

The Implicit Model

1. Basic Concept:

With this approach, programmers write codes using a familiar sequential programming language, and then compiler is responsible to convert automatically it into a parallel codes (ex. KAP from Kuck and Associates, FORGE from Advanced Parallel Research).

2. Features :

- Simpler semantics: no deadlock; always determinate.
- Better portability due to sequential program.
- Single thread of control makes testing, debugging, correctness verification easier.
- Disadvantages: Extremely difficult to develop autoparallel compiler; Autopar always is low-efficiency.

The Data-Parallel Model

1. Basic Concept:

Data-Parallel model is the native model for SIMD machines. Data parallel programming emphasizes local computations and data routing operations. It can be implemented either on SIMD or on SPMD. Fortran90 and HPF are examples.

2. Features :

- Single thread: as far as control flow is concerned, a data parallel program is just like a sequential program.
- Parallel synchronous operation on large data structure (ex. Array etc.).
- Loosely synchronous : there is a synchronization after every statement.
- Single address space: all variables reside in a single address space.
- Explicit data allocation: users allocating data may reduce communication overhead.
- Implicit communication: users don't have to specify communication operations.

The Shared-Variable Model

1. Basic Concept:

The shared-variable programming is the native model for PVP, SMP and DSM machines. There is an ANSI X3H5 standard. The portability of programs is problematic.

2. Features :

- Multiple threads: A shared variable program uses either SPMD (Single- Program-Multiple-Data) or MPMD (Multiple-Program-Multiple-Data).
- Asynchronous: Each process executes at its own pace.
- Explicit synchronization: special synchronous operations (barrier,lock, critical region, event) are used.
- Single address space: all variables reside in a single address space.
- Implicit data and computation distribution: because data can be considered in SM , there is no need to explicitly distribute data and computation.
- Implicit communication: communication is done implicitly through reading/ writing of shared variables.

The Message-Passing Model

1. Basic Concept:

The message passing programming is the native model for MPP, COW. The portability of programs is enhanced greatly by PVM and MPI libraries.

2. Features :

- Multiple threads: A message passing program uses either SPMD (Single- Program-Multiple-Data) or MPMD (Multiple-Program-Multiple-Data).
- Asynchronous operations at different nodes.
- Explicit synchronization: special synchronous operations (barrier,lock, critical region, event) are used.
- Multiple address space: The processes of a parallel program reside in different address space.
- Explicit data mapping and workload allocation.
- Explicit communication: The processes interact by executing message passing operation.

Comparison of Parallel Programming Models

| Models | Implicit | Dat.Par. | Sha.Var. | Mes.Pas. |
|----------|-------------|----------|---------------|---------------|
| Arch. | SIMD | SIMD | SMP,DSM | MPP,COW |
| Thread | Single | Single | Multiple | Multiple |
| Compt. | Synch. | Synch. | Asynch. | Asynch. |
| Synch. | Auto. | Loosely | Explicit | Explicit |
| Comm. | Implicit | Implicit | Implicit | Explicit |
| Ad.Sp. | Global | Single | Single | Multiple |
| Dat.All. | Implicit | Explicit | Implicit | Explicit |
| Porta. | Better | Good | Bad | <u>Better</u> |
| Progr. | <u>Best</u> | Good | <u>Better</u> | Bad |

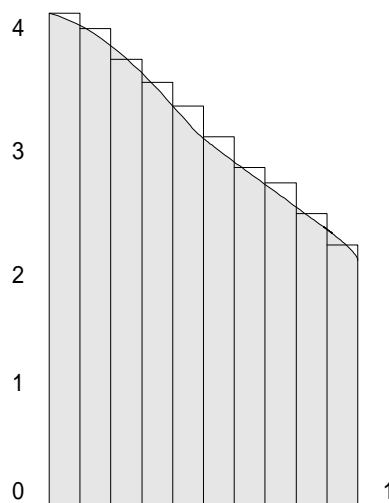
p Computation

☞ Integration formula of p :

$$p = \int_0^1 \frac{4}{1+x^2} dx \approx \sum_{i=0}^{N-1} \frac{4}{1 + \left(\frac{i+0.5}{N}\right)^2} \times \frac{1}{N}$$

☞ A sequential C code to compute p :

```
#define N 1000000
main(){
    double local,pi=0.0,w;
    long i;
    w=1.0/N;
    for (i=0;i<N;i++){
        local=(i+0.5)*w;
        pi=pi+4.0/(1.0+local*local);
    }
    printf("pi is %f\n",pi*w);
}
```



ANSI X3H5

☞ **Parallel Construct:**Using parallel construct to specify parallelism of X3H5 program.

