Object Oriented Parallel Programming Languages

TEMPUS SJEP No. 08333-94

Activity No.1.

DR. MÁRAY TAMÁS

Technical University of Budapest

Parallel and Concurrent Programming

- potentially parallel execution of parts of a computation
- interleave the execution of components of a program on a single processor (quasi parallelism) or distribute it among several processors (real parallelism)

Parallel and Concurrent Programming

- concurrent programming
  - system programming for single or multiprocessor machines (both shared or distributed memory architecture) - eg. operating systems
  - concurrent, independent, asynchronous processes
- parallel programming
  - speed up the execution of algorythms which require high computational power
- main problems:
  - decomposition of the problem, creating parallel executable algorythms (discovering data and functional parallelism)
  - synchronization and communication of processes, mutual exclusion, etc.

Parallel and Concurrent Programming

- We have a lot of high level languages and tools for programming serial machines, however,
- Parallel machines differ from serial machines in many ways
- There are different parallel HW architectures
  - eg. shared memory computers, distributed memory computers, etc.
- There are different parallel programming paradigms
  - eg. data parallelism, task parallelism, etc.

Parallel and Concurrent Programming Languages

- General, high level, high power, effective, parallel languages are still missing, especially those, which could integrate the different concepts.
- At the moment, it is difficult to write *portable* parallel programs because of the lack of suitable languages and tools.

Object Oriented Paradigm

- Today it is a very fashionable programming (and SW designing!) paradigm
- High number of languages, tools and implementations exist
- The best known and probably mostly used OO language is C++, however many others have been developed:
  - eg. Simula, Smalltalk, Eiffel, Enfin, Java, Python, ...
- OO technology is suitable for building large software systems

Object Oriented Paradigm

- **Main goals are achieving high level of**
  - **data abstraction**
  - **data encapsulation**
  - **software modularity**
  - **code reuse**
- **Supporting modern software engineering and design methods**

Object Oriented Paradigm

- **OO development is based on the following main features:**
  - **classes and instances**
  - **inheritance**
  - **polymorphism**
- **Object based concept**
  - **using objects without inheritance**
  - **object based <-> object oriented**

Object orientation and parallelism

- It is obvious that new tools to be developed in order to satisfy the PP requirements, will include the OO paradigm!
- Large and complicated systems which are built on OO philosophy often need high computation power --> parallel processing
  - (also, take into consideration the effectiveness(less) of known OO language implementations...)
- The object philosophy and the parallel execution complement each other in very natural way
  - if objects are individual, why could not their execution be individual? ie. parallel?

OO and parallel

- Concurrency is a natural consequence of the concept of objects
- Object oriented philosophy fits especially well to parallel implementations
- It is obvious that the different objects can be implemented by different, parallel executed segments although other type of models have also been developed
- Interaction and communication interfaces of the objects are well defined and controlled

Concurrency and Object Orientation

- similarity between the basic constructs of object orientation and concurrency
- several analogies exist between objects and processes (classes and process types), eg:
  - local variable space
  - persistent data
  - encapsulated behavior
  - restrictions on how modules can exchange information
  - communication mechanism based on some form of message passing

OOPP language design criteria

- encapsulation, data abstraction and inheritance
- minimality of mechanism (smallest possible extension)
- full use of object oriented techniques (including inheritance!)
- enabling full use of available HW parallelism
- support for many forms of concurrency
  - eg.: shared memory, multitasking, distributed processing, etc.
- support for reuse of non-concurrent software
- promote OO software development

Creating OOPP languages

- Three distinct approches:
- 1. Design a new concurrent OO language
  - eg.: Hybrid, POOL, SR
- 2. Extend an existing OO language
  - Inheritance from special classes (Eiffel//, Presto, pC++)
  - Special keywords, modifiers or preprocessing (uC++, CC++, CEiffel)
  - Extension to the syntax and semantics (ACT++, Actalk)
- 3. Design a library. Use an existing OO language and provide concurrency abstractions through external libraries
  - eg.: ABC++, Eiffel with Class Library, Java

