Systolic arrays vs. multithreaded multiprocessors

- Project Idea:
  - Develop models/methods to compare application performance between systolic arrays vs multithreaded multiprocessor systems
- Deliverable:
  - Students will develop a model incorporating application and architectural parameters that facilitate design space exploration of the application mapping problem.
- Mentor:
  - Will Plishker (plishker@eecs.berkeley.edu)

Implementing Tipi Irregular Datapaths on FPGA Fabrics

- Mentors: Andrew Mihal mihal@eecs.berkeley.edu
- and Scott Weber sjweber@eecs.berkeley.edu
- Research the FPGA as an implementation target for Tipi processing elements
- Take advantage of FPGA features:
  - Block RAMs
  - Fast carry chains
  - Etc.
- Improve the Tipi Verilog code generator to produce efficient implementations
- What kind of programmable machines are a good match for the capabilities of the FPGA?

Cycle-Accurate Multiprocessor Simulator Generation

- Mentor: Andrew Mihal mihal@eecs.berkeley.edu
- Given a heterogeneous multiprocessor of Tipi processing elements
- And the assembly code for each PE
- Produce a compiled-code simulator that is cycle-accurate
- Produce statistics/visualization of PE usage to guide design space exploration

MoCs for Network Processing

- Mentor: Andrew Mihal mihal@eecs.berkeley.edu

- Click is a good MoC for the *data plane* of packet routers
- What MoC is good for the *control plane*?
- How do you capture the interaction between these abstractions?
- Implement this MoC in the Cairn application development environment

Cairn Data Type Optimization

- Mentor: Andrew Mihal mihal@eecs.berkeley.edu

- Basic data type in the Cairn ADE is a bit vector
- Given a complete Cairn model, optimize the widths of these data types
  - Find unused bits
  - Apply a model transform to prune them
- Questions:
  - What part of the transform is model-of-computation independent?
  - What improvements are possible when we know the model of computation?

Compiler Math Library Synthesis

- Mentors: Andrew Mihal mihal@eecs.berkeley.edu
- and Matt Moskewicz moskewcz@eecs.berkeley.edu
- Canonical example of a math library: software floating point
- Generalize this concept to irregular Tipi processing elements and combinational operations
  - Machine has: 1-bit left-shift function unit
  - App wants: left-shift by N
  - Machine has: 32-bit adder
  - App wants: Add two 231-bit numbers
- Automatically generate the appropriate math library given an architectural model
- Apply transforms to application code
- Formulate as a logic synthesis problem?

Stream Processor Design Exploration with Tipi

Mentors: Scott Weber sjweber@eecs.berkeley.edu and Matthew Moskewicz moskewcz@eecs.berkeley.edu

- Tipi: toolset to quickly create and use PEs (Processing Elements)
  - Designer creates datapath only
  - Horizontal microcode based control is automatically generated
  - Also tools for cycle simulation, hardware generation
  - Some support for microcode sequence generation
- Project: Create stream processor with complex control
  - Achieve complex (layered) control by composition of multiple Tipi PEs
    - Layered control -> control flow constructs, smaller code size
  - Application: GPU, DSP type kernels/benchmarks or similar
  - Goal: perform functional testing, gather performance metrics
  - Work breakdown:
    - Design of processor
      - Layout of functional units, stream gates, register files, pipelining, …
      - Design space exploration: consider code size, performance, area
    - Fighting with toolset
      - adding some simple (fast to compute) estimates for area / speed
      - compare estimates with ‘real’ numbers from synthesis (which is slow)

Memory management for dataflow models of computation

- Dataflow models of computation expose a significant amount of concurrency in system design by reducing shared state between parts of a system. Dataflow components exchange messages with immutable values, in order to ensure that side effects do not have unintended consequences. Unfortunately, implementing this semantics in a single processor system often results in undesirable dynamic memory allocation for large components. In particular, algorithms such as the Fast Fourier Transform, cannot be easily implemented 'in-place'. There are some techniques known as region-based memory allocation that statically analyze a program (or dataflow) graph to understand when memory can be reused. Come up with a scheme for applying these techniques to dataflow models that address the desire to safely implement algorithms such as the FFT without allocating extra memory. Prototype in Ptolemy II.

- Mentor: Stephen Neuendorfer, neuendor@eecs.berkeley.edu

Reconfiguration constraints in higher-order dataflow models.

- Generic system-level models are often parameterized in order to increase reusability and configurability. Run-time parameter reconfiguration is a powerful tool for leveraging parameterization. However, while some parameters can be reconfigured at any time, other parameters must be reconfigured at specific execution points, or not reconfigured at all because of their meaning in a model. Currently, Ptolemy II includes a system for analyzing reconfiguration to determine if safety constraints are satfisfied which relies on a 'statically structured' model. However, many interesting models are not statically structured, such as those with mobile components that are 'plugged-in' at run-time. While we should not expect arbitrarily restructured models to be statically analyzable, statically structured models with mobile components can probably be analyzed under certain constraints. Figure out what those constraints should be and implement a combined static/runtime analysis system for quickly checking the reconfiguration constraints of a mobile component when it is plugged in.
- Mentor: Stephen Neuendorfer, neuendor@eecs.berkeley.edu