Inside a parallel construct includes either parallel block,parallel loop,or single process.

```

program main                                ! The program begins in sequential mode
  A                                          ! A is executed by only the base thread
  parallel                                  ! Switch to parallel mode
    B                                       ! B is replicated by every team member
    psections                             ! Starts a parallel block
      section
        C                                  ! One team member executes C
      section
        D                                  ! Another team member executes D
    end psections                         ! Wait till both C and D are completed
  psingle                                  ! Temporarily switch to sequential mode
  E                                         ! E is executed by one team member
  end psingle                             ! Switch back to parallel mode
  pdo i=1,6                                ! Starts a pdo construct
    F(i)                                   ! The team members share the 6 iterations of F
  end pdo no wait                          ! No implicit barrier
  G                                         ! More replicate code
end parallel                              ! Switch back to sequential mode
H                                          ! H is executed by only the initial process
...                                       ! There could be more parallel constructs
end

```

☞ **Implicit barrier(fence operation):**Located parallel,end parallel,end psection,end pdo,end psingle forces all memory accesses up to this point to become consistent.

☞ **Thread interaction and synchronization,including four types of synchronization variables:**Latch,Lock,Event and Ordinal.

POSIX Threads(Pthreads)

Pthreads standard was established by IEEE standards committee which is similar to Solaris Threads.

☞ Thread Management Primitives :

| Function Prototype | Meaning |
|--|-------------------------------|
| <code>int pthread_create(pthread_t * thread_id, pthread_attr_t * attr, void * (*myroutine)(void *), void * arg)</code> | Create a thread |
| <code>void pthread_exit(void * status)</code> | A thread exits |
| <code>int pthread_join(pthread_t thread, void ** status)</code> | Join a thread |
| <code>pthread_t pthread_self(void)</code> | Returns the calling thread ID |

☞ Threads Synchronization Primitives :

| Function | Meaning |
|--|---|
| <code>pthread_mutex_init(...)</code> | Creates a new mutex variable |
| <code>pthread_mutex_destroy(...)</code> | Destroy a mutex variable |
| <code>pthread_mutex_lock(...)</code> | Lock (acquire) a mutex variable |
| <code>pthread_mutex_trylock(...)</code> | Try to acquire a mutex variable |
| <code>pthread_mutex_unlock(...)</code> | Unlock (release) a mutex variable |
| <code>pthread_cond_init(...)</code> | Creates a new conditional variable |
| <code>pthread_cond_destroy(...)</code> | Destroy a conditional variable |
| <code>pthread_cond_wait(...)</code> | Wait (block) on a conditional variable |
| <code>pthread_cond_timedwait(...)</code> | Wait on a conditional variable up to a time limit |
| <code>pthread_cond_signal(...)</code> | Post an event, unlock one waiting process |
| <code>pthread_cond_broadcast(...)</code> | Post an event, unlock all waiting process |

Shared-Variable Parallel Code to Compute π

The following code is a C-like notation :

```
#define    N    1000000
main() {
    double local, pi = 0.0, w ;
    long i ;
A:    w = 1.0 / N ;
B:    #pragma parallel
        #pragma shared ( pi, w)
        #pragma local ( i, local )
        {
            #pragma pfor iterate (i = 0; N ; 1)
            for ( i = 0; i < N; i++) {
P:        local = ( i + 0.5 ) * w ;
Q:        local = 4.0 / ( 1.0 + local * local ) ;
            }
C:        #pragma critical
            pi = pi + local ;
        }
D:    printf("pi is    %f\n", pi*w) ;
} /* main() */
```

MPI:Message Passing Interface

- ☞ **Message-Passing Library approach to parallel programming:**A collection of processes executes program written in standard sequential language augmented with calls to library of functions to send/receive message.
- ☞ **Computation:**In MPI programming model,a computation consists of one or more heavy weigh processes that communicate by calling library routines.The number of processes in an MPI computation is normally fixed.
- ☞ **Communication mechanism :**
 - ✍ Point-to-point communication operation.
 - ✍ Collective communication operations (broadcast,summation etc.).
- ☞ **Communicator:**to allow the MPI programmers to define modules that allow subprograms to encapsulate communication operations.
- ☞ **Basic MPI:**Although MPI is a complex system including more than 200 functions,we can solve a wide range of problems using just six of its functions!
- ☞ **Both C language binding and Fortran language binding for MPI.**

MPI Basics

`MPI_INIT(int *argc, char ***argv)`

Initiate a computation.

