
MOTECC-90
Modern Techniques in Computational Chemistry
Input/Output Documentation
MOTECC™-90

Input/Output Documentation


Published by

IBM Corporation
Center for Scientific & Engineering Computations
Department 48B/428
Neighborhood Road
Kingston, NY 12401 USA

MOTECC is a Trademark of the International Business Machines Corporation
October 1990
ICAP/3090 from MOTECC-90

Input Documentation - Version 1.0

David Folsom, Marian Klonowski, Nikos Pitsianis, Stephen Rondeau,
Antoine Trannoy, Silvia Veronese and Sylvain Waserman

With

Thierry Delbecque, Laura Evans, Pierre Ferron, Philippe Mouton

and Jukka Saarinen.

IBM Corporation
Center for Scientific & Engineering Computations
Department 48B/428
Neighborhood Road
Kingston, NY 12401 USA

Introduction

ICAP, an acronym for "loosely Coupled Array of Processors" is a Multiple-Instruction-Multiple-Data (MIMD) parallel computing system utilizing a master/slave topology. The system was originally conceived in early 1983 as a tool for a specific set of applications. However, over the course of the past 6 years the ICAP systems have evolved and developed into general purpose parallel computing systems for a broad range of scientific and engineering applications.

The first ICAP system, appropriately called ICAP-1, consisted of an IBM 4381 mainframe and 10 FPS-164 Attached Processors (APs). In this system the master was considered to be the IBM mainframe, and the APs were the slave processors. Eventually, a fast bus and shared memories between APs were added to improve communication paths between processors. A schematic representation of this system is given in Figure 1 on page 996.

In parallel to the development of the ICAP-1 system a second system was being assembled, ICAP-2. ICAP-2 was similar to ICAP-1 in the sense that it was also a master/slave system, however in the case of ICAP-2 the master was a larger IBM 3081 mainframe, and the slaves were more powerful FPS-264 Attached Processors. This machine was approximately 3-4 times more powerful than the ICAP-1 system.

Along with the development of the ICAP hardware systems was the development of the ICAP software for parallel processing. The main philosophy behind the software development was to modify the operating system software as little as possible, and to build the ICAP system on top of the IBM operating system. In this fashion the major portions of the software could be made operating system independent, and only particular pieces of the software would need modification in order to run ICAP on different operating systems.

IBM operating systems have inherent support for parallel execution on multiprocessors. In the IBM Virtual Machines VM/XA System Product operating system, each user runs his application in his own virtual machine (VM). Distinct VM's can be scheduled by the operating system to run simultaneously on different CPUs of the multiprocessor. A standard feature of the VM/XA operating system is the Inter-User Commu-
ication Vehicle (IUCV), which provides the facility to communicate messages and data between VMs. Thus, to achieve parallel execution of a single application program only requires that there be some mechanism to partition different portions of the application across several VMs, with each of the VMs connected to a master VM through the use of IUCV.

The current ICAP system, known as ICAP/3090, has been designed in an analogous fashion to the previous ICAP systems. The philosophy adopted is a master/slave topology, however in the ICAP/3090 configuration each processing node is an IBM 3090 CPU, and its associated hardware features. The central feature of ICAP/3090 is that multiple systems can be coupled to provide a two level hierarchy; "cluster" and CPU. Before we describe the configuration of the ICAP/3090 system, it is worthwhile to review the characteristics of the IBM 3090 family of processors.

The IBM 3090 vector multiprocessor family encompasses a variety of models ranging from a two-processor system (Model 200) to a six-processor complex (Model 600). Within a single 3090 it is possible to increase the throughput of a large workload by using the multiple processors on independent jobs. This has been the traditional motivation behind the development of such systems. An important corollary of this approach is that memory must be increased proportionately. This contributed to the development of expanded storage on the 3090 to alleviate the effect of paging that occurs when multiple jobs compete for real memory. Overall, the 3090 multiple-processor systems have been quite successful in increasing system throughput. It should also be pointed out that within a single IBM 3090 multi-processor all system resources such as memory, I/O, etc., can be shared by all processors. Therefore, it is also possible to decrease the turn-around time for a single job by utilizing multiple processors (and system resources) on that job. Finally, vector capabilities have been introduced to the 3090s in an integrated architectural manner with the addition of vector registers and 171 new instructions, providing each processor with scalar and vector processing. Overall, the design of this system emphasizes a balance of memory access, I/O, and processing capabilities.

