High-Performance Computing

COMP 40730

Alexey Lastovetsky

(B2.06, alexey.lastovetsky@ucd.ie)

Course Subject

- Parallel computing technologies
  - Aimed at *acceleration of solving a single problem on available computer hardware*
  - The course focuses on *software tools for developing parallel applications*: optimising compilers, parallel languages, parallel libraries
- Logical view vs. cooking book
  - *Ideas, motivations, models* rather than *technical details*
  - We will follow the evolution of hardware architecture
  - Carefully selected programming systems

Course Outline

- Vector and Superscalar Processors
  - Programming and performance models
  - Optimising compilers
  - Array-based languages (Fortran 90, C[])
  - Array libraries (**BLAS**)
- Shared-Memory Multiprocessors
  - Programming and performance models
  - Parallel languages (Fortran 95, **OpenMP**)
  - Threads libraries (**Pthreads**)

Course Outline (ctd)

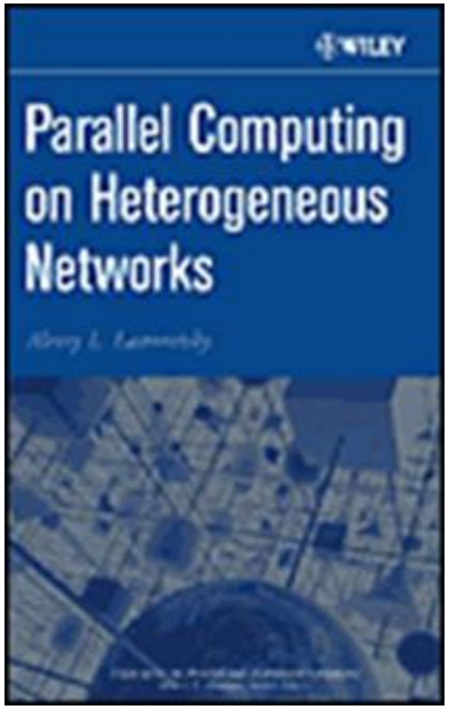
- Distributed-memory multiprocessors
  - Programming and performance models
  - Parallel languages (High Performance Fortran)
  - Message passing libraries (**MPI**)
- Networks of Computers
  - Hardware and programming issues
  - Parallel computing (HeteroMPI, mpC)
  - High-performance Grid computing (NetSolve/GridSolve)

Course Output

- You will be able to orient yourselves in parallel computing technologies
  - Mainly theoretical course
  - Some practical assignments (OpenMP, MPI)
    - A cluster of four 2-processor workstations
    - School network of computers

References

- Reading materials
  - A.Lastovetsky. *Parallel Computing on Heterogeneous Networks*. John Wiley & Sons, 423 pp, June 2003, ISBN: 0-471-22982-2.

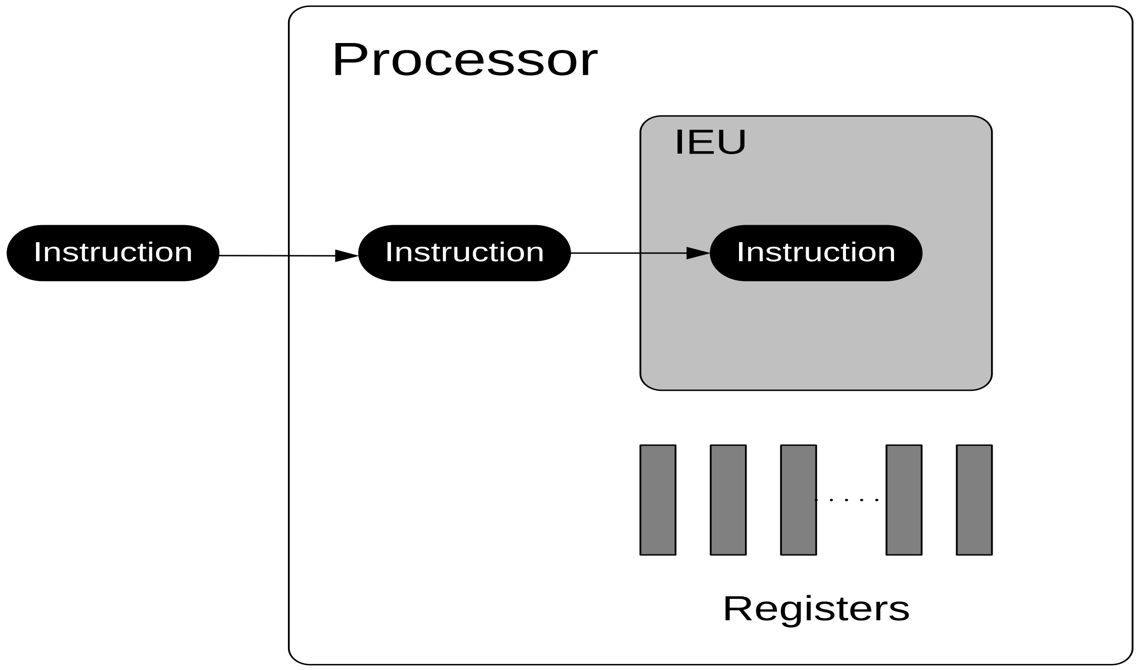


- The course website (lecture notes, assignments, etc.) https://csmoodle.ucd.ie/moodle/course/view.php?id=491