Fundamental problems - Anomalies

- One of the most important benefit of OO is code reuse, for which inheritance is the basic tool
- concurrency does not blend well with inheritance!
- synchronization code cannot be inherited easily
- inheritence anomalies
  - state partitioning anomaly
  - history-only sensitive anomaly
  - state modification anomaly

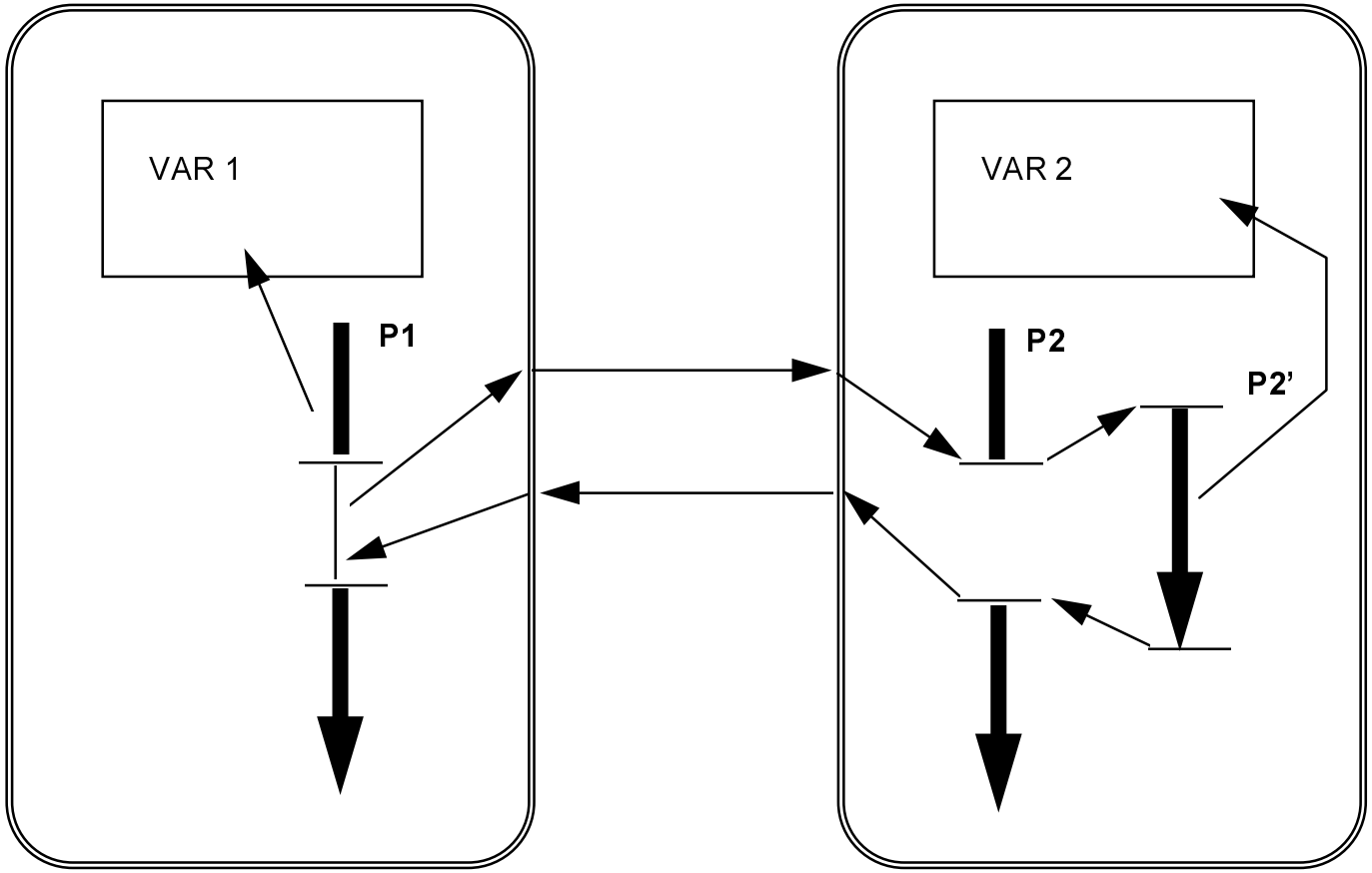
OO parallel programming models

- **Orthogonal approach**
  - **concurrent execution is independent of objects. Traditional synchronization constructs must be used**
- **Homogeneous approach**
  - **all objects are considered to be active entities that have control over concurrent invocations**
- **Heterogeneous approach**
  - **both active and passive objects exist**

