Chapter 6

Results and Discussion

Parallel MOPAC was first developed on a Sun 4 workstation cluster using PVM. It was ported to a Thinking Machine CM5 and an Intel iPSC/860. The code was changed to use MPI in order to work with PNNL PeIGS later. The code was further ported to a DEC Alpha farm with a Giga switch, the IBM SP2, the SGI Power Challenge, as well as a LINUX workstation cluster.

A prime concern, performance is the most important factor of a parallel program. Even if we have shown that our approach should improve the performance, we undeniable still need some benchmarks to illustrate how good parallel MOPAC is for real world applications.

In this chapter, we will demonstrate the performance improvement of parallel MOPAC using small and large data files. The experiments were performed on SGI Power Challenge shared memory parallel machines in NPAC and NCSA and an IBM SP2 distributed memory parallel machine in Argonne National Laboratory. All results are average as derived from at least 5 trials.

It will cost too many computer resources to run through all of the data files several times. The biggest data may run weeks. It will exceed the maximum allowed reservation time for these parallel machines. Moreover, the node-hours accumulate rapidly when many nodes are used. Obviously, we would run out of our quota before we could finish our experiments. The big data benchmark experiences are stopped at some carefully chosen cut points where all benchmarking subroutines are called for at least 5 times.

Jobs running on other nodes may interfere the benchmarks. In order to avoid unnecessary interference, we always allocate all 16 processors of the SGI Power Challenge for every trial. An accurate observation again is that it is too costly to allocate all 80 IBM SP2 nodes for every trial. Fortunately, the SP2 switch can be partitioned into smaller partitions such that there is no interference between partitions. Therefore, we can still get reasonable results.

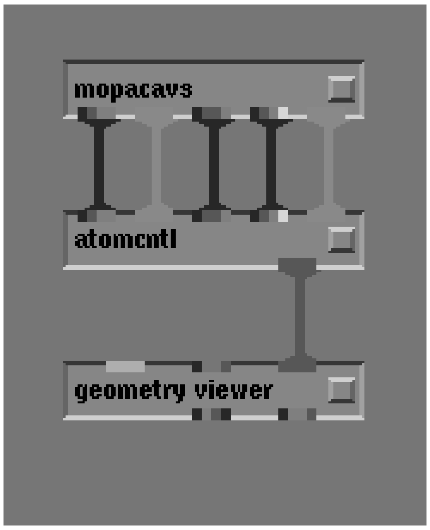
6.1 Running Parallel MOPAC

It is better to use the graphical user interface to run parallel MOPAC via AVS. Nevertheless, for users who do not have AVS or do not want to use graphical user interface, parallel MOPAC additionally provides a command line user interface. The procedures for getting, configuring, compiling, and installing parallel MOPAC are described in Appendix A.

The parallel MOPAC graphic user interface consists two AVS modules, mopacavs and geomcntl. Module mopacavs is the real user interface. It serves as the controller and server of the parallelized MOPAC tasks. It is also the bridge between MOPAC and AVS. It receives the internal molecule geometry information and thereby sends it to the geomcntl molecule. Module geomcntl transforms the molecule geometry information into AVS geometry objects. The AVS built-in module "geometry view" then takes the objects and displays them on the screen.

A pre-configured AVS network for running parallel MOPAC is provided so that users can simply load and run it. The AVS network for running parallel MOPAC is shown in Figure 6.1(a).

After the network is connected, a blank geometry viewer window and a MOPAC execution control panel window pop up. The MOPAC execution control panel shown



(a) AVS network

(b) AVS control panel

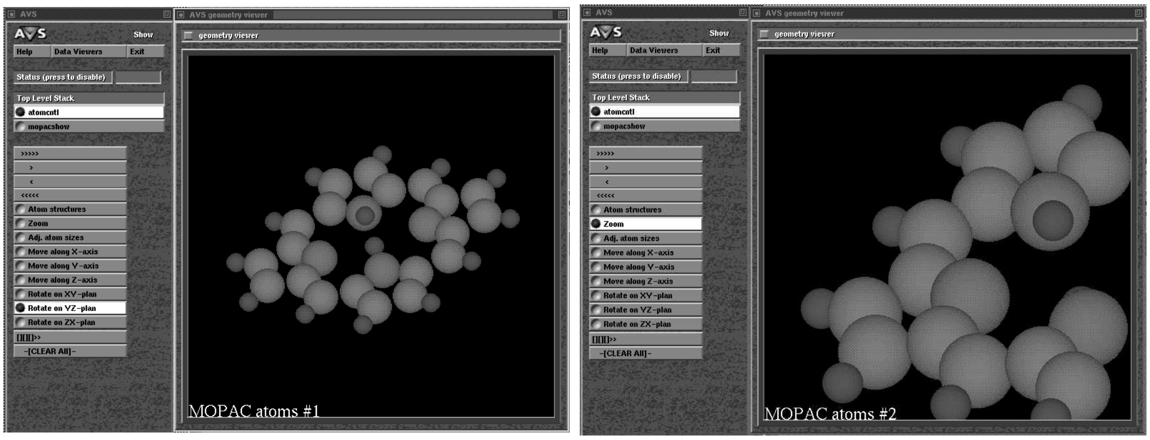
Figure 6.1: Parallel MOPAC AVS network and control panel

in Figure 6.1(b) requests the input file name and number of nodes to be used. The default number of nodes is 0 which means the MOPAC program will detect the number of nodes automatically.

When MOPAC reads in a new molecule structure or changes an existing one, the molecular structure will be displayed in the geometry viewer window. The user can exploit the MOPAC geometry control panel to scale, move, or rotate the molecule structure in order to get a clear view. Combining these associations, Figure 6.2 shows how a protein molecule structure can be moved, scaled, and rotated by using the parallel MOPAC geometry control panel.

For data which generates multiple molecule structures, module geomcntl keeps all molecule structures. Users can view them one by one by clicking on the MOPAC geometry control panel. The user can also click on the "slide show" button on the MOPAC geometry control panel to comprehensively view all molecule structures one after another. This is useful to the chemist for visualization of changes in structures or reaction mechanisms. Figure 6.3 shows the MOPAC graphic user interface in AVS.

For users who do not use the MOPAC graphic user interface, A command line user



(a) rotation (b) zooming

Figure 6.2: Parallel MOPAC geometry control

interface can be used to run parallel MOPAC. The command line format is as follows:

```
% mopacterm #proc
```

The optional argument #proc describes the number of nodes you want to use. This command is a shell script to determine the architecture of the machine and starts the correct binary of the MOPAC sequential computation module. The MOPAC sequential computation module then issues the mpirun command to start the MOPAC parallel computation module.

A more complicated command line user interface is also provided for users who can not use mpirun command to start parallel jobs. This means that IBM SP2 batch system users need to use spsubmit command to run parallel jobs. The command line format is as follows:

```
% mopacnode.rs6000.AIX 1:-1:0:0:filename
```

This command is actually used by the MOPAC sequential computation module to start the MOPAC parallel computation module. The ":"-separated command line

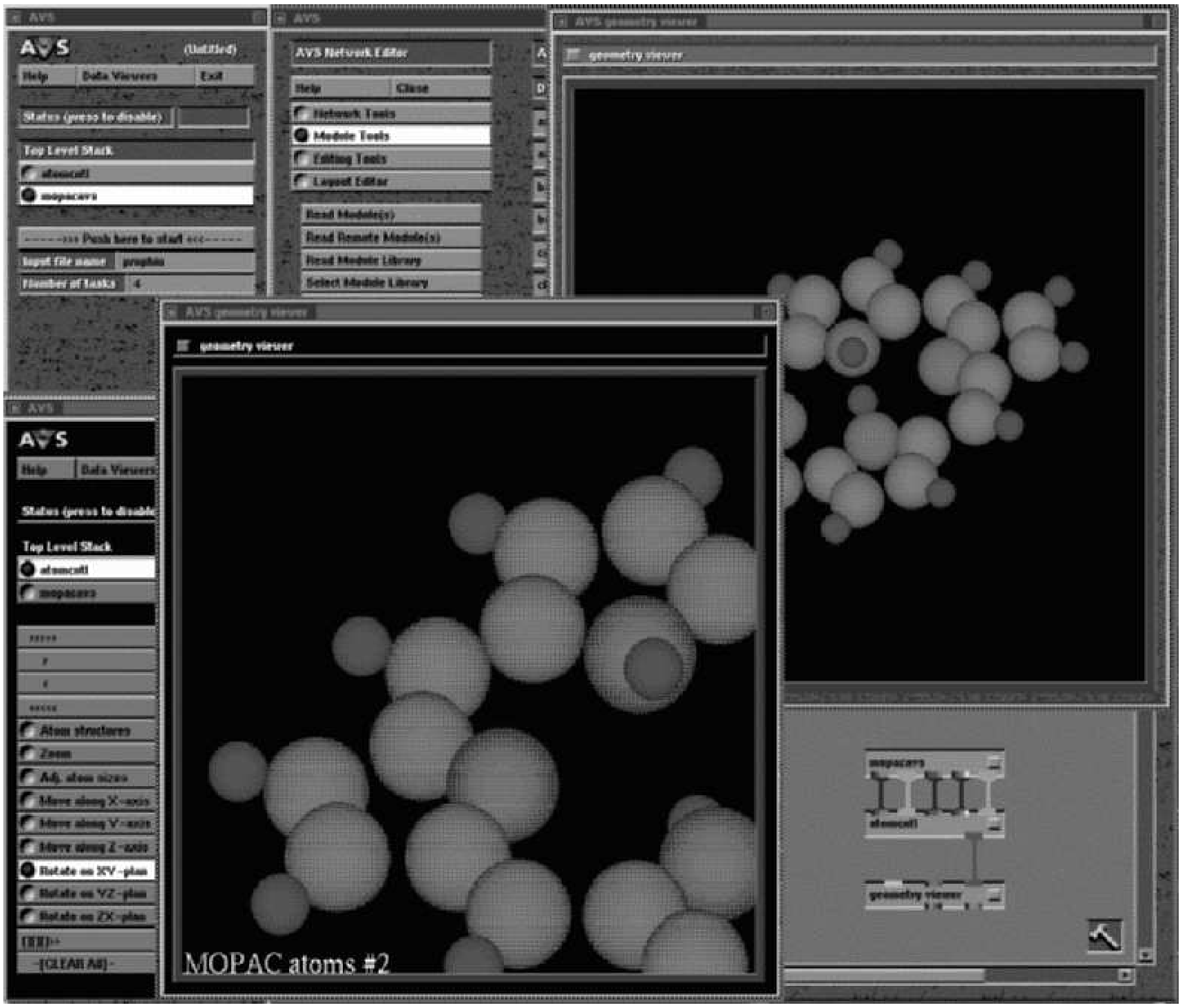


Figure 6.3: Parallel MOPAC AVS user interface

parameter "1:−1:0:0:filename" is used for the MOPAC sequential computation module to pass parameters to the MOPAC parallel computation module when parallel MOPAC is started by the graphic and above command line user interfaces. Special values "1:−1:0:0" is used to tell the MOPAC parallel computation module that it is run in reverse starting mode. In the reverse starting mode, the MOPAC parallel computation module is started by batch command spsubmit. The number of nodes to be used is controlled by the batch command. The batch command itself allocates nodes for parallel jobs only. Presently engaged, the MOPAC sequential computation module must share with one of these node. By the description in Section 4.3, the last node is used to host the MOPAC sequential computation module and the distribution in Figure 5.7(b) is used.

All important MOPAC results are written into disk files in ASCII text. Users may still examine results by checking those files. The molecule structures are also saved in disk files with extension ".atom." An AVS module mopacshow is provided to display the molecule structures files generated by MOPAC command line user interface.

6.2 Performance with Small Data Sets

At the time we finished the initial implementation of parallel MOPAC, we were using an Intel iPSC/860 and a Sun 4 workstation cluster to run the benchmarks. Although it was widely used in the early 1990's, Intel iPSC/860 has very little memory. There is 8 MB of real memory on each node. Virtual memory is not supported by Intel iPSC/860. More than 3 MB of real memory is used by the operating system which leaves less than 5 MB for applications. The MOPAC executable takes another 1 MB. Consequently, the maximum data size that we can run is very limited. Moreover, the computing power of Intel iPSC/860 nodes is relatively low compared with today's processors. The Sun 4 workstation cluster has virtual memory support and is capable of running larger data sizes. However, the slow processors and Ethernet still seriously limit the data size. In this connection, the "small" data files are actually the largest data size we can run on these machines.

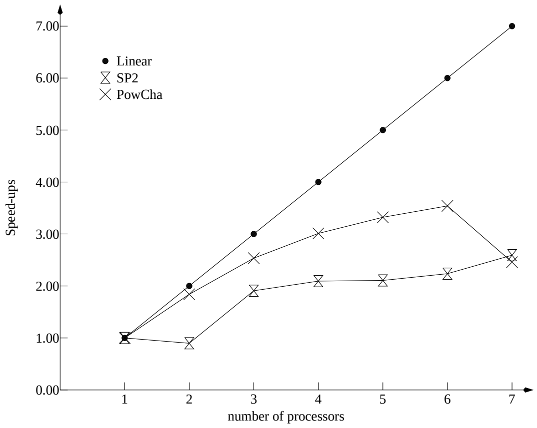
As better hardware became available, we were able to run larger data files. Access to an IBM SP2 and an SGI Power Challenge was obtained when parallel MOPAC was completed. The performance of parallel MOPAC running large data files is shown in Section 6.3. In the mean time, we re-ran the smaller data files to show the behavior of parallel MOPAC running smaller data files.

The data size of computation in parallel MOPAC is decided by the numbers of heavy and light atoms in the input molecule structure. It roughly equals to $4 \times number\_of\_heavy\_atoms + number\_of\_light\_atoms$. The 5 test data files have sizes between 100 and 200. The total execution time of the sequential version ranges from less than 1 minute to about 25 minutes on the 8-node SGI Power Challenge in NPAC. The raw results are shown in Appendix D. Figures 6.4, 6.5, and 6.6 show the speed-up of subroutines DENSIT, DIAG, and HQRII respectively. Figure 6.7 shows the overall speed-up of parallel MOPAC.

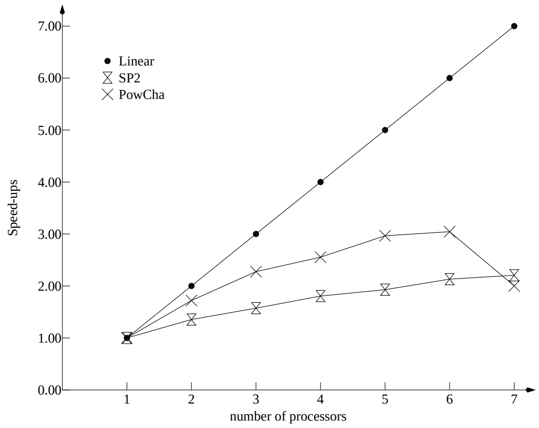
The computation statements in the main loop of subroutine DENSIT are simpler than those of subroutine DIAG and HQRII. Even though the load is balanced and the only communication is the initial data distribution and final result collection, the execution time saved by parallel execution is not significant. Moreover, since the data size is not large, the communication is, nevertheless, still a non-trivial portion. In all cases, the communication drags down the overall speed-up. Until then, the maximum speed-up ranges from 1.8 to 2.8 using 6 nodes.