Programming Systems for Serial Scalar Processors

Serial Scalar Processor

- Starting-point of evolution of parallel architectures
  - Single control flow with serially executed instructions operating on scalar operands



Serial Scalar Processor (ctd)

- One instruction execution unit (IEU)
- Next instruction can be only started after the execution of the previous instruction in the flow has been terminated
- A relatively small number of special instructions for data transfer between main memory and registers
- Most of instructions take operands from and put results to scalar registers
- The total time of program execution is equal to the sum of execution times of its instructions
- The performance of that architecture is determined by the clock rate

Basic Program Properties

- A lot of languages and tools have been designed for programming SSPs
- C and Fortran are the most popular among professionals
- What is so special in these two languages?
  - They support and facilitate the development of software having certain properties considered basic and necessary by most professionals

Basic Program Properties (ctd)

- Fortran is used mostly for scientific programming.
- C is more general-purpose and widely used for system programming
  - C can be used for programming in the Fortran-like style
  - Fortran 77 can be converted into C (GNU Fortran 77 compiler is implemented as such a convertor).

- The same program properties make Fortran attractive for scientific programming, and C for general-purpose and, especially, for system programming.

Efficiency

- C allows one to develop highly efficient software.
- C reflects the SSP architecture with completeness resulting in programs of the assembler’s efficiency
  - Machine-oriented data types (short, char, unsigned, etc.)
  - Indirect addressing and address arithmetics (arrays, pointers and their correlation)
  - Other machine-level notions (increment/decrement operators, the sizeof operator, cast operators, bit-fields, bitwise operators, compound assignments, registers, etc.)
- C supports *efficient* programming SSPs

Portability

- C is standardised as ANSI C
  - All good C compilers support ANSI C
  - You can develop a C program running properly on any SSP
- C supports *portable* programming SSPs
- Portability of C applications
  - Portability of source code
  - Portability of libraries
    - Higher level of portability for SSPs running the same OS, or OSs of the same family (Unix)
  - GNU C compiler

Modularity

- C allows a programmer to develop a program unit that can be separately compiled and correctly used by others without knowledge its source code

- C supports *modular* programming

- Packages and libraries can be only developed with tools supporting modular programming

Easy to Use

- A clear and *easy-to-use* programming model ensures reliable programming
- Modularity and easy-to-use programming model facilitate the development of really complex and useful applications

- The C language design
  - Provides a balance between efficiency and lucidity
  - Combines lucidity and expressiveness

Portable efficiency

- Portably efficient C application
  - A portable C application, which runs efficiently on any SSP having a high-quality C compiler and efficiently implemented libraries
- C
  - Reflects all the main features of each SSP affecting program efficiency
  - Hides peculiarities having no analogs in other SSPs (peculiarities of register storage, details of stack implementation, details of instruction sets, etc.)
- C supports *portably efficient* programming SSPs

Basic Program Properties (ctd)

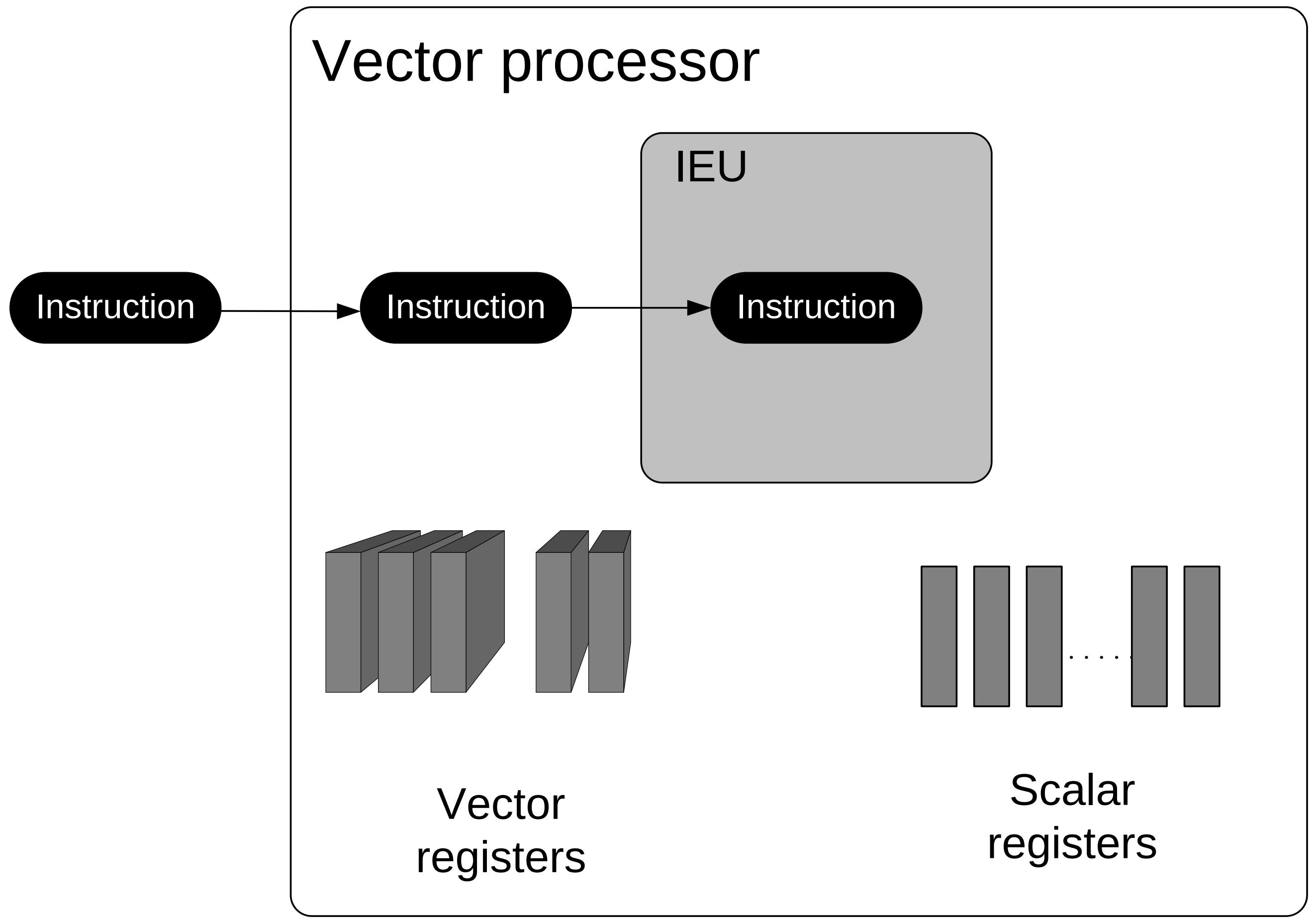
- There many other properties important for different kinds of software (fault tolerance, testability, etc.)
- 5 primary properties
  - *Efficiency*
  - *Portability*
  - *Modularity*
  - *Easy-to-use programming model*
  - *Portable efficiency*
- We will assess parallel programming systems mainly using the 5 basic program properties

