Kanor : a DSL for Declarative Communication

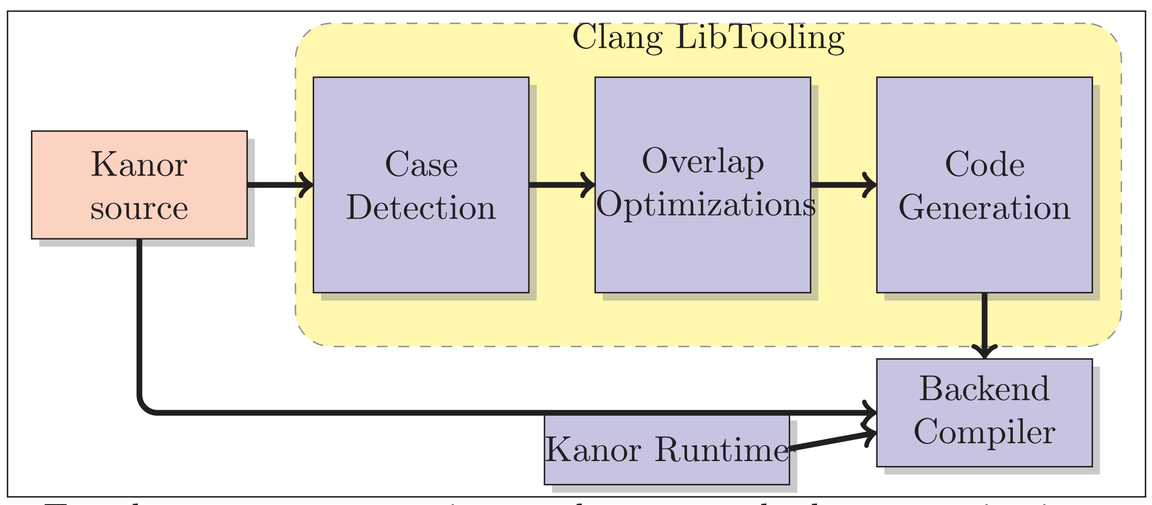
Nilesh Mahajan nnmahaja@indiana.edu

Motivation

Writing efficient parallel programs continues to be a challenge for the programming community. Writing efficient and correct parallel programs remains a challenging activity for programmers. Our approach, Kanor

- is a domain specific language embedded in C++
- follows the partitioned address space model like MPI
- provides constructs similar to list comprehensions for communication
- guarantees properties like deadlock freedom, deterministic evaluation

Kanor System



- Template metaprogramming used to type check communication statements.
- Kanor programs can be executed without involving our Clang passes.
- Clang LibTooling passes needed by optimizations that need code rewriting.

Communication Properties

- **Across Processes**
  - *Global* - all processes know the communication pattern
  - *Corresponding* - the sender and receiver know each other
  - *Sender* - only a sender knows its receivers
- **Across Statement Instances**
  Certain characteristics of the communication statement such as process sets, message sizes, message contiguity might not change between invocations. Allows Kanor runtime to cache the pattern for later use.

Detection of Collectives

1 **Input**: Communication Statement S
2 **Output**: Set of Collective Calls C
3 G = build from S;
4 n = number of vertices in vertex set V(G);
5 **if** *each vertex v in V(G) has degree n* **then**
6 **if** *send and receiver buffers contiguous* **then**
7 C = {Alltoall};
8 **else**
9 C = {AllGather};
10 return;
11 **foreach** *v in V(G) with no incoming edges* **do**
12 **if** *send and receiver buffers contiguous* **then**
13 C = C ∪ {broadcast};
14 **else**
15 C = C ∪ {scatter};

Figure 1: Algorithm to detect MPI collectives.

