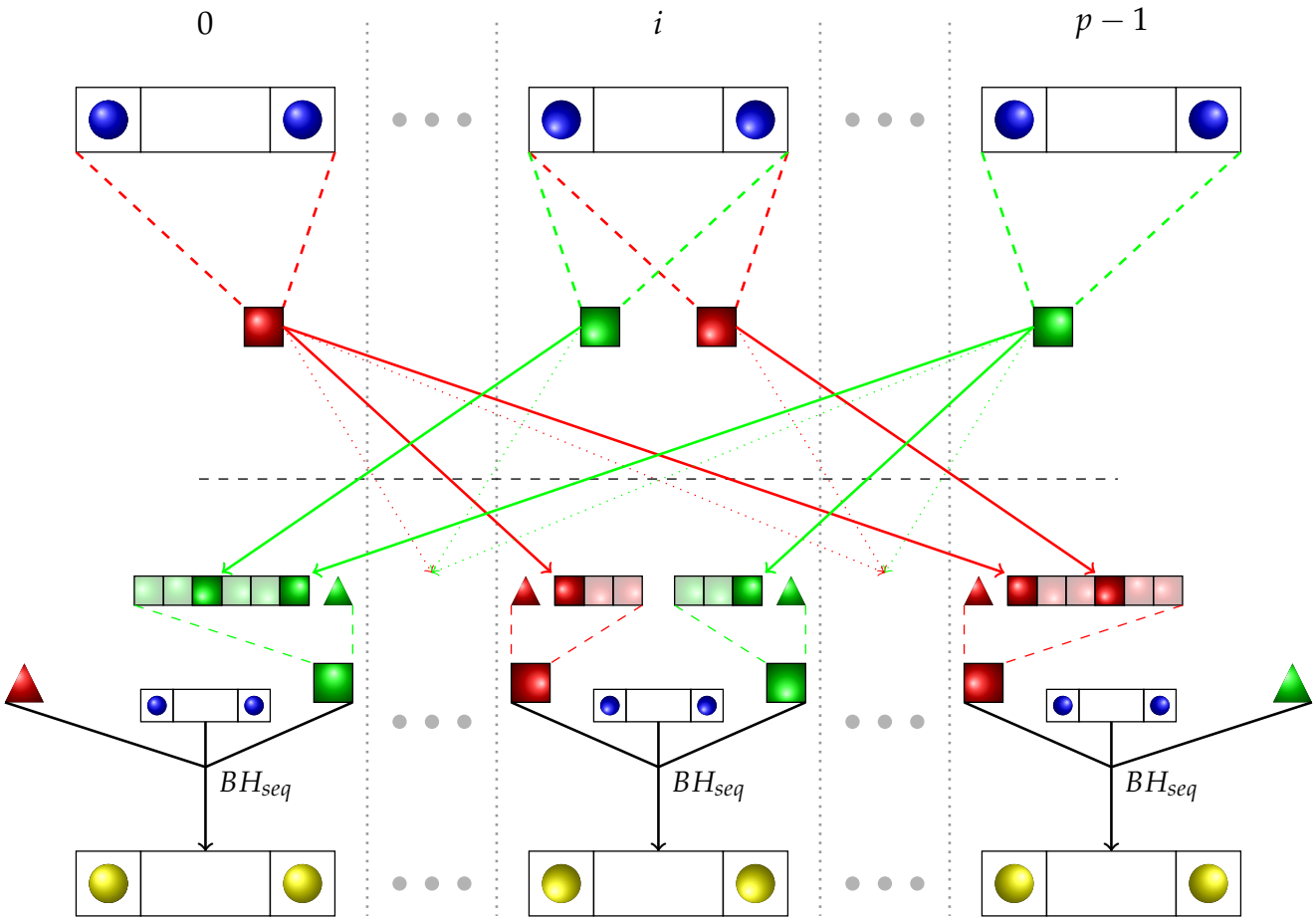
- Powerlist axiomatisation in Coq
- Programming with powerlists in Coq
- Reasoning about powerlists in Coq
- Applications

Based on [15, 14]