Since the communication cost is still significant and the data sizes are small, the communication data chunk is small and the frequency is high. This communication pattern favors the machine with smallest communication latency. The SGI Power Challenge's shared memory architecture has shorter latency than IBM SP2's distributed memory architecture. It looks as if this is the reason why the SGI Power Challenge outperforms the IBM SP2.

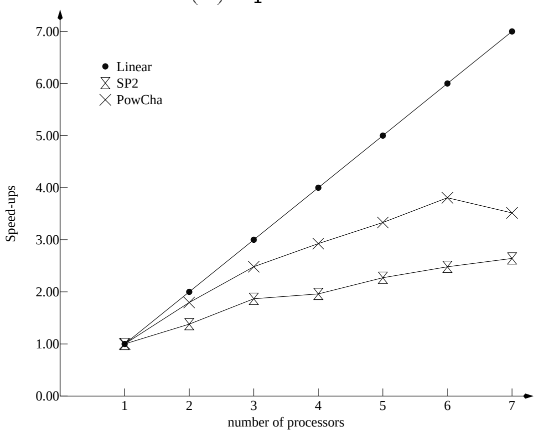
We can also observe an interesting phenomenon in Figure 6.4. The speed-up of the SGI drops sharply at 7 nodes. This is because the benchmarks are run on an 8 node SGI Power Challenge. The high volume of small data chunks fills up the bandwidth of the shared memory bus and degrades the performance. The same phenomenon



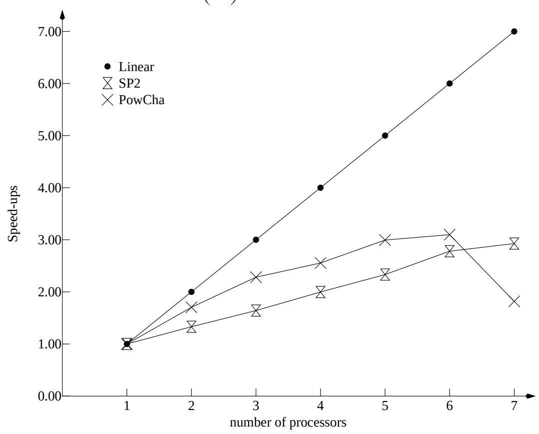
(a) apsbtest



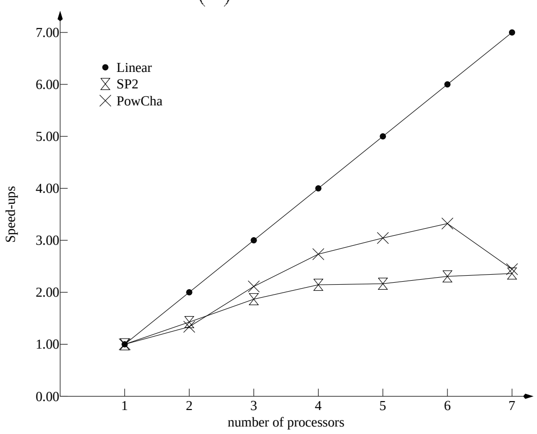
(b) chlorin



(c) metenk

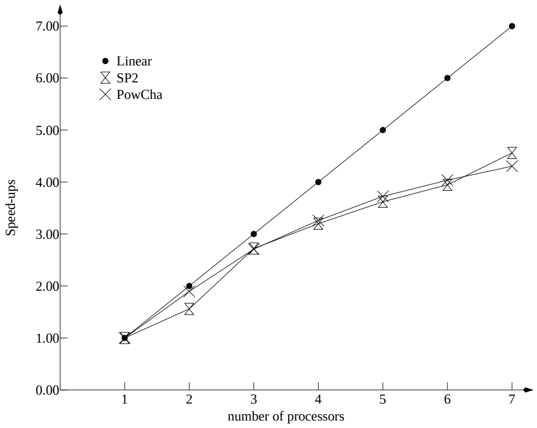


(d) porphin

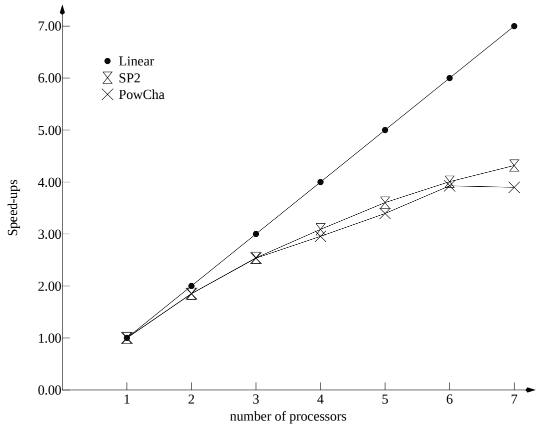


(e) tetrabenz

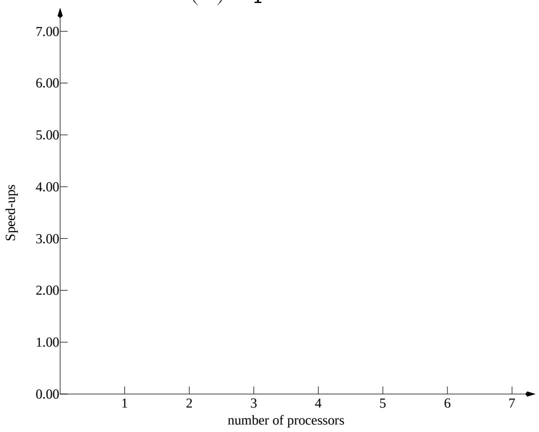
Figure 6.4: Speed-ups of subroutine DENSIT with small data sets



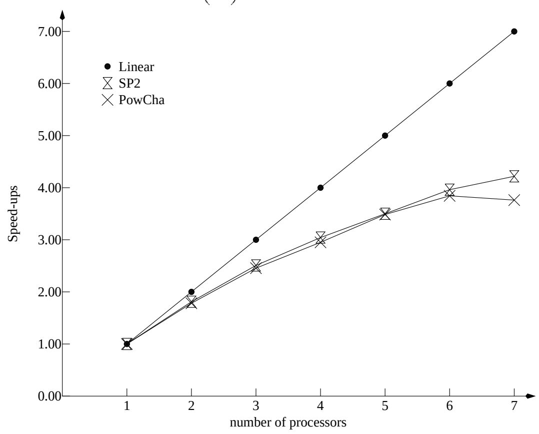
(a) apsbtest



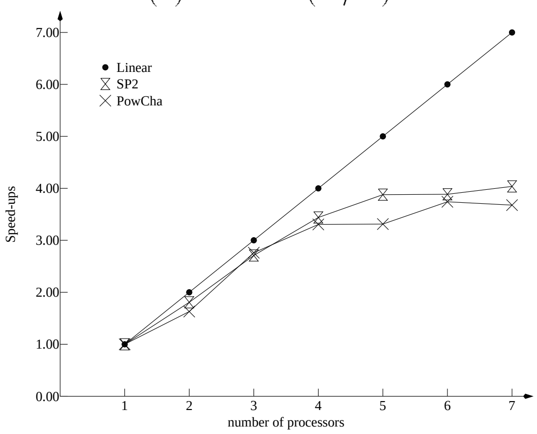
(b) chlorin



(c) metenk (N/A)

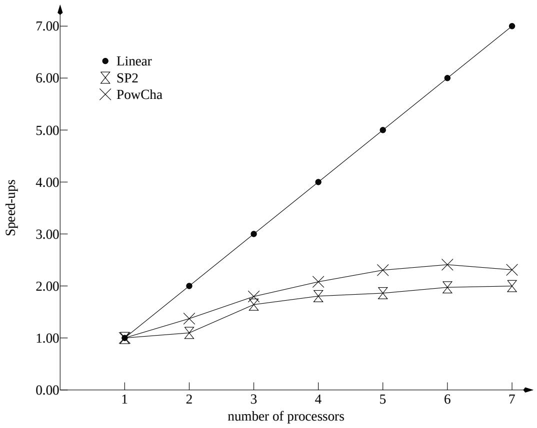


(d) porphin

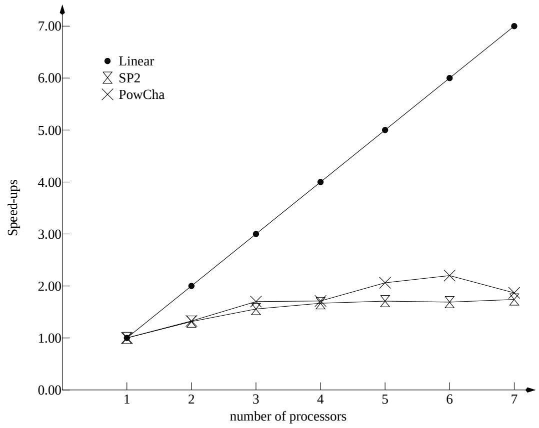


(e) tetrabenz

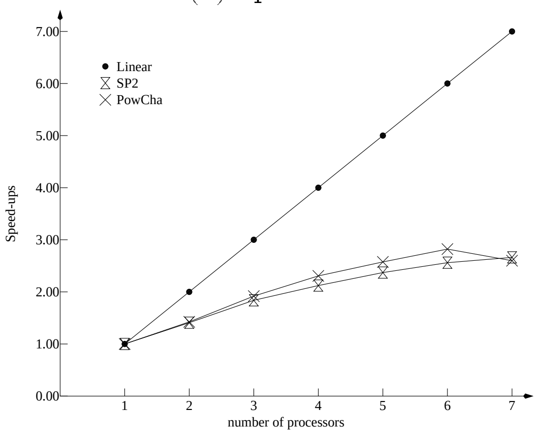
Figure 6.5: Speed-ups of subroutine DIAG with small data sets



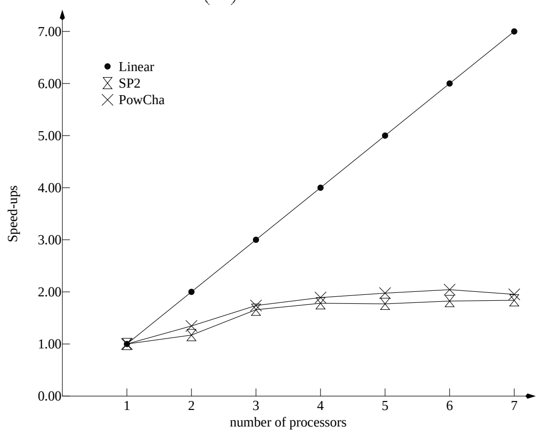
(a) apsbtest



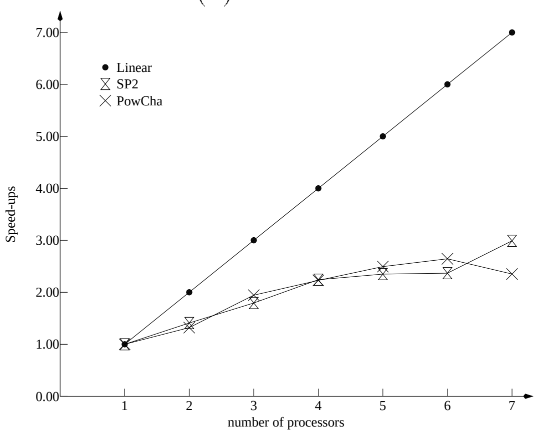
(b) chlorin



(c) metenk

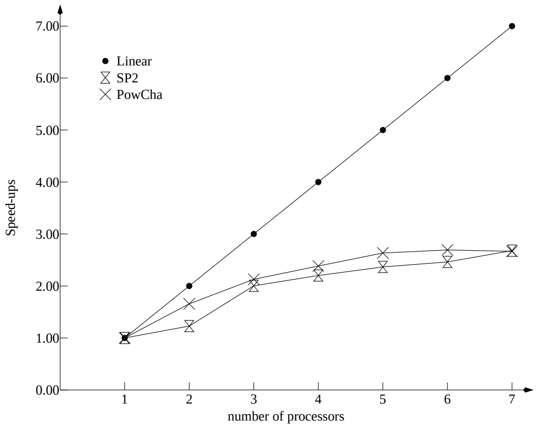


(d) porphin

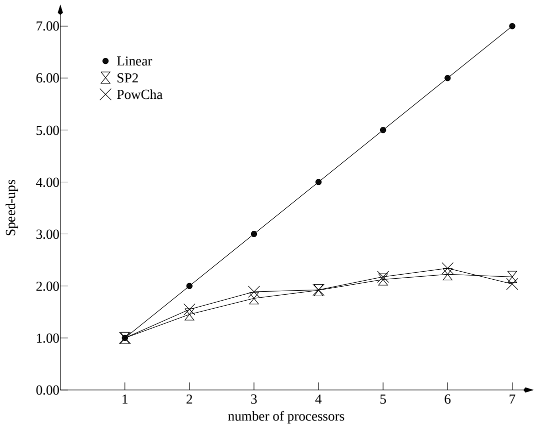


(e) tetrabenz

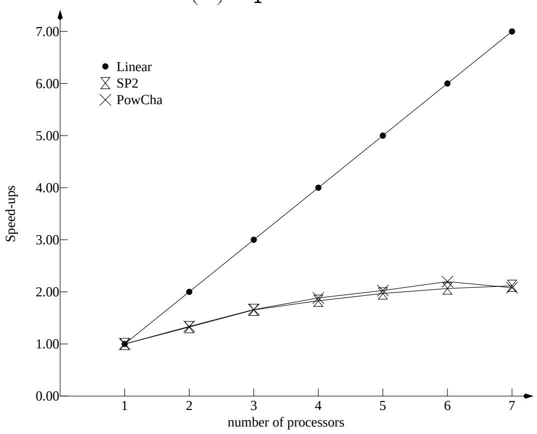
Figure 6.6: Speed-ups of subroutine HQRII with small data sets



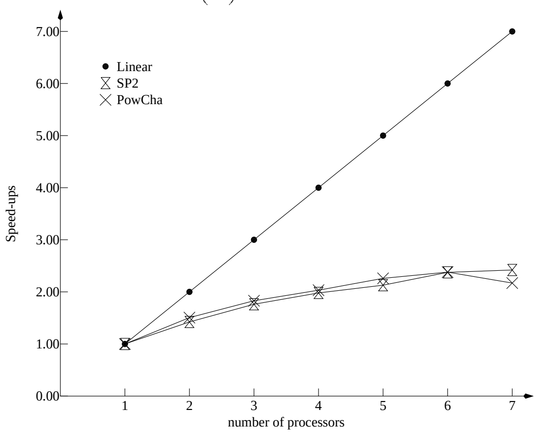
(a) apsbtest



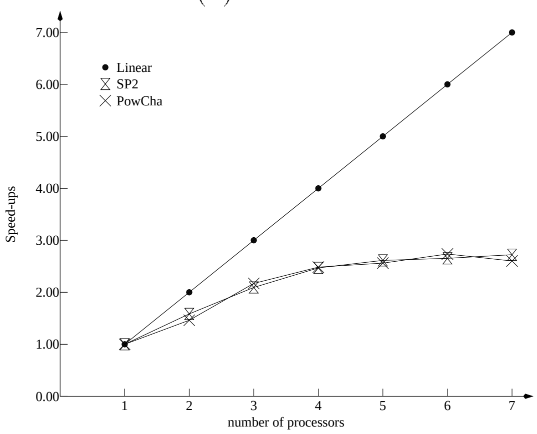
(b) chlorin



(c) metenk



(d) porphin



(e) tetrabenz

Figure 6.7: Speed-ups of parallel MOPAC with small data sets

happens to subroutine DIAG and HQRII, too. In short, since their computation stage is much longer than DENSIT's, the phenomenon is not as significant as in subroutine DENSIT.