The ICAP approach to parallel processing with IBM 3090s is to increase the level of parallelism available by coupling several 3090s together to form a system that is not massively parallel, but rather is modular and can be expanded to match the degree of parallelism that a set of applications can support. In this sense, the controlling software must also be modular and flexible, and easily adapted and expanded to take advantage of higher levels of parallelism as well as new hardware features that may become available.

The current ICAP/3090 system consists of two IBM 3090/400 Base machines, one IBM 3090/600 E machine and one IBM 3090/600 S machine, for a total of 20 processing nodes spread across four clusters. Each processing node has a scalar and vector processor, and the total memory on the four machines (main plus expanded) is 4.75 Gigabytes. Disk packs are shared by the four machines and amount to roughly 300 Gigabytes. The four 3090s are currently linked together via an IBM 3088 channel-to-channel connector which allows complete point-to-point communication among all 3090 systems. The maximum rate for transmission of data along the standard IBM channel is 4.5 Mbytes/second; therefore the maximum rate for exchange of data between IBM 3090s is of the same order of magnitude. This speed has obvious implications in determining the optimal partitioning of a parallel application, but this topic has been discussed in detail elsewhere and will not be addressed here.

The ICAP/3090 system software has been designed in a modular and flexible fashion and to support all parallel and vector capabilities on the system. Among the features worth mentioning are 31 bit addressing (2 Gbytes of virtual memory), shared memory within a single 3090 machine, parallel I/O and of course parallel processing across multiple 3090 systems.

Each parallel application uses several Virtual Machines (VM's) dedicated for the run. The VM's may be distributed across any of the 3090 systems in the ICAP complex. Each VM is initialized with a separate copy of the application program and data area, although the application may also designate special COMMON block areas in Shared Segments, which can be shared by the VM's within the same 3090 system. In this way both, distributed-memory with message passing and shared-memory facilities are available to the application.

One of the VM's is initialized first and becomes the 'Master'; controlling when parallel subroutines are executed on the remaining 'Slave' VM's. The application may have many cycles where parallel routines are initiated and completed on the Slaves. As part of each cycle, the Master controls the transfer of any neces-
sary data to and from the Slaves. Slave VMs may each execute a different part of the 'Slave Code'. The application program has explicit control of these services through the use of ICAP Directives.

There are several parts to the ICAP system providing a sort of 'shell' around VS FORTRAN for the functions just described above:

- Precompiler and Directives
- Parallel Run Utilities
- Run Scheduler and Resource Manager
- Commands to Start, Monitor and Stop a Job
- Communications Software

Directives are special FORTRAN Comment statements which the user adds to a program to control the ICAP parallel functions. The ICAP Precompiler scans the program, converting Directives to FORTRAN statements, and producing two new source files: one file for the Master VM and a second file for the Slave VM's. Both files are compiled with the VS FORTRAN compiler.

The parallel run utilities control the initiation of parallel subroutines, along with the transfer of data between Master and Slaves.

The ICAP/3090 scheduler assigns the resources (VMs and Storage) used by an application, and initializes them for a run. A set of user commands are provided to start (or abort) a parallel run and to query on the run status. When starting a run, the user specifies how many Slaves in which system(s) the application should run on.

The following scenario will illustrate the steps used to develop an application under ICAP/3090, test it and run it for productive use.

Assume that the application exists in its sequential (non-parallel) form. The first step is deciding how to break up the program into subroutines which can be executed concurrently. The program is altered to add ICAP directives and precompiled.

When the program Precompiles without errors, the Master and Slave files are compiled by FORTRAN. The user creates the Master-Slave EXEC file to define the files and the programs loaded for the Master and Slave VM's and then the application can be executed.

The structure of this document, intended to serve as a User Guide to the ICAP/3090 system, is as follows:

Chapter two has a complete description of the Precompiler Directives and their usage with simple examples.

Chapter three describes the User Interface, the available commands, the messages-responses of the system and directions on how to write the Master-Slave EXEC with examples, together with a short description of the Scheduler-Resource Manager, its purpose and function from a user's point of view.