Outline

Bulk Synchronous Parallel Homomorphisms I

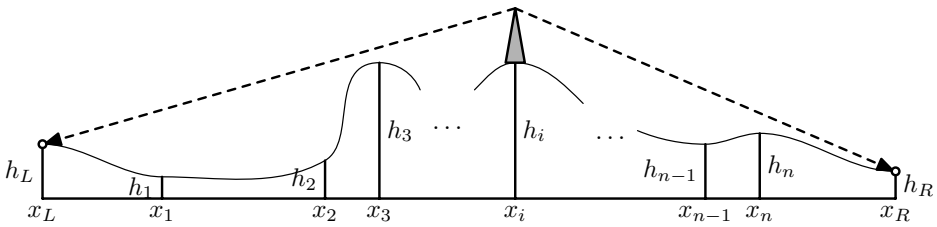
BSP Homomorphism



Bulk Synchronous Parallel Homomorphisms II

Applications

- Tower building



- Array packing

$$\mathit{pack}\ p\ [x_0, \ldots, x_n] = [\underbrace{x_{i_0}, \ldots, x_{i_m}}_{\forall x.\ p\ x = \text{true}}, \underbrace{x_{k_0}, \ldots, x_{k_{m'}}}_{\forall x.\ p\ x = \text{false}}]$$

- All nearest smaller values

$$\begin{aligned}\mathit{ansv}\ &[3,1,4,1,5,9,2] \\ = &[(\bot,1),(\bot,\bot),(1,1),\\ &(\bot,\bot),(1,2),(5,2),(1,\bot)]\end{aligned}$$

- Sparse matrix-vector multiplication

Bulk Synchronous Parallel Homomorphisms III

The Lecture

- BSP homomorphisms theory
- BSP homomorphisms in Coq
- The BH skeleton:
  - in Coq
  - in OSL
- Applications

Outline

Generate-Test-Aggregate in Coq I

Generate-Test-Aggregate K. Emoto, S. Fischer, Z. Hu

User's perspective:

- **G** Generator to produce all candidates
- **T** Tester for validity check of candidates
- **A** Aggregator to build the final result

Optimization:

- Filter embedding: **G** **T** **A** $\Rightarrow$ **G** **A**
- Semiring Fusion: **G** **A** $\Rightarrow$ D&C

