

MCNPX RUNNING PARALLEL UNDER PVM

Franz X. Gallmeier
Oak Ridge National Laboratory
P.O. Box 2008, MS 6474
Oak Ridge, Tennessee, 37830, U.S.A.
(865) 574-9675
gallmeierfz@ornl.gov

Phillip D. Ferguson
Oak Ridge National Laboratory
P.O. Box 2008, MS 6474
Oak Ridge, Tennessee, 37830, U.S.A.
(865) 241-5702
fergusonpd@ornl.gov

SUMMARY

Gaps in the MCNPX code release 2.1.5 and release 2.2.3 were closed to enable running the code in multitasking mode on distributed memory parallel machines via the Parallel Virtual Machine (PVM) software. Performance tests were performed to check the runtime behavior of the code. These tests show that the code scales well on small sized cluster machines, and provides a significant speedup of SNS nuclear design analyses.

I. BACKGROUND

With the recent computer software and hardware developments, parallel machines become much more accessible to users who demand the application software to keep up and make use of the new capabilities. In this context it was seen that Monte Carlo applications, like the multi particle high-energy transport code MCNPX,¹ could gain significantly from parallel platforms. The benefit for the nuclear analyst is a shorter turn around of calculations, results with smaller statistical uncertainties, and the capability to tackle problems that were out of reach before.

Although the MCNPX code is based on MCNP4B,² which already provides parallel capabilities,³ MCNPX could not make use of it. The high-energy physics models, which MCNPX inherited from LAHET⁴ or which were added later, were not supported in the parallel scheme. An effort was launched to overcome this limitation to speed up the Spallation Neutron Source (SNS)⁵ nuclear design efforts.

II. IMPLEMENTATION

This task was realized in six implementation steps:

- The MCNPX parallel implementation inherited from the MCNP4B code was replaced by the more flexible and transparent MCNP4C⁶ modules,

- The initialization of the high energy physics models was added to the daughter processes,
- The option of writing files of high energy reaction histories was updated for the parallel mode,
- The mesh-tally feature was updated to be supported in the parallel mode,
- A new tally was added to sample the isotope production in the high-energy mode of the code.

A. The MCNPX Multitasking Strategy

The MCNPX code was built on the MCNP4B code, which is equipped with distributed and shared memory parallel capabilities. In MCNP4B, all multitasking coding both for the parent and daughter processes is pressed into one subroutine. For the MCNP4C version of the code, the multitasking was reworked separating the parent and daughter tasks, which resulted in a more transparent implementation. Except for some improvements with regard to allowing shared memory application in distributed systems, the strategy of multitasking seems to be unchanged from the MCNP4B version. Hence it was decided to update MCNPX with the implementation of MCNP4C.

The multitasking strategy is briefly described as follows:

- One parent process spawns and controls up to 200 daughter processes.
- The parent defines subtasks (the length of which can be influenced by the user) and updates the runtime information after finishing those.
- The parent divides each subtask into micro-tasks (number of available daughter processors times 5-27 depending on the differences in cpu speed of the daughter processes), and distributes the micro-tasks to the daughters until all micro-tasks are

done. This scheme supports an even balanced load on the daughter processes and minimizes processor dead times, which is especially important for heterogeneous systems, and systems with uneven loads on processors.

- The parent continuously monitors the daughter processes, detects and removes stalled daughter processes, and redistributes the micro-tasks of stalled daughter processes to available daughter processors.
- Upon finishing all subtasks, the parent evaluates and prints the runtime statistics of the summary tables and tallies.

B. Extending Multitasking to the High Energy Modules

One of the parent's first actions after spawning the daughter processes is to broadcast the problem information to the daughter processes. MCNP4B/C distinguishes between fixed, variable and ephemeral storage, particle storage, and tally storage. Hence, it is an easy task to determine what the daughter processes need for problem setup.

In the extension of the code to MCNPX this clear storage management was not continued. The new functionality was basically achieved by adding the high-energy subroutines from the LAHET code system and providing subroutines that build the link to the legacy code. All LAHET reaction models share information via common blocks that evolved historically resulting in unstructured and undocumented data arrays. None of the models accumulates particle information. The high-energy physics models merely determine the outcome of particle interaction with matter on a per particle basis. Sorting the numerous common blocks in fixed and variable variables would take an enormous effort and was thought to be unnecessary for distributed memory multitasking. For shared memory multitasking it is essential to allow parallel tracking of individual particles. As we intended to apply the code mainly on distributed memory multi-processor platforms, we made the decision to limit multitasking in MCNPX to the distributed memory capability via PVM.⁷

The piece missing to run MCNPX in the distributed memory parallel mode is the initialization of the high-energy models including the material description of the physics models. It was found that this initialization could be achieved solely from the

common MCNP4B storage in the daughter processes after having received all the information from the parent process.

