Using CCA Events for Polygraph and FPGAs

Ian Gorton, Daniel Chavarria, Manojkumar Krishnan, Jarek Nieplocha

Background

AT PNNL, we are working to accelerate key computationally intensive routines from the *Polygraph* application using FPGAs. We expect this application could be an excellent use case to explore requirements and implementation options for a CCA event service.

Polygraph description:

Polygraph has been developed at PNNL to analyze protein spectra obtained from mass spectrometry experiments. For each protein spectrum, the application stores position and intensity arrays from a tandem mass spectrometry run to be analyzed. Each spectrum is around 100 to 400 pairs of peak location, peak intensity numbers. The data also includes the weight of the peptide and its charge for each spectrum.

For each input, *Polygraph* scans through a reference database of several million proteins, the FASTA file (several GB in size), and generates a list of potential matching peptides based on weight. Typically, there will be thousands to millions of candidates (one protein may generate several candidates) per spectrum. For each candidate, the application then computes a projected spectrum based on the residues involved and then compares that projection to the actual spectrum and assigns it a score based on statistically generated datasets and the number of peaks that "match" (are close enough). The top matches, (their name and matching subpeptide) are printed for each spectrum and a summary of the top hits for all of the spectra is printed as well.

Polygraph routines for FPGA

We have identified three routines in Polygraph that appear amenable to FPGA acceleration. These are:

fp_generate
fp_extract
fp_set-hypoth

We currently have an implementation of fp_generate running. We have found that in order to amortize FPGA set up costs, Polygraph would have to be modified to work on a batch of proteins (currently 512 in test) instead of being called once for each individual protein structure (as occurs in the existing sequential and MPI version)

In addition to passing a batch of proteins to fp_generate on the FGGA, the follow modifications are needed, and should be executed on a CPU which can directly access the FGPA memory:

1. Convert 32-bit integers to 21-bit numbers which are more efficient for this problem on the FPGA
2. Pack the values in to FPGA memory banks for more efficient access
3. Initialize and call the fp_generate routine on the FPGA
4. Unpack the results
5. Convert to 32-bit integers

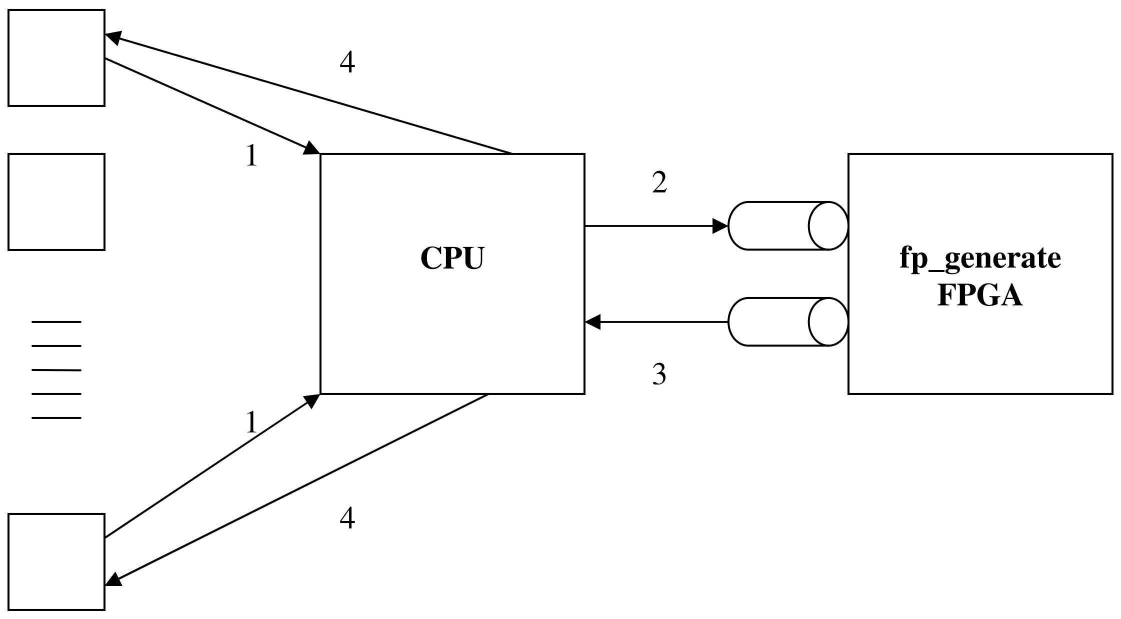
Steps 1,2, 4, 5 must be executed on the CPU attached to the FPGA.

Note these additional functions are all side-effects of the FPGA implementation of the routine. Hiding these behind a CCA SIDL interface would seem a sensible abstraction mechanism. This means the same CCA fp_generate interface could be implemented by versions running on the FPGA and pure (perhaps parallel) software versions, without the need for the caller to know the implementation details.

Hardware considerations

On the Cray machine we are working on, only 6 of the 18 (dual CPU) nodes have attached FPGAs. While this is an artifact of the configuration of this machine, we expect that in general HPC machines will typically, for cost reasons, have less FPGA's available than processing nodes.

Hence, in order to get full performance benefits from the FPGAs, we anticipate that a possible architecture for Polygraph will be as follows:



One CPU is dedicated as a 'server' to handle requests from other processors to execute fp_generate on the FPGA. These other processors send requests asynchronously (step 1 in diagram) to the server CPU. The server converts and packs the requests ready for

FPGA processing, and submits them one at time to the FPGA for processing. Concurrent request are queued on the server and executed on the FPGA when it becomes free. (step 2).

When the FPGA completes processing a batch of inputs, it informs the server CPU, which unpacks/converts the results (step 3), and returns these asynchronously to the sending processor (step 4).

This design aims to maximize the FPGA usage, and dedicates the attached CPU to performing the necessary pre- and post-processing on FPGA requests. If a sufficient volume of requests can be generated by the other processing nodes in the application, then the attached CPU is likely to be kept busy. The requesting processes can also continue with other useful work while they wait for a response from the FPGA.

The design also extends to support a pool of FPGA-attached server CPUs. Requests for processing could be load-balanced across these servers to reduce response latencies.

What's this got to do with CCA Events?

We could implement fp_generate as a CCA component that simply subscribed to a named topic/channel. This component would be executed on a processing node with an attached FPGA.

Other nodes would publish 'request' events to the named topic, and continue with other processing while listening on a 'reply' topic for the results to be returned.

This would require the event service in this use case to be based on some underlying RMI protocol, as the publishers and subscribers do not execute in the same address space.

This raises the question of how the event service is implemented in this scenario, for example:

- a centralized (single?) process/thread. If it is a process/thread, where does it execute?
- A peer-to-peer implementation, in which SIDL generated stubs implement event service features (eg Manta Ray - http://sourceforge.net/projects/mantaray/). This potentially may create a higher performance and more robust event service?

A possible 'smart' would be to have 'load balancing topics/channels'. They are programmatically identical to a normal pub-sub topic. Behaviorally they accept published events and only send each event to exactly one subscribers, chosen on the basis of some load balancing algorithm (eg round-robin). I saw this in part of TIBCO's pub-sub technology, and found it very useful on occasions.