`argc`, `argv` are required only in the C language binding,
where they are the main program's arguments.

`MPI_FINALIZE()`

Shut down a computation.

`MPI_COMM_SIZE(comm, size)`

Determine the number of processes in a computation.

| | | |
|-----|-------------------|---|
| IN | <code>comm</code> | communicator (handle) |
| OUT | <code>size</code> | number of processes in the group of <code>comm</code> (integer) |

`MPI_COMM_RANK(comm, pid)`

Determine the identifier of the current process.

| | | |
|-----|-------------------|--|
| IN | <code>comm</code> | communicator (handle) |
| OUT | <code>pid</code> | process id in the group of <code>comm</code> (integer) |

`MPI_SEND(buf, count, datatype, dest, tag, comm)`

Send a message.

| | | |
|----|-----------------------|--|
| IN | <code>buf</code> | address of send buffer (choice) |
| IN | <code>count</code> | number of elements to send (integer ≥ 0) |
| IN | <code>datatype</code> | datatype of send buffer elements (handle) |
| IN | <code>dest</code> | process id of destination process (integer) |
| IN | <code>tag</code> | message tag (integer) |
| IN | <code>comm</code> | communicator (handle) |

`MPI_RECV(buf, count, datatype, source, tag, comm, status)`

Receive a message.

| | | |
|-----|-----------------------|--|
| OUT | <code>buf</code> | address of receive buffer (choice) |
| IN | <code>count</code> | size of receive buffer, in elements (integer ≥ 0) |
| IN | <code>datatype</code> | datatype of receive buffer elements (handle) |
| IN | <code>source</code> | process id of source process, or <code>MPI_ANY_SOURCE</code> (integer) |
| IN | <code>tag</code> | message tag, or <code>MPI_ANY_TAG</code> (integer) |
| IN | <code>comm</code> | communicator (handle) |
| OUT | <code>status</code> | status object (status) |

Message-Passing Code to Compute π

The following code is a C notation with MPI :

```
#define N 1000000
main() {
    double local, pi, w;
    long i, taskid, numtask;
A:   w = 1.0 / N;
    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &taskid);
    MPI_Comm_size( MPI_COMM_WORLD, &numtask );
B:   for ( i = taskid; i < N; i = i+numtask) {
        P:   local = ( i + 0.5 ) * w ;
        Q:   local = 4.0 / (1.0 + local * local) ;
    }
C:   MPI_Reduce (&local, &pi, 1, MPI_Double,
                MPI_MAX, 0, MPI_COMM_WORLD);
D:   if (taskid == 0) printf("pi is  %f\n", pi*w) ;
    MPI_Finalize();
} /* main() */
```

PVM:Parallel Virtual Machine

- ☞ **PVM is a public-domain software system developed at Oak Ridge National Laboratory that was originally designed to enable to network of heterogeneous Unix computers to be used as a large-scale message-passing parallel computer. Now, PVM has been implemented for non-Unix platforms(Windows NT, Windows 95).**
- ☞ **Comparison with MPI: The main difference between PVM and MPI is that PVM is a self-contained system while MPI is not. PVM is not a standard. MPI has more powerful support for message passing.**
- ☞ **PVM system consists of two parts: a PVM daemon (pvmd) resided on every computer, a user-callable library(libpvm3.a) linked to user application.**
- ☞ **Process management in PVM uses a set of process management functions. Communication with PVM uses a set of communication functions.**