Vector and Superscalar Processors

Vector Processor

- Vector processor
  - Provides single control flow with serially executed instructions operating on both vector and scalar operands
  - Parallelism of this architecture is at the instruction level
  - Like SSP
    - VP has only one IEU
    - The IEU does not begin executing next instruction until the execution of the current one has completed
  - Unlike SSP
    - Instructions can operate on both scalar and vector operands
    - Vector operand is an ordered set of scalars located on a vector register

Vector Processor (ctd)



Vector Processor (ctd)

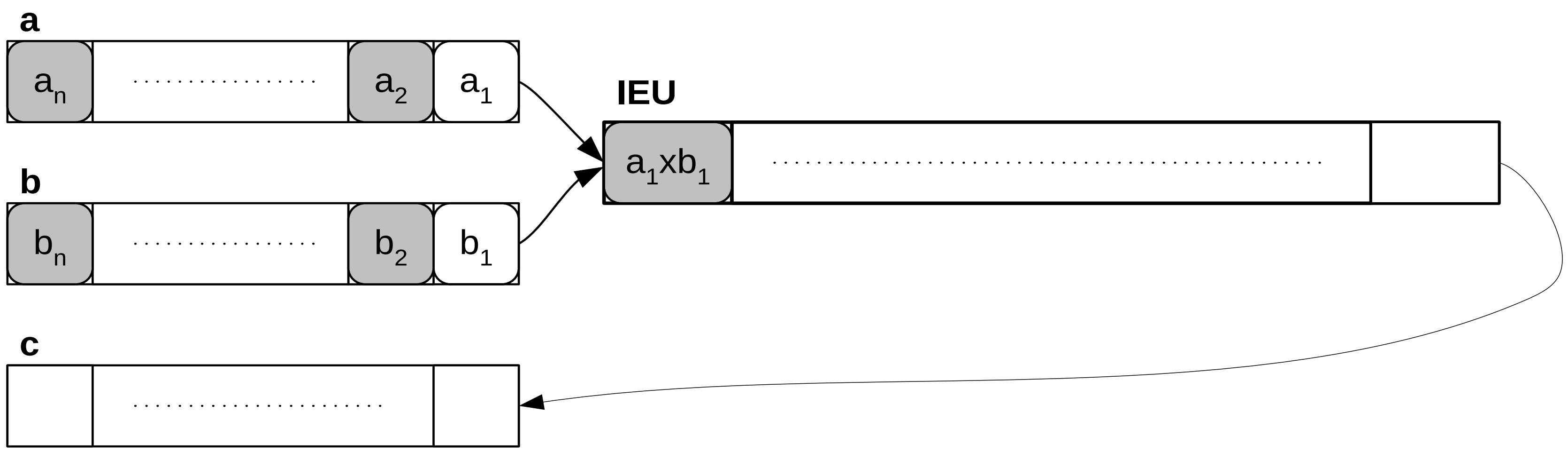
- A number of different implementations
  - ILLIAC-IV, STAR-100, Cyber-205, Fujitsu VP 200, ATC
- Cray-1 is probably the most elegant vector computer
  - Designed by Seymour Cray in 1976
  - Its processor employs data pipeline to execute vector instructions
  - 80 MFLOPS