At the end of this document you will also find an appendix with a Quick Reference chart.
Figure 1. Schematic representation of ICAP-1.
Guide to Precompiler use

Presentation

The Precompiler is a FORTRAN language conversion program invoked by a command of the form:

\[ \text{LCAPCOMP } \text{fm} < \text{FORTRAN} < \text{fm} > \]

where 'fm' is a character string of up to six characters that is used as the generic application name for the parallel run. One or two letters are appended to this application name to identify the specific input or output files shown in Figure 2. 'fm' is the file mode of the user's disk. Input to the Precompiler is either one or two FORTRAN source files. The Precompiler generates four output files.

\[
\begin{array}{c}
\text{filenameM FORTRAN} \rightarrow \\
[\text{filenameS FORTRAN} \rightarrow ]
\end{array} \]

\[
\begin{array}{c}
\text{LCAPCOMP} \\
\rightarrow \text{filename}$M$ \text{FORTRAN} \\
\rightarrow \text{filename}$S$ \text{FORTRAN} \\
\rightarrow \text{filename} \text{MEM} \\
\rightarrow \text{filename} \text{LIS}
\end{array}
\]

Figure 2. Input and Output files of the ICAP Precompiler.

The 'filenameM' FORTRAN file contains program source for the master program and may also contain slave routines. The 'filenameS' FORTRAN file is optional and only contains source for slave routines. Both files can also contain FORTRAN-like statements called "directives". To generate output code, the Precompiler replaces the directives in the input code by FORTRAN calls to ICAP utilities.

Directives start with the characters "CS" in the first two columns, making them transparent to the FORTRAN compiler since they satisfy the FORTRAN convention for comments. They must not contain labels. However, they begin in column 7, like FORTRAN statements, to allow continuation lines by specifying a character different from '-' or '0' in column 6. Any continuation lines also have to start with "CS" in columns 1 and 2.

The output 'MEM' file is required for ICAP execution and contains the shared memory size determined by the precompiler. The 'LIS' file is an information file containing all the messages sent to the screen during precompilation. It also contains a cross-reference table for common blocks declared with different sizes in several FORTRAN units. The format of this table, at the at the end of the 'LIS' file, is the following:
CROSS REFERENCE FOR COMMON BLOCKS DECLARED WITH DIFFERENT SIZES

COMMON /IPAIR/

In module JKINT the common block size is 4096 bytes.
In module PKINT the common block size is 4096 bytes.
In module SLHST the common block size is 4 bytes.
In module SLHSTU the common block size is 4 bytes.
In module SLDER the common block size is 4096 bytes.

COMMON ...

-------------
In module ...

Basic Directives

We illustrate now the different directives starting with the example below. There are three matrices (A, B and C) of N*N elements and the matrix product \( C = A \cdot B \) is performed. A sequential code is:

```fortran
REAL*8 A(N,N), B(N,N), C(N,N)
DO 10 I= 1, N
  DO 10 J= 1, N
    C(I,J) = 0.00
    DO 20 K= 1, N
      C(I,J) = C(I,J) + A(I,K) * B(K,J)
  20    CONTINUE
10   CONTINUE
```

In the following code, the routine is parallelized among NSL processes (or slaves). Each slave, with identifier ISL (from 1 to NSL), is going to compute approximately 1/NSL-th of the work:

```fortran
DO 10 I= ISL, N, NSL
  DO 20 J= 1, N
    C(I,J) = 0.00
    DO 20 K= 1, N
      C(I,J) = C(I,J) + A(I,K) * B(K,J)
  20    CONTINUE
10   CONTINUE
```

