Single-core vs. Multicore vs. Manycore Processor

- All have different purposes and different architectures
- Single-core is a microprocessor with a single core
- Multicore devices have 2-8 cores in them
- Manycore consists of thousands of cores
Manycore Processors

- A processor that consists of a large number of cores
- Designed for a high degree of parallel processing
- Able to handle thousands of threads simultaneously
Different Types of Instruction Streams

- **Single**
  - SISD: Single Instruction, Single Data
  - SIMD: Single Instruction, Multiple Data

- **Multiple**
  - MISD: Multiple Instruction, Single Data
  - MIMD: Multiple Instruction, Multiple Data
SIMD Parallel Processing

- GPUs use Single Instruction, Multiple Data (SIMD)
- A single instruction stream is applied to multiple separate data structures
- Threads execute the same instruction on different data
- Synchronous Programming
MIMD Processing

- Hammerblade uses Multiple Instruction, Multiple Data (MIMD)
- Asynchronous programming
  - Allows multiple things to happen concurrently
- More effective than SIMD in terms of performance
Hammerblade Architecture
Nodes

- Each node is a single System-on-Chip
- Multiple Nodes are interconnected
- Each node is architected from an array of tiles connected by a 2-D mesh network
Tile Groups

- Each tile contains a core
- Tile Group - subarray of tiles
  - Execute a single program
- Tile Groups are launched using Grids
  - Allow iterative invocations of Tile Groups

Architecture for the Manycore
Threads Overview in GPUS

- Threads grouped into thread blocks
- Grid is made of thread blocks
- In GPU, threads blocks are dispatched to the Streaming Multiprocessor (SM)
- Kernel Grid dispatched by GPU Unit
Execution Model of HammerBlade vs GPU
Basejump Manycore Accelerator Network

- 2D mesh network
- Single global memory space is shared by all nodes on the network
- Each tile is allocated a local address space
  - Private data memory in each core
- Global Memory space is addressed by the node's coordinates and a local address
  - \(<X\ cord, Y\ cord, local\ address>\)
Transaction Ordering

- Ordered Network
  - Sequential order
- XY dimension ordered routing
  - Travel along one dimension first, then the other
- Mesh nodes can route packets in 5 directions
  - P=0, S, N, E, W
Simulation

- Synopsis VCS and the RISC-V toolchain are used to simulate the architecture of the Hammerblade
  - Synopsis is a Verilog simulator
- Set up by cloning github repositories

<table>
<thead>
<tr>
<th>Clone and Compile the RISC-V Toolchain (and RTL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ git clone <a href="mailto:git@bitbucket.org">git@bitbucket.org</a>:taylor-bsg/bsg_manycore.git</td>
</tr>
<tr>
<td>$ git clone <a href="mailto:git@bitbucket.org">git@bitbucket.org</a>:taylor-bsg/bsg_ip_cores.git</td>
</tr>
<tr>
<td>$ cd bsg_manycore/software/riscv-tools</td>
</tr>
<tr>
<td>$ make checkout-all # Takes ~ 5 mins</td>
</tr>
<tr>
<td>$ make build-riscv-tools # Takes ~ 6 mins</td>
</tr>
</tbody>
</table>
Programming in CUDA-Lite

- CUDA-Lite allows Hammerblade to mimic the structure of a GPU
  - Easy transition from CUDA to CUDA-Lite
- C++
- Single Program, Multiple Data (SPDM) paradigm
  - Tasks are split up and run simultaneously on multiple processors
- CUDA known variables and its own hardware specific variables
- Example of CUDA known variables:
  - gridDim
  - blockDim
  - BlockIdx (position of block)
Sample Code

```c
/***************************************************************
 * Define tg_dim_x/y: number of tiles in each tile group
 * Calculate grid_dim_x/y: number of tile groups needed
 ***************************************************************/

hb_mc_dimension_t tg_dim = { .x = 0, .y = 0 };  
hb_mc_dimension_t grid_dim = { .x = 0, .y = 0 };  
if (!strcmp("v0", test_name)){  
    //strcmp is used to compare string arguments
    tg_dim = { .x = 1, .y = 1 };  //tile group dimensions
    grid_dim = { .x = 1, .y = 1 };  //grid dimensions
} else if (!strcmp("v1", test_name)){  
    tg_dim = { .x = 2, .y = 2 };  
    grid_dim = { .x = 1, .y = 1 };  
} else if (!strcmp("v2", test_name)){  
    tg_dim = { .x = 4, .y = 4 };  
    grid_dim = { .x = 1, .y = 1 };  
} else if (!strcmp("v3", test_name)){  
    tg_dim = { .x = 2, .y = 2 };  
    grid_dim = { .x = 2, .y = 2 };  
} else {  
    bsg_pr_test_err("Invalid version provided!\n");  
    return HB_MC_INVALID;
```
Project

- Goal: Learning how to program in CUDA_Lite
- Progress: Got simulation running successfully and working on coding the transpose of a Matrix to learn how to use the different functions and variables in CUDA-Lite
  - Comfortable with VIM
- Challenges: Initially did not have much experience with Linux, VIM, or programming in CUDA (programming in CUDA-Lite without knowing CUDA is challenging)
Future

- Work on more programs in CUDA-Lite throughout the rest of the quarter
- Will be continuing research with Marcus and Professor Wong over the Summer and throughout the school year
- Use the simulation to study different aspects of the Hammerblade
References


Thank you