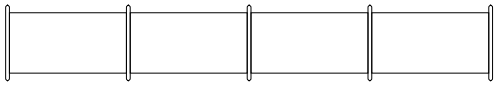
Pipelined Computers



These machines can execute in parallel the various subfunctions of an operation in a manner similar to a factory assembly line.

Pipelined Vector Computers

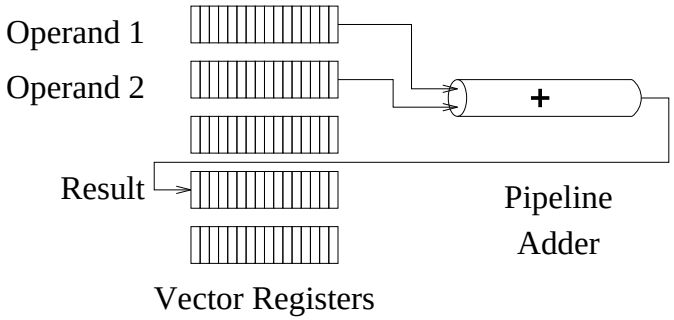
Although similar to pipelined scalar or superscalar computers, a pipelined vector computer is distinguished by its ability to manipulate vector data structures.

Note that pipelined scalar and superscalar computers are not considered to be parallel computers although they use concurrency to increase performance.

Vector Manipulation

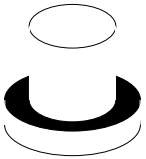
A single vector instruction may initiate an operation (e.g. FP addition) on two vectors stored in vector registers to produce a result which may be stored in a new vector register.

```
V3(0:15) = V0(0:15) + V1(0:15)
```



- In each cycle a new pair of scalar operands will be presented to the pipeline.
- After an initial delay (determined by the pipeline length), results will be produced at a rate of one per cycle.

Cray X-MP (1982)



Main processor unit (4 processor machine):

- Performance: 200 Mflops/s (approx.)
- Height: 2m
- Diameter: 1.5m
- Weight: 5 Mg

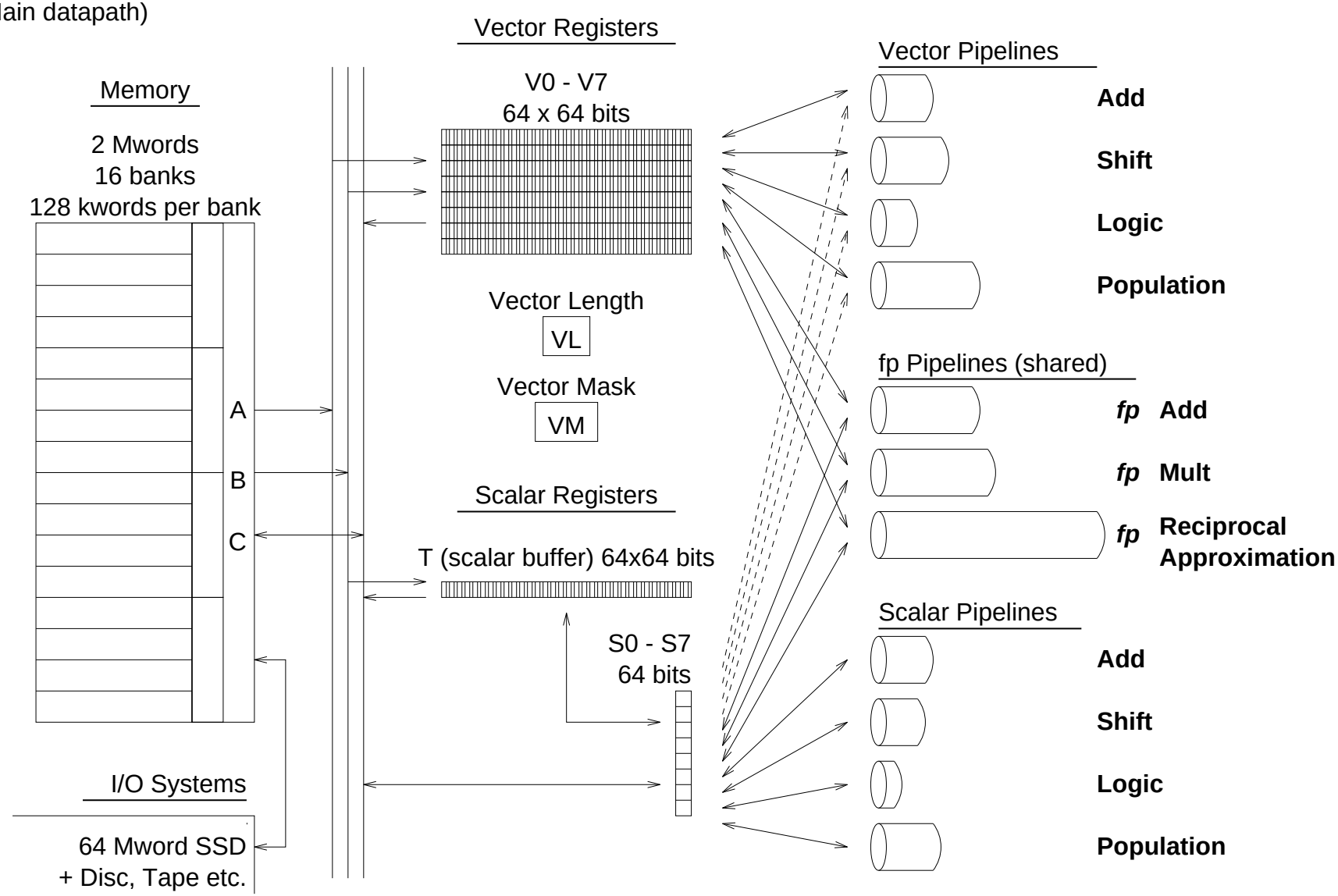
Support:

