Course Website
- [http://jatinga.iitg.ernet.in/~asahu/cs523/](http://jatinga.iitg.ernet.in/~asahu/cs523/)
- Course Contents
- Text and Reference Books
- All lecture slides
- Summery of each class with references
- Other information
  - Simulators, Benchmarks, Source Code,
  - Referred Papers, EBooks

Books: Text
- Some Recent Papers

Books: References
- P Pacheco "An Introduction to Parallel Programming", Morgan Kaufmann Publishers, 2011
- Introduction to Parallel Programming", Morgan Kaufmann Publishers, 2011

Data Parallel Architectures
- SIMD Processors
  - Multiple processing elements driven by a single instruction stream
- Vector Processors
  - Uni-processors with vector instructions
- Associative Processors
  - SIMD like processors with associative memory
- Systolic Arrays
  - Application specific VLSI structures

GPU Architecture
- Mix of Data Parallel and Function Parallel
  - Only limited functions can be run (Kepler Architecture)
  - Other wise it is data parallel
- How to execute a Program
  - Load data and function to GPU
  - Triggers GPU
  - Get back result
- GIGA thread scheduler
- Nvidia GTX 3072 cuda core, 8 thread/core
- Many shared data/global object/PC data
Systolic Arrays [H.T. Kung 1978]

Simplicity, Regularity, Concurrency, Communication

Example:

\[
\begin{bmatrix}
\begin{array}{cccc}
A_{i,0} & A_{i,1} & 0 & 0 \\
A_{i,2} & A_{i,3} & 0 & 0 \\
A_{i,4} & A_{i,5} & 0 & 0 \\
0 & 0 & A_{i,6} & 0 \\
0 & 0 & 0 & A_{i,7} \\
\end{array}
\end{bmatrix}
\begin{bmatrix}
B_{i,0} & B_{i,1} & 0 & 0 & 0 \\
B_{i,2} & B_{i,3} & 0 & 0 & 0 \\
B_{i,4} & B_{i,5} & 0 & 0 & 0 \\
0 & 0 & B_{i,6} & B_{i,7} & 0 \\
0 & 0 & 0 & B_{i,8} & B_{i,9} \\
\end{bmatrix}
\]

T=0

Why Process level Parallel Architectures?

MIMD Architectures

Design Space
- Extent of address space sharing
- Location of memory modules
- Uniformity of memory access
  - UMA, NUMA

Issues from user’s perspective
- Specification / Program design
  - explicit parallelism or
  - implicit parallelism + parallelizing compiler
- Partitioning / mapping to processors
- Scheduling / mapping to time instants
  - static or dynamic
- Communication and Synchronization

Parallel programming models
- CUDA
- Pthread
- MPI
- Cilk
- Concurrent control flow
- Functional or logic program
- Vector/array operations

With shared variables or message passing
Relationship between programming model and architecture?
Issues from architect’s perspective

• Coherence problem in shared memory with caches
  - Software Coherence: Synchronization and Lock
  - Lock: One person should access
  - Critical Section, Amdhal’s Law, Modified Law
• Efficient interconnection networks

Cache Coherence Problem

Multiple copies of data may exist
⇒ Problem of cache coherence
Options for coherence protocols
• What action is taken?
  - Invalidate or Update
  - Lazy protocol
• Which processors/caches communicate?
  - Snoopy (broadcast) or directory based
• Status of each block?
• Memory Consistency Problem: Pthread Level

Interconnection Networks

• Architectural Variations:
  - Topology
  - Direct or Indirect (through switches)
  - Static (fixed connections) or Dynamic (connections established as required)
  - Routing type: store and forward/worm hole
• Efficiency:
  - Delay
  - Bandwidth
  - Cost

Processor Design: Basic

• Microprocessor Vs Processor
  - Fetch, Decode, Register Access, Execute, Write Back
  - Single Cycle Design: All state in one cycle
  - Multi Cycle Design: Each state in one cycle Multiple T State, 8085 Microprocessor
• 8085 Microprocessor
• RISC Processor
• Simple MIPS Processor with 9 Instructions
• One Instruction Set Computer

To be discussed

Understanding a given Processor
(Example 8085)

Designing a Processor
(Example Tiny MIPS with 9 Instructions)
**8085 Microprocessor**

