Contents

Topics

1. Locality and Arrays
2. Taxonomy
3. Vector Programming (Vector)
4. Atomic Instructions and Locks
5. Open MP
6. Very Long Instruction Word (VLIW)
7. Multithreading Architectures (Multithreaded Architecture)
8. MPI (Distributed Memory Parallel Machine)
9. Dynamic Instruction Scheduling (Out of Order Superscalar)

Processors & Architectures

- Multicore
- GPUs
- Vector
- VLIW
- Multithreaded Architecture
- Distributed Memory Parallel Machine
- Out of Order Superscalar

Tables

Oisin's table.

Core Type	Single Thread	Chip Area	Power	Coding	Compiler Demands	General Purpose
OoO Superscalar	4	1	1	4	4	4
Multicore Scalar	1	3	3	2	2	2
VLIW	3	3	3	3/1*	1	2
Vector	N/A	3	4	3/1*	1	2
GPU	-3	4	4	1.5	1	1

- compiler/programmar

Summary

Dario's notes.

Architectures

- VLIW (Compiler optimisation), Vector SIMD, MIMD (threaded), OoO Superscalar (hardware optimiser)
- Vector saves energy on Instruction Fetch, very efficient for array operations
- MIMD avoids stalling by switching tasks, using multiple register sets, task switched strategies, OoO to mix instructions from across threads

- Multicore born of power wall, ILP (instruction level parallelism) wall, memory wall, can use shared or distributed memory
  - Easy programming, hard cache, bad hardware scaling
- Multiprocessor also an option with multiprocessing, NUMA (non uniform memory access) allows sharing and efficient individual work
- Distributed memory multi PC, explicitly move data, hardware is very simple, configuration of layout a bit more complicated, also clusters

Low Power Processing

- Power and energy very important on mobile
- Parallelism > high frequency as it uses less power (low voltage)

Locking

- Software level locking
  - Atomic compare & swap
  - Atomic increment (can be used for ticket lock)
- Hardware level locking
  - Single bus
- Algorithms
  - Spin lock
  - FCFS sleep lock
  - Hybrid (futex)

  OoO have to handle conflicts

OMP

Reduction

- Reduction to one variable
- **Useful for the exam**

```
int sum

#pragma omp parallel for reduction (+:sum)
{
    for(int i = 0; i < n; i++) {
```

```
        sum += a[i];
    }
}
```

- Uses arithmetic or boolean operators

Scheduling

- Not all iterations created equal
- Static: all equal
- Dynamica: iterations chunked and queued
- Guided: chunks shrink over time

Conditional

- Stick an if at the end

```
merge_sort(int* arr, int size, int start) {
    if(size > 2) {
        #pragma omp parallel sections if (sie > 128)
        {
            ...
        }
    }
    ...
}
```

- Careful with shared variables

SIMD

```
#pragma omp simd safelen(4)
```

- You can vectorise access as long as you don't modify things less than 4 items apart

VLIW

- Originally for control signals
- Now common, IA64, Myriad2, PS3 (IBM Cell)

- Compiler decides what's issued and executed simultaneously to avoid stalls and data hazards
- Issue packets of multiple instructions to be executed at once
- If one can't be used, replaced with `noop`
- Apply loop unrolling to avoid hazards and overhead + register renaming
- List scheduling
  - Build dependency graph
  - Start at deepest dependency with no incoming edges
- Speculation
  - Assume store doesn't affect following load
  - Guess at outcome of conditional branch
  - Need hardware/software to check results and fix incorrect ones
  - Prediction (conditional instructions) elimnates branches
- A lot of pressure on compilers
  - Usual optimisations
  - Dependence analysis
  - Compile time prediction
  - Scheduling and register allocation
- – Less power, better scaling (instruction per bundle)
- – Lock step, big delay on hazard, code bloat with `noop`s, fixup loop unrolling

Multithreading

- Hardware per thread (register file, PC, buffers)
- Shared (larger) hardware, TLB (translate lookaside buffer), cache, branch prediction, memory (virtual)
- Switch strategy
  - Round robin each instruction (hides stalls)
  - Switch on L2 cache miss (pipeline startup means this is slow anyway)
- Mix with ILP

MPI

- Message passing interface
- For clusters, MIMD setups, distributed memory with own address spaces
- Hides architecture details, message passing, buffering
- Provides message management (send, broadcast, secure)
- Scoping for messages (communicator)
- Messages sent to/received from ranks (identifier within a communicator)
- Asyncronous versions with wait exist
- Barrier and broadcast for sync, etc.
- Reduce for inverse of broadcast
- Scatter/gather to distribute & get data

Basic Vector Instructions

```
__m128 _mm_load_ps(float* src);
__m128 _mm_loadu_ps(float* src);
__m128 _mm_load1_ps(float* src);
__m128 _mm_setr_ps(float a, float b, float c, float d);
__m128 _mm_set1_ps(float a);

_mm_store_ps(float* dst, __m128 value);
_mm_storeu_ps(float* dst, __m128 value);

__m128 _mm_add_ps(__m128 a, __m128 b);
__m128 _mm_sub_ps(__m128 a, __m128 b);
__m128 _mm_mul_ps(__m128 a, __m128 b);
__m128 _mm_div_ps(__m128 a, __m128 b);
```