Process structure of objects

- **Actor modell / CSP (Communicating Sequential Processes)**
  - **active objects**
  - **methods are scheduled for execution when messages are accepted**
  - **splitting the program according to the thread of control**
  - **a process is associated with an object**
  - **only that process is allowed to access the variables of the object**
  - **explicit control**

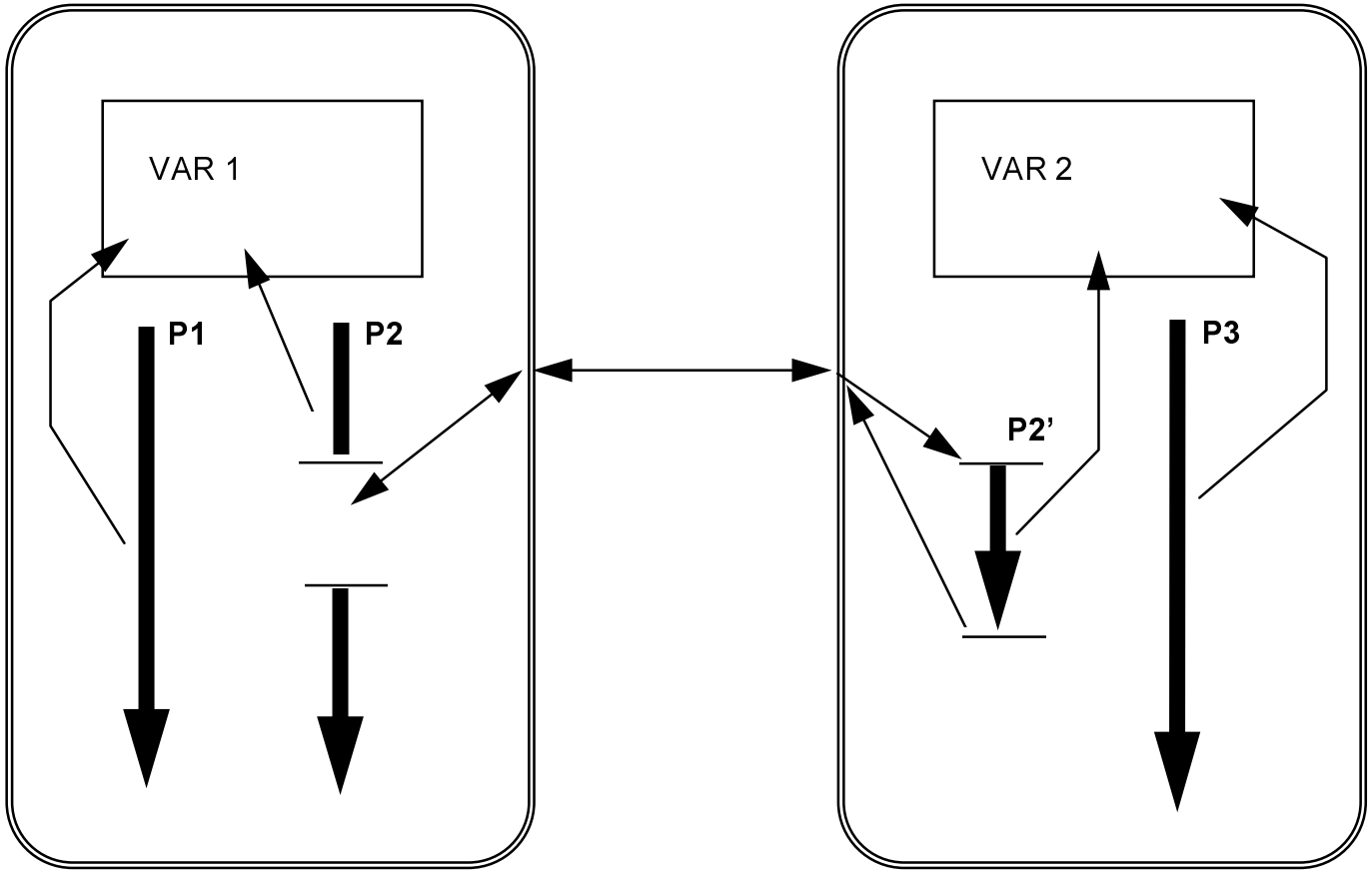
OOPP models - CSP like model



Process structure of objects

- **MONADS model / DP (Distributed Processes)**
  - **passive objects**
  - **process migration between objects**
  - **implicit control**
  - **several processes can access the variable space of an object**
  - **synchronization (mutual exclusion) mechanism is needed within the object**
  - **communication is generally implemented by RPC**

OOPP models - DP like model



Present efforts

- There are many research teams all over the world whose target is designing new OOPP languages.
- Cooperation exists between some of the research groups
  - eg. Europa project
- The high number of emerging OOPP languages is really frightening!
  - pC++, CC++, Mentat, MPC++, UC++, EC++, Charm++, ICC++, HPC++, ABC++, COOL, MC++, C**, Amelia, ORCA, para++, Emerald, Dowl, Amber, javaPVM, CEiffel, Eiffel//, Presto, SR, ABCL, POOL, Voyager, etc.
- Surely, most of them will not survive

OOPP languages

- A great variety of OOPP languages exist
- Most of the OOPP languages are derivated from C++, or are extensions to C++ :-(
- They cover the different parallel HW architectures and different parallel programming paradigms
- Some of them tries to fit to the existing standards (like MPI, PVM) some of them don't
- Some of them try to provide portability of applications and support different HW and SW environments, some of them are very special

Problems with implementation

- main problem is inefficiency, caused by mismatch between services that operating systems provide and those that languages offer
- fine grained language model <-> coarse grained system model (distributed address space!)
- interprocess communication mechanisms are not designed for fine grained object models