Cray-1 Vector Processor

- Consider the execution of a single vector instruction that
  - performs multiplication of two vector operands
  - takes operands from vector registers **a** and **b** and puts the result on vector register **c** so that $c_i = a_i \times b_i \ \ (i=1,\dots,n)$
- This instruction is executed by a pipelined unit able to multiply scalars
  - the multiplication of two scalars is partitioned into m stages
  - the unit can simultaneously perform different stages for different pairs of scalar elements of the vector operands

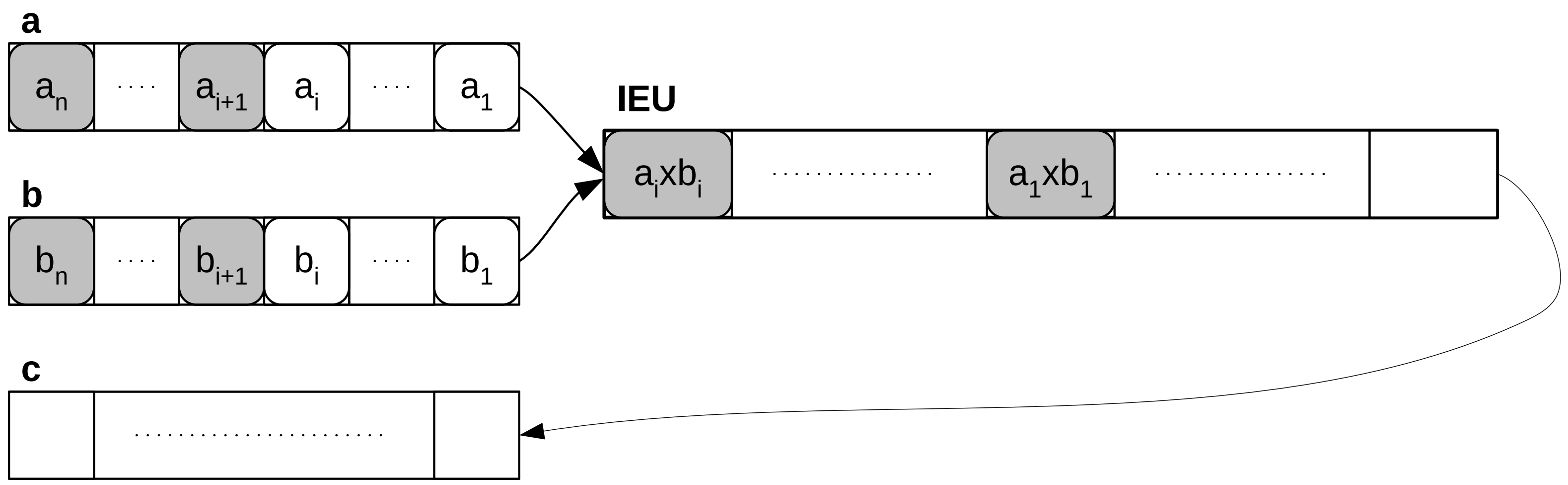
Cray-1 Vector Processor (ctd)

- At the first step, the unit performs stage 1 of the multiplication of elements a_1 and b_1



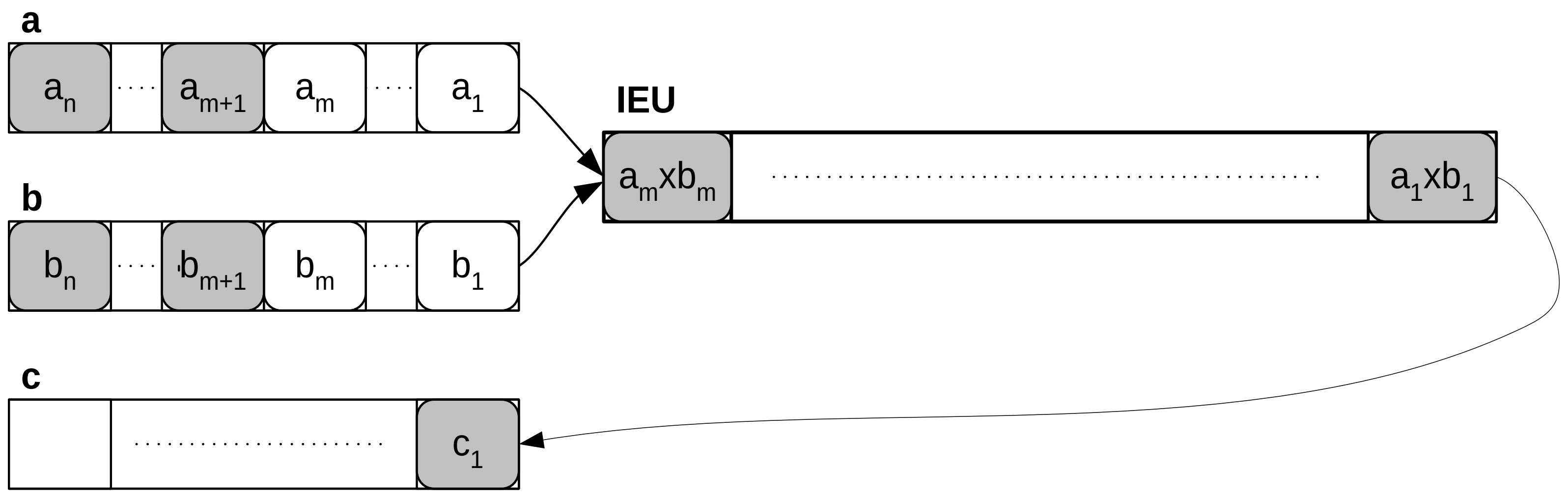
Cray-1 Vector Processor (ctd)

