Integrating TyBEC cost model with CHREC cost model

The TyBEC cost model can estimate resources and performance of an application (1 or more kernels) on a single device. It works from the IR code and a perl-based compiler parses it to generate the estimates (on a roofline model).

At CHREC, a VisualSim based model has been developed to model the performance of (as far as I can understand) 3D -FFT application. However, as part of that process, microbenchmarks have been used to estimate paramters that lead to an overall performance estimate. The results from these microbenchmarks can be used to create an integrated cost model that estimate the cost of an different varations of application's implementation on a cluster. This observation forms the basis of this work.

Scope

We will cost designs expressed in the IR code, and targetted at Novo-G# cluster initially. Also:

1. For device-parallelism, we will limit the partitioning to task-parallelism initially (different kernels to different devices), and data-parallelism will be left for later.
2. We will limit the model to a single (CPU) node with a device cluster (rather than a cluster of nodes).

Notes on the Current TyBEC Cost Model

The TyBEC model assumes a single-device implementation, parses a given IR code, and generates a performance and resource cost-estimate.

It required the prior execution of microbenchmarks on the device which calculate the resource and performance for various primitive instructions, and also estimate sustained bandwidth to the global memory.

Notes on the Current Novo-G# Cost Model

Initially, the Anton machine was modelled, implemented 3D FFT.

- Can actually run the algorithm (and verify results), or turn that off to speed things up, and just get estimates.
- The followign modelling parameters were used (based on papers)
  - system frequency
  - internal bandwidth
  - external bandwidth
  - synch delay
  - package writing delay
  - wire delay, x, y, z
  - tranceiver delay
  - FFT calculation time on 1/4 *Geometry cores* (GCs) [That is, the computation time, in cycles].
- FFT algorithm and routing algorithm are implemented as scripts inside the VSim blocks.

Then, once this Anton model was verified, a similar model for Novo-G# was built.

- The Novo-G# cluster is configured as a 3D-torus.
- Various PHY options are possible, with different latencies and other parameter. However, for our purpose, we would just fix the PHY in our model depending on what is currently being used in the cluster (Interlaken, 118-151 ns latency)
- Packet size will have an impact on the performance.
- The implementation uses a Nios-II soft processor alongwith Altera 1D FFT Megacore IP.
- The available figures in the thesis are for a 2x2x2 torus.

- Resource estimates for Gidel's PCI interface (BSP) and the 3D-torus network stack are available (i.e. resource utilization excluding application logic)
- Overhead of using inter-FPGA linkes has minimal impact on the resources available for application acceleration.
- The various modelling parameters used (and hence available) for Novo-G# cluster are:
  - System frequency
  - FFT Latency (cycles)
  - Num cores
  - Nios core latency
  - Packet generation latency
  - Inter-FPGA latency (Interlaken PHY)
  - Router latency
  - Ext Rx/Tx latency
  - Channel rate (per channel of a link)
  - Channel width (= 4 in each direction, so 6x4)
  - Number of routers
  - Packet buffer length
- A script running in the VM in the applicaiton layer generates packets based on the psuedo-code.

Suggested Approach for a TyBEC–Nov-G# Cost Model

Two possible approaches:

1. *Analytical+Empirical (Basic):* Use the latency, bandwidth and resource parameters available from the work at CHREC on the Novo-G# cluster, and intergrate them into the curernt perl-based TyBEC backend.
2. *Functional+Empirical (Advanced):* Build a functional model, most likely on SystemC and one that involves a network model as well, and "run" the application (i.e, at least realistic number/size of packets, data can be dummy).