PVM Program to Compute π

```
#define n16          /* number of tasks */
#include "pvm3.h"
main (int argc, char ** argv)
{
    int mytid, tids[n], me, i, N, rc, parent;
    double mypi, h, sum=0.0, x ;
    me = pvm_joyingroup( "PI" );
    parent = pvm_parent();
    if (me == 0) {
        pvm_spawn("pi", (char**)0, 0, "", n-1, tids);
        printf("Enter the number of regions: ");
        scanf("%d",&N);
        pvm_initsend( PvmDataRow );
        pvm_pkint(&N,1,1);
        pvm_mcast(tids,n-1,5);
    } else {
        pvm_recv(parent, 5);
        pvm_upkint(&N, 1, 1);
    }
    pvm_barrier("PI", n);  /* optional */
    h = 1.0 / (double) N;
    for (i = me + 1; i <= N; i += n) {
        x = h * ((double)i - 0.5);
        sum += 4.0 / (1.0 + x*x);
    }
    mypi = h * sum;
    pvm_reduce( PvmSum, &mypi, 1, PVM_DOUBLE, 6, "PI", 0);
    if ( me == 0 ) printf("pi is approximately %.16f\n", mypi);
    pvm_lvgroup("PI");
    pvm_exit();
}
```


HPF:High-Performance Fortran

- ☞ **HPF is a language standard designed as an extension of Fortran 90.F90's array language and HPF's data distribution directives provide a powerful notation for data-parallel computations in science and engineering.**
- ☞ **To support data-parallel programming,HPF introduces a FORALL construct,INDEPENDENT directive etc. which allow HPF to be more flexible in specifying array sections and parallel computation patterns.**
- ☞ **To achieve top performance,HPF offers ALIGN, DISTRIBUTE which enable user to tell the compiler how data should be allocated among processors.**
- ☞ **The most attractive feature of data-parallel approach as exemplified in HPF is that the compiler takes on the job of generating communication code which bring on two advantages :**
 - ☞ **First,it allows programmer to focus on the tasks of identifying opportunities for concurrent execution and determining efficient partition,agglomeration and mapping.**
 - ☞ **Second,it simplifies the task of exploring parallel algorithms,in principle,only data distribution directives need be changed.**

Gaussian Elimination in HPF

```
parameter n = 32
real A(n,n+1), x(n)
integer i, pivot_location(1)
!HPF$ PROCESSOR Nodes(4)
!HPF$ ALIGN x(i) WITH A(i,j)
!HPF$ DISTRIBUTE A(BLOCK,*) ONTO Nodes
do i = 1, n-1
  ! pivoting
  pivot_location = MAXLOC( ABS( A(i : n, i) ) )
  swap( A(i, i:n+1), A(i-1+pivot_location(1), i : n+1) )
  ! triangularization
  A(i, i:n+1) = A(i, i:n+1) / A(i,i)
  FORALL ( j = i+1 : n, k = i+1 : n+1 ) A(j, k) = A(j, k) - A(j, i) * A(i, k)
end do
! back substitution
do i = n, 1, -1
  x(i) = A(i, n+1)
  A(1:i-1, n+1) = A(1:i-1, n+1) - A(1:i-1, i)* x(i)
end do
```

Two Ways to Parallelize Compiler

- ➡ **SIMDizing:**It is also called vectorizing which explores parallelism at the instruction level and is the basis of the most widely used vector-super-computers.For example loop in program can be replaced by a vector instruction that has the effect of performing these simple operations in parallel or in pipelined fashion. The longer vector, the better speedup.
- ➡ **MIMDizing:**It is also called parallelizing which explores parallelism at the task level and is the basic parallelizing approach for MIMD multiprocessors.For example,the code can be divided into some tasks for MIMD multiprocessor,with each processor assigned to one task. The good speedup depends on the ratio between the actual computation time and the total overhead incurred.

SIMDizing : Vectorizing

☞ Procedure of the vectorizing :

- ☞ Dependence analysis
- ☞ Parallelization
- ☞ Mapping

☞ Dependence Analysis :

- ☞ Objective:To determine whether two statements or different interactions of a loop can run in parallel.
- ☞ Types :
 - ☞ *Anti-dependence(first read and then write)*
 - ☞ *Flow-dependence(first write and then read)*
 - ☞ *Output-dependence(first write and then write)*
- ☞ Testing :
 - ☞ *It is an NP-complete problem:time-consuming and not practical.*
 - ☞ *Apply only to certain class of problems:perfectly nested loop,separable etc.*
 - ☞ *Common testing techniques:GCD test,Banerjee bound test,| -test etc.*