  - eg.: RPC does not translate well into distributed object systems (it is based on client-server mechanism, target must be located, etc.)
- object distribution, object mapping, efficient load balancing

Case studies

- **CC++ (Compositional C++)**
  - **one of the most widely used OOPP language**
  - **derived from C++**
- **Java**
  - **one of the newest general purpose OO language which allows concurrent programming, too**

CC++
(Compositional C++)

- Caltech University, USA
- Task parallel extension to C++
- Implements six basic abstractions:
  - The *processor object* is a mechanism for controlling locality
  - The *global* pointer, for linking together processor objects
  - The *thread* for specifying parallel execution
  - The *sync* variable for synchronizing thread execution
  - The *atomic* function for controling the interleaving of threads in processor objects
  - The *transfer* functions, for allowing data structures to be transferred between processor objects

CC++ (cont.)

- **Introduces atomic new keywords:**
  - **par - for parallel blocks**
  - **spawn - for spawning tasks**
  - **parfor - parallel loops**
  - **sync - synchronization of variables**
  - **global - global pointers to far objects**
  - **atomic - mutual exclusion within an object**
- **These keywords are used to compose concurrent programs on distributed environments**

CC++ (cont.)

- **The abstraction level of this language is not too high --> it is powerful!**
- **Compiler is available for most of the UNIX workstations**

Java

- New, high level, general purpose OO language
- Architecture independent -> portable applications
  - Java applications run on the Java Virtual Machine
- It is more simple and the OO concept is more clear than in C++
- It naturally supports networked environment
- Java allows writing parallel/concurrent applications - with using the “Synchronization” library

Java (cont.)

- Java implements threads
- There are additional tools for supporting Java parallel programming
  - JavaPVM - PVM interface for Java programs
  - Voyager - agent-enhanced distributed programming
- The Java “Synchronization” library supports all the traditional synchronization and communication mechanisms:
  - semaphore, monitor, randezvous, message passing, etc.