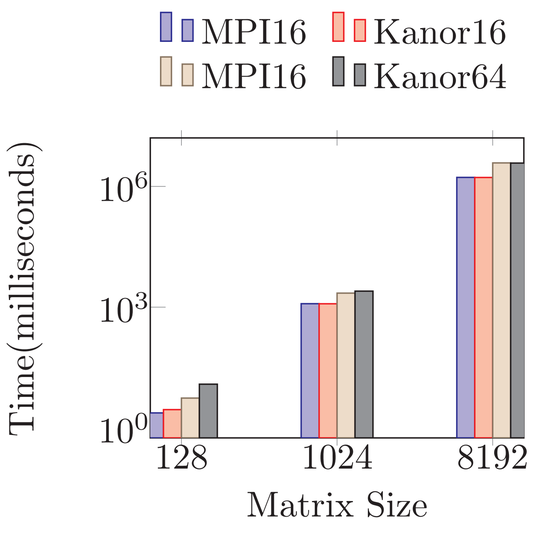


Figure 2: Cholesky

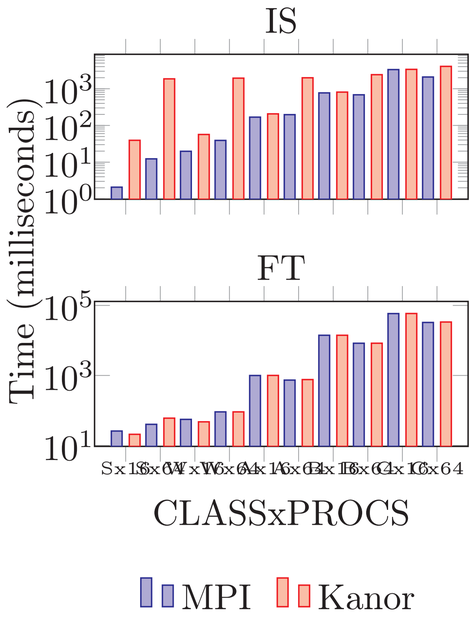
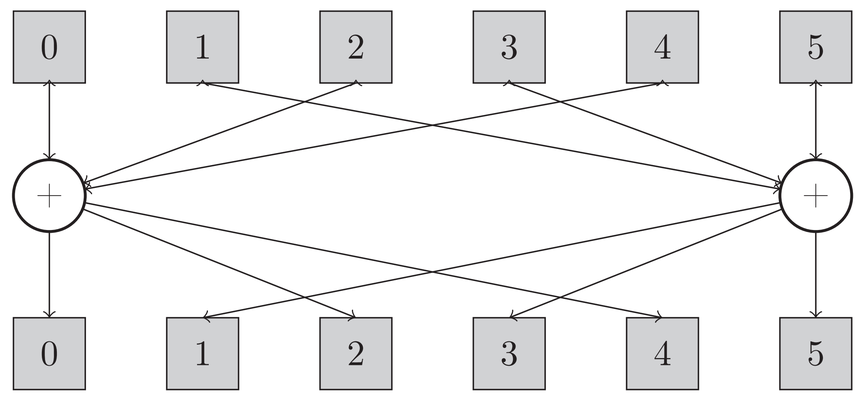


Figure 3: NAS Benchmarks

MPI vs Kanor

```
int rval;
std::vector<int> sbuff;
...
std::vector<MPI_Request> reqs;
int rmdr = me % 2;
for (int i = 0; i < nprocs; i++) {
  if (i % 2 == rmdr) {
    MPI_Request req;
    MPI_Isend(&sb[i], 1, MPI_INT, i, 0, MPI_COMM_WORLD, &req);
    reqs.push_back(req);
  }
}
MPI_Waitall(reqs.size(), reqs.data(), MPI_STATUSES_IGNORE);
for (int i = 0; i < nprocs; i++) {
  if (i % 2 == rmdr) {
      int r;
      MPI_Recv(&r, 1, MPI_INT, i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
      rval += r;
  }
}
```

```
int rval;
kanor::CommBuff<int> sbuff;
...
Topology t;
rval _at_ rcvr << std::plus<int>() << sbuff[rcvr] _at_ sndr |
  _for_each(sndr, t.world) & _for_each(rcvr, t.world) &
  _if ((sndr % 2) == (rcvr % 2)) _with t & GLOBAL;
```



Optimizations

BSP nature of Kanor avoids static send/receive matching problem.

```
// sb is sent buffer
// rb is received buffer
... = rb;
sb = ...;
...;
for (i=0; i < N; i++) {
  rb _at_ r << sb _at_ s | ...;
  ... = rb;
  // compute
  sb = ...;
}
```

Figure 4: Loop with comm statement

```
... = rb;
sb = ...;
start_send(sb, ...);
...;
for (i=0; i < N; i++) {
  wait(...);
  ... = rb;
  // compute
  sb = ...;
  start_send(sb, ...);
}
```

Figure 5: Loop with look ahead

```
... = rb;
sb = ...;
start_send(sb, ...);
for (i=1; i < min(B,N); i+=B) {
  for (int b = 0; b < B; b++) {
    ... = rb;
    sb = ...;
    ...
  }
  start_send(sb, B, ...);
}
```

Figure 6: Look-ahead loop strip mined

- **Async Window Expansion** - Use asynchronous calls to start communication early and overlap it with computation
- **Strip Mining** - Adjust the granularity of communication with computation that produces or uses the communicated data
- **Buffer Expansion** - The communication buffer is copied so that the computation uses one copy of the buffer while the communication works with other copy.

Conclusions

- Kanor allows specification of communication patterns in a declarative way.
- Kanor communication statements are deadlock-free and deterministic.
- BSP nature of Kanor makes it easier for compiler, runtime to perform optimizations.
- Interesting future directions include optimizing for graph based algorithms in Kanor, mechanizing Kanor semantics and proving correctness of transformations.

Acknowledgements

I would like to thank my advisors Dr. Arun Chauhan and Dr. Andrew Lumsdaine for help with this research.