– EXTRACT FROM FUTURE ECA PAPER –

In the FAIR timing system, the data master issues a time-sensitive command as a tuple of values, (x, y, z). Front-end controllers implement these commands by executing actions to control the physical devices which direct the beam. To determine if a command is relevant, front-end controllers match their local configuration against the issued command tuples. Examples of tuple fields thus far include: Machine, EventCode, Sequence, Beam Process, Production Chain.

This communication pattern, content-based publish-subscribe, has been extensively studied in database circles [1]. The only novel feature in the FAIR timing system is that the command stream must be processed within hard real-time constraints, on the order of nanoseconds. This necessitates a hardware implementation, restricting both the format of the tuples and the complexity of the matching rules we can implement.

Constant-time hardware circuits can only process a fixed amount of data. Therefore, in the FAIR timing system, it is necessary to set an upper limit on the tuple size. In the current implementation, this limit is set to 64-bits, although this can be modified by reprogramming the FPGA. It further simplifies the hardware design if there is also a limit imposed on the number of fields within the 64-bit tuple. Given the current real-time budget, eight 125MHz clock cycles are available, suggesting a cap of eight fields per tuple.

In a typical database query, one selects tuples using a boolean function. If the function returns true, the tuple is included in the result set. In content-based publish-subscribe systems, the same function serves to accept/reject or filter the message. Unfortunately, complex functions cannot be executed in constant (or real) time. Worse, the front-end controllers must support multiple filter functions; for example, a single front-end controller might control two accelerator devices.

A hardware query processing engine is implemented as a pipeline. Given our budget, each stage of that pipeline must complete in eight cycles. Unfortunately, there will likely be more than eight filters per front-end controller. Therefore, it must be possible to reject more than one filter per cycle.

To solve the matching problem efficiently, we can again take inspiration from the considerable work on databases. While databases queries do offer the ability to evaluate an arbitrary function to select tuples, in practice they will use an index for performance. An index is, in some sense, a lexically sorted array of the tuples, suitable for binary search. We use the same approach in our hardware query processing engine.

In a database one can construct multiple indexes for the same tuple. These indexes typically differ in the ordering of the tuple fields. For example, one index might sort tuples in the form (x, y, z) while another sorts them as (y, z, x). The first index makes it possible to execute queries for $x = 5 \cap y = 3$ or $x = 2$ efficiently as these prefixes of the key are listed together. The second index makes it possible to execute queries for $y = 5$ or $y = 3 \cap z = 7$. Both indexes support queries for a concrete tuple, $x = 5 \cap y = 6 \cap z = 7$.

While indexes are usually constructed over tuples of data, one can also construct an index over queries. If there is an index which would support the query

for tuples, then that query can be placed into the index. Consider a query watching for a tuple with $y = 5 \cap z = 7$. An index over data tuples ordered as (y, z, x) would serve to process this query. If we add $x = *$ to the query, then we could insert $(5, 7, *)$ into an analogous query table. As long as the ordering matches, the queries in the index are always prefixes. Due to this property, given a data tuple, it is possible to hierarchically search the table for matching queries. This is the core idea we use in the hardware implementation.

In hardware, we can make two copies of a circuit to compute two results in parallel. Thus, multiple indexes can be searched for matching queries simultaneously. The savings in time simply costs us a more expensive chip to host the additional logic. For the FAIR project, so far only two tuple orderings appear necessary. Thus, by doubling the query processor's size, we can support all queries that are prefixes of these two orderings. Naturally, in our implementation, the number of indexes is left configurable so that if requirements change, the front-end controller FPGAs can be reprogrammed to accommodate them.

– CUT HERE FOR CTT –

References

[1] Patrick Th Eugster, Pascal A Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 35(2):114–131, 2003.