☞ Parallelization :




- ☞ Objective:spread iterations of a loop across processors.
- ☞ Method :
 - ☞ *Examine loop dependence and convert sequential-loops to parallel-loops with necessary synchronization.*
 - ☞ *Elimination of cyclic dependence*
 - ☞ *Loop fusion,distribution etc.*

☞ Mapping :

Data and Program \mapsto **Virtual processors** \mapsto **Physical architectures.**

MIMDizing:Parallelizing

Three Approaches :

-  **Library Subroutines:**In addition to standard sequential libraries,the new function libraries are added to support parallelism and interaction(PVM Library,MPI Library etc.).
-  **New Language Constructs:**The programming language is extended with some new constructs to support parallelism and interaction(Forall etc.).
-  **Compiler Directives(also called pragmas in C):**there are formatted comments carrying additional information to help the compiler do a better job in optimization.

Practical Approach :




The practical applications should be coded using standard Fortran or C,plus a standard message passing library,such as PVM or MPI etc.

Performance Analysis






Performance Analysis basic steps :

-  **Data collection**
-  **Data Transformation**
-  **Data Visualization**

Difficulty :

-  **Performance data including execution time, communication cost and so on tend to be complex,huge and multidimensional.**
-  **Until now few standards exist.**
-  **The various public domain and commercial tools take different approaches,use different data file formats and provide different display techniques.**

Issues to be considered :

-  **Accuracy:**Depending on the sampling techniques,accuracy of clock etc.
-  **Simplicity:**Automatically collect with little or no programmer intervention.
-  **Flexibility:**Tools can be extended easily,different views of the same data should be provided.
-  **Intrusiveness:**Performance data collection inevitably introduces some overhead,we need to be aware of this overhead.
-  **Abstraction:**A good performance tool allows data to be examined at a level of abstraction appreciate for the programming model of the parallel program.

Data Collection

☞ **Meaning :**

Data Collection is the process by which data about program performance are obtained from an executing program.

☞ **Basic data collection techniques :**



- ✍ ***Profiles* record the amount of time spent in different parts of a program at relatively low cost which should be the first consideration technique. Profiles typically are gathered automatically.**
- ✍ ***Counters* record either frequencies of events or cumulative times including the number of procedure calls, total number of message etc. In fact, a counter is a storage location that can be incremented each time. The insertion of counters may require programmer intervention.**
- ✍ ***Traces* record each occurrence of various specified events such as time-stamped event which is the most detailed and low-level approach to performance data collection and widely used to study of program behavior. The disadvantages of traced-based approach are easy to generate the huge volume of data, thereby leading to probe effect.**

☞ **Comments :**





- ✍ **Profile and counter are easier to obtain and to analyze while traces can show fine detail.**
- ✍ **Tracing should be used with care and only if other data collection techniques are not available or do not provide sufficient information.**

Data Transformation and Visualization

Why need?

-  The raw data produced by profiles, counters or traces are rarely in the form required to answer performance questions. Hence data transformation are applied often with the goal of reducing total data volume. Transformations can be used to determine mean values or other higher-order statistics or to extract profile and counter data from traces.
-  Although data reduction techniques can be used in some situations to compress performance data, it is often necessary to be able to explore the raw multi-dimensional data, this process can benefit from the use of data visualization techniques.

Transformation and display formats :

-  Zero-dimensional(scalar) data display format.
-  One-dimensional data display format: histogram.
-  Two-dimensional data display format: color and two-dimensional matrix.
-  Traces data display format: histogram or Gantt Chart or space-time diagram.

Performance Analysis Tools:Examples

Paragraph :

It is a portable and interactive trace analysis and visualization package developed at Oak Ridge National Laboratory for message-passing programs,the user instructs paragraph to construct various displays including processor utilization. Communication volumes and patterns.A disadvantage of paragraph is that the relationship between performance data and program source is not always clear.

Upshot :

It is a trace analysis and visualization package developed at Argonne National Laboratory for message-passing programs. Upshot's display tools are designed for showing the state of the event.Upshot provides fewer displays than does paragraph,but has some nice features.The ability to scroll and zoom its displays is particularly useful.