C. Extending the History Tape Capability

Some of the particle transport information, like isotope production/destruction, gas production, etc., is gained from the MCNPX code high-energy physics model regime by post-processing a history tape, which contains the events of nuclear reaction outcomes.

Similar to writing a surface source file, the history records are written on a scratch unit by the daughter processes and transferred to the parent upon finishing a subtask. The parent finally writes the master history file.

Fortran compiled codes on LINUX platforms suffer from a file size limit of 2 Gigabyte. With current computers, this is a real limitation, especially as we envision large scale computing with the multitasking capability. Already the predecessor of MCNPX, the LAHET code, allowed extending this limit by writing continuation files. This capability was revived for the MCNPX code, not only for the parent's history tape, but also for the daughters writing their scratch units.

D. Extending the Meshtally Feature

The meshtally is a new feature of the MCNPX code and as such is not only completely decoupled from the general tally management, but also from the input processing. Only the storage is kept under the MCNP management. The tally arrays were extended to allow for multitasking, and coding has been added to initialize the meshtally feature on the daughter processes, and to transfer the tally subtask results from the daughter to the master process.

E. Isotope Production Tally

Generating isotope production rates for activation analyses is a frequent task to be performed by MCNPX in the design of nuclear and/or accelerator facilities. Writing and post-processing a history file as described in section C was found to be a real bottleneck to accomplish this task, especially because the post-processing has to be performed with a serial code. It seemed natural to sample for the isotope production online in MCNPX with the benefit avoiding storage requirements of huge history files and the long post-processing step. An additional benefit is that the user now can run as long as necessary to produce results with

statistical errors not dependent on file size limits of history files.

The isotope production tally is invoked by a single keyword followed by the cell numbers of cells to be investigated. If no cell number is provided, the code performs the analysis for all material cells in the problem.

For large problems, the limiting factor of the isotope production tally might be the memory required for sampling. Therefore one is directed to get as accurate an estimate of the storage requirement for each cell as possible. For setting up the storage, MCNPX determines the maximum mass and charge numbers for each cell from the material description. It then screens the "gamma-library" PHTLIB for the number of possible isotopes, groundstates, as well as isomeric states, below this maximum mass and charge also considering the added projectile mass and charge. A contingency is provided for produced isotopes that may be generated by the physics reaction models, but are not listed in the PHTLIB.

III. TESTING

The program development was performed on two platforms: the IBM/AIX4.2 in IBM/RS6K workstations and on Intel-Redhat LINUX 6.2, applying the IBM XL-Fortran/C and the Portland Group standard Fortran PGF77/GnuCC compilers, respectively. Both platforms provide decent debugging tools, which was of great advantage in tracing down programming errors. The choice of platforms was directed by the availability of IBM/SP2 and Intel-Beowulf LINUX clusters. Surprisingly, the MCNPX executables created with the IBM/AIX-f77 and LINUX-pgf77 compilers produce identical results.

It is MCNP quality assurance philosophy to be able to obtain reproducible results in two successive calculations performed with the same executable and with the same input files. This philosophy is also extended to multitasking meaning that a serial and a multitasking run have to give exactly the same results. This concept was the QA guideline for the implementation task.

Simple test cases were continuously run and compared against output produced by the original

serial codes in order to monitor the stages of program development to accomplish this task. Deviations from the nominal output were easily picked up and resolved by code corrections. Further alpha test runs were performed with large-scale SNS problems to check reliability, performance and matching of serial and multitasking results. After finishing the test positive, the codes were released for production runs.

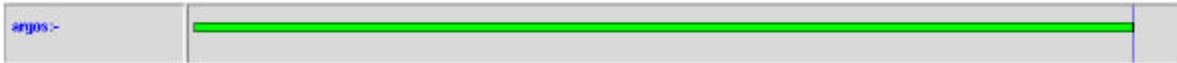
A valuable tool was found to be the XPVM⁸ tool, which is a real time performance monitor for PVM applications, as well as a PVM debugger provided by the authors of PVM. This tool writes a file of all the PVM activities that can be assessed manually and converted to charts of the runtime performance, the processor utilization and the message queue. The XPVM tool takes a long time analyzing a PVM-activity file scaling with the size of this file. For this reason, only problems with a small number of source particles were investigated.