- The *i*-th step ($i=2,\dots,m-1$)



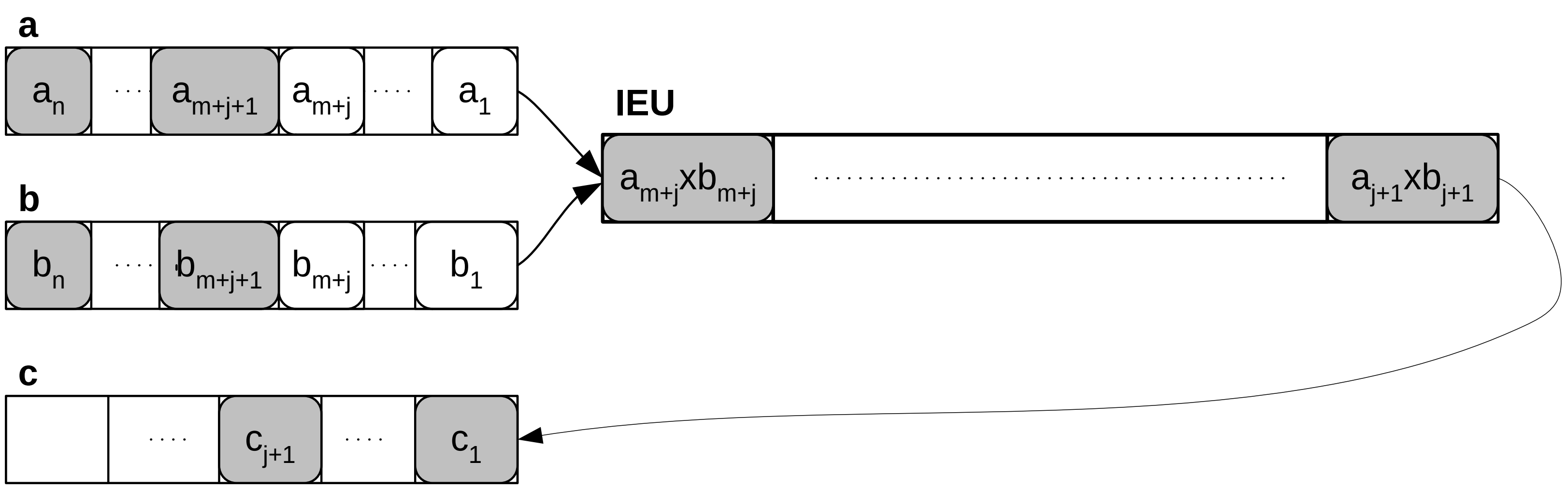
Cray-1 Vector Processor (ctd)

- The *m*-th step



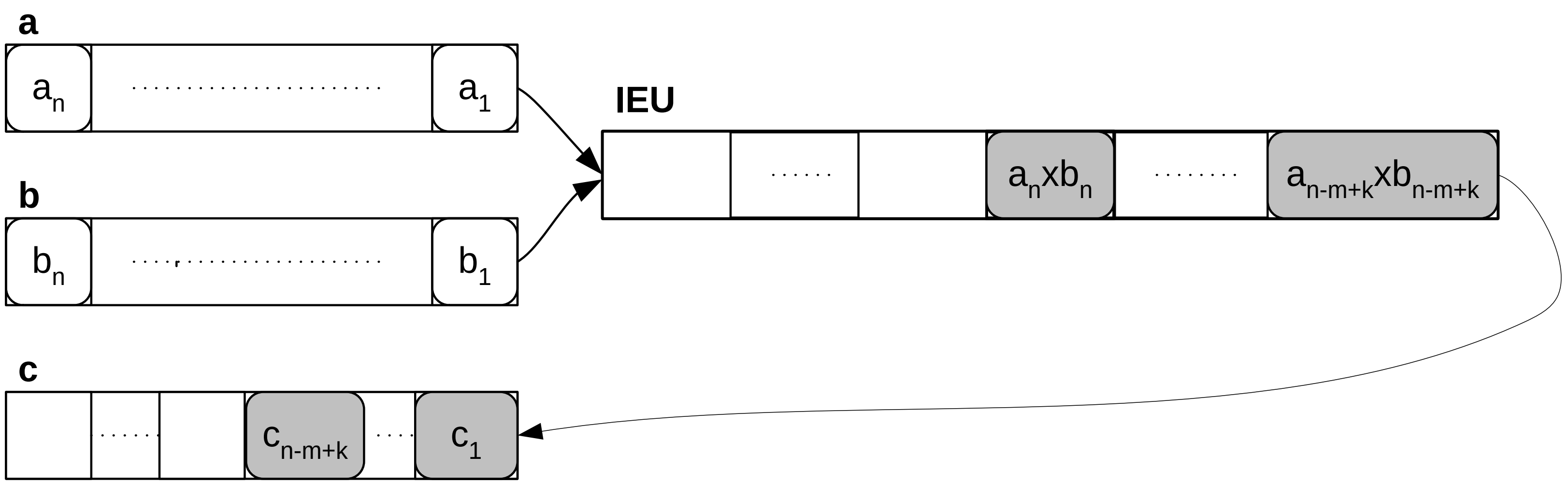
Cray-1 Vector Processor (ctd)

