Introducing Parallel Processing into the Curriculum of the Hungarian Higher Educational Institutions of Technology
Tempus SJEP No. 08333-94

Activity No 1: Parallel Programming Mobility Grant and Partner No. 5

The parallel processing is a challenging new area of computer science. The fast progress in computer technology resulted commercial appearance of affordable multiprocessor based computers. The application of parallel computing rapidly spreads in many areas of engineering sciences. Without understanding and learning this new technique Hungarian engineers and scientist will be left behind and will not be able to design and produce competitive new industrial products.

Introduction to parallel processing techniques have been introduced into curriculum of nearly every higher educational institutions of technology in EC. Moreover special parallel processing centers have been established in most of these countries in order to support the training of students and industrial staff to understand and apply this new kind of technology. This Tempus project gives possibility to introducing the parallel processing into the Hungarian higher educational institutions. The project covers different activities (activity No 1-6). This report displays the results of one part of the activity No. 1: **Parallel Programming**. The aim of activity No. 1 to develop complete curriculum of the Parallel Programming course.

Parallel Programming Course

The main subjects are:

- Introduction to parallel processing. Classification of models.
- Concurrent, cooperative processes: basic concepts, problems.
- Cooperation via shared memory.
- Cooperation via message passing.
- Object based and object oriented models.
- Data parallel models.
- **Language and system tools for parallel programming.**
- Problems of real-time and distributed systems.
- Execution models of declarative languages.
- Programming in parallel C.
- Mapping, debugging, performance analysis.

Objectives

The goal of the Parallel Programming course to provide the students with the knowledge and skills to specify, design and implement programs by using several parallel models. The main subjects are:

Introduction to parallel processing. Classification of models.

Concurrent, cooperative processes: basic concepts, problems.

Cooperation via shared memory.

Cooperation via message passing.

Object based and object oriented models.

Data parallel models.

Language and system tools for parallel programming.

Problems of real-time and distributed systems.

Execution models of declarative languages.

Programming in parallel C.

Mapping, debugging, performance analysis.

The duration of course 1 semester (15 times 4 hours lectures and labs).

My task was to develop teaching materials about the **language and system tools for parallel programming**. More specific about the PVM, XPVM, HeNCE, EXPRESS, LINDA, P4. This topic will be completed with MPI by G. Dozsa. I was familiar with my task, because I have had experiences with PVM before I have come to London.

Overview of Parallel Programming

Why is parallel processing important ?

- To achieve higher performance ? Just for performance ?
  - parallelism is just a technological fix
  - not relevant to the applications engineer
  - hide it (like other details of hardware e.g. registers, cache,...)
- It's a common and natural mechanism for abstracting 'real-world' components
- It simplifies our engineering

Parallel processing myths

- harder than serial processing
- fragile (implicitly non-deterministic semantic)
- deadlock problems
- resource connection problems
- difficult to debug

Common experiences ... but not necessary ones !

Why is parallel processing important ?

To achieve higher performance ? If just for performance, we should hide it like other details of hardware e.g.. registers, cache, MMU's.

If just for performance it would not be relevant to the application engineer.

It is a common and natural mechanism for abstracting 'real-world' components.

There is some myths about parallel processing.

These are common experiences, but not necessary ones !

Architectures

Parallel architectures: Flynn's taxonomy

INSTRUCTIONS \ DATA	Single	Multiple
Single	Single Instruction Single Data **SISD** (serial machines)	Single Instruction Multiple Data **SIMD**
Multiple	Multiple Instruction Single Data **MISD**	Multiple Instruction Multiple Data **MIMD**

We classify the computer architectures by the Flinn's taxonomy.

The classification based on:

- the number of instructions and
- the data streams used in the architecture of machine

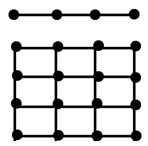
There are 4 main classes of architectures: SISD, SIMD, MISD, MIMD.

