Communication Safe Application Parallelisation with Session Types

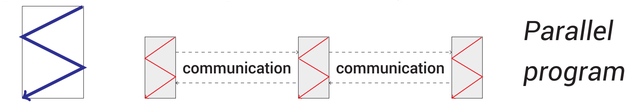
Nicholas Ng, José Gabriel F. Coutinho, Nobuko Yoshida and Wayne Luk

Department of Computing, Imperial College London

Imperial College London

Background

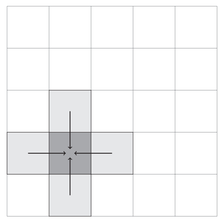
- **Parallel programming** difficult to master, error prone
- Parallel application = computation + synchronisation



- Aim: **Simplify** parallel programming
  - Minimal effort to develop parallel program from sequential code
  - Safety guarantees to ensure parallel application is correct
- Proposed approach: Separation of concern
  - **Developer** focuses on functional computation code
  - **Parallel expert** focuses on parallelising and communication
- Communication safety guaranteed by **Session Types**

Developer input: Sequential code

- Defines **functional behaviour**
- Isolates computation in **kernels**
- C source code with annotations
  - e.g. Label a section of code
  - e.g. Partitioning instructions



Example: 5-point stencil

Sequential code

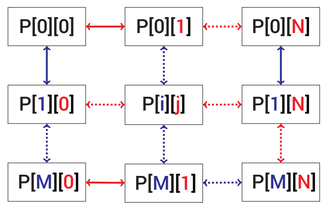
```
int main(int argc, char *argv[])
{ ...
  for (int h=0; h<H; h++)
    for (int w=0; w<W; w++)
      if (1<=h && h<=H-2 && 1<=w && w<=W-2) {
        tmp[h*W+w] = (mtx[(h-1)*W+w] + mtx[(h+1)*W+w]
                    + mtx[h*W+w]
                    + mtx[h*W+(w-1)] + mtx[h*W+(w+1)])/5;
      }
  return EXIT_SUCCESS;
}
```

Our proposal: Pabble protocol description language

- Pabble: Parameterised Scribble [5]
  - **Scalable** communication protocol description language
  - Specific for **parallel programming**
  - Use **parameters** on participants to scale protocols
  - Guarantees **communication safety** and **deadlock free**
- Formal basis: Multiparty Session Types (MPST) [4]
  - **Typing system** for communication
  - Idea: Communication interactions are **dual** (Send vs. Receive)
  - Parametric variant: Parameterised MPST [2]
- Derived from Scribble project [3]
  - Developer-friendly protocol language for distributed systems
  - Academia-industry collaboration to make MPST accessible

Parallel expert input: Communication topology

- Parallel **interaction structure**
  - i.e. Communication topology
- Pabble protocol language
  - Non-application specific
  - Scalable parallel protocols
  - Communication safety for free!



Example: M-by-N mesh for partitioned subproblems

Mesh topology

```
global protocol Mesh(role P[0..M][0..N]) {
  rec Iterate {
    Up(int)    from P[i:1..M][j:0..N] to P[i-1][j];
    Down(int)  from P[i:0..M-1][j:0..N] to P[i+1][j];
    Left(int)  from P[i:0..M][j:1..N] to P[i][j-1];
    Right(int) from P[i:0..M][j:0..N-1] to P[i][j+1];
    continue Iterate;
} }
```

References

- J. M. P. Cardoso, T. Carvalho, J. G. F. Coutinho, W. Luk, et al. Lara: an aspect-oriented programming language for embedded systems. In *AOSD*, pages 179–190, 2012.
- P.-M. Deniélou, N. Yoshida, A. Bejleri, and R. Hu. Parameterised multiparty session types. *LMCS*, 8, 2012.
- K. Honda, A. Mukhamedov, G. Brown, T.-C. Chen, and N. Yoshida. Scribbling interactions with a formal foundation. In *ICDCIT 2011*, volume 6536 of *LNCS*, pages 55–75, 2011.
- K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *POPL'08*, volume 5201 of *LNCS*, pages 273–284, 2008.
- N. Ng and N. Yoshida. Pabble: Parameterised scribble for parallel programming. In *PDP 2014*, 2014.

Output: MPI Parallel application

- Code generation by aspect-oriented compilation [1]
- Parallelise for distributed execution by MPI
- Computation: analyse & extract from **sequential code**
- Communication: define with **Pabble protocol**

Example:

Parallelised MPI application

```
#include <mpi.h>
int main(int argc, char *argv[])
{ MPI_Init(&argc, &argv);
  while (1) {
    tmp = calculate_subproblem(mtx, H, W);
    i = rank/W; j = rank%W;
    if (1<=i&&i<=M&&0<=j&&j<=N)    MPI_Recv(rank-W,Up);
    if (0<=i&&i<=M-1&&0<=j&&j<=N)  MPI_Send(rank+W,Up);
    if (0<=i&&i<=M-1&&0<=j&&j<=N)  MPI_Recv(rank+W,Down);
    if (1<=i&&i<=M&&0<=j&&j<=N)    MPI_Send(rank-W,Down);
    ...
  }
  MPI_Finalize();
  return EXIT_SUCCESS;
}
```