Architectures

George Bosilca
bosilca@eecs.utk.edu

What kind of parallel architecture
– Shared memory or distributed memory
– What kind of network? Which topology?
– What tools can we use?
– How the user level programs interact with the hardware?

Asynchronous vs. synchronous
• Allow overlapping communication computation
• Hiding latencies
• Additional cost for management
• No overlapping
• No additional cost
• All latencies included in the final time
Architectures

- Vector architecture
- Multi flow architectures
- Shared memory
- Distributed memory

Vector architecture

- Specialized on computation on arrays
- One instruction can be applied to several data from the same arrays (loops)
- Load/Store through vector registers

\[
\begin{align*}
\text{For } i = 0 \text{ to } 64 \text{ do} \\
a[i] &= c[i] + d[i] \\
b[i] &= a[i] \cdot f[i]
\end{align*}
\]

Correct sequential semantic: \(a[0], b[0], a[1], b[1], \ldots\)
Vector architecture

• Vector operation = pipeline
• Operation applied directly on the vector registers
• Instruction with strong semantics: one instruction applied on the whole vector register

Vector architecture - example

For i = 0 to 64 do
   c[i] = a[i] + b[i]

Cost:
- sequential: 64 * 4 cycles
- Vector: 63 + 4 cycles

Vector architecture - example

Total cost of the vectorized loop:

\[ T_{\text{init}} + 63 \quad (\text{LoadV A} \parallel \text{LoadV B}) \]
+ 4 + 63 \quad (\text{AddV})
+ \text{StoreV} + 63 \quad (\text{StoreV})
= 2 \times T_{\text{init}} + 193 \text{ cycles}

• Chaining = linking the pipelines together
  \[ T_{\text{init}} \times 4 \times T_{\text{init}} + 63 = 2 \times T_{\text{init}} + 67 \]
• How about the memory access?
Vector architecture

- Several banks to sustain the high bandwidth
- Components “state of the art” from the technology point of view
- First vector processor: Cray1 (12 vector units + chain MAC)
- Vector multiprocessor: CrayT90 32 procs (1024 memory banks)
- Vendors: SGI/Cray, Fujitsu, NEC, Hitachi

Multiflow architecture

- Hyper-Threading it’s a new idea?
- Basic idea: do something else while waiting for memory latency or how to deal with cache misses and data dependencies
- When to switch?
  - On every load operation
  - On cache miss
  - On every instruction (no cache locality)
  - On instruction block
- How to switch?
  - Context switch too expensive: thread approach

Multi flow architectures
Multiflow architecture

- TERA MultiThreaded Architecture
  - Heavily alternate multi threaded
  - No caches (direct access to the memory)
  - Change the flow after each load
  - One memory access ~ 100 cycles
  - 16 protection domains (register, status, CP)
    sharing 128 threads by processor ...
- Up to 256 processors !!!

Shared Memory

- Each processor have his own cache (one or several levels)
- They can access the whole shared memory
- How about consistency ? How can a data be on several processors in same time.

Shared memory

- Allow fine grain resources sharing
- Communications are implicit in load/store on shared addresses
- Synchronization is performed by operations on shared addresses
Shared memory - ShMem

Uniform Memory Access
Non Uniform Memory Access
Cache Coherent – Non Uniform Memory Access
Cache Only Memory Access

ShMem – Shared Cache

- Alliant FX-8 (8x68020 512KB); Encore & Sequent (2xN32032)
- Advantages
  - Identical to uni processor systems
  - Only one copy of any cached block
    - Smaller storage size
  - Fine-grain sharing
  - Potential for positive interference
  - Can share data within a line without “ping-pong”
  - No false sharing for long data

ShMem – Shared Cache

- Drawbacks
  - Sharing cache bandwidth between processors
  - Increase latencies for ALL accesses
  - Potential for negative interference
    - One proc flush data for another

Many L2 caches are shared today!
ShMem – Bus based approach

- Cheap, usual components => dominate the market
- Attractive as servers and convenience parallel computers
  - Fine grain resource sharing
  - Uniform access using Load/Store
  - Automatic data movement and coherent cache replication
  - Cheap and powerful extension
- **Sequential access**

ShMem - caches

- Caches become critical
  - Reduce average latency (replication closer to proc)
  - Reduce average bandwidth
  - Manage consistency
- Data goes from producer to consumer to memory
- Many processors can share the memory efficiently
- Concomitant read accesses to the same location

ShMem – cache coherence example
ShMem – cache coherence example

A = 1
Shared Memory

A = 1
L2
A = 1
L2

Processes have different values for A

Intolerable from the programmer point of view
ShMeme – cache coherence example2

\[ A = 1 \]

\[ A = 0 \]

\[ A = 10 \]

Still intolerable.

ShMeme – Caches and coherence

- Caches play an important role in all cases
  - Reduce average access time
  - Reduce bandwidth on shared interconnection
- Private caches create a coherence problem
  - Copies of the same data on several caches
  - A write may not become visible to other processors
- Solutions
  - Another memory organization
  - Detect and take actions to avoid this problem

ShMeme – Cache coherence protocols

- 2 main categories:
  - Invalidation
    - Any write preceded by a block invalidation for all others processors
  - Broadcast (diffusion)
    - Before any write all caches containing the same data will be invalidated
ShMem – Snoop protocols

• Snooping (or monitoring) the bus
• set of states
• state-transition diagram
• actions

![Cache line representation](image)

Ordering (memory consistency)

/*Assume initial values of A and B are 0*/

(1a) A = 1;
(2a) print B;
(1b) B = 2;
(2b) print A;

- What’s the intuition?
  - Whatever it is, we need an ordering model for clear semantics
    • across different locations as well
    • so programmers can reason about what results are possible

Lamport give the definition of a multi processor sequentially consistent:
- The result of all executions is the same as the sequential atomic execution of each instruction
- The operations of each processor appear in the sequential order as specified by the program.

Ordering (memory consistency)

/*Assume initial values of A and B are 0*/

(1a) A = 1;
(2a) print B;
(1b) B = 2;
(2b) print A;

- What matters is order in which operations appear to execute, not the chronological order of events
- Possible outcomes for (A,B): (0,0), (1,0), (1,2)
- What about (0,2)?
  - program order => 1a->1b and 2a->2b
  - A = 0 implies 2b->1a, which implies 2a->1b
  - B = 2 implies 1b->2a, which leads to a contradiction
- What is actual execution 1b->1a->2b->2a?
  - appears just like 1a->1b->2a->2b as visible from results
  - actual execution 1b->2a->2b->1a is not
ShMem – Cache & Directories

- **Centralized**
  - Keep the state and the tag of each block of data for all caches
  - For each memory access the controller check the tag and state of all blocks

- **Distributed**
  - Each processor keep a directory for the data in his cache
  - Update this data depending on the information on the bus.

- Strongly depend on the interconnection network. (broadcast)