- The serial machines have SISD architectures. They are often referred to as von Neumann architecture machines, and also known as SERIAL or SCALAR machines.
- No specific MISD machine exists.
- CPP DAP, MASPAR, Connection Machine CM-2, CRAY-4 are examples of SIMD machines.
- Examples of MIMD machines are Parsys, Meiko, Transtech, Cray Y-MP, Cray T3D, Convex C3

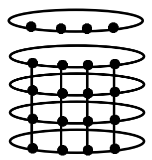
Characteristics of architectures

- Processor granularity
- Homogenous or heterogeneous
- Latency and bandwidth of interconnection
- Topology

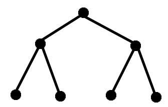
Meshes



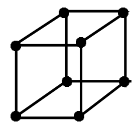
Rings and Toruses



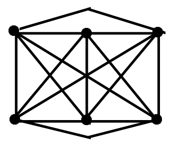
Trees



Hypercubes



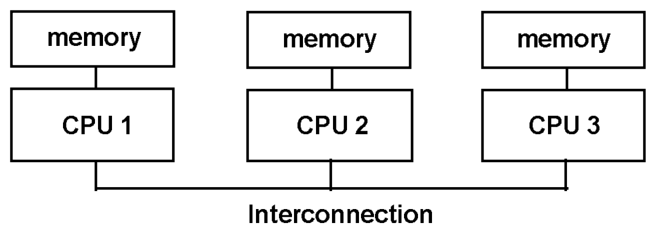
Fully-connected



Basic Concepts of Cooperation

- Cooperation via shared memory
- Cooperation via virtual shared memory (paging)
- Cooperation via message passing

Distributed Memory Concept



- Multiple processors work on a same problem
- Memory and address space are local to each processor
- Data is shared by passing messages
- Can be loosely or tightly coupled

  WAN → LAN → multiproc. computer

Important parallel processing concept is the distributed memory concept.

This is the easiest way to connect serial computers to getting parallel programming environment.

The main characteristics of distributed memory computers are:

Multiple processors work on a same ...

Why use Distributed Memory Computers ?

Benefits of parallel programming

- Decreased wallclock time

Benefits of distributed memory computers

- Scalable
- Cost effective
- Increased reliability through redundancy
- Can incorporate specialized processors

Disadvantages of distributed memory computers

- Communication is expensive
- Not all algorithms are appropriate
- Existing serial and shared memory applications may need to be restructured
- Effort is required to achieve real speed-up
- Programs can be difficult to debug

Why use
Distributed Memory Computers ?
(cont)

Suitable applications

- Applications distributed across networks
- Large grain size applications
  high ratio of computation to communication
  for each iteration
    for j data points
      do k calculations
    share bytes of data
      with y other processors
  end iteration

Language and System Tools for Distributed Memory Computers

Several tools are available:

PVM, MPI, Express, Linda, ISIS, ParaScope, EUI, NX, P4, Para++, HeNCE, etc.

Common concepts are:

- Process-level parallelism
- Parallelism is provided through subroutine libraries for C, C++, FORTRAN, LISP, Ada, etc.

PVM

Parallel Virtual Machine (PVM)

- Basic model: message passing

  sending process packs data into message buffer,specifies a receiver, sends the message

  receiving process receives the message, unpacks the buffer

  The sender must to known whom to send the message
- Provides communication and synchronization primitives
- Uses daemons to communicate between different processors in a network
- Supports Heterogeneous run-time environment (uses XDR)
- Public domain
- Tools: XPVM, HENCE

PVM made by Oak Ridge National Laboratory

Express

Express

- Basic model: message passing

  The low-level message passing is similar to the PVM paradigm

  The high-level message passing:

  Sequential program is distributed across N nodes

  The host node coordinates the work across the other nodes

- Express also provides higher level routines to simplify coordination of processes, CUBIX I/O, data decomposition, averaging, etc.
- Plotix library provides parallel graphics system

Linda

Linda

- Basic model: logical shared memory
  shared variable = tuple
  logical shared memory = tuple space
  operations:

  out → adds tuple to tuple space
  read → reads a tuple
  in → reads a tuple and deletes it from tuple space

  any process can access any tuple

  i.e the "sender" does not have to identify a "receiver"

  although each tuple is physically located on specific processor, the programmer does not need to know where it is stored