The computation statements in the main loop of subroutine DIAG are more complicated than those of subroutine DENSIT. The communication factor does not affect the overall performance as much as that of subroutine DENSIT. The speed-up for SGI Power Challenge and IBM SP2 are about the same. The speed-up curves are also smoother than that of subroutine DENSIT. A summary indication, the maximum speed-up ranges from 3.7 to 4.5 at 6 or 7 nodes.

Subroutine HQRII has some small data communication in the middle of the computation. The overall performance is degraded somewhat. Like subroutine DENSIT, the small data chunk communication favor the machine with short communication latency. In contrast, the communication in the middle of computation also implies synchronization, which can not benefit from low latency communication. The SGI Power Challenge outperforms the IBM SP2, but not as significant as in subroutine DENSIT. The maximum speed-up ranges from 1.7 to 2.9.

It is more significant that the speed-up curves promptly drop at 2 nodes. The parallel version usually needs to do some extra initialization and communication to calculate and distribute the work load. The extra initialization is not needed in sequential programs. This extra work makes the speed-up curves show a small drop at 2 nodes. The drop in the speed-up curves is not significant for subroutine DENSIT and DIAG. However, since we need to do a matrix transformation to adapt PeIGS into HQRII in the beginning of the subroutine, the speed-up curve drop in subroutine HQRII is more significant than in the cases of subroutine DENSIT and DIAG.

Figure 6.7 shows the overall speed-up curves of parallel MOPAC. The speed-up is lower than individual parallelized subroutines because the speed-up of MOPAC includes non-parallelized parts.

Parallelization in each attempt does not benefit small data sets very much. The initial data distribution and final result collection destroy the performance improvement gained by parallelization. In fact, the calculation and initialization for work load distribution, although small, add up more overhead. The performance does not scale up very well and the speed-up curves reach the maximum at small number of nodes. That is, the results discussed above suggest that small data sets should use small number of processors or just use the sequential version because the parallel version does not run much faster. If parallel version is still preferred, apparently, a small machine with low latency communication should be used.

6.3 Performance with Big Data Sets

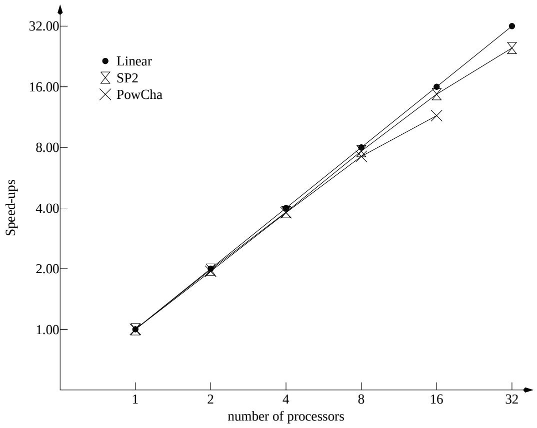
Figures 6.8, 6.9, and 6.10 show the speed-up curves of parallel MOPAC running big data sets. The given data sizes range from 265 to 1308. The benchmarks are run on a 16 node SGI Power Challenge and an 80 node IBM SP2.

Figure 6.8 shows the speed-up curves of subroutine DENSIT. As the data size gets big, the execution time tends to dominate the total execution. The communication time becomes negligible and the performance scales up much better. While the size of the data chunks becomes large the advantage of low latency communication of shared memory machine does not give much benefit. The IBM SP2 outperforms the SGI Power Challenge for the 3 bigger data sets. A speed-up of 25 is observed at 32 nodes. Higher speed-ups are possible if more nodes are used.

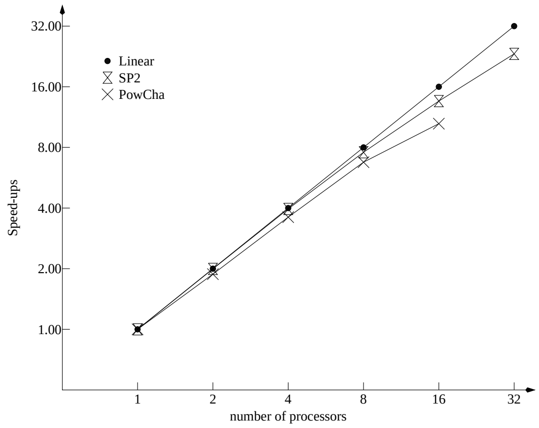
The performance drop when all nodes are used on SGI Power Challenge is still true, but it is improved for bigger data sets.

Like the discussion in previous section, subroutine DIAG favors high bandwidth over low latency. The IBM SP2 outperforms the SGI Power Challenge for all 5 data sets in Figure 6.9.

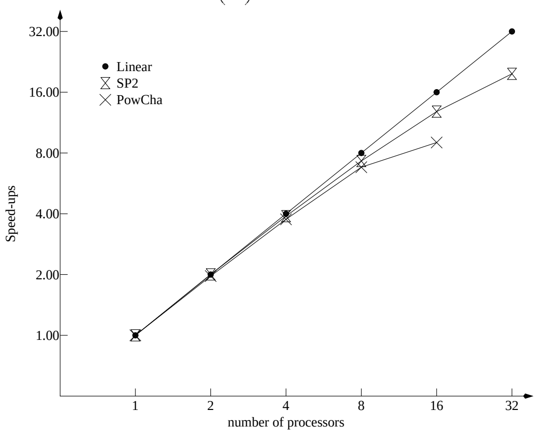
The communication in the middle of computation causes synchronization and reduces the speed-up. The maximum speed-up of subroutine HQRII at 32 nodes is only 14.7.



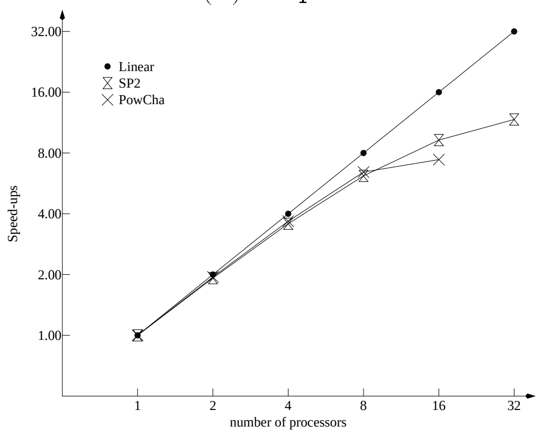
(a) 1crn



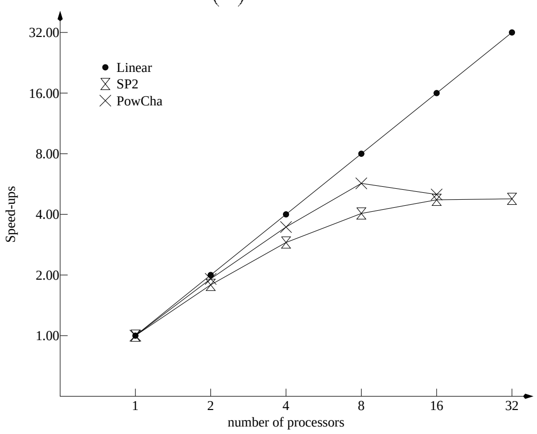
(b) vcop_4



(c) c60_3

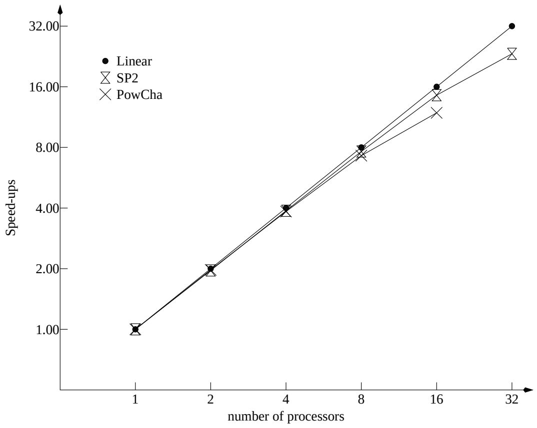


(d) c60_2

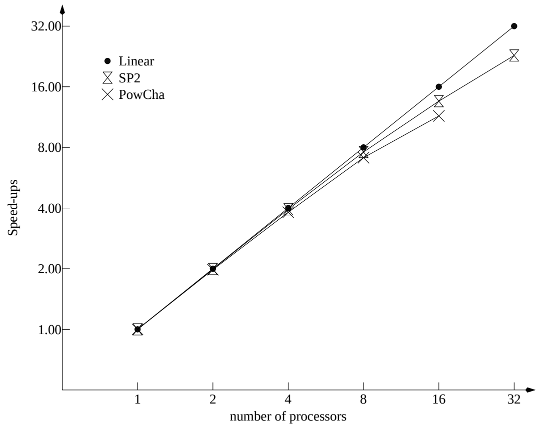


(e) porphyrn

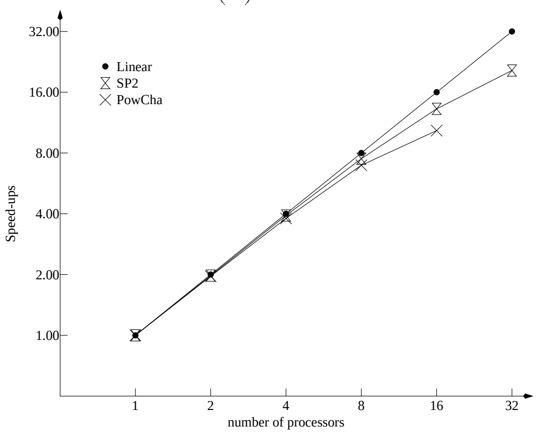
Figure 6.8: Speed-ups of subroutine DENSIT with big data sets



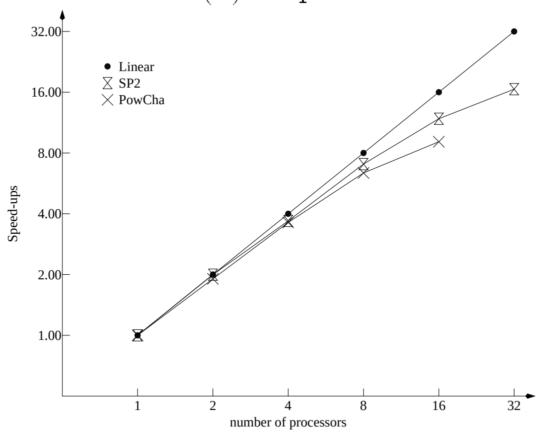
(a) 1crn



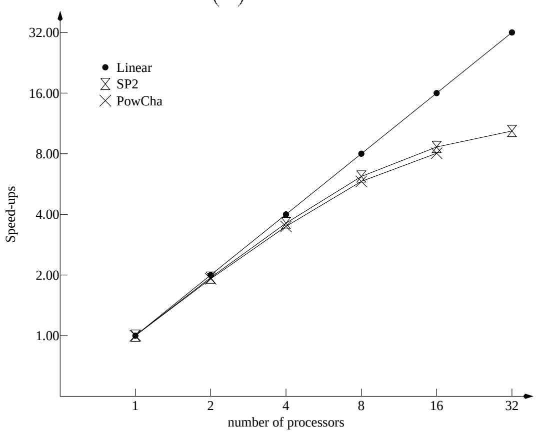
(b) vcop_4



(c) c60_3

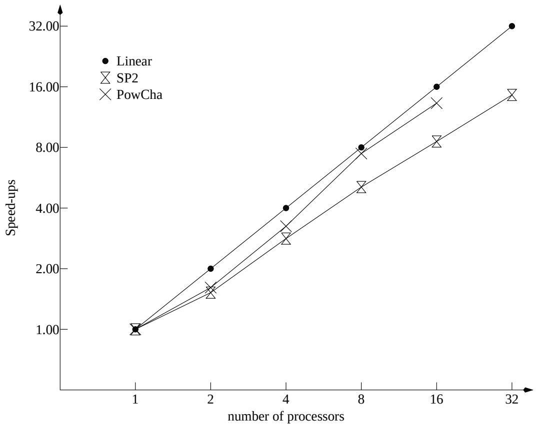


(d) c60_2

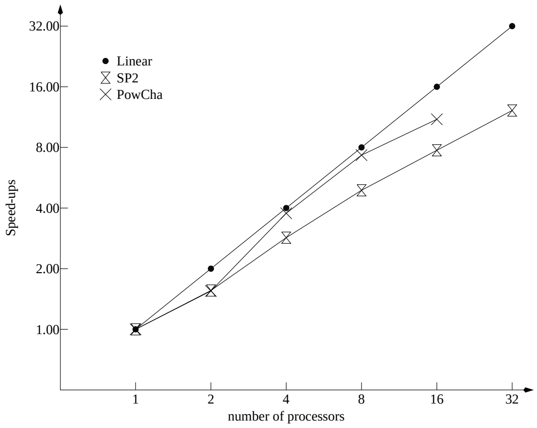


(e) porphyrn

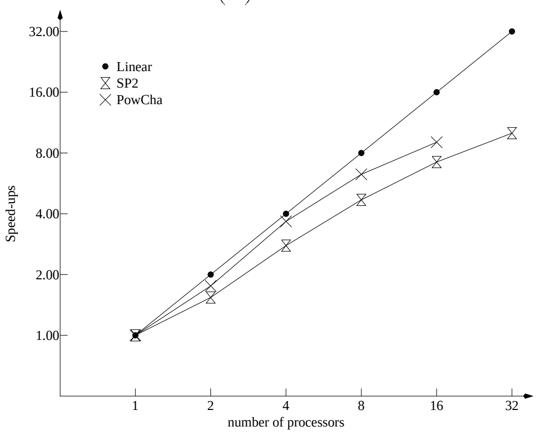
Figure 6.9: Speed-ups of subroutine DIAG with big data sets



