VECTORIZATION, OPTIMIZATION AND
SUPERCOMPUTER ARCHITECTURE

Willi Schönauer and Hartmut Häfner
Rechenzentrum der Universität Karlsruhe, Postfach 6980, D-7500 Karlsruhe 1, F.R.G.

ABSTRACT

The basic architecture of vector computers is discussed in general and in more detail for some relevant vector computers. The relationship between arithmetic operations, memory bandwidth and realistic performance is pointed out. The key for an optimal vector computer algorithm is the data structure. Some basic problems are discussed as examples for the selection of optimal data structures and algorithms. The basic principles and rules are extracted from these examples. Finally the design of software for vector computers independent of a special architecture is briefly mentioned.

1. THE ARCHITECTURE OF VECTOR COMPUTERS

Large scale computations, i.e. number crunching, always means a series production of numbers. So we can use the "assembly line" principle for this series production: the arithmetic pipeline or (vector) pipe. In Figure 1.1 a 5-stage addition pipeline is depicted: 5 operand pairs are in different stages of execution in the pipeline, thus the operations must be independent. The following Fortran loop

\[ c_i = a_i + b_i \]

Figure 1.1. 5-stage addition pipeline.
\begin{align*}
\text{do } & 10 \text{ i = 1,1000} \\
& \text{d(i) = a(i) + b(i) * c(i)} \\
10 & \text{ continue}
\end{align*}

results in a compound vector operation in which the addition and multiplication pipes work in parallel. This is called supervector speed because two results are produced in each cycle time $\tau$. But before the operation can start, the addresses of the operands and the vector length must be loaded to certain registers and (eventually) the operands must be loaded into vector registers. Then the pipes must be filled, the result address must be loaded to a register and the result stored from a vector register back to memory.

This situation leads to the following tuning formula for the processing of a vector of $n$ elements:

$$
t_n = \tau_{\text{eff}} \left( n + n_{1/2,\text{eff}} \right).
$$

The effective cycle time $\tau_{\text{eff}}$ is determined by the bottlenecks in the processing of the data and is (for a single pipeline) usually much larger than the hardware cycle time $\tau$. The value $n_{1/2,\text{eff}}$ is Hockney's "half performance length" ([1], see also [2]), that represents the wasted (fictitious) operations for the startup of the vector operation under consideration. If $n = n_{1/2,\text{eff}}$, half the time is useful and half the time is wasted, thus only half of the effective peak performance is obtained. Another effect of $n_{1/2,\text{eff}}$ is, that if $n$ is below some breakeven length $n_\omega$ it is cheaper to execute the $n$ operations in scalar mode. Values $n_{1/2,\text{eff}}$ range between 20 (CRAY-1) and 200 (Fujitsu VP200) for simple operations, values of $n_\omega$ are in the range of 4 to 10. The consequence of $n_{1/2,\text{eff}}$ for the programmer is the following

Rule: Make vectors as long as possible!

Figure 1.2. Overview of the main components of a vector computer.