Examples of runtime performance are presented in Fig. 1 for a test problem of 200 source protons of 1-GeV energy impinging on lead sphere in water environment. Results for the serial run, for 2-, 4-, and 8- processor multitasking runs are compared. And in detail discussed to demonstrate the program flow. The colors of the plots mean:

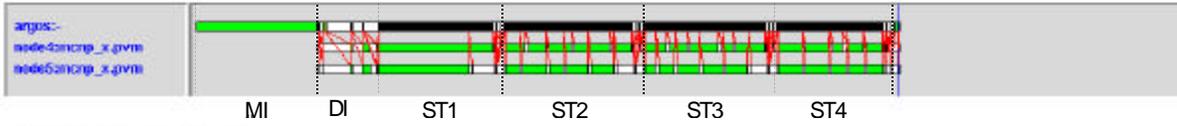
-  Green color indicates an active process performing user activities,
-  Yellow color shows periods when a process performs only PVM tasks,
-  White color shows the process in waiting stage,
-  Black means that the colors are not resolved on the time scale because the process periodically flips from green to yellow and,
-  Red lines indicate messages passed from one process to the other.

In principle PVM can establish and manage multiple distributed memory processes on a single processor, which practically does not make many sense as only one process can be active at a time. It is assumed further that a daughter process is served by a separate processor.

Single Processor Run:



Three Processor Run:



Five Processor Run:



Nine Processor Run:

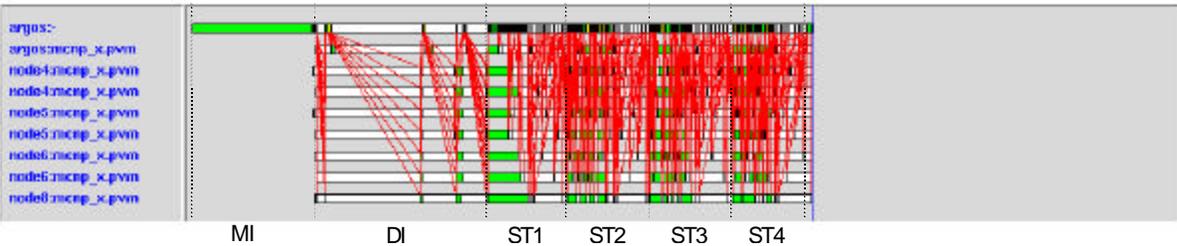


Fig. 1: Runtime Chart generated with XPVM for a 200 source particle problem run serial and with a different number of tasks with MCNPX. The phases of the multitasking runs are indicated with master initialization (MI), daughter initialization (MI), and subtask 1-4 (ST1-ST4).

In the nine-processor runtime graph, computer names appear twice for different processes because the computers are dual processor machines.

All the cases were run on the Beowulf cluster Argos of the SNS Experimental Facilities Division of Oak Ridge National Laboratory. All runtime charts are on the same time scale. This means that in this case not much time is gained by performing the problem in the multitasking mode. This is not typical, but an artifact of the small number of source particles tracked.

For all multitasking runs the master process needs some time to set up the problem reading and interpreting the input before it spawns the daughter processes. The time period it needs depends on how fast it reads the input data from the hard disk. Because of other activities on the machine that runs the master process it took the five-processor run a little longer than the others.

After spawning, the master broadcasts data arrays to the daughter nodes needed for their initialization. The length of this process depends on the available network bandwidth causing differences in the length of the initialization phase of the daughter processes.

After the master received the message ready to go from all daughter processes, it launches the subtasks, here 50 particles per subtask, requesting $50/(\text{number of daughter processes})$ from each processor at for the first subtask. For routine calculations the first subtask is very short, at maximum 200 source particles long, just as much as is needed to check if all daughter processes are performing well.

For the following subtasks the master splits the subtask size into $(\text{number of daughter processes}) \times$ (5 to 27) so called microtasks. One microtask is

submitted to a daughter process at a time, which will receive another microtask upon request from the master until all microtasks are preformed. Needless to say that the master manages the distribution. The master also tracks how many microtasks each process performed and increases the average microtasks per process from 5 to maximal 27 in case of imbalance. This enables the master to respond to changes in the machine loads minimizing waiting times from unfinished processes. If one process doesn't respond for a long time, the master will not serve it and will redistribute the lost microtasks to the other available processes.