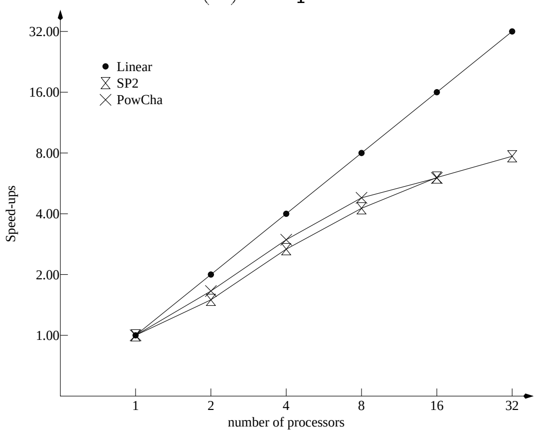
(a) 1crn



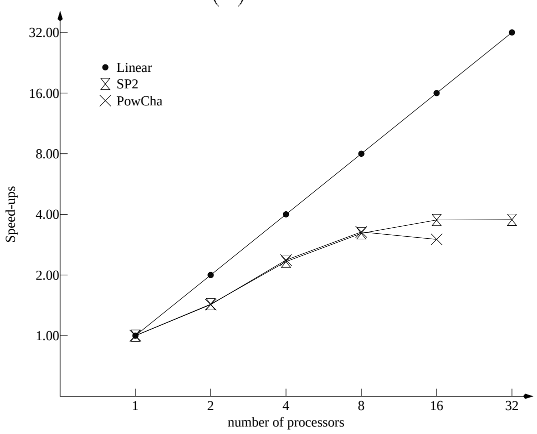
(b) vcop_4



(c) c60_3

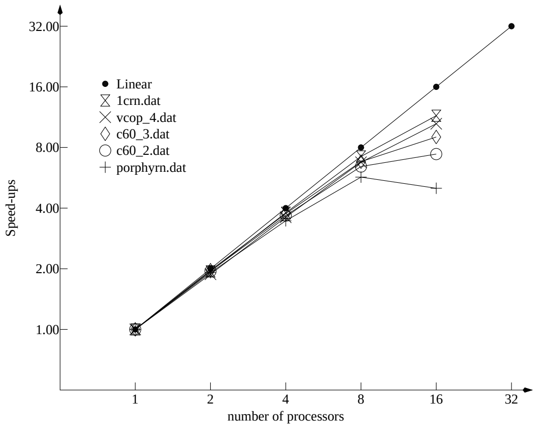


(d) c60_2

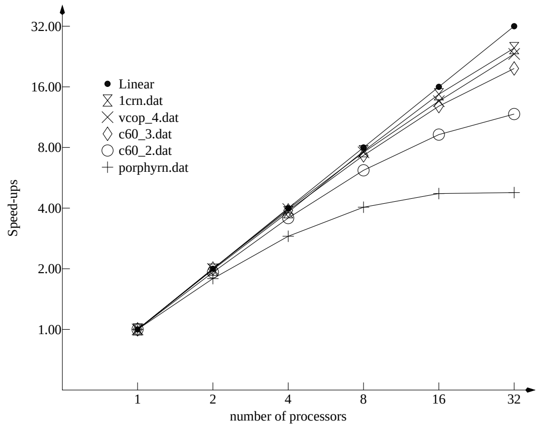


(e) porphyrn

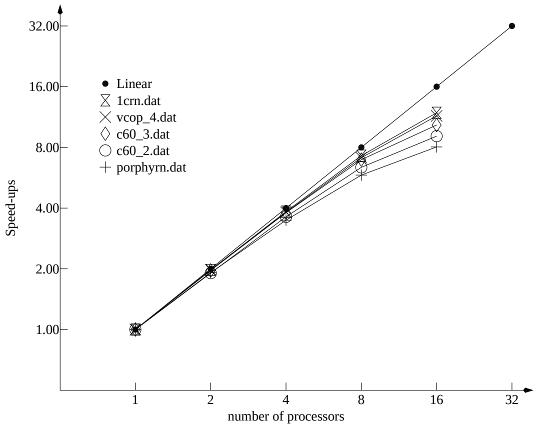
Figure 6.10: Speed-ups of subroutine HQRII with big data sets



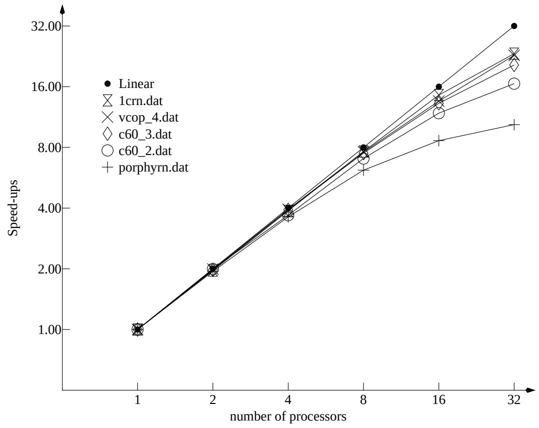
(a) densit/Power Challenge



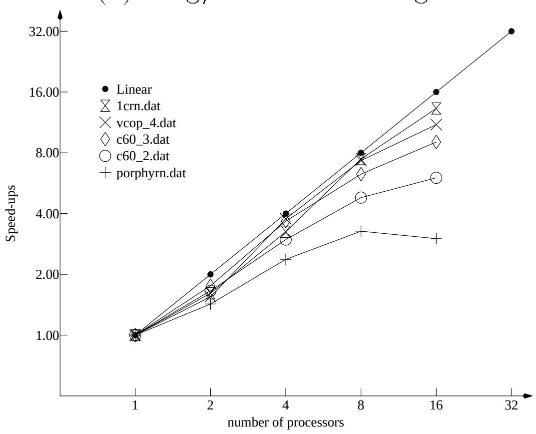
(b) densit/SP2



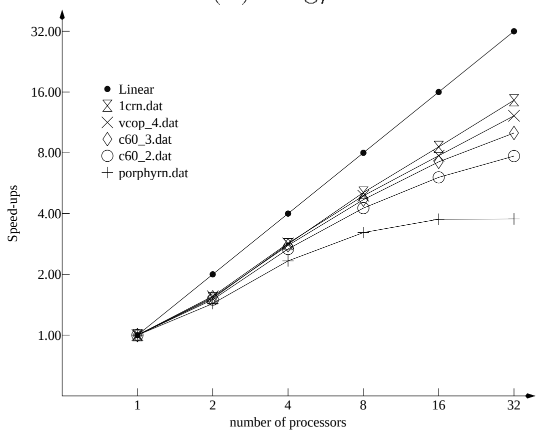
(c) diag/Power Challenge



(d) diag/SP2

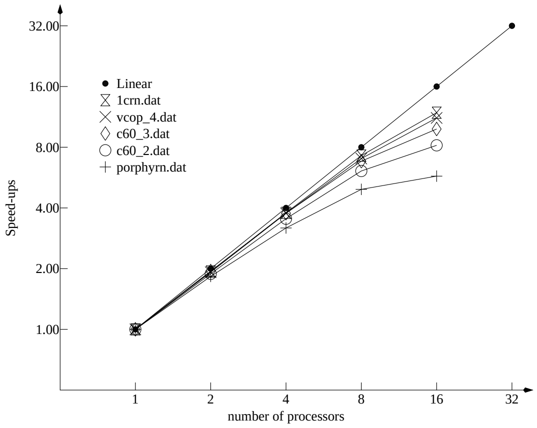


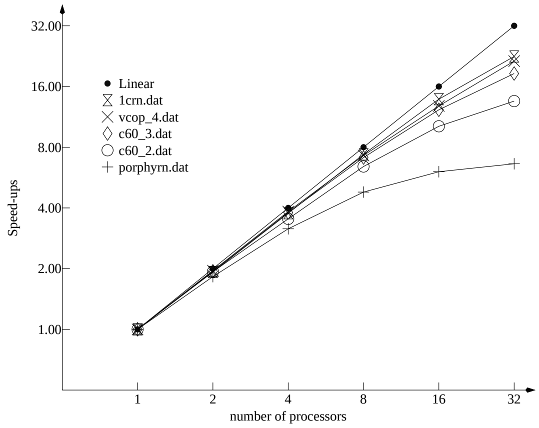
(e) hqrii/Power Challenge



(f) hqrii/SP2

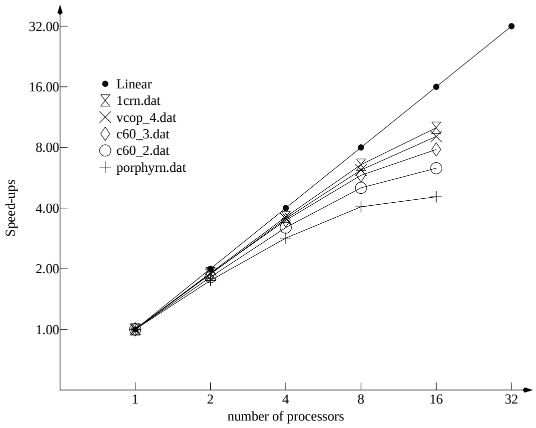
Figure 6.11: Speed-ups of subroutines/machines

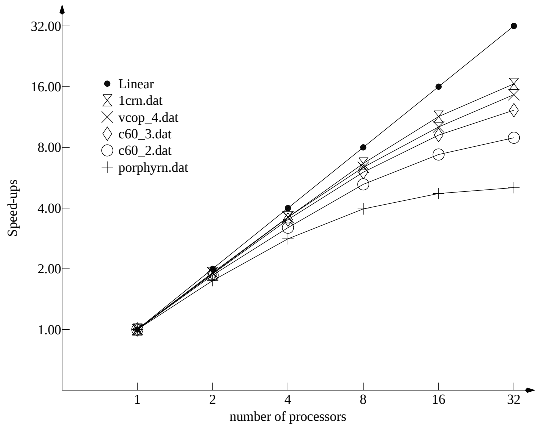




(a1) Power Challenge (a2) SP2

(a) Best case - Assuming the non-parallelable part is $O(1)$





(b1) Power Challenge (b2) SP2

(b) Worst case - Assuming the non-parallelable part is $O(N^2)$

Figure 6.12: Projected Speed-ups of parallel MOPAC

Since the big data benchmark is stopped at some cut points, we need to estimate the overall performance improvement. In this respect, since the highest computational complexity of the computationally intensive parts is $O(N^3)$, we assume the non-parallelized parts of MOPAC have computational complexity of O(1) and O(N^2). The projected overall speed-up is shown in Figure 6.12.

Chapter 7

Summary

The subject of this dissertation is to improve the performance of a legacy application by cogently parallelizing it in a computer scientist's way. Getting the knowledge of the area of the legacy application, parallelizing the code and keeping the legacy application validated are the most strategic topics of this dissertation. Although parallelizing and optimizing a program involve many technical issues, getting the knowledge of the area of the legacy application can be more difficult and time-consuming for a computer scientist. In essence, keeping the program validated is mandatory to a legacy application. In this dissertation, we developed a procedure for a computer scientist to parallelize legacy applications efficiently and correctly apply the procedure to a semi-empirical quantum chemistry package MOPAC.

It is not always possible for a computer scientist to be a domain expert of the field of the application he intends to work on. By following our procedure, one can work on a legacy application with only basic knowledge in that field. The sequential analyses, including time profiling analysis, program flow analysis and computational complexity analysis, provide a way for computer scientists to correctly identify the target subroutines to work on without the domain expertise. By combining the sequential analyses and the Amdahl's law, we can successfully estimate the performance for different input data sizes. The parallel analyses optimize the parallel code by resolving the data dependency, which maximize the parallelizable parts, and reducing the communication, which minimize the communication overhead. The applicable methods we used in these parallel analyses transform the parallel code in a systematic way that guarantee that the program semantics of the legacy application are not changed and the variables access are not troubled in the complex parallelization. The inter-module communication binds the sequential and parallel modules together and finally the graphic user interface gives users a meaningful view of intact scientific results.

The sequential and parallel analyses described in chapter 4 provide a systematic way for computer scientists to improve the performance of a legacy application with minimum domain expertise and keep the legacy application certified.

In the example of MOPAC shown in chapter 5, we identified 4 computational intensive parts in time profiling and keep only the 3 which will still be computational intensive in production. The computational intensive parts then can be parallelized and optimized directly by the methods described in chapter 4 or by using a pre-existed optimized parallel software. The data dependence problems are resolved. Thereafter, the parallel code is optimized to minimize the communication costs. The parallel and sequential modules are integrated by graphic user interface enabled AVS modules.

The benchmark shown in Chapter 6 shows the performance is improved as expected. The features of the performance curves are also discussed in Chapter 6. Small data shows less improvement due to the sequential part still has significant weight. As larger input data is used, the weight of sequential part becomes insignificant and better performance improvement is obtained.

The performance improvement makes interactive execution of MOPAC possible. The AVS graphic user interface provided by our parallel MOPAC enables the users to visualize the traditional text output for users interactively execute MOPAC in a visualized environment. A traditional command line user interface is also precisely provided for users not using graphic interface.

In addition to the IBM SP2/AIX and SGI Power Challenge/IRIX, parallel MOPAC has also been ported to Sun Sun4/Solaris, DEC Alpha/OSF1, IBM PC/Linux, IBM

PC/FreeBSD, and Thinking Machine CM5. Although we introduced about 10,000 lines of code, more than 70% of them are used for visualization and the inter-module communication. The replaced sequential code is less than 1,000 lines, which is about 3% of the whole sequential version. The rest of 97% code is kept unchanged and validated. Of heightened interest, the correctness of parallel MOPAC is confirmed by comparing the outputs with the outputs of sequential version.

Tried Approaches Before the parallel MOPAC implementation was finalized, we had also tried some different works. Some of them are replaced by more responsible methods while some are for other reasons. These works, although not included in the current implementation, take the approach for responsible investigation to not waste resource. Citing implications, some of them may still be picked up again in the future.

- For a given input data, each parallel module has its own optimized machine size. The best machine size for DENSIT may not be best for DIAG1. Moreover, a MOPAC input file can contain several molecules in different sizes. The communication library used in the initial implementation of parallel MOPAC was PVM. PVM was commonly used on workstation clusters. Above all, it has a very practical feature which allows the parallel program dynamically to change the number of nodes during execution. The optimized machine sizes can be calculated and reconfigured for each parallel module before they are called.

  We switched to MPI later to adopt PeIGS and port parallel MOPAC to more platforms. In most MPI implementations, the machine size is decided at the start-up and cannot be dynamically reconfigured. The code for dynamically machine size optimization has to be abandoned.

- In the results from early benchmark, small data was used due to slow machines we used. The results reinforced very high communication costs which seriously limit the scalability. To overcome this problem, we made some well-charted studies on the performance impact by computation-communication ratio of parallel computers and concluded a high computation-communication ratio is mandatory to parallel programs.

A parallel machine with high computation-communication ratio improves the scalability of a parallel program. Parallel programs with a large amount of frequently data communications can run slower than their sequential counterparts on parallel machines with a low computation-communication ratio. In this manipulation, large input data size or small machine size is not affected by the machine computation-communication ratio that much because the amount and frequency of communication compared to computation have been reduced. Moreover, for parallel programs having frequent communications, the communication latency affects the overall performance more than the computation-communication ratio. What emerges is that the previous conclusion hence needs to be corrected.

- Before we realized that the computational complexity can dramatically change the work load distribution pattern, the time profiling analysis shows DCART is a time-consuming part of MOPAC. Even though we found DCART has lower computational complexity and will not be time-consuming when large data is used, DCART still takes some work load for smaller data.

  Parallelizing DCART was attempted. About 50 common blocks with 100 global variables in 20 subroutines were analyzed for data dependency. Some global variables used in both sequential and parallel parts need to be synchronized. Certain subroutines used in both sequential and parallel parts need to be split into two versions. The data dependence problems of some variables are found and resolved by the methods described in Section 4.2.

  Our initial analysis shows subroutine DCART can be parallelized without technical problems. It warrants raising the question that the parallelization of subroutine DCART involves the rewriting of 11% of MOPAC code that may introduce bugs and invalidate the application. Because of this train of thought, the parallelization of subroutine DCART was then dropped.

Future Work Although parallel MOPAC delivers good performance improvement for small effort and proves our approach is a cost efficient way of improving performance for legacy applications, there are still some topics that can be worked on:

- Parallelizing legacy applications using a shared memory implementation. Traditionally shared memory machines can connect few nodes together, typically 8 or 16 maximum. Today, with the new hardware technology, large SMPs have been widely used as enterprise servers. Sun HPC 10000 [85] server can connect up to 64 UltraSPARC. IBM S80 Enterprise server [56] is a 24-way PowerPC RS64 III SMP. HP 9000 V2600 Enterprise Server [49] supports up to 32 PA-8600.

  The NUMA architecture further expands the number of processors a shared memory machine can connect. SGI has been working on its ccNUMA architecture for some time. The SGI Origin 2000 can connect up to 256 nodes and 512 mips R10000 processors with its ccNUMA architecture. In addition to SMP, HP also provides its ccNUMA based HP 9000 V2600 Enterprise Server which can connect up to 128 PA-8600 processors. IBM started adapting the NUMA architecture to AIX for PowerPC and Monterey for IA-64 in 1998. The first PowerPC and IA-64-based NUMA platforms are planned to be delivered in 2001.