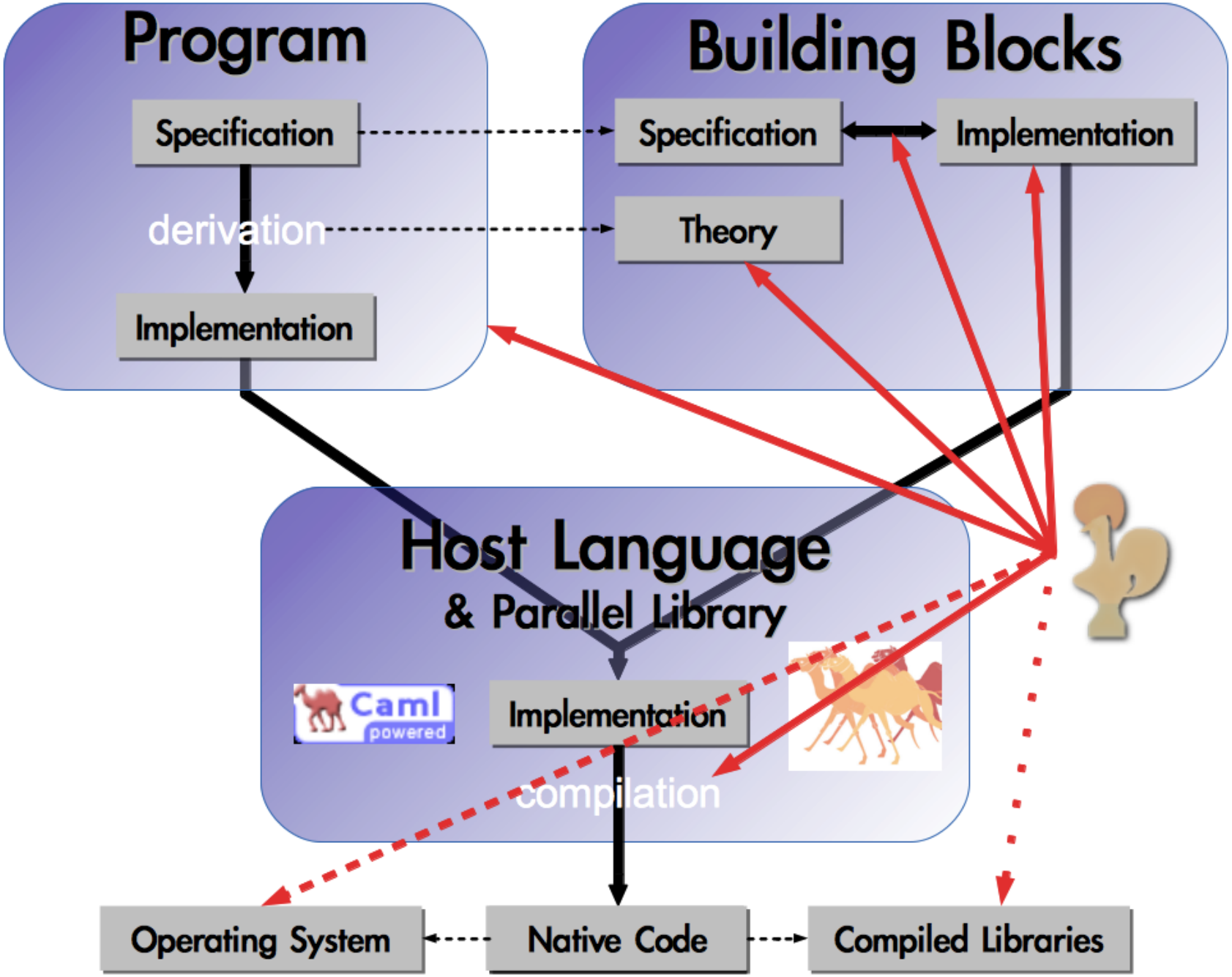
Generate-Test-Aggregate in Coq II

The Lecture

- Overview of GTA
- GTA in Coq:
  - Data structures
  - GTA optmisation theorem
  - Support for automatic parallelisation
  - Program extraction?

Based on [8, 9] and ongoing work

Outline


Program
Specification
derivation
Implementation
Building Blocks
Specification
Implementation
Theory
Host Language
& Parallel Library
Caml
powered
Implementation
compilation
Operating System
Native Code
Compiled Libraries

Outline

Bibliography I

[1] M. Bamha and M. Exbrayat. Pipelining a Skew-Insensitive Parallel Join Algorithm. *Parallel Processing Letters*, 13(3):317–328, 2003.

[2] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development*. Springer, 2004.

[3] R. Bisseling. *Parallel Scientific Computation. A structured approach using BSP and MPI*. Oxford University Press, 2004.

[4] A. Braud and C. Vrain. A parallel genetic algorithm based on the BSP model. In *Evolutionary Computation and Parallel Processing GECCO & AAAI Workshop*, Orlando (Florida), USA, 1999.

[5] M. Cole. *Algorithmic Skeletons: Structured Management of Parallel Computation*. MIT Press, 1989. Available at `http://homepages.inf.ed.ac.uk/mic/Pubs`.

[6] M. Cole. Parallel Programming with List Homomorphisms. *Parallel Processing Letters*, 5(2):191–203, 1995.

Bibliography II

[7] D. C. Dracopoulos and S. Kent. Speeding up genetic programming: A parallel BSP implementation. In *First Annual Conference on Genetic Programming*. MIT Press, July 1996.

[8] K. Emoto, S. Fischer, and Z. Hu. Generate, Test, and Aggregate – A Calculation-based Framework for Systematic Parallel Programming with MapReduce. In *ESOP*, volume 7211 of *LNCS*, pages 254–273. Springer, 2012. doi:10.1007/978-3-642-28869-2_13.

