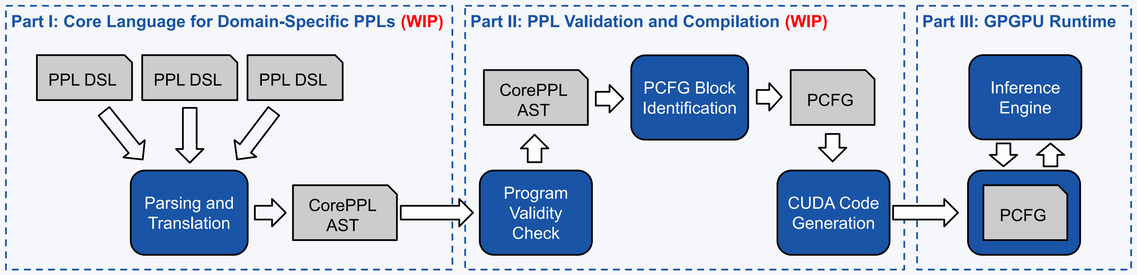
Compilation of Universal Probabilistic Programs to GPGPUs

Daniel Lundén[1], Joey Öhman[1], David Broman[1]

[1]KTH Royal Institute of Technology

Overview

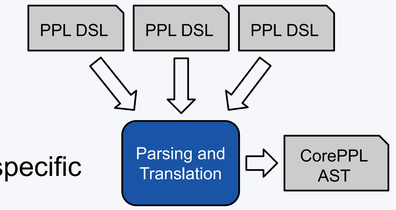
- This work-in-progress is concerned with **universal** probabilistic programming languages (PPLs).
  - Examples: ***WebPPL, Anglican, Birch.***
- Currently, few PPL systems make use of HPC hardware.
  - Examples: ***LibBi*** (CUDA, not universal), ***Pyro*** (PyTorch), ***Edward*** (TensorFlow).
- Our goal: Develop a ***highly efficient*** framework for compiling universal probabilistic programs written in a core PPL down to ***pure*** CUDA GPGPU code.



The figure gives a high-level overview of the overall toolchain. The gray rectangular boxes are artifacts, such as programs, abstract syntax trees (ASTs), or probabilistic control flow graphs (PCFGs). These artifacts are processed by the rounded boxes in blue.

Part I: Core Language for Domain-Specific PPLs (Work in progress)

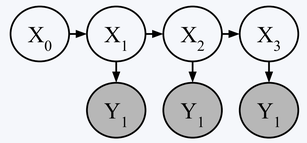
CorePPL AST



- A small universal PPL based on the lambda calculus.
- A target ***intermediate language*** when translating from domain-specific PPLs.
- Also envisioned as an intermediate language for ***optimization*** phases.

PPL DSL

```
1   let observe = lam v. lam dist. weight (logpdf v dist) in
2
3   let lgss xprev data = match data with
4     | [] -> xprev
5     | y : ys ->
6       let x = sample (normal (xprev + 2) 1) in
7       observe y (normal x 1);
8       resample ();
9       lgss x ys in
10
11  let data = [24.39, 25.47, 29.62] in
12  let x0 = sample (normal 0 100) in
13  lgss x0 data
```



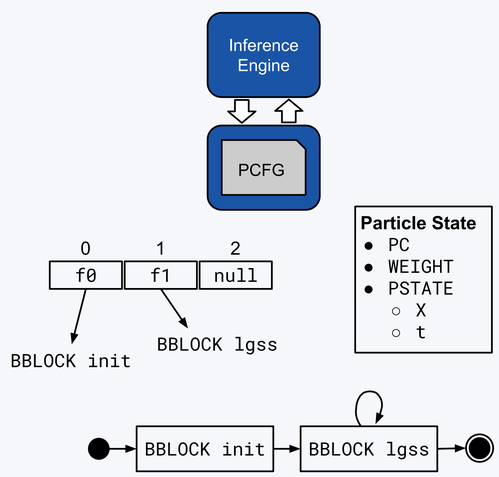
Part III: GPGPU Runtime

- Currently supports highly efficient ***SMC inference*** using CUDA (also supports OpenMP).
  - This implementation has been evaluated on a number of complex models from ***phylogenetics*** (with impressive results)
- We are working on adding support for more inference algorithms (e.g., ***MCMC***).

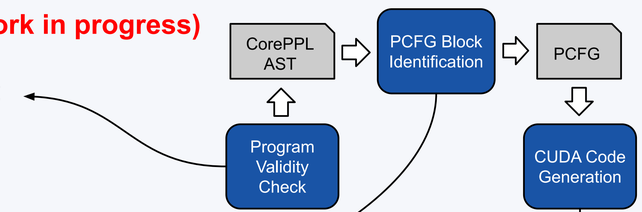
PCFG

```
1  BBLOCK(init, {
2      PSTATE.x = SAMPLE(normal, 0.0, 100);
3      PSTATE.t = 0;
4
5      PC++;
6      BBLOCK_CALL(lgss, NULL);
7  })
```

```
1  BBLOCK(lgss, {
2      floating_t* dataP = DATA_POINTER(data);
3
4      PSTATE.x = SAMPLE(normal, PSTATE.x + 2.0, 1);
5      OBSERVE(normal, PSTATE.x, 1.0, dataP[PSTATE.t]);
6
7      if(++PSTATE.t == NUM_OBS)
8          PC++;
9  })
```



Part II: PPL Validation and Compilation (Work in progress)



- Rule out higher-order programs that cannot be compiled to the GPU.
  - Not allowed: ***Closures***, data structures requiring ***garbage collection***, etc.
- Transform the CorePPL AST into a PCFG.
  - Explicitly manage a ***call stack*** to support complex control flow involving BBLOCKs and inference algorithms (e.g., resampling in SMC).
- Generate the actual ***CUDA C++ code***, which is later compiled using the CUDA compiler.

This is an efficient (and, for CUDA, necessary) alternative to using ***continuation-passing style*** (as is done in WebPPL and Anglican).

This project is financially supported by the Swedish Foundation for Strategic Research (ASSEMBLE RIT15-0012, SSF FFL15-0032).