  Shared memory machines have the favorable advantage of higher communication bandwidth and lower latency than distributed memory machines. Communicationally intensive applications can take advantage of shared memory machines. It would be interesting to do some research on the shared memory programming model.

- Web based user interface. Presently, network computing is the most rapidly growing topic of computer science. The enhancement is that with a web based user interface, users can use computer resources remotely. The Gateway System [47] developed at Northeast Parallel Architecture Center (NPAC) is a seamless interface designed enabling scientists and engineers to utilize High Performance Computing (HPC) resources like computational engines, software, and visualization systems. Users can use the browser based graphic user interface to access the remote HPC resources through a secure web server and object broker middleware. The Gateway System not only provides graphic user interface for accessing computational engines but also provides secure accesses on these HPC resources. By adapting the Gateway System, a secure "parallel MOPAC computation web site" can be setup on a computer center with large computation power to serve users anywhere on the internet securely.

- Determining and using the optimized number of nodes dynamically. Using too many processors to run a small data set could be slower than using a small number of processors. Likewise, we cannot know the unpredictable optimal number of processors before the data is read in. The initial implementation of parallel MOPAC used PVM. PVM allows dynamically configuring the size of the parallel virtual machine at run time. We could dynamically configure the machine size to optimize the execution time based on the size of input data. The MPI implementations we use are needed to define the size of the parallel machine when the program starts. The real optimal number of processors for a certain data set is determined, at this juncture, by the size of data, the communication pattern, and the characteristics of the communication subsystem.

- Subroutine DCART was picked by our time profiling analysis and dropped because increasing the input data size decreases the weight of subroutine DCART. However, subroutine DCART is still a time-consuming part for smaller molecules. In this special context, despite the high risk of changing a large portion of MOPAC may invalidate MOPAC, it may be worth it to parallelize subroutine DCART if the validation problem can be resolved.

Appendix A

Installing Parallel MOPAC

Requirements The current version of parallel MOPAC requires that MPI and PeIGS should be installed before installing MOPAC. Before beginning installation, the following should be kept in mind:

- Platform: Parallel MOPAC supports the following platforms:
  - IBM [345]86 PC cluster running Linux (both a.out and ELF)
  - IBM [345]86 PC cluster running FreeBSD
  - IBM RS6000 workstation cluster running AIX
  - IBM SP-2 running AIX
  - Sun workstation cluster running SunOS 3.x, 4.x
  - Sun workstation cluster running SunOS 5.x (Solaris)
  - DEC Alpha farm running OSF/1
  - SGI MIPS running IRIX
- Have MPI installed. MPI is a communication interface. Parallel MOPAC uses MPI to communicate among processes. There are at least four freely distributed implementations on the internet. Parallel MOPAC uses mpich. You can get

mpich via anonymous FTP from info.mcs.anl.gov:/pub/mpi/mpich.tar.Z. IBM and SGI MPI are also supported in parallel MOPAC.

- Have PeIGS installed. PeIGS is an optimized parallel eigensystem solver developed by PNNL. Parallel MOPAC takes advantage of PeIGS's optimized parallel eigensystem solvers' performance. PeIGS is distributed as part of NWCHEM. NWCHEM is available at:
  http://www.emsl.pnl.gov:2080/docs/nwchem/nwchem.html.

- Have GNU make installed. PeIGS requires GNU make version 3.68 or higher to install correctly.

The tar and gzip-ed parallel MOPAC source code is called mopac.<mmmyy>.tar.gz. Where "mmm" and "yy" are the month and year of the release date. Create a MOPAC home directory and unpack mopac.<mmmyy>.tar.gz in the MOPAC home directory. We will call the MOPAC home directory ${MOPAC_HOME} in the remaining portion of this document. Parallel MOPAC is distributed in gzip compressed tar file. You can use either one of the following commands to unpack it.

- `gtar xfz mopac.dec95.tar.gz`

- `gzcat mopac.dec95.tar.gz | tar xf -`

To justify this, two files and one directory will appear in the current directory. The directory structure looks like Figure A.1. Where configure is the configuration script, Makefile is the top level makefile of MOPAC. Directory Src holds the second level makefiles and all source files.

Configure parallel MOPAC As a sort of prelude, MOPAC needs to know the architecture and operating system of the target machine and where the required software PeIGS and MPI are. Shell script *configure*, which resides in the top level of the parallel MOPAC directory tree, can automatically detect the type of machine and operating system as well as get the correct settings for compiling parallel MOPAC.

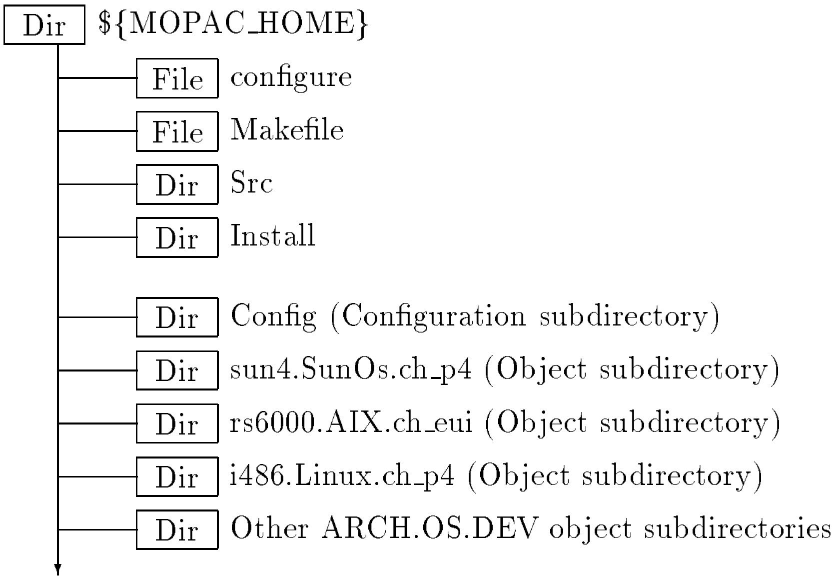


Figure A.1: MOPAC directory structure

It also attempts to find the required software in their normal places if, in short, they are not specified in the command line options of configure. Users can use command line options to force configure to use the settings users want parallel MOPAC to use.

You will need to change working directory to ${MOPAC_HOME} to run configure. The syntax of *configure* is as follow:

- Command :
  `configure [-p PeIGS-dir][-m MPI-dir][-d MPI-device][-i install-dir]`
  configure is a shell script. According to what is presumably the simplest interpretation, it creates a configuration subdirectory, *Config*, and generates a machine specific configuration file, *make.incl.MACHNAME* for making MOPAC. Where *MACHNAME* is the machine base name. e.g. if you are configuring MOPAC on machine nova.npac.syr.edu, you will get a configuration file make.incl.nova. *configure* also creates an object file directory, *ARCH.OS.DEV*, to hold object files. Where *ARCH* is the architecture name, *OS* is the operating

system name, and *DEV* is the MPI device name. e.g.. If you are configuring MOPAC on a IBM SP-2 using ch_eui, configure will create subdirectory rs6000.AIX.ch_eui for you.

- Options:

-p PeiGS-dir : The location of PeIGS. MOPAC will look for PeIGS libraries libpeigs.a, liblapack.a, and libblas.a in this directory. If you have PeIGS libraries installed in directory /foo/bar/Pnl/peigs2.1-SunOs, then, to be resourceful, you should use: -p /foo/bar/Pnl/peigs2.1-SunOs. *configure* will search for PeIGS in some default place if the -p option is not used. The default directories are:

1. ${MOPAC_HOME}/peigs-OS
2. ${MOPAC_HOME}/Peigs-OS
3. ${MOPAC_HOME}/PEIGS-OS
4. ${MOPAC_HOME}/peigs
5. ${MOPAC_HOME}/Peigs
6. ${MOPAC_HOME}/PEIGS
7. ${HOME}/Pnl/peigs2.1

PeIGS became the official eigensolver of Chemistry package NWCHEM developed by PNNL. The libraries were moved to new locations. The new locations depend on the platform that NWCHEM runs on. Environment variables *NWCHEM_TOP* and *NWCHEM_TARGET* must be specified to compile NWCHEM properly. The following new locations will be searched for PeIGS libraries before the traditional locations are searched.

1. /usr/local/lib/${NWCHEM_TARGET}
2. /usr/local/nwchem/lib/${NWCHEM_TARGET}
3. ${NWCHEM_TOP}/lib/${NWCHEM_TARGET}

-m MPI-dir : This option specifies the location of MPI. Some default locations will be searched if this option is not used.

1. ${HOME}/[mpi,mpich]
2. /usr/local/[mpi,mpich]
3. /usr/[mpi,mpich]
4. /usr/local
5. /usr

If MPI is not installed in the above locations, you need to specify it by -m option to force the configure script to look for it.

-d MPI-device : This option specifies the device MPI is using. configure shell script can determine the MPI device for most architectures automatically. This option is used only for machines supporting more than one MPI communication device and the configure shell script cannot determine MPI device correctly. For example, configure checks /dev/cmni for CM-5. If you are configuring and compiling MOPAC on a regular SUN 4 without a CM-5 NI installed, you need to specify "-d ch_cmmd" if you do not want to use default device. On this occasion, the default device of IBM SP-2 is ch_eui. If you want to use ch_p4 instead of ch_eui, you need to specify "-d ch_p4."

-i install-dir : This option specifies the directory where you want to put parallel MOPAC. The default directory is ${MOPAC_HOME}/Install.

--checkconfig : This option is used in MOPAC internally.

Compile parallel MOPAC After configure, you can just use make to compile parallel MOPAC.

- Command: `make [nodeprogram] [termprogram] [avsprogram] [nodeclean] [termclean] [avsclean] [clean]`
- All object files and the final executables will be kept in the object file directory. The executables are:

  mopacnode : MOPAC node program.

mopacterm : MOPAC host program for people that do not have AVS.

mopacavs : MOPAC host program for people that have AVS.

geomcntl : AVS module which controls the view of atom structures.

mopacshow : AVS module which displays static atom structures.

- Options:

  nodeprogram : generate mopacnode.

  termprogram : generate mopacterm.

  avsprogram : generate mopacavs, geomcntl, and mopacshow.

  cleannode : clean mopacnode and associated object files.

  cleanterm : clean mopacterm.

  cleanavs : clean AVS associated files.

  clean : clean all object files.

  If no option is given, mopacnode and mopacterm are generated.

Install parallel MOPAC You can use "make install" to install parallel MOPAC. "make install" will install the intended parallel MOPAC modules into the destination directory specified in *configure*.

Appendix B

Data Visualization and AVS Parallel Modules

The increasing power of supercomputers and graphic systems has made it possible for scientific and engineering users to use real-time interactive graphic visualization on their data and results. In the areas as diverse as fluid dynamics, molecular modeling, geophysics, and other computer-aided engineering, scientists and engineers no longer need to use their imagination to figure out what their data looks like from numbers and tables. It is now possible to show the physical meaning of those numbers and tables by the means of graphics. Two dimensional graphics show flat pictures. Three dimensional graphics enable users to see data preferably from different viewpoints. Of value, animations even show the progress and change of data over time.

Visual data requires three processes to be able to display on screen.

Filtering It may be difficult to show the raw data on a screen. For example, the air flow analysis over the wings of an airplane contains one million data points. It is impossible to display all one million data points on screen. For practical reasons, some data points must be merged or removed before producing the graph.

Mapping The data can be represented in either geometric or image representations.

Geometric representation uses triangles, lines, spheres, and other geometric primitives to represent the data while image representation uses colored pixels to display two dimensional pictures.

Rendering The geometric image representations must be converted into the graphic languages or use a graphic software to display the pictures on graphic devices like screens or printers.

This brings us to the programming for displaying graphic data which is complicated and time-consuming. For it is the contention that scientists and engineers would rather use the familiar number and table representation if they would otherwise need to spend a lot of time on graphical programming. Application Visualization System (AVS) is a visualization system which provides powerful but easy to use programming tools and libraries for graphic programming [2].

AVS Introduction

From this frame of constructs, one of the most important features of AVS is its user interface. All this refers to the fact that the AVS user interface is designed to be easy to use such that the users can concentrate on their specialty without traditionally wasting much time on graphical programming. AVS users can namely construct their visualization applications by combining software components into executable flow networks, which consist of modules and links, to implement specific functions in the visualization processes.

The flow networks are built from an elemental menu of modules by using a visual programming interface called the AVS Network Editor. The AVS system allows users to dynamically connect software modules to create data flow networks for scientific computations. The modules are the AVS computational units. Modules pass data of mutually agreed upon types to each other. AVS supports a wide range of ready-to-use standard modules. Many general purpose applications can be constructed by using only the standard modules. Users can build their own customized modules in C or FORTRAN to meet their needs [1].

AVS can display visualization data in either a pixel-based method or a geometry-based method. A pixel-based visualization strategy is simply taking a raw data value and translating it into a number that represents a color. AVS accomplishes this translation with a table lookup called a colormap. Colormaps can be defined, saved, and retrieved by users. AVS also provides a Colormap Editor tool to generate colormaps easily and quickly.

Pixel-based visualization shows only two dimensional data. Higher dimensional data needs to be mapped to two dimensions in order to be displayed. Such data can be converted to pixels by slicing or blending. Slicing makes a two dimensional cross-section through a three dimensional block of pixels. If a three dimensional block of pixels is passed through a colormap whose auxiliary field contains opacity or transparency data, pixels can be blended along the line of sight. By following this procedure, it produces a two-dimensional picture of what appears to be in three-dimensional space.

As a general proposition, a geometry-based visualization strategy maps the raw data into the vertices of geometric objects. The values of data are used to assign colors to the vertices by using the AVS colormap. There are several techniques to create geometric descriptions from raw data. For example, an atom of a molecule can be potentially represented as a sphere. Color and transparency can be used to represent the type of atom.

AVS supports three viewer subsystems to render visual data on the screen.

Image Viewer subsystems The AVS Image Viewer subsystem is an interactive tool for displaying, manipulating, and processing images.

Geometry Viewer subsystems The AVS Geometry Viewer is an interactive tool for manipulating and viewing one or more three dimensional objects of the fundamental AVS data type geometry.

Graph Viewer subsystems The AVS Graph Viewer subsystem is an interactive tool for creating XY or contour plots of data.

Users can use the three viewers interactively from the AVS main control panel. The three viewers also exist as AVS built-in modules and can be connected to users' AVS networks in the AVS Network Editor.

AVS Modules

The module is the AVS computational unit. Each module can perform its function independently. Several modules can also work together to achieve a more complex function. Modules take typed data as inputs and produce typed data as output. For the most part, modules specify to the AVS kernel what data inputs they expect to receive from other modules and data outputs they send out to other modules. User interface parameters are also defined by modules to allow user interactions.

There are two kinds of modules, subroutine modules and coroutine modules. A subroutine module acts like a subroutine. Its computation function is invoked by AVS whenever its inputs or parameters change. AVS maintains a run queue. The flow executive activates a subroutine module only when it is in the run queue. A module is put into the run queue when one of its inputs or parameters has been modified. A coroutine module executes independently. The anticipation is that it obtains inputs from AVS and sends outputs to AVS as needed. Coroutine modules are mainly used in simulations or animations.