- 175 kVA generator
- Freon Cooling

We shall consider the single processor CRAY X-MP/12.

Cray X-MP/12

(Main datapath)



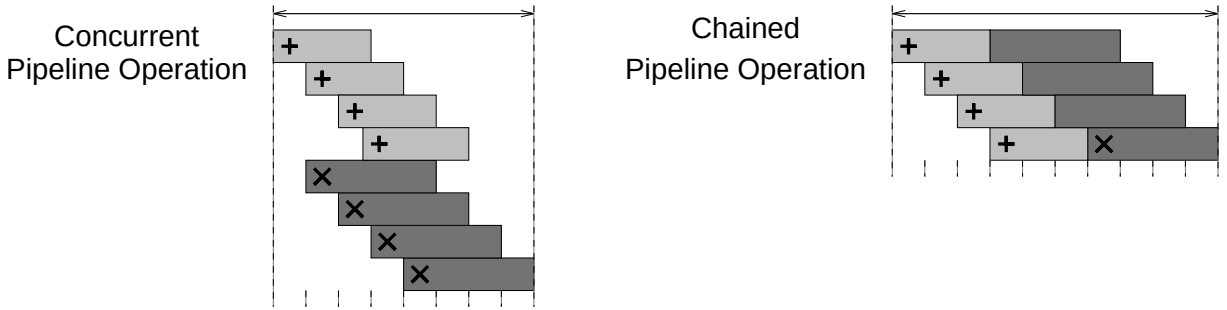
Uniprocessor Cray X-MP

- A Supercomputer - *Speed at all costs*
- 8 vector registers
  - 64 words per vector
  - 64 bits per word.
- Multiple pipes
  - Vector pipes (inc. shared FP pipes)

```
V5(0:31) = V1(0:31) * V3(0:31)
V5(0:63) = V2(0:63) * S3
```

    VL register determines vector length.
  - Scalar pipes optimized for scalar only operation.
- Memory
  - 4 ports - 16 banks for maximum bandwidth.

Concurrent & Chained Operations



- Concurrent Pipeline Operation. *independent operations*

  Where different registers and different pipes are used, a new vector instruction may be issued every cycle.

```
V0(0:3) = V1(0:3) + V2(0:3)
V3(0:3) = V4(0:3) * V5(0:3)
```