- Step *m+j* (*j*=1,…,*n-m*)



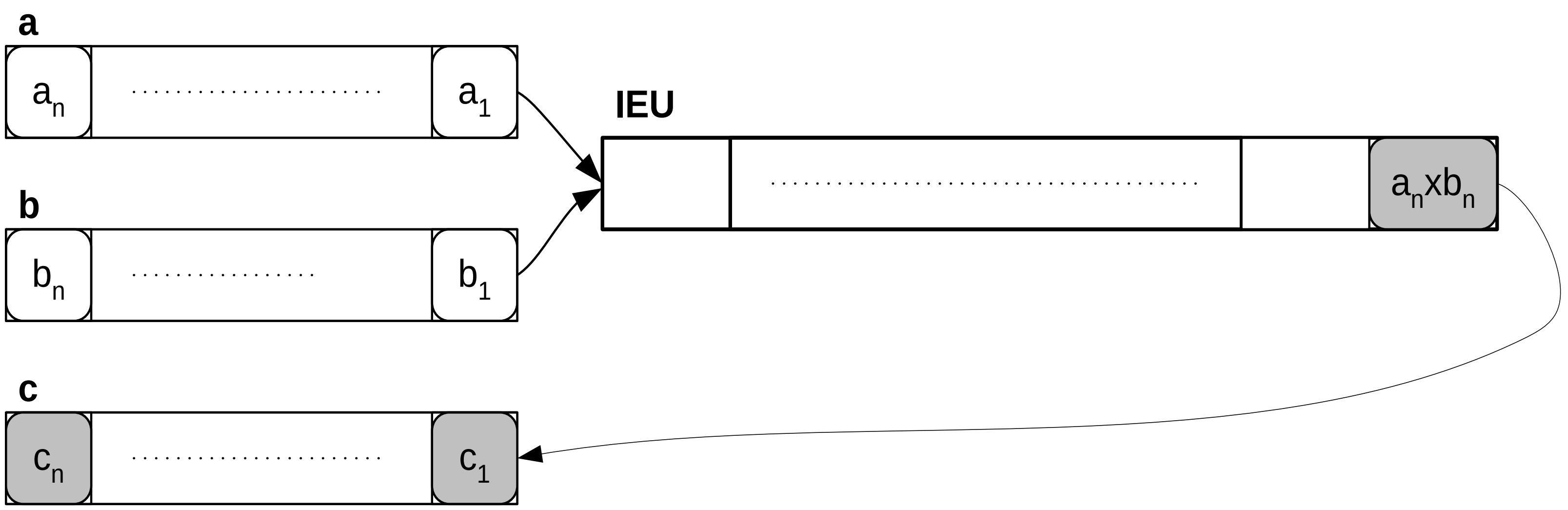
Cray-1 Vector Processor (ctd)

- Step $n+k-1$ ($k=2,\ldots,m-1$)



Cray-1 Vector Processor (ctd)

- Step *n*+*m*-1



Cray-1 Vector Processor (ctd)

- It takes $n+m-1$ steps to execute this instruction
- The pipeline of the unit is fully loaded only from m-th till n-th step of the execution
- Serial execution of n scalar multiplications with the same unit would take nxm steps
- Speedup provided by this vector instruction is