[9] K. Emoto, S. Fischer, and Z. Hu. Filter-embedding semiring fusion for programming with MapReduce. *Formal Asp. Comput.*, 24(4-6): 623–645, 2012. doi:10.1007/s00165-012-0241-8.

[10] L. Granvilliers, G. Hains, Q. Miller, and N. Romero. A system for the high-level parallelization and cooperation of constraint solvers. In Y. Pan, S. G. Akl, and K. Li, editors, *Proceedings of International Conference on Parallel and Distributed Computing and Systems (PDCS)*, pages 596–601, Las Vegas, USA, 1998. IASTED/ACTA Press.

Bibliography III

[11] Z. Hu, H. Iwasaki, and M. Takechi. Formal derivation of efficient parallel programs by construction of list homomorphisms. *ACM Trans. Program. Lang. Syst.*, 19(3):444–461, 1997. ISSN 0164-0925. doi:10.1145/256167.256201.

[12] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: formal verification of an OS kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (SOSP'09)*, pages 207–220. ACM, 2009. ISBN 978-1-60558-752-3. doi:10.1145/1629575.1629596.

[13] X. Leroy. A formally verified compiler back-end. *Journal of Automated Reasoning*, 43(4):363–446, 2009. doi:10.1007/s10817-009-9155-4.

[14] F. Loulergue, V. Niculescu, and S. Robillard. Powerlists in Coq: Programming and Reasoning. In *First International Symposium on Computing and Networking (CANDAR)*. IEEE Computer Society, 2013. to appear.

Bibliography IV

[15] J. Misra. Powerlist: A structure for parallel recursion. *ACM Trans. Program. Lang. Syst.*, 16(6):1737–1767, November 1994.

[16] A. Morihata, K. Matsuzaki, Z. Hu, and M. Takeichi. The third homomorphism theorem on trees: downward & upward lead to divide-and-conquer. In Z. Shao and B. C. Pierce, editors, *Symposium on Principles of Programming Languages (POPL)*, pages 177–185. ACM Press, 2009. doi:10.1145/1480881.1480905.

[17] K. Morita, A. Morihata, K. Matsuzaki, Z. Hu, and M. Takeichi. Automatic Inversion Generates Divide-and-Conquer Parallel Programs. In *Conference on Programming Language Design and Implementation (PLDI)*, pages 146–155. ACM Press, 2007. doi:10.1145/1250734.1250752.

[18] R. O. Rogers and D. B. Skillicorn. Using the BSP cost model to optimise parallel neural network training. *Future Generation Computer Systems*, 14(5-6):409–424, 1998.

Bibliography V

[19] J. Sevcík, V. Vafeiadis, F. Z. Nardelli, S. Jagannathan, and P. Sewell. CompCertTSO: A Verified Compiler for Relaxed-Memory Concurrency. *Journal of the ACM*, 60(3):22, 2013. doi:10.1145/2487241.2487248.

[20] J. Tesson. *Environnement pour le développement et la preuve de correction systématiques de programmes parallèles fonctionnels*. PhD thesis, LIFO, University of Orléans, November 2011. URL `http://hal.archives-ouvertes.fr/tel-00660554/en/`.

[21] J. Tesson, H. Hashimoto, Z. Hu, F. Loulergue, and M. Takeichi. Program Calculation in Coq. In *Thirteenth International Conference on Algebraic Methodology And Software Technology (AMAST2010)*, LNCS 6486, pages 163–179. Springer, 2010. doi:10.1007/978-3-642-17796-5_10.

[22] The Coq Development Team. The Coq Proof Assistant. `http://coq.inria.fr`.

[23] L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103, 1990. doi:10.1145/79173.79181.

Coq Hands-On

- To try during the presentation, and/or
- For Coq hands-on after the lecture

Installing Coq

Coq website: `http://coq.inria.fr`
Recommended installation:

- Windows installer for Coq's website
- Linux: Ubuntu packages (`coq`, `coqide`, `proofgeneral-coq`)
- MacOS: MacPorts or Homebrew

Recommended IDE: emacs + ProofGeneral