- 8 Bit CPU
- 3-6Mhz
- Simpler design: Single Cycle CPU
- ISA = Pre x86 design (Semi CISC)
- 40 Pin Dual line Package
- 16 bit address
- 6 registers: B, C, D, E, H, L
- Accumulator 8 bit

**8085 Microprocessor Architecture**

- Address Bus: Consists of 16 address lines: \( A_0 - A_{15} \)
  - Address locations: 0000 (hex) – FFFF (hex)
  - 64K (16-bit) of memory, each byte has 8 bits
- Data Bus: Consists of 8 data lines: \( D_0 - D_7 \)
  - Operates in bidirectional mode
  - The data bits are sent from the MPU to I/O & vice versa
  - Data range: 00 (hex) – FF (hex)
- Control Bus:
  - Consists of various lines carrying the control signals such as read / write enable, flag bits

**8085 Bus Structure**

- Registers:
  - Six general purpose 8-bit registers: B, C, D, E, H, L
  - Combined as register pairs to perform 16-bit operations: BC, DE, HL
  - Registers are programmable (load, move, etc.)
- Stack Pointer (SP)
- Accumulator & Flag Register
  - (Zero, Sign, Carry, Parity, AuxCarry)
- Program Counter (PC)
  - Contains the memory address (16 bits) of the instruction that will be executed in the next step.
**How instruction executed**

- All instructions (of a program) are stored in memory.
- To run a program, the individual instructions must be read from the memory in sequence, and executed.
  - Program counter puts the 16-bit memory address of the instruction on the address bus
  - Control unit sends the Memory Read Enable signal to access the memory
  - The 8-bit instruction stored in memory is placed on the data bus and transferred to the instruction decoder
  - Instruction is decoded and executed

**Copy/Mem/IO operation**

- **MVI R, 8 bit** // load immediate data
- **MOV R1, R2** // Example MOV B, A
- **MOV R M** // Copy to R from 0(HL Reg) Mem
- **MOV M R** // Copy from R to 0(HL Reg) Mem
- **LDA 16 bit** // load A from 0(16bit)
- **STA 16 bit** // Store A to 0(16bit)
- **LDAX Rp** // load A from 0(Rp), Rp=RegPair
- **STAX Rp** // Store A to 0(Rp)
- **LXI Rp 16bit** // load immediate to Rp
- **IN 8bit** // Accept data to A from port 0(8bit)
- **OUT 8 bit** // Send data of A to port 0(8bit)

**Arithmetic Operation**

- **ADD M** // Add A to A + 0(HL)
- **SUB R** // Sub A to A - B.reg
- **SUI 8bit** // Sub A = A - 8bit
- **SUB M** // Sub A= A - 0(HL)
- **INR R** // R = R+1
- **INR M** // 0(HL)=0(HL)+1
- **DCR R** // R = R-1
- **DCR M** // 0(HL)=0(HL)-1
- **INX Rp** // Rp=Rp+1
- **DCX Rp** // Rp=Rp-1

**Other Operations**

- **Logic operations**
  - **ANA R** ANI 8bit ANA M
  - **ORA, ORI, XRA, XRI**
  - **CMP R** // compare with R with ACC
  - **CPI 8bit** // compare B bit with ACC
- **Branch operations**
  - **JMP 16bit, CALL 16 bit**
  - **JZ 16bit, JNZ 16bit, JC 16bit, JNC 16 bit**
  - **RET**
- **Machine Control operations**
  - **HLT, NOP, POP, PUSH**

**Instruction to Micro-Instructions**

- **ADD B** // ACC = ACC + B
- **Things need to do to execute this**
  - Fetch Instruction from Memory and put into IR
  - Put Higher address to AB-15
  - Put Lower Address to AO-47
  - Get back data/instruction from memory to BUS
  - Activate BUS to IR gate to store data in IR
- **Addition need to be done in ALU**
  - Operands should be in TempR and ACC, Result will put to BUS
- **Execute**
  - Activate MUX to select Register B
  - Put value to B to BUS
  - Activate Gate of Temp Reg, so that data from BUS will go to TempR
  - Do the Operation ADD
- **Again result from BUS put to ACC**
  - Activate Gate for ALU, so that data from ALU come to BUS
  - Activate Gate for ACC, so that data from BUS go to ACC
Designing a Processor
(Example Tiny MIPS (9 Instructions))

Next Class.....