$$S = \frac{n \times m}{n+m-1} = \frac{m}{1+\frac{m-1}{n}}$$

- If n is big enough, $S \approx m$

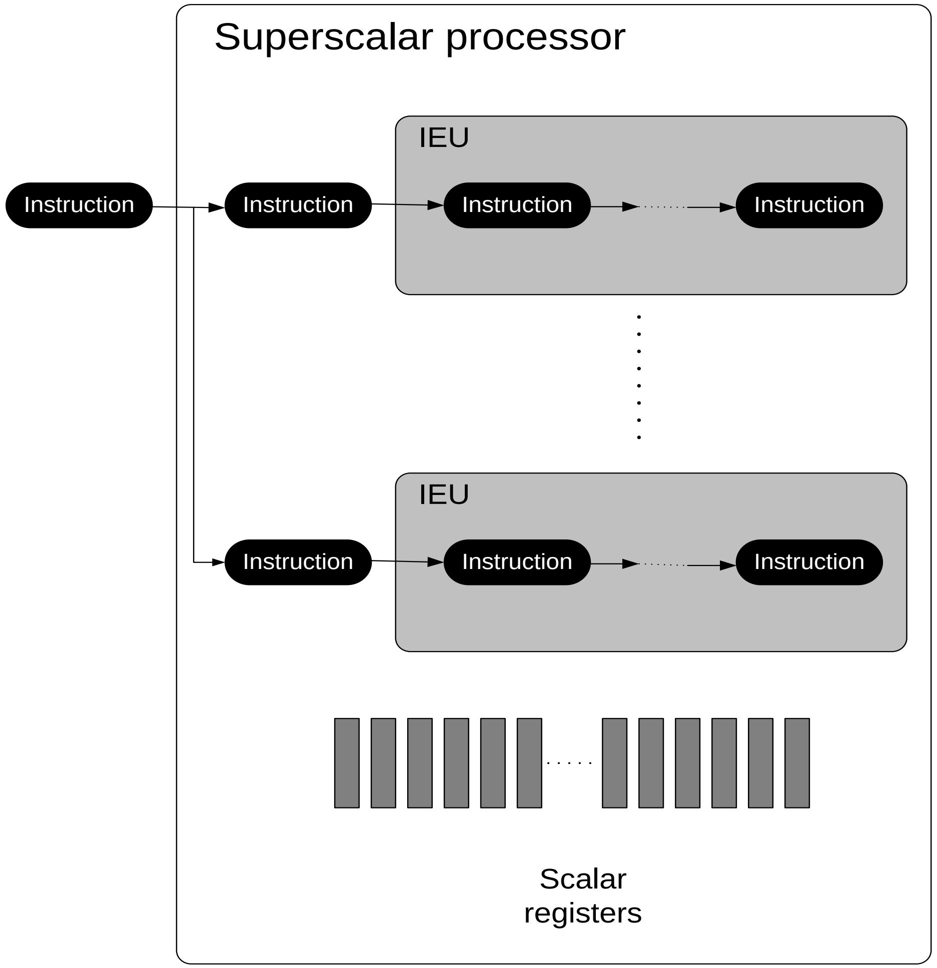
Vector Processor (ctd)

- VPs are able to speed up applications, whose computations mainly fall into basic element-wise operations on arrays.
- The VP architecture includes the SSP architecture as a particular case ($n=1$, $m=1$)

Superscalar Processor

- Superscalar processor
  - Provides single control flow with instructions operating on scalar operands and being executed in parallel
  - Has several IEUs executing instructions in parallel
  - Instructions operate on scalar operands located on scalar registers
  - Two successive instructions can be executed in parallel by two different IEUs if they do not have conflicting operands
  - Each IEU is characterized by the set of instructions it executes
  - Each IEU can be a pipelined unit

Superscalar Processor (ctd)

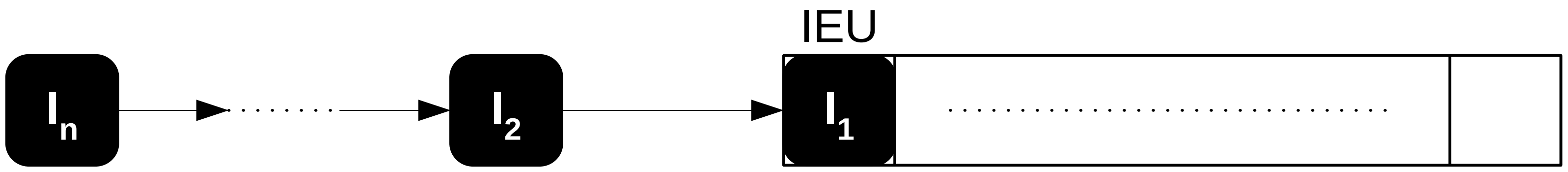


Instructions Pipeline (ctd)

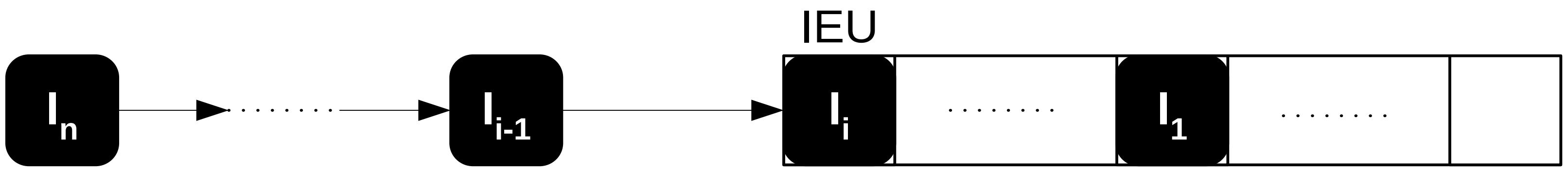
- A pipelined IEU can execute simultaneously several successive instructions each being on its stage of execution
- Consider the work of a pipelined IEU
  - Let the pipeline of the unit consist of m stages
  - Let n successive instructions of the program, $I_1,\ldots, I_n$, be performed by the unit
  - Instruction I_k takes operands from registers a_k, b_k and puts the result on register c_k $(k=1,\ldots,n)$
  - Let no two instructions have conflicting operands

Instructions Pipeline (ctd)