ISIS, ParaScope

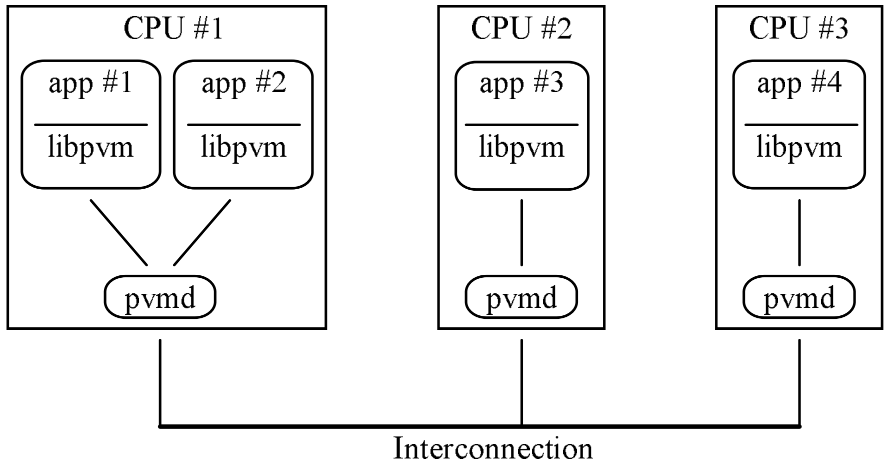
ISIS

- Basic model: virtual synchrony
  event driven process group
  replicated data
  uses RPC calls

ParaScope

- NOT an API
  Editor for generating FORTRAN D code

Basic Concepts of PVM



app # → Part of the application   libpvm → Language if.

pvmd → PVM daemon

PVM has two main parts

Language interface called libpvm

Simple subroutine calls that the application programmer may embed in concurrent or parallel application code.

Includes primitive operations:
- initiate and terminate processes
- pack send, receive and broadcast messages
- synchronize via barriers
- query configuration of the virtual machine

Architecture dependent daemon

One pvmd per user runs on each machine configured into parallel virtual machine. Each users have their own pvmds running.

Server - Client model. Pvmd works as a server.

Each processes (each part of the parallel applications) communicate with each other through their daemons.

Parallelizing for PVM

Strategies for parallelization:

"Embarrassing" parallelism

- Running several instances of same task with different parameters
- Suitable only if serial execution time of each instance is acceptable

Parallelize on a loop

- The computation is independent of the previous iteration

Parcel out tasks (master / slave)

- Supervisor task runs on one node
- Useful when work of main thread is much less than work of independent tasks
- Easy to run on arbitrary number of processors, if there are many independent tasks

Parallelizing for PVM (cont)

Strategies for parallelization (cont);

"Rolling " execution

- Each node hands off data to next at end of its serial section
- Useful when serial portion of code is not trivial but is still significantly sorter than parallel part
- All nodes run identical code - good load balance
- Especially suited to ring topology

"Regional"parallelism

- Data dependences are localized in portions of arrays
- Useful when serial time is greater than parallel time
- Requires internode communication
- Most difficult

Debugging and Tracing Tools for PVM

Several tools are available:

PraGraph, Xab, abmon, topg

Common concepts are:

- Intercepts the PVM calls, and generates trace records.
- Replays the trace records.

ParaGraph

- Graphical display package for visualizing the behavior and performance of parallel programs on message passing multiprocessor architectures.
- Uses PICL (Portable Instrumented Communications Library) traces as input.

Debugging and Tracing Tools for PVM (cont)

Xab

- Set of tools for analyzing and debugging PVM programs
- Major components:

  Xab libraries, abmon, xab, topg
- Provides visual and textual feedback about program execution bay using tracing information
- Not really a "debugger"

HeNCE
(Heterogeneous Network Computing Environment)

Main characteristics:

- Graphical parallel programming environment
- Interface for creating, compiling, and debugging parallel programs
- The programmer does not need to use any parallel programming primitives
- Relies on the PVM system