User AVS modules are implemented as separate UNIX programs. They communicate with the AVS kernel using the Berkeley UNIX socket mechanism and shared memory. UNIX domain sockets are persistently used where possible for efficiency. TCP domain sockets are used on remote modules or where UNIX domain sockets cannot be used.

Besides primitive data such as integers, floating points, and strings, AVS supports the following aggregate data types:

Field One, two and three dimensional grids of numbers with scaler values or vectors, integer or floating point values at each grid point. The grids can be:

- **Uniform** the grids are regular
- **Rectilinear** the grids' physical spaces are orthogonal but the distances between grid points can be variables.
- **Irregular** no restriction on the correspondence between computational space and physical space.

Unstructured Cell Data geometric objects composed of discrete cells with associated data.

Molecular Data Type molecular and quantum structures.

Images Two dimensional pictures.

Geometric Data Three dimensional objects that describe the geometry.

An AVS module communicates with other modules via parameters and ports. A parameter is a variable that has a constant value during an invocation of a module. AVS users can change the value of parameters by manipulating their associated widgets. A widget is a virtual input device such as a dial or a push button. Besides parameters, a module can have zero or more input and output ports. Like arguments of a subroutine, a port is a channel through which data pass to or from other modules.

When an AVS network is constructed, modules are connected to each other via ports, though it is not necessary for all input and out ports of a module to be connected to other modules. AVS ports are color coded by its data type. Ports with the same data type can be connected to pass data from one module to another. Several modules can be connected to perform more powerful functions. An AVS network is used to configure modules together into a strong visualization application. An AVS Network Editor subsystem is ultimately used to help users create AVS networks.

AVS communications

The purpose of constructing a network is to provide a data processing pipeline. The output of one module becomes an input of another particularly in regard at each step.

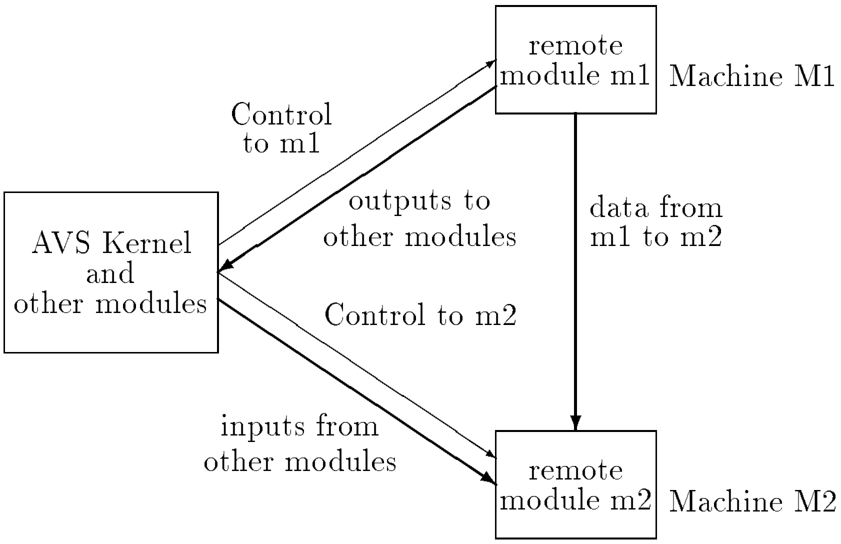


Figure B.1: AVS Directly Module Communication

Data enters AVS, flows through the modules of a network, and finally is rendered on a display.

AVS modules, as described in the previous section, are activated only when one of their parameters or input ports is changed. AVS uses a flow executive to supervise data movement and to invoke modules in the correct order. AVS uses a remote procedure call mechanism to establish communications between modules. When a module starts, AVS creates a new process in which that module runs. AVS also sets up a connection between the module and the AVS kernel by using Berkeley UNIX socket mechanism.

AVS supports remote execution of modules. An AVS network can distribute its modules on several different machines. These machines may or may not have the same architecture. AVS uses eXternal Data Representation (XDR) format to pass data between the machines of a heterogeneous network.

All data passing between modules is controlled by the AVS kernel. Remote module m1 needs to send its output data back to the AVS kernel and the AVS kernel re-sends the data out to the down stream remote module m2. To reduce the communication overhead, AVS supports Direct Module Communication (DMC). As shown in Figure B.1, suppose an AVS network contains a remote module m1 running on machine M1 and remote module m2 running on machine M2. An output port of m1 connects to an input port of m2. An additional direct module communication socket between m1 and m2 will be created by AVS to send data directly from m1 to m2 without sending data through the AVS kernel. AVS performs these operations of socket communication transparently to users.

Remote modules are coarse grained programs which are distributed on several hosts and work cooperatively. The communication between modules is controlled, but not transfered, by the AVS kernel. The AVS DMC provides convenient socket communication between the remote modules running in parallel. Users can easily create connections between modules by clicking on the ports of the icons of the modules. The communication is transparent to users even if the modules are run on heterogeneous hosts. The AVS kernel is also smart enough to use shared memory to make the connections if both modules are on the same host.

Appendix C

Computational Complexity of Subroutine DHC

The computational complexity of subroutine DCART cannot be determined by just analyzing the consonant loop structure of the subroutine itself. There are subroutine calls inside of loops. The computational complexity of a loop with subroutine calls equals the product of the computational complexity of the loop and the computational complexity of the subroutines called inside the loop. The computational complexity of subroutine DCART cannot be determined without knowing the computational complexity of the subroutines called inside the loop.

The most complicated subroutine called by subroutine DCART is subroutine DHC. Figure C.1 shows that subroutine DHC is called inside of an $O(n^2)$ two-level DO-loop of DCART. The computational complexity of the two-level DO-loop is $O(n^2 \times computational\ complexity\ of\ DHC)$. We will break into these subroutines in "depth first" order to find out their computational complexity and derive the overall computational complexity of subroutine DHC.

Subroutine JAB There is a two-level DO-loop in subroutine JAB. The outer loop has constant lower bound and upper bound of 1 and 4. The higher bound of inner

- SUBROUTINE DHC
  - SUBROUTINE FOCK2
    - SUBROUTINE JAB
    - SUBROUTINE KAB
  - SUBROUTINE H1ELEC
    - SUBROUTINE DIAT
      - SUBROUTINE DIAT2
        - SUBROUTINE SET
          - SUBROUTINE AINTGS
          - SUBROUTINE BINTGS
      - SUBROUTINE COE
      - FUNCTION SS
        - SUBROUTINE BFN
      - SUBROUTINE GOVER
  - FUNCTION HELECT
  - SUBROUTINE ROTATE
    - SUBROUTINE REPP
  - SUBROUTINE SOLROT
    - SUBROUTINE ROTATE
      - SUBROUTINE REPP

Figure C.1: Subroutine DHC calling tree

loop is the index of the outer loop. The two-level loop will iterate a fixed number of $\frac{(1+4)\times 4}{2} = 10$ times. The computational complexity of this subroutine is $O(1)$.

Subroutine KAB There are two two-level DO-loops in subroutine KAB. Both two-level DO-loops iterate a fixed number of $4 \times 4 = 16$ times. The computational complexity of this subroutine is $O(1)$.

Subroutine FOCK2 Besides the loops with constant lower bound and upper bound, this subroutine has some DO-loops with variable bounds. A one-level DO-loop with variable $MAXORB$ upper bound makes the computational complexity of subroutine FOCK2 at least $O(n)$. However, this $O(n)$ loop is inside of an "IF" structure which is executed only when FOCK2 is first time called. This $O(n)$ loop should not be included in the established computational complexity analysis since it is used only once in the whole program execution.

A two-level DO-loop with input argument $NUMAT$ as the upper bound of the outer loop and the index of the outer loop as the upper bound of the inner loop produces a $O(NUMAT^2)$ computational complexity. Since $NUMAT$ is a foreseen argument, we need to find out what is passed in from the caller of subroutine FOCK2. Subroutine FOCK2 forms the two-electron two-center repulsion part of the Fock matrix. It is called in two subroutines. The input argument $NUMAT$ equals the number of atoms when subroutine FOCK2 is called from subroutine ITER and makes the computational complexity of subroutine FOCK2 $O(n^2)$. However, when subroutine FOCK2 is called by subroutine DHC in a derivative, it equals a small fix number 2.

Although subroutine FOCK2 has some loops related to the input molecule size, they are not in effect in the way called from DHC. Therefore, the computational complexity of subroutines FOCK2 itself is $O(1)$. The computational complexity of the subroutines called by FOCK2 are both $O(1)$. By now, the overall computational complexity of this subroutine is $O(1)$.

Subroutine AINTGS There is one one-level DO-loop perceived in subroutine AINTGS. The upper bound of the DO-loop is the number of types of the overlaps between atomic orbitals of atoms. There are six of them. The DO-loop iterates maximum of six times. The computational complexity of this subroutine is $O(1)$.

Subroutine BINTGS There is one two-level DO-loop in subroutine BINTGS. The outer loop iterates a maximum number of six times. The inner loop has a fixed upper bound of 6, 7, 12, or 15 depending on an input argument. No matter what value the input argument is, the upper bound of this loop is a small constant. The computational complexity of this subroutine is $O(1)$.

Subroutine SET This subroutine has no loop at all. The computational complexity of the subroutines it calls are both $O(1)$. The computational complexity of this subroutine is $O(1)$.

Subroutine DIAT2 Like subroutine BINTGS, this subroutine has only one loop with constant bounds. The computational complexity of the subroutines it calls is $O(1)$. In accordance with this, the overall computational complexity of this subroutine is $O(1)$.

Subroutine COE This subroutine has only one loop with constant bounds. The computational complexity of this subroutine is $O(1)$.

Subroutine BFN Like subroutine BINTGS, the outer loop of the only two-level DO-loop in subroutine BFN iterates a maximum number of six times and the inner loop has a fixed upper bound of 6, 7, 12, and 15 depends on a input argument. Both inner and outer loops are $O(1)$. The computational complexity of this subroutine is $O(1)$.

Function SS The DO-loops in function SS have either fixed bounds or small variable upper bounds. The computational complexity of the subroutine it calls is $O(1)$. The computational complexity of this subroutine is $O(1)$. However, the six-level DO-loop with maximum 13 iterations for each level may produce up to $13^6 = 4826809$ iterations and take a lot of CPU time. In any event, this is an important reason that makes subroutine DCART look like a computationally intensive subroutine in the time profile of MOPAC.

Subroutine GOVER There are two two-level DO-loops in another two-level DO-loop. This produces two four-level loops with constant number of $4 \times 4 \times 6 \times 6 = 576$ iterations. The computational complexity of this subroutine is $O(1)$.

Subroutine DIAT The DO-loops in subroutine DIAT have either fixed bounds or small variable upper bounds. The computational complexity of the subroutines it calls are all $O(1)$. The computational complexity of this subroutine is $O(1)$.

Subroutine H1ELEC All DO-loops in subroutine H1ELEC have either fixed bounds or small variable upper bounds except for two two-level DO-loops. The two two-level DO-loops have variables for their upper bounds. Fortunately, in such a distinction, the two two-level DO-loops are both applied on a 9×9 matrix. The maximum number of iterations the two two-level DO-loops are 81. The computational complexity of the subroutines it calls are all $O(1)$. The overall computational complexity of this subroutine is $O(1)$.

Function HELECT There is one two-level DO-loop in function HELECT. The upper bound of the inner loop depends on the index of the outer loop. The upper bound of the outer loop N is an argument of subroutine HELECT. Like subroutine FOCK2, we need to examine what values are passed in. In addition to subroutine DHC, subroutine HELECT is called by subroutine ITER and DERI1, too. Argument N is passed with $NORBS$, the number of atomic orbitals, when HELECT is called in

ITER and DERI1 and has computational complexity $O(n^2)$. However, when called in subroutine DHC, HELECT is used to calculate the energy contributions from those pairs of atoms that have been moved by subroutine DERIV. The value of N is derived from $NATORB$, which is the number of atomic orbitals per atom. The maximum possible value of $NATORB$ is 9, which makes the computational complexity of the two-level DO-loop and subroutine HELECT $O(1)$.

Subroutine REPP The DO-loops in subroutine REPP have fixed bounds. The computational complexity of this subroutine is $O(1)$.

Subroutine ROTATE The four DO-loops in subroutine ROTATE have fixed bounds. The computational complexity of the subroutine it calls is $O(1)$. The computational complexity of this subroutine is $O(1)$.

Subroutine SOLROT The DO-loops in subroutine DIAT have either fixed bounds or small variable upper bounds. The computational complexity of the subroutine it calls is $O(1)$. The computational complexity of this subroutine is $O(1)$.

Subroutine DHC The DO-loops in subroutine DHC have either fixed bounds or small variable upper bounds. The computational complexity of all subroutines it calls is $O(1)$. Therefore, the overall computational complexity of subroutine DHC is $O(1)$.

Appendix D

Results of Parallel MOPAC

This appendix contains tables of time and speed-up of running parallel MOPAC by using different data sets. The time unit in the tables is second. The time shown in the small data sets is the total time of the specific parallel modules while the time shown in the big data sets is the average time of one instance of call to the parallel modules. The results in the first column (one node) are measured by the sequential MOPAC. The benchmark of small data files is run on a 8-node SGI Power Challenge and a 80-node IBM SP2. The benchmark of small data files is run on a 16-node SGI Power Challenge and a 80-node IBM SP2. To prevent shared memory interference from other processes in SGI Power Challenge machines, the machines were reserved for exclusive use when running benchmark.