Shown below is the complete code to run this example on the ICAP-system. Individual directives are described in the remainder of this section. Program MAIN and subroutines ABIN and COUT run on the Master VM; subroutine MULT runs on each Slave VM. Note that COMMON /AMATRIX/ and arrays 'B' and 'C' have separate copies in the master and each slave.
PROGRAM MAIN
IMPLICIT REAL*8 (A-H,O-Z)
C
C INFORMATION GENERATED BY LCAP-SYSTEM
INTEGER*4 NSL,ISL,SLVSY(0:32),NSYS,LDRLSV(1:8),NUMSLV(1:8)
CHARACTER*8 VMNAME(0:32), SYSNAM(1:8)
COMMON /LCAP$/ NSL,ISL,SLVSY,VMNAME,NSYS,SYSNAM,LDRLSV,NUMSLV
C
C USER'S DATA
PARAMETER ( NMAX=16 )
DIMENSION A(NMAX*NMAX), B(NMAX*NMAX), C(NMAX*NMAX)
COMMON /AMATRIX/ A
C
C READ IN MATRIX DIMENSION N AND INITIALIZE ARRAYS A AND B
CALL ABIN(N,A,B,C)
C
C$ START
C$ EXECUTE ON ALL:MULT(N,B,C)
C$ ADDING C
C$ WAIT FOR ALL
C$ FINISH
C
C PRINT THE RESULT
CALL COUT(N,C)
STOP
END

SUBROUTINE COUT(N,C)
IMPLICIT REAL*8 (A-H,O-Z)
DIMENSION C(*)
DO 10 I= 1, N
  WRITE(6,30) (C(N*(J-1)+I), J= 1, N)
10 CONTINUE
30 FORMAT(6(F8.2,2X))
RETURN
END

SUBROUTINE ABIN(N,A,B,C)
IMPLICIT REAL*8 (A-H,O-Z)
DIMENSION A(*), B(*)
WRITE(6,*) 'ENTER N'
READ(5,*) N
DO 10 I= 1,N
  DO 20 J= 1,N
    A(N*(I-1)+J)= 1.0
    C(N*(I-1)+J)= 0.0
    IF (J.LE.I) THEN
      B(N*(I-1)+J)= 1.0
    ELSE
      B(N*(I-1)+J)= 0.0
    ENDIF
20 CONTINUE
10 CONTINUE
RETURN
END
C$ SLROUTINE
SUBROUTINE MULT(N,B,C)

-----------
IMPLICIT REAL*8(A-H,O-Z)

C
C INFORMATION GENERATED BY LCAP-SYSTEM
INTEGER*4 NSL,ISL,SLVSYS(0:32),NSYS,LDRSLV(1:8),NUMSLV(1:8)
CHARACTER*8 VMNAME(0:32), SYSNAM(1:8)
COMMON /LCAPS/ NSL,ISL,SLVSYS,VMNAME,NSYS,SYSNAM,LDRSLV,NUMSLV
C
C USER'S DATA
C
PARAMETER ( NMAX=16)
COMMON /AMATRIX/ A(NMAX*NMAX)
DIMENSION B(N,N), C(N,N)
C$ ALLOCATE B(NMAX*NMAX), C(NMAX*NMAX)
C
C$ SLIN /AMATRIX/
C$ SLIN N, B
C$ SLOUT C
C
DO 10 I= ISL, N, NSL
  DO 10 J= 1, N
    IK= I
  DO 20 K= 1, N
    C(I,J)= C(I,J) + A(IK)*B(K,J)
    IK= IK + N
10 CONTINUE
20 CONTINUE
RETURN
END

Run-Time Configuration

Run time information specified by the user, such as the number of slaves and which system(s) they are to run on..., is made available to the FORTRAN application via the COMMON /LCAPS/. This common should be declared by the user in those FORTRAN units containing EXECUTE and/or WAIT directives. The common block contains configuration information for the parallel run and is initialized prior to the start of the program. The meaning of each field is described below:

- NSL contains the number of slaves for the run (1-32).
- ISL contains the index (1:32) for a slave, or 0 for the master.
- SLVSYS(0:32) contains the relative system number(1-8) for the master and each slave.
- VMNAME(0:32) contains the userids of the VM's on which the master and slaves are running.
- NSYS contains the number of systems on which the job is scheduled.
- SYSNAM(1:8) contains the system names.
- LDRSLV(1:8) contains for each system the index of the leader slave which is the slave with the lowest index (ISL) on its system. If there are no slaves running on the same system as the master, then LDRSLV for that system is equal to zero. A simple use of LDRSLV can be found in the example of Section 2.3.3.
- NUMSLV(1:8) contains the number of slaves for each system.