- Chained Pipeline Operation. *dependent operations*

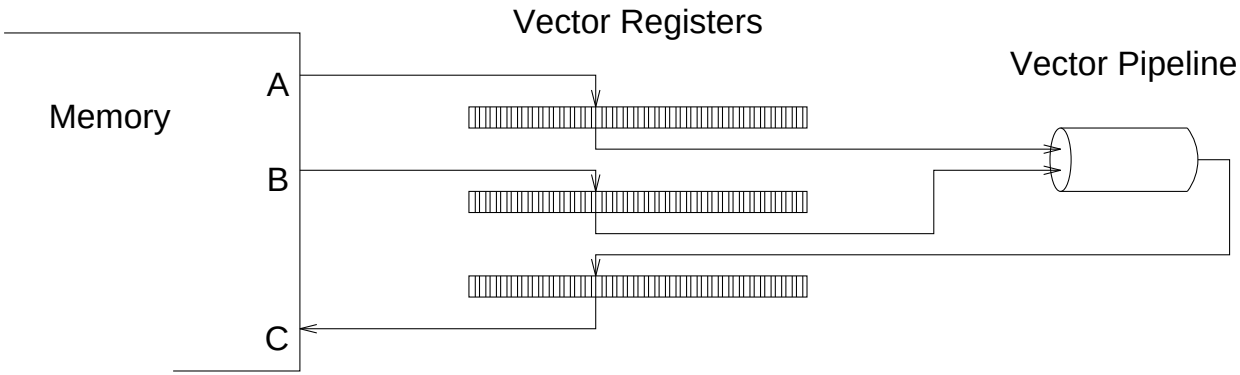
  Where the second vector instruction uses the result of the first, the operations are chained together.

```
V0(0:3) = V1(0:3) + V2(0:3)
V3(0:3) = V0(0:3) * V5(0:3)
```

CPU Power *vs* Memory Bandwidth

Memory access is also pipelined giving one access per cycle on ports A, B & C in the absence of memory bank conflicts.

We can chain memory access and pipeline operations:[1]



Where more than one pipeline is concurrently active, the architecture relies on the use of local register values to maintain the power bandwidth balance.

[1] A conflict on memory read will result in a *bubble* in the pipe, a conflict on write will result in buffering in a vector register.

Benefits of Vector Processors

For scientific applications exhibiting a high degree of data parallelism, vector processors offer significant benefits over superscalar processors.

- Fewer instructions to be executed

  Superscalar processors suffer from the *Flynn Bottleneck* at the instruction fetch and decode stage. With many fewer instructions to decode in a vector machine it is much easier to keep the pipelines busy.

- Hardware controlled loops

  Short loops as might be generated by the coding of vector arithmetic cause serious performance degradation in superscalar machines. With hardware controlled loops this problem does not occur.

- No stage iteration

  Stage iteration as might be found in an integer multiplication pipe on a superscalar machine would seriously impede vector operation and is not used in vector machines.

- # Pipeline performance

  Consider an l stage pipeline performing an operation on a vector of length n. The time taken for each stage is τ (normally 1 clock cycle). There is also an additional set-up time $s\tau$ to prepare the pipe for calculation.

  Thus the time taken to produce the first result is

  $$t_1 = [s + l]\tau$$

  then at a rate of one result per cycle giving

  $$t_{pipe} = [s + l + (n - 1)]\tau$$

  or

  $$t_{pipe} = t_0 + n\tau$$

  where t_0 is the perceived start-up time ($[s + l - 1]\tau$).

Performance Measurement

- ## Asymptotic Performance, r_∞

  The asymptotic rate is a useful measure of parallel computer performance, it indicates the rate of operations for a very large vector.

  $$r_\infty \equiv \lim_{n \to \infty} n/t$$

  Thus for our simple pipeline

  $$r_{\infty pipe} = \tau^{-1}$$

- ## Half–performance Length, $n_{1/2}$

  The performance of our computer with shorter vectors is indicated by the half–performance length; the length of vector that will result in half the asymptotic performance.