ParAide :

The ParAide system developed by Intel's Supercomputer System Division is specialized for Paragon parallel computer. The modified versions of the standard Unix prof and gprof tools provide profiling on a per-node basis.The system performance visualization system provides information regarding the utilization of processors,communication network and memory bus.

IBM's Parallel Environment :

The IBM AIX Parallel Environment is specialized for IBM computers,in particular for SP multi-computer.A variant of the standard Unix prof and gprof commands can be used to generate and process multiple profile data.The VT can be used to display a variety of different trace data.

Definition, Goal and Concepts

☞ Definition :

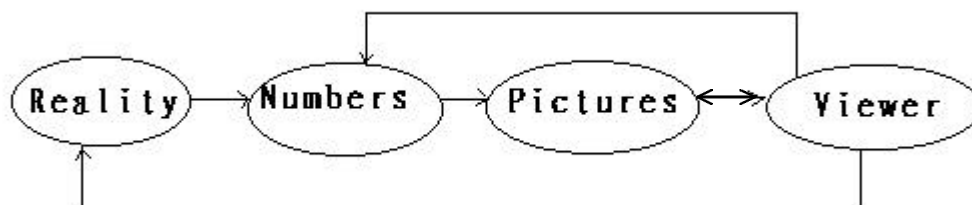
Visualization of scientific data describes the application of graphical methods to enhance interpretation and meaning of scientific data.

☞ Goal :

The goal of scientific visualization is to provide concepts, methods and tools to create expressive and effective visual representation from scientific data.

☞ Concepts :






- ✍ **Concepts and tools of scientific visualization are based on other disciplines: psychology/perception, computer graphics, artists and graphic designers.**
- ✍ **Scientific visualization is essentially a mapping process from one domain (a real phenomenon) into another domain (numbers) into yet another domain (pictures) and further into a fourth domain (the subjective interpretation of the viewer) :**







Characterization of Scientific Data

Visual Cues :

Visual cues are elements of a picture. Examples of visual cues/perceptual elements are below:

-  **Spatial:** position, motion.
-  **Shape:** length, depth, area, volume, thickness.
-  **Orientation:** angle, slope.
-  **Density.**
-  **Color:** colors, contrast.

Quantitative Type :

-  **Pointers(x_i, y_i):** Expressive visualization for point data sets are scatter plots and glyphs.
-  **Scalars** are usually samples of a continuous function:
 $y_i = f_i(x_1, x_2, \dots, x_n)$.
-  **Vectors:** possible displays for vectors are arrow indicating length and direction of each vector component.
-  **Tensor field:** Tensor is an extension of the vector and matrix expression. In an n-dimensional tensor field, the tensor's principal directions and magnitudes should be calculated.

Visualization Techniques

Scatter Plots :

The scatter plot uses position as its primary visual cue. It uses a linear mapping between data values and y-position, often a logarithmic scale is being used.



Glyphs :

Glyphs (called textons in perception research) have been invented to specifically express the complex data sets. One glyph is composed of individual parts: for example, a pair of circles, one vertical line, brightness and curve, and is usually identified by viewer as one figure.

Linear Graphs :

It is used to display continuous information and is an effective visual representation of scalar data sets.

Histograms, Pie Charts :

-  A histogram looks like a discrete line graph. The area of each rectangle is of special meaning.
-  If information is compared to some total number, we can use a pie chart to express relationship between each data value and the whole data set.

Contour Plot (isolines) :

It can be used to show lines of constant values inside the two-dimensional data set. It is also called isolines because they connect points of equal values.

More About Visualization Techniques

Image Display :

It is frequently chosen as visual representations of quantitative 2-dimensional data. To indicate the data value, gray levels or color can be chosen.

Surface View :

The numeric values of the data set are treated as elevations, and visually depicted as a terrain showing peaks for local maxima and valley for local minima in the data set.

Iso surfaces :

If the data are available in a regular lattice in 3D, we can use isosurfaces to visually represent volumetric data. Similar to isolines, surfaces of constant values are identified and illuminated.

Ray tracing of volumes :

If we want to display all of the elements inside a volume, we can use ray tracing to project each volume element (voxel) from 3D to 2D. Ray-traced images of volumes can assume a transparent, cloud like appearance (translucency).

Animation :

Using animation to move from one picture to the next, we can show a continuous sequence of visual representations allowing the viewer to observe changes between pictures.