At the end of each subtask the daughter processes send the results back to the master, which updates its summary and tally arrays subsequently. Upon finishing all subtask the master performs the statistical analysis of the tally information and finishes the run with the printout.

As our test cases use only a small subset of the MCNPX capabilities, later on rigorous checks were performed employing the complete test case suits provided with the MCNPX package (34 test problems for MCNPX_2.1.5, and 42 test problems for MCNPX_2.2.3). In three test problems, tracking changes were identified, meaning that the basic summary tables show differences comparing serial versus multitasking runs. In four additional problems, differences in tallies were noted, besides numerous differences in tally fluctuation charts and other peripheral output. These test reveal that some work is waiting to resolve all these issues.

IV. PERFORMANCE

A large test problem was performed on the LINUX Beowulf cluster Argos to investigate the performance of the multitasking mode of MNCPIX.

As test problem served the SNS target station model requesting information about the neutron leakage distributions from the moderators located on the top and the bottom of the flat mercury target due to 1GeV protons. The model consists of over 300 cells, 6 sets of point detectors, and 4 sets of surface current tallies. The serial run of the problem consumed about 6200 minutes of cpu time for 1,000,000 source protons. The problem was run

serial and in the multitasking mode on 2, 4, 8, and 16 processors. The wall clock times of the runs were monitored and evaluated.

This test problem used the default subtasking scheme, starting with a size of 200 particles for the first subtask, followed by a size of 1000 particles up to 20,000 particles, by a size of 2000 particles up to 40,000 particles, by a size of 4000 particles up to 80,000 ... ending with a size of 64,000 particles up to 1,000,000 source particles. This scheme involves many rendezvous of master and daughter processes.

Figure 2 presents the scaling behavior of the runs involving different numbers of processors in terms of the numbers of source particles calculated per minute (throughput). The theoretical curve assumes a linear increase of the throughput with the number of processors. In practice the throughput, calculated the (number of calculated source particles/wall clock time) is lower because the master process never contributes in particle tracking, and PVM overhead time and processor waiting times sum up. The processor efficiency, plotted in Fig.3, is calculated as the ratio of practical and theoretical throughput. It is obvious that the processor efficiency is especially low for the minimum processor multitasking case with 64%, and rises as processors are added to the run, because the overhead time from the master processor is then distributed through more processors.

Also of interest is to compare the results not adding the master process to the timing. This gives an indication about the penalty paid by the PVM message passing and waiting times. Figure 4 presents the throughput plotted versus the productive processors showing almost a lineup of theoretical and practical curves. The efficiency of the productive processors does in no considered case drop below the 90%.

It would be of interest to extend these performance curves to higher numbers of processors. Also of interest would be to study how the performance can be improved by reducing the numbers of subtasks. Now, since the multitasking MCNPX has been made available to the neutronics group of the SNS Project, the computer cluster is very loaded, and it is almost impossible to perform such studies.

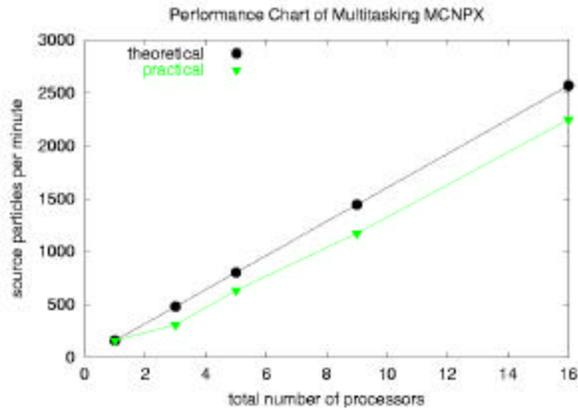


Fig. 2: Average calculational speed of a large-scale problem run with MCNPX in serial and multitasking mode with different numbers of processors.

MCNPX in serial and multitasking mode with different numbers of processors.

