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

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

**References**

  - **Lara:** an aspect-oriented programming language for embedded systems.
  - Parameterised multiparty session types.
    LMCS, 8, 2012.
  - Scribbling interactions with a formal foundation.
    In ICDCIT 2011, volume 6536 of LNCS, pages 55–75, 2011.
- K. Honda, N. Yoshida, and M. Carbone.
  - Parallel asynchronous session types.
    In POPL'12, 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:**
```c
MPI_Recv;
MPI_Send;
MPI_Finalize();
return EXIT_SUCCESS;
```