The data files used for benchmark are listed in Figure D.1.

legend	data file name	number of light atoms	number of heavy atoms	data size $n_{light} + 4 \times n_{heavy}$
a	apsbtest.dat	38	25	138
b	chlorin.dat	16	24	112
c	metenk.dat	30	41	194
d	porphin.dat	14	24	110
e	tetrabenz.dat	20	42	188

(a) Small data files

legend	data file name	number of light atoms	number of heavy atoms	data size $n_{light} + 4 \times n_{heavy}$
A	1crn.dat	0	327	1308
B	vcop_4.dat	279	169	955
C	c60_3.dat	0	180	720
D	c60_2.dat	0	120	480
E	porphyrn.dat	33	58	265

(b) Big data files

Table D.1: Data files used for benchmark

unit: sec		Number of nodes						
		1	2	3	4	5	6	7
a	SP2	304.411	339.272	159.552	145.435	144.453	136.136	117.446
	PowCha	196.484	106.864	77.488	65.265	59.169	55.507	79.931
b	SP2	11.967	8.841	7.607	6.626	6.204	5.616	5.424
	PowCha	5.608	3.260	2.463	2.195	1.891	1.842	2.801
c	SP2	16.444	11.916	8.806	8.385	7.239	6.625	6.225
	PowCha	8.446	4.702	3.401	2.886	2.535	2.218	2.403
d	SP2	62.272	46.813	37.920	31.176	26.693	22.403	21.278
	PowCha	26.219	15.388	11.494	10.265	8.757	8.463	14.422
e	SP2	192.419	134.876	103.009	89.735	88.869	83.396	81.478
	PowCha	135.358	101.319	64.097	49.520	44.449	40.735	55.371

(a) run time

		Number of nodes						
		1	2	3	4	5	6	7
a	SP2	1.000	0.897	1.908	2.093	2.107	2.236	2.592
	PowCha	1.000	1.839	2.536	3.011	3.321	3.540	2.458
b	SP2	1.000	1.354	1.573	1.806	1.929	2.131	2.206
	PowCha	1.000	1.720	2.276	2.555	2.966	3.045	2.002
c	SP2	1.000	1.380	1.867	1.961	2.272	2.482	2.642
	PowCha	1.000	1.796	2.483	2.926	3.331	3.807	3.515
d	SP2	1.000	1.330	1.642	1.997	2.333	2.780	2.927
	PowCha	1.000	1.704	2.281	2.554	2.994	3.098	1.818
e	SP2	1.000	1.427	1.868	2.144	2.165	2.307	2.362
	PowCha	1.000	1.336	2.112	2.733	3.045	3.323	2.445

(b) speed-up

Table D.2: Subroutine DENSIT run time and speed-ups with small data sets

unit: sec		Number of nodes						
		1	2	3	4	5	6	7
a	SP2	796.259	511.074	292.599	249.008	219.945	201.816	174.623
	PowCha	564.587	297.926	208.437	173.169	151.613	139.908	131.189
b	SP2	24.829	13.411	9.771	8.036	6.888	6.197	5.752
	PowCha	17.894	9.665	7.057	6.055	5.270	4.558	4.591
c	SP2	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	PowCha	0.000	0.000	0.000	0.000	0.000	0.000	0.000
d	SP2	119.026	65.711	47.461	39.121	34.014	30.044	28.217
	PowCha	86.004	48.262	35.030	29.141	24.677	22.382	22.860
e	SP2	813.644	450.142	300.387	236.582	209.818	209.370	201.550
	PowCha	607.140	372.497	219.820	183.664	183.281	162.258	165.133

(a) run time

		Number of nodes						
		1	2	3	4	5	6	7
a	SP2	1.000	1.558	2.721	3.198	3.620	3.945	4.560
	PowCha	1.000	1.895	2.709	3.260	3.724	4.035	4.304
b	SP2	1.000	1.851	2.541	3.090	3.605	4.007	4.317
	PowCha	1.000	1.851	2.535	2.955	3.395	3.926	3.898
c	SP2	NA	NA	NA	NA	NA	NA	NA
	PowCha	NA	NA	NA	NA	NA	NA	NA
d	SP2	1.000	1.811	2.508	3.043	3.499	3.962	4.218
	PowCha	1.000	1.782	2.455	2.951	3.485	3.843	3.762
e	SP2	1.000	1.808	2.709	3.439	3.878	3.886	4.037
	PowCha	1.000	1.630	2.762	3.306	3.313	3.742	3.677

(b) speed-up

Table D.3: Subroutine DIAG run time and speed-ups with small data sets

unit: sec		Number of nodes						
		1	2	3	4	5	6	7
a	SP2	112.896	102.863	68.763	62.548	60.635	57.188	56.474
	PowCha	88.278	64.331	49.109	42.430	38.298	36.635	38.197
b	SP2	2.231	1.696	1.433	1.338	1.306	1.320	1.282
	PowCha	1.802	1.359	1.061	1.053	0.874	0.819	0.965
c	SP2	68.340	48.532	37.178	32.227	28.821	26.699	25.669
	PowCha	57.940	40.696	30.214	25.123	22.509	20.541	22.301
d	SP2	3.918	3.348	2.364	2.200	2.215	2.149	2.131
	PowCha	2.941	2.186	1.694	1.557	1.489	1.440	1.507
e	SP2	14.884	10.554	8.297	6.639	6.334	6.286	4.976
	PowCha	11.518	8.718	5.922	5.167	4.616	4.355	4.899

(a) run time

		Number of nodes						
		1	2	3	4	5	6	7
a	SP2	1.000	1.098	1.642	1.805	1.862	1.974	1.999
	PowCha	1.000	1.372	1.798	2.081	2.305	2.410	2.311
b	SP2	1.000	1.315	1.557	1.667	1.709	1.690	1.740
	PowCha	1.000	1.326	1.699	1.712	2.062	2.200	1.867
c	SP2	1.000	1.408	1.838	2.121	2.371	2.560	2.662
	PowCha	1.000	1.424	1.918	2.306	2.574	2.821	2.598
d	SP2	1.000	1.170	1.657	1.781	1.769	1.823	1.838
	PowCha	1.000	1.345	1.737	1.890	1.976	2.042	1.952
e	SP2	1.000	1.410	1.794	2.242	2.350	2.368	2.991
	PowCha	1.000	1.321	1.945	2.229	2.495	2.644	2.351

(b) speed-up

Table D.4: Subroutine HQRII run time and speed-ups with small data sets

unit: sec		Number of nodes						
		1	2	3	4	5	6	7
a	SP2	1367.938	1114.397	682.985	621.592	578.152	555.458	510.527
	PowCha	969.034	585.026	455.363	406.419	367.736	359.898	363.343
b	SP2	47.602	32.771	27.008	24.820	22.401	21.390	21.885
	PowCha	31.987	20.645	16.929	16.609	14.691	13.666	15.693
c	SP2	99.589	75.229	60.186	54.412	50.576	48.242	47.053
	PowCha	79.758	59.708	48.034	42.336	39.360	36.299	38.371
d	SP2	229.426	161.305	130.152	115.871	107.769	96.618	94.800
	PowCha	143.974	95.618	78.701	70.853	63.712	60.486	66.376
e	SP2	1159.408	731.598	553.803	469.579	443.620	436.824	425.806
	PowCha	859.377	587.956	395.903	345.839	335.464	313.960	330.130

(a) run time

		Number of nodes						
		1	2	3	4	5	6	7
a	SP2	1.000	1.228	2.003	2.201	2.366	2.463	2.679
	PowCha	1.000	1.656	2.128	2.384	2.635	2.693	2.667
b	SP2	1.000	1.453	1.763	1.918	2.125	2.225	2.175
	PowCha	1.000	1.549	1.889	1.926	2.177	2.341	2.038
c	SP2	1.000	1.324	1.655	1.830	1.969	2.064	2.117
	PowCha	1.000	1.336	1.660	1.884	2.026	2.197	2.079
d	SP2	1.000	1.422	1.763	1.980	2.129	2.375	2.420
	PowCha	1.000	1.506	1.829	2.032	2.260	2.380	2.169
e	SP2	1.000	1.585	2.094	2.469	2.614	2.654	2.723
	PowCha	1.000	1.462	2.171	2.485	2.562	2.737	2.603

(b) speed-up

Table D.5: Total run time and speed-ups with small data sets

unit: sec		Number of nodes					
		1	2	4	8	16	32
A	SP2	432.344	218.564	113.306	56.430	29.437	17.359
	PowCha	164.185	84.680	43.309	22.757	14.276	NA
B	SP2	178.345	89.421	45.069	23.576	13.136	7.655
	PowCha	60.147	32.025	16.692	8.905	5.736	NA
C	SP2	74.994	37.437	19.267	10.267	5.848	3.804
	PowCha	22.481	11.439	5.989	3.306	2.494	NA
D	SP2	11.610	6.047	3.253	1.885	1.253	0.991
	PowCha	6.209	3.194	1.691	0.965	0.837	NA
E	SP2	1.117	0.625	0.385	0.277	0.236	0.234
	PowCha	0.894	0.467	0.258	0.157	0.178	NA

(a) run time

		Number of nodes					
		1	2	4	8	16	32
A	SP2	1.000	1.978	3.816	7.662	14.687	24.905
	PowCha	1.000	1.939	3.791	7.215	11.500	NA
B	SP2	1.000	1.994	3.957	7.565	13.577	23.297
	PowCha	1.000	1.878	3.603	6.754	10.485	NA
C	SP2	1.000	2.003	3.892	7.304	12.823	19.717
	PowCha	1.000	1.965	3.754	6.801	9.013	NA
D	SP2	1.000	1.920	3.569	6.159	9.263	11.714
	PowCha	1.000	1.944	3.672	6.436	7.417	NA
E	SP2	1.000	1.786	2.898	4.039	4.722	4.774
	PowCha	1.000	1.913	3.460	5.697	5.019	NA

(b) speed-up

Table D.6: Subroutine DENSIT run time and speed-ups for big data sets

unit: sec		Number of nodes					
		1	2	4	8	16	32
A	SP2	462.818	236.085	119.434	60.631	31.872	19.851
	PowCha	317.223	160.885	82.575	43.601	26.733	NA
B	SP2	188.194	94.464	47.775	24.811	13.870	8.230
	PowCha	124.316	62.888	32.636	17.492	10.853	NA
C	SP2	76.651	38.805	19.550	10.225	5.792	3.739
	PowCha	50.344	25.661	13.281	7.245	4.877	NA
D	SP2	21.404	10.728	5.808	3.038	1.811	1.292
	PowCha	14.270	7.498	3.944	2.242	1.570	NA
E	SP2	3.353	1.727	0.927	0.544	0.388	0.323
	PowCha	2.223	1.161	0.638	0.381	0.277	NA

(a) run time

		Number of nodes					
		1	2	4	8	16	32
A	SP2	1.000	1.960	3.875	7.633	14.521	23.315
	PowCha	1.000	1.972	3.842	7.276	11.867	NA
B	SP2	1.000	1.992	3.939	7.585	13.568	22.866
	PowCha	1.000	1.977	3.809	7.107	11.454	NA
C	SP2	1.000	1.975	3.921	7.496	13.234	20.498
	PowCha	1.000	1.962	3.791	6.949	10.323	NA
D	SP2	1.000	1.995	3.686	7.046	11.817	16.562
	PowCha	1.000	1.903	3.618	6.365	9.088	NA
E	SP2	1.000	1.941	3.617	6.168	8.639	10.366
	PowCha	1.000	1.915	3.485	5.829	8.039	NA

(b) speed-up

Table D.7: Subroutine DIAG run time and speed-ups for big data sets

unit: sec		Number of nodes					
		1	2	4	8	16	32
A	SP2	567.300	373.266	201.211	111.643	66.334	38.937
	PowCha	814.361	505.417	250.637	109.269	61.399	NA
B	SP2	222.215	142.709	78.000	45.361	28.728	18.221
	PowCha	291.704	188.225	77.465	39.845	26.442	NA
C	SP2	94.487	61.405	33.979	20.160	13.104	9.425
	PowCha	110.436	62.884	30.124	17.618	12.215	NA
D	SP2	29.858	19.930	11.186	7.016	4.941	3.874
	PowCha	28.719	17.336	9.655	5.987	4.777	NA
E	SP2	5.698	3.979	2.446	1.770	1.519	1.515
	PowCha	4.702	3.296	1.982	1.438	1.565	NA

(a) run time

		Number of nodes					
		1	2	4	8	16	32
A	SP2	1.000	1.520	2.819	5.081	8.552	14.570
	PowCha	1.000	1.611	3.249	7.453	13.263	NA
B	SP2	1.000	1.557	2.849	4.899	7.735	12.195
	PowCha	1.000	1.550	3.766	7.321	11.032	NA
C	SP2	1.000	1.539	2.781	4.687	7.211	10.025
	PowCha	1.000	1.756	3.666	6.268	9.041	NA
D	SP2	1.000	1.498	2.669	4.256	6.043	7.707
	PowCha	1.000	1.657	2.974	4.797	6.011	NA
E	SP2	1.000	1.432	2.330	3.219	3.751	3.760
	PowCha	1.000	1.426	2.372	3.269	3.005	NA

(b) speed-up

Table D.8: Subroutine HQRII run time and speed-ups for big data sets

		Number of nodes					
		1	2	4	8	16	32
A	SP2	1.000	1.924	3.764	7.370	13.838	22.531
	PowCha	1.000	1.935	3.782	7.267	11.856	NA
B	SP2	1.000	1.954	3.834	7.278	12.809	21.380
	PowCha	1.000	1.920	3.761	7.030	11.170	NA
C	SP2	1.000	1.940	3.793	7.106	12.258	18.551
	PowCha	1.000	1.944	3.762	6.818	9.845	NA
D	SP2	1.000	1.929	3.530	6.417	10.156	13.544
	PowCha	1.000	1.884	3.534	6.098	8.174	NA
E	SP2	1.000	1.825	3.151	4.792	6.042	6.622
	PowCha	1.000	1.830	3.180	4.944	5.746	NA

(a) Best case - Assuming the non-parallelable part is $O(1)$

		Number of nodes					
		1	2	4	8	16	32
A	SP2	1.000	1.895	3.596	6.653	11.367	16.511
	PowCha	1.000	1.905	3.612	6.569	10.015	NA
B	SP2	1.000	1.912	3.602	6.367	10.092	14.598
	PowCha	1.000	1.881	3.538	6.181	9.068	NA
C	SP2	1.000	1.887	3.505	6.023	9.204	12.227
	PowCha	1.000	1.892	3.478	5.820	7.810	NA
D	SP2	1.000	1.858	3.197	5.246	7.373	8.928
	PowCha	1.000	1.817	3.200	5.039	6.309	NA
E	SP2	1.000	1.746	2.815	3.960	4.722	5.049
	PowCha	1.000	1.749	2.837	4.057	4.549	NA

(b) Worst case - Assuming the non-parallelable part is $O(N^2)$

Table D.9: Projected overall speed-ups for big data based on Table 5.1

Bibliography

[1] Advanced Visual Systems Inc., 300 Fifth Ave., Waltham, MA 02154. *AVS Developer's Guide*. Release 4, May 1992.

[2] Advanced Visual Systems Inc., 300 Fifth Ave., Waltham, MA 02154. *AVS User's Guide*. Release 4, May 1992.

[3] S. G. Akl. *The Design and Analysis of Parallel Algorithms*. Prentice Hall, Englewood Cliffs, NJ, 1989.

[4] R. Alasdair, A. Bruce, J. G. Mills, and G. Smith. CHIMP/MPI User Guide. Technical Report EPCC-KTP-CHIMP-V2-USER.1.2, Edinburgh Parallel Computing Centre, 1994.

[5] R. J. Allen and I. J. Bush. Parallel Application Software on High Performance Computers: Parallel Diagonalisation Routines. Technical report, The CCLRC HPCI Centre at Daresbury Laboratory, August 1996. soft copy available at http://www.dl.ac.uk/TCSC/Subjects/Parallel_Algorithms/diags/diags.ps.

[6] I. Angus, G. Fox, J. Kim, and D. Walker. *Solving Problems on Concurrent Processors*, volume II. Prentice Hall, Englewood Cliffs, NJ, 1990.

[7] K. K. Baldridge. Promises and Perils of Parallel Semiempirical Quantum Methods. In T. G. Mattson, editor, *Parallel Computating in Computational Chemistry*. Chapter 8, pages pp. 97-113. American Chemical Society, 1995.

[8] K. K. Baldridge. Parallel Implementation of Semiempirical Quantum Methods for the Intel Platforms. *Journal of Mathematical Chemistry*, 19:pp. 87–109, 1996.

[9] R. J. Bartlett, N. Y. Öhrn, G. D. Purvis III, and M. C. Zerner. Florida School on Applied Molecular Orbital Theory. Lecture Notes, May 1990.

[10] D. Basak and D. K. Panda. Designing Clustered Multiprocessor Systems under Packaging and Technological Advancements. *IEEE Transactions on Parallel and Distributed Systems*, 7(9):pp. 962–978, 1996.

[11] D. J. Becker, T. Sterling, D. Savarese, B. Fryxell, and K. Olson. Communication Overhead for Space Science Applications on the Beowulf Parallel Workstation. In *Proceedings of the 1995 International Conference on Parallel Processing*, Pentagon City, VA, August 1995.

[12] Y. Beppu and I. Ninomiya. HQRII: A Fast Diagonalization Subroutine. *Computers and Chemistry*, 6(2):pp. 87–91, 1982.