- At the first step, the unit performs stage 1 of instruction I_1

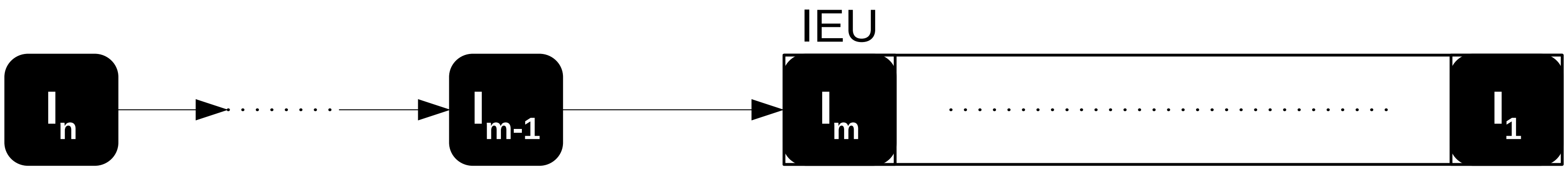


- Step i ($i=2,\ldots, m-1$)

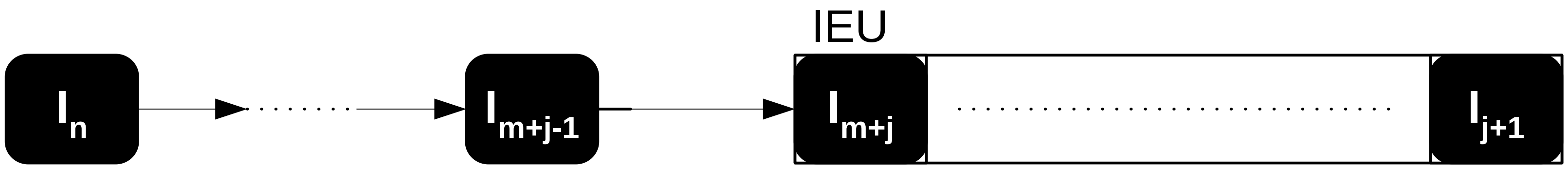


Instructions Pipeline (ctd)

- Step *m*

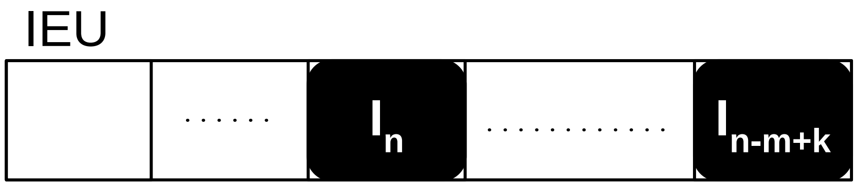


- Step *m*+*j* (*j*=1,…,*n*-*m*)

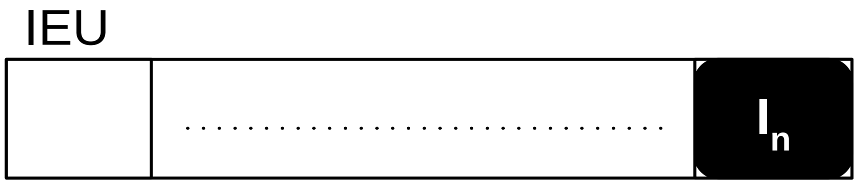


Instructions Pipeline (ctd)

- Step $n+k-1$ ($k=2,\ldots,m-1$)



- Step $n+m-1$



Instructions Pipeline (ctd)

- It takes $n+m-1$ steps to execute n instructions
- The pipeline of the unit is fully loaded only from m-th till n-th step of the execution
- Strictly serial execution by the unit of n successive instructions takes $n \times m$ steps
- The maximal speedup provided by this unit is

$$S_{IEU} = \frac{n \times m}{n+m-1} = \frac{m}{1+\frac{m-1}{n}}$$

- If n is big enough, $S_{IEU} » m$

Superscalar Processor (ctd)

- The maximal speedup provided by the entire superscalar processor having K parallel IEUs is

$$S_{proc} \approx \sum_{i=1}^{K} m_i$$

- SPs are obviously able to speed up basic element-wise operations on arrays
    - Successive instructions execute the same operation on successive elements of the arrays

Superscalar Processor (ctd)

- To efficiently support that type of computation, the processor should have at least

$$R \times \sum_{i=1}^{K} m_i$$

registers (each instruction uses *R* registers)

- CDC 6600 (1964) - several parallel IEUs
- CDC 7600 (1969) - several parallel pipelined IEUs
- Modern microprocessors are superscalar
- The superscalar architecture includes the serial scalar architecture as a particular case ($K=1$, $m=1$).

Vector and Superscalar Architectures

- Why are vector and superscalar architectures united in a single group?

- The most successful VPs are very close to superscalar architectures
  - Vector pipelined unit can be seen as a specialized clone of the general-purpose superscalar pipelined unit
  - Some advanced superscalar processors (Intel i860) are obviously influenced by the vector-pipelined architecture

Programming Model

- These architectures share the same programming model
  - A good program for vector processors widely uses basic operations on arrays
  - A program intensively using basic operations on arrays is perfectly suitable for superscalar processors
  - More sophisticated mixtures of operations able to efficiently load pipelined units of SPs are normally
    - Not portable
    - Quite exotic in real-life applications
    - Too difficult to write or generate

Programming Systems

- Vector and superscalar processors are an evolution of the serial scalar processor → no wonder that programming tools for the architectures are mainly based on C and Fortran

- The programming tools are
  - Optimising C and Fortran 77 compilers
  - Array-based libraries
  - High-level parallel extensions of Fortran 77 and C