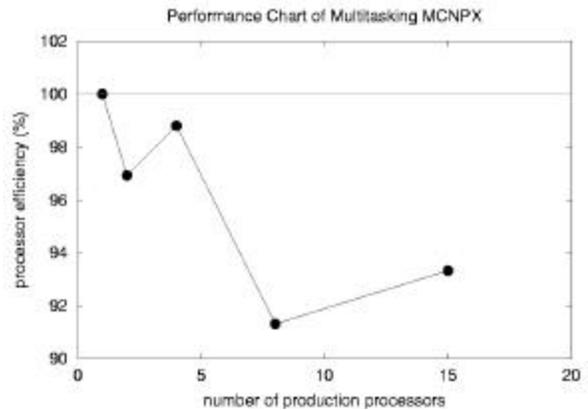


Fig. 5: Efficiency of the daughter processors for a large scale problem run with MCNPX in serial and multitasking mode involving a different numbers of processors.

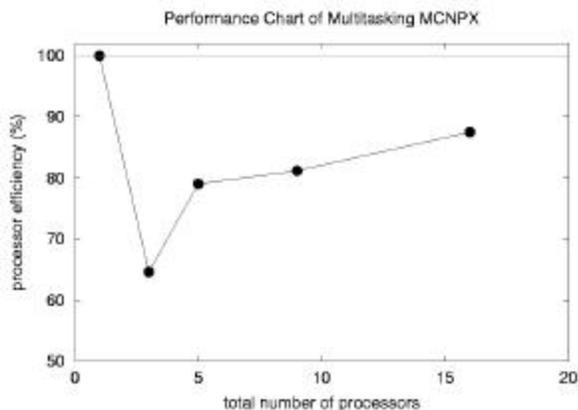


Fig. 3: Average processor efficiency of a large-scale problem run with MCNPX in serial and multitasking mode involving a different number of processors.

V. CONCLUSION

After closing some gaps in the MCNPX code considering the applicability of the code to run on distributed memory parallel computers under PVM, performance tests have proven that significant speedups can be achieved in transport calculations on a Beowulf-type computer cluster. Some work remains to be completed to close the coding gaps for shared memory systems, and to clean up inconsistencies in some of the test problems distributed with the MCNPX package.

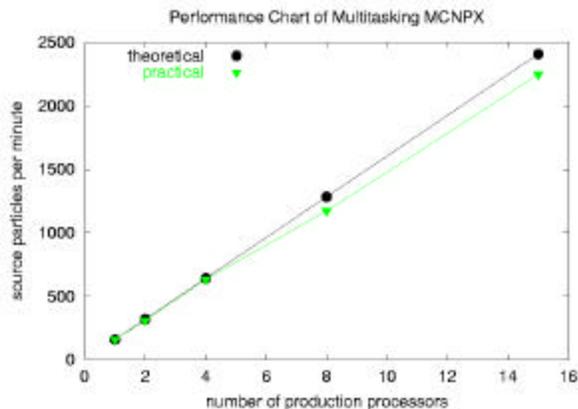


Fig. 4: Calculational speed of the daughter nodes for a large scale problem run with

VI. ACKNOWLEDGMENT

This work was supported by the U.S. Department of Energy through the Spallation Neutron Source (SNS) Project.

SNS is managed by UT-Battelle, LLC, under contract DE-AC05-00OR22725 for the U.S. Department of Energy.

VII. REFERENCES

1. H. G. Hughes et. al., "MCNPX for Neutron-Proton Transport," *International Conference on Mathematics & Computation, Reactor Physics & Environmental Analysis in Nuclear Applications*,

American Nuclear Society, Madrid, Spain,
September 27-30, 1999.

2. R. E. Prael, H. Lichtenstein, "User Guide to LCS:
The LAHET Code System," Los Alamos National
Laboratory, LA-UR-89-3014 (1989).

3. J. F. Briesmeister, "MCNP- A general Monte
Carlo n-particle transport code, Version 4B," Los
Alamos National Laboratory, LA-12625-M (1997).

4. G. W. McKinney, "A Practical Guide to Using
MCNP with PVM," *Trans. Am. Nucl. Soc.*, 72, 397
(1994).

5. R. L. Kustom, "An overview of the Spallation
Neutron Source Project," *Proceedings of the XX
International Linac Conference*, Monterey (2000).

6. J. F. Briesmeister, "MCNP- A general Monte
Carlo n-particle transport code, Version 4C," Los
Alamos National Laboratory, LA-13709-M (2000).

7. A. Geist et. al., "PVM3 User Guide and
Reference Manual," ORNL/TM-12187, Oak Ridge
National Laboratory (1994).

8. J. A. Kohl, G. A. Geist, "XPVM 1.0 Users Guide,"
ORNL/TM-12981, Oak Ridge National Laboratory
(1996).