  $$r_{n_{1/2}} \equiv r_\infty/2$$

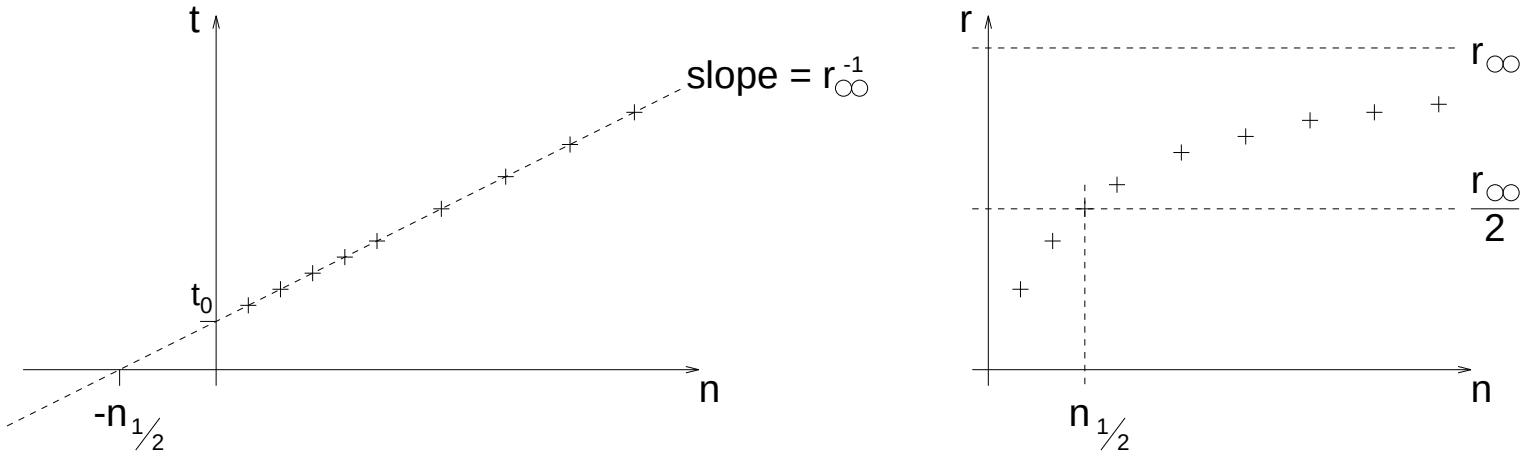
  Thus for our simple pipeline

  $$n_{1/2} = t_0/\tau$$

Performance Measurement

The performance of our generic machine is fully defined by r_∞ and $n_{1/2}$.

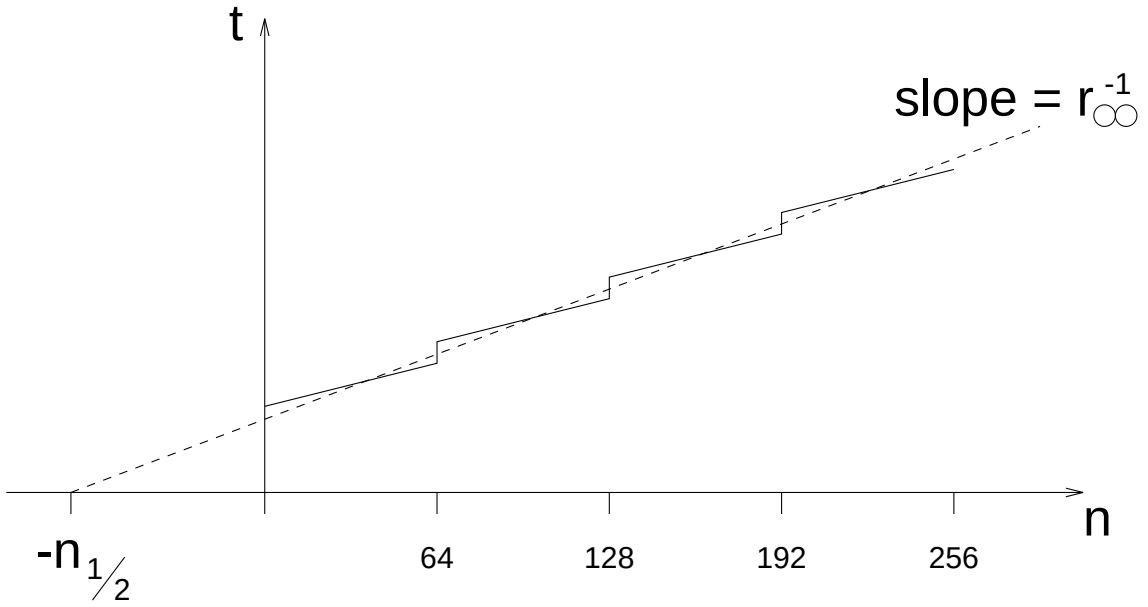
$$t = \frac{n_{1/2} + n}{r_\infty}$$



For most real machines this will not be the case, but it is usually possible to approximate their behaviour to that of a generic machine.

Performance Measurement

In the case of the Cray X-MP the performance will be modified due to the maximum vector length (64). This will result in an additional overhead every time a new internal vector instruction is started.



The solid line indicates the actual performance while the dashed line indicates the equivalent generic performance, a best fit line giving values for r_∞ and $n_{1/2}$.