The Mitrion Platform

- Mitrion Processor
- Mitrion-C SDK
The Purpose Of A Processor Architecture

- A processor is an abstraction layer.
- Completely separates hardware from software.
- A machine, built in hardware that performs your program, written in software.
- Allows the programming language to ignore hardware design considerations.
The Philosophy Behind The Mitrion System

• Do not attempt to build hardware directly from a program

• Instead:
  – Insert a machine between the program and the hardware
  – Compile language into code for the machine
  – Adapt the machine according to code
  – Instantiate the adapted machine in an FPGA
A New Processor Architecture Specifically For FPGAs

Architecture design goal:
- High silicon utilization
- Take advantage of FPGA re-configurability

Goal achieved by:
- Allow processor to be massively parallel
- Allow processor to be fully adapted to algorithm
A Cluster On A Chip

- Fully distributed architecture
- Similar in design to parallel computers, but within a single chip
- High chip utilization through FPGA
  - Fine grain PEs
  - Each PE fully adapted to algorithm
  - Network topology specific for algorithm
- No Instruction Stream, instead Data Stream
Using The Mitrion Processor
Compiling A Mitrion Program

Mitrion-C
Source code

Compiler

Processor
Machine-code

Processor Configurator

Processor
Architectures

Processor
HW-Design
(VHDL IP Core)

FPGA

Mitron Development Toolkit
No User Hardware Implementation Or Integration

- Fixed HW interface
  - No integration considerations, just `main()`

- Algorithm is insulated from all timing and HW considerations
  - No timing requirements, combinatorial path balancing, pipe length balancing, data path synchronization, selection of architecture specific features, etc

- Runs at fixed clock-rate
The architecture allows automatic scaling with FPGA size and features. Just re-configure the processor for the new platform.

As FPGA sizes continue to grow, Mitrion will allow for easy upgrades to the next generation of performance available.

Using HDLs, a fundamental redesign will typically be needed for each hardware upgrade.
Mitrion-C

- ANSI-C can not be automatically parallelized.
  - All parallel C languages are super-subsets of C
- In general, one can think as usual. The language is intrinsically parallel
- The language is designed to preserve any parallelism in your program
- The language is designed to help you reveal parallelism in your program
Data-dependency graph

- Gives a full description of all parallelism in an algorithm
- Shows exactly what operations have to be done after each other, i.e. in sequence
- Mitrion-C allows a complete data-dependency graph to be created from the program
A C-family Language

- Basic syntax is the same as for other C-family languages
- Examples:
  - Blocks are surrounded by `{ }`
  - Assignment with `=`
  - Statements end with `;`
  - `if, for, while`
  - Most of the usual C operators
  - C-style comments (though nestable)
Types

• Basic types
  - `int/uint`: signed/unsigned integer
  - `boolean`: boolean value (true/false)
  - `float`: Floating point real value
  - `bits`: Bit vector format

• Free bit width
  - `int:24`: 24 bit signed integer
  - `uint:19`: 19 bit unsigned integer
  - `float:24.8`: IEEE-754 single precision float

• Collections
  - `int:24[100]`: Vector (indexable collection)
  - `int:14<100>`: List (no index)
Language constructs

Operators

if(a>b) ...

while(i<10) ...

for(i in <0..999>) ...

foreach (e in vector) ...

int:8 function(int:8 a) ...
A C-family Language

• Important differences
  – No pointers
  – No dynamic allocation
  – Static general recursion only
    • Though loop structures may be dynamic
Compiler, Simulator And Debugger
Contact Information

Mitrionics AB
Ideon Science Park
SE-223 70 Lund
www.mitrion.com

stefan.mohl@mitrion.com