[13] H. J. Bernstein and M. Goldstein. Parallel Implementation of Bisection for the Calculation of Eigenvalues of Tridiagonal Symmetric Matrices. *Computing*, 37:pp. 85–91, 1986.

[14] B. H. Besler, K. M. Merz, Jr., and P. A. Kollman. Atomic Charges Derived from Semiempirical Methods. *Journal of Computational Chemistry*, 11(4):pp. 431–439, 1990.

[15] R. C. Bingham, M. J. S. Dewar, and D. H. Lo. Ground States of Molecules. XXVI. MINDO/3. Calculations for Hydrocabons. *Journal of the American Chemical Society*, 97(6):pp. 1294–1301, March 1975.

[16] L. S. Blackford, J. Choi, A. Cleary, A. Petitet, R. C. Whaley, J. Demmel, I. Dhillon, K. Stanley, J. Dongarra, S. Hammarling, G. Henry, and D. Walker. ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers - Design Issues and Performance. In *Supercomputing '96 Proceedings CD-ROM*, Pittsburgh, PA, November 1996.

[17] S. H. Bokhari. Communication Overhead on the Intel iPSC/860 Hypercube. Technical Report NASA Contractor Report: ICASE Interim Report No. 10, NASA Langley Research Center, May 1990.

[18] G. Burns, R. Daoud, and J. Vaigl. LAM: An Open Cluster Environment for MPI. Technical report, Ohio Supercomputer Center, undated.

[19] R. Butler and E. Lusk. User's Guide to the P4 Parallel Programming System. Technical Report ANL-92/17, Argonne National Laboratory, 1992.

[20] R. Butler and E. Lusk. Monitors, Messages, and Clusters: The p4 Parallel Programming System. *Parallel Computing*, 20:pp. 547–564, April 1994. Also Argonne National Laboratory Mathematics and Computer Science Division preprint P362-0493.

[21] P. M. Campbell and E. A. Carmona. Hierarchical Domain Decomposition with Unitary Load Balancing for Electromagnetic Particle-In-Cell Codes. *IEEE*, pages pp. 943–950, 1990.

[22] Center of Excellence in Space Data and Information Sciences. *Beowulf Project at CESDIS*. web page document at http://cesdis.gsfc.nasa.gov/beowulf.

[23] H. Y. Chang, S. Utku, M. Salama, and D. Rapp. A Parallel Householder Tridiagonalisation Stratagen Using Scattered Square Decomposition. *Parallel Computing*, 6:pp. 297–311, 1988.

[24] J. Choi, J. Dongarra, and D. Walker. The Design of a Parallel Dense Linear Algebra Software Library: Reduction to Hessenberg, Tridiagonal and Bidiagonal Forms. *Numerical Algorithms*, 10:pp. 379–400, 1995.

[25] D. E. Comer and D. L. Stevens. *Internetworking with TCP/IP. Vol II: Design, Implementation, and Internals*. Prentice Hall, Englewood Cliffs, NJ, 1991.

[26] S. Dandamudi and D. Eager. Hierarchical Interconnection Networks for Multicomputer Systems. *IEEE Transactions on Computers*, 39(6):pp. 786–797, June 1990.

[27] A. L. Decegama. *The Technology of Parallel Processing Volume I: Parallel Processing Architectures and VLSI Hardware*. Prentice Hall Inc., Englewood Cliffs, NJ, 1989.

[28] M. J. S. Dewar and G. P. Ford. Ground States of Molecules. 44. MINDO/3 Calculations of Absolute Heat Capacities and Entropies of Molecules without Internal Rotations. *Journal of the American Chemical Society*, 99(24):pp. 7822–7834, November 1977.

[29] M. J. S. Dewar and E. F. Healy. AM1: A New General Purpose Quantum Mechanical Molecular Model. *Journal of the American Chemical Society*, 107(13):pp. 3902–3909, 1985.

[30] M. J. S. Dewar and W. Thiel. Ground States of Molecules. 38. The MNDO Method. Approximations and Parameters. *Journal of the American Chemical Society*, 99(15):pp. 4899–4907, July 1977.

[31] I. Dhillon, G. Fann, and B. Parlett. Application of a New Algorithm for the Symmetric Eigenproblem to Computational Quantum Chemistry. In *Proceedings of the SIAM 8th Conference on Parallel Processing for Scientific Computing*, Minn, MN, March 1997.

[32] J. Dongarra and D. Sorensen. A Fully Parallel Algorithm for the Symmetric Eigenproblem. *SIAM Journal of Scientific Computing*, 8:pp. 139–154, 1987.

[33] A. Edelman. Optimal Matrix Transposition and Bit Reversal on Hypercubes: All-to-All Personalized Communication. *Journal of Parallel and Distributed Computing*, 11:pp. 328–331, 1991.

[34] D. M. Elwood, G. I. Fann, and R. J. Littlefield. *PeIGS, Parallel Eigensystem Solver*. Pacific Northwest Laboratory, July 1995.

[35] G. Fann and R. Littlefield. Parallel Inverse Iteration with Reorthogonalization. In *6th SIAM Conference on Parallel Computing*, pages pp. 409–413, Norfolk, VA, March 1993. SIAM.

[36] G. Fann, R. Littlefield, and D. Elwood. Performance of a Fully Paralle Dense Real Symmetric Eigensolver in Quantum Chemistry Applications. In *Simulation MultiConference 1995*. Society for Computer Simulation, 1995.

[37] Frank J. Seiler Research Laboratory, United States Air Force Academy, CO 80840. *MOPAC Manual*, DEC-3100 edition, December 1990.

[38] Fujitsu Company. *MOPAC 2000 home page*. web page document at http://www.fujitsu.co.jp/hypertext/softinfo/product/indust/winmopac/mopac2000/-index_e.html.

[39] Fujitsu Company. *WinMOPAC V2.0 home page*. web page document at http://-www.fujitsu.co.jp/hypertext/softinfo/product/indust/winmopac/index_e.html.

[40] A. Geist, A. Beguelin, J. J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine*. The MIT Press, Cambridge, MA, 1994. Also available at http://www.netlib.org/pvm3/book/pvm-book.ps.

[41] The Gigabit Ethernet Alliance. *Gigabit Ethernet Technical Overview*, May 1999. soft copy available at http://www.gigabit-ethernet.org/technology/-whitepapers/gige_0698/papers98_toc.html.

[42] G. Golub and J. M. Ortega. *Scientific Computing an Introduction with Parallel Computing*. Academic Press, Inc., 1250 Sixth Avenue, San Diago, CA 92101-4311, 1993.

[43] J. R. Goodman. Using Cache Memory to Reduce Processor-Memory Traffic. In *Proceedings of the 10'th Annual International Symposium on Computer Architecture*, Stockholm, Sweden, 1983.

[44] W. Gropp and E. Lusk. Some Early Performance Results with MPI on the IBM SP1. Early draft. Comes with MPICH 1.0.12 package, June 1995.

[45] W. Gropp and E. Lusk. User's Guide for mpich, a Portable Implementation of MPI. Technical Report ANL/MCS-TM-ANL-96/6, Mathematics and Computer Science Division, Argonne National Laboratory, 1996.

[46] W. D. Gropp and B. Smith. Chameleon Parallel Programming Tools Usrs Manual. Technical Report ANL-93/23, Argonne National Laboratory, March 1993.

[47] T. Haupt, E. Akarsu, G. Fox, A. Kalinichenko, H. Kim, P. Sheethalnath, and C. Youn. The Gateway System: Uniform Web Based Access to Remote Resources. Technical Report SCCS-837, Northeast Parallel Architectures Center at Syracuse University, February 1999.

[48] B. Hendrickson and D. Womble. The Torus-Wrap Mapping for Dense Matrix Calculations on Massively Parallel Computers. *SIAM Journal of Scientific Computing*, 15:pp. 1201–1226, 1994.

[49] Hewlett-Packard Company. *HP 9000 V-Class Enterprise Server Specifications.* web page document at http://www.unixsolutions.hp.com/products/-servers/vclass/specifications.html.

[50] Hewlett-Packard Company. *HP 9000 V2600 Enterprise Server White Paper.* web page document at http://www.unixsolutions.hp.com/products/servers/vclass/-v2600_wp.html.

[51] C. T. Ho and M. T. Raghunath. Efficient Communication Primitives on Hypercubes. *Concurrency: Practice and Experience*, 4(6):pp. 1–31, 1992.

[52] K. Hwang, Z. Xu, and M. Arakawa. Benchmark Evaluation of the IBM SP2 for Parallel Signal Processing. *IEEE Transactions on Parallel and Distributed Systems*, 7(5):pp. 522–535, 1996.

[53] Intel Corporation, Beaverton, Oregon. *Touchstone Delta System User's Guide*, 1991.

[54] Intel Corporation, Beaverton, Oregon. *iPSC/860 System Calls Reference Manual*, 1992.

[55] Intel Corporation, Beaverton, Oregon. *iPSC/860 System User's Guide*, 1992.

[56] International Business Machines Corporation. *IBM RS/6000 Enterprise Server Model S80 Specification Sheet.* web page document at http://-www.rs6000.ibm.com/hardware/enterprise/s80_specs.html.

[57] International Business Machines Corporation. *MPI-F: An MPI implementation for IBM SP*.

[58] International Business Machines Corporation. *RS/6000 SP Directory*. web page document at http://www.rs6000.ibm.com/resource/features/1999/sp.html.

[59] International Business Machines Corporation. *IBM AIX PVMe User's Guide and Subroutine Reference*, third edition, July 1994.

[60] D. M. Koppelman. Reducing PE/Memory Trafic in Multiprocessors by the Difference Coding of Memory Addresses. *IEEE Transactions on Parallel and Distributed Systems*, 5(11):pp. 1156–1168, November 1994.

[61] M. H. Lee and S. R. Seidel. Concurrent Communication on the Intel iPSC/2. Technical Report CS-TR 9003, Michigan Technological University, July 1990.

[62] P. C. Liewer, E. W. Leaver, V. K. Decyk, and J. M. Dawson. Dynamic Load Balancing in a Concurrent Plasma PIC Code on the JPL/Caltech Mark III Hypercube. *IEEE*, pages pp. 939–942, 1990.

[63] T. H. Lin, T. Haupt, and G. C. Fox. Parallelizing MOPAC on distributed computing systems within AVS framework. Technical Report Syracuse Center for Computational Science, SCCS-744, Northeast Parallel Architectures Center at Syracuse University, 1995.

[64] R. J. Littlefield and K. J. Marchhoff. Investigating the Performance of Parallel Eigensolvers for Large Processor Counts. *Theor. Chim. Acta*, 84:pp. 457–473, 1993.

[65] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, June 1995. on-line manual available at http://www.mpi-forum.org/docs/mpi-11.ps.

[66] nCUBE Corporation, Foster City, CA. *nCUBE 2 Programmer's Guide 2.0*, 1990.

[67] L. M. Ni and P. K. McKinley. A Survey of Wormhole Routing Techniques in Direct Networks. *IEEE Computer*, 26(2):pp. 62–76, February 1993.

[68] Parasoft Corporation, Pasadena, CA. *Express User's Guide*. Version 3.2.5, 1992.

[69] M. J. Quinn. *Designing Efficient Algorithms for Parallel Computers*. McGraw-Hill Book Company, 1987.

[70] M. T. Raghunath and A. Ranade. Designing Interconnection Networks for Multi-Level Packaging. In *Proc. of Supercomputing '93*, pages pp. 772–781. IEEE Computer Society Press, 1993.

[71] S. Ranka and J. C. Wang. Static and Runtime Scheduling of Unstructured Communication. *International Journal of Computing Systems in Engineering*, 1993.

[72] S. Ranka, J. C. Wang, and G. C. Fox. Static and Runtime Algorithms for All-to-Many Personalized Communications on Permutation Networks. In *Proceedings of the 1992 International Conference on Parallel and Distributed Systems*, pages pp. 211–218, HsinChu, Taiwan, December 1992.

[73] S. Ranka, J. C. Wang, and G. C. Fox. Static and Runtime Algorithms for All-to-Many Personalized Communications on Permutation Networks. *IEEE Transactions on Parallel and Distributed Systems*, 5(12):pp. 1266–1274, December 1994.

[74] S. Ranka, J. C. Wang, and M. Kumar. Personalized Communication Avoiding Node Contention on Distributed Memory Systems. In *Proceedings of the 1993 International Conference on Parallel Processing*, volume I, pages pp. 241–244, St. Charles, IL, August 1993.

[75] S. Ranka, J. C. Wang, and M. Kumar. Irregular Personalized Communication on Distributed Memory Machines. *Journal of Parallel and Distributed Computing*, 25(1):pp. 58–71, Febrary 1995.

[76] J. Saltz, R. Das, R. Ponnusamy, D. Mavriplis, H. Berryman, and J. Wu. PARTI Procedures for Realistic Loops. In *Proceedings of Sixth Distributed Memory Computing Conference*, Portland, Oregon, 1991.

[77] J. Saltz, S. Petiton, H. Berryman, and A. Rifkin. Performance Effects of Irregular Communications Patterns on Massively Parallel Multiprocessors. Technical Report NASA Contractor Report 187514, ICASE Report No. 91-12, Institute for Computer Applications in Science and Engineering at NASA Langley Center, January 1991.

[78] SGI. *SGI Origin Family Technical Overview.* web page document at http://-www.sgi.com/origin/tech_info.html.

[79] A. J. Smith. Cache Memories. *ACM Computing Surveys*, 14(3):pp. 473–530, September 1982.

[80] W. Stallings. *Handbook of Computer Communications Standards - Local Network Standards*, volume 2. Howard W. Sams & Company, Indianapilis, Indiana, USA, First edition, 1988.

[81] T. Sterling, D. Becker, D. Savarese, et al. BEOWULF: A Parallel Workstation for Scientific Computation. In *Proceedings of the 1995 International Conference on Parallel Processing.* Volume I, pages pp. 11-14, August 1995.

[82] R. W. Stevens. *UNIX Network Programming.* Prentice Hall, Englewood Cliffs, NJ, 1990.

[83] J. J. P. Stewart. Optimization of Parameters for Semiempirical Methods II. Applications. *Journal of Computational Chemistry*, 10(2):pp. 221–264, March 1989.

[84] C. B. Stunkel, D. G. Shea, B. Abali, M. G. Atkins, C. A. Bender, D. G. Grice, P. Hochschild, D. J. Joseph, B. J. Nathanson, R. A. Swetz, R. F. Stucke, M. Tsao, and P. R. Varker. The SP2 High Performance Switch. *IBM Systems Journal*, 34(2):pp. 185–204, 1995.

[85] Sun Microsystems, Inc. *SUN HPC 10000 specifications.* web page document at http://www.sun.com/servers/hpc/products/hpc10000_spec.html.

[86] V. Sunderam. PVM: A Framework for Parallel Distributed Computing. *Concurrency: practice and experience*, 2(4), December 1990.

[87] A. Sussman, J. Saltz, R. Das, S. Gupta, D. Mavriplis, R. Ponnusamy, and K Crowley. PARTI Primitives for Unstructured and Block Structured Problems. *Computing Systems in Engineering*, 3:pp. 73–86, 1992.

[88] A. Szabo and N. S. Ostlund. *Modern Quantum Chemistry.* McGraw-Hill, New York, First edition, 1989. Revised.

[89] Thinking Machines Corporation, Cambridge, MA. *CMMD Reference Manual.* Version 3.0, December 1992.

[90] Thinking Machines Corporation, Cambridge, MA. *The Connection Machine CM-5 Technical Summary*, October 1991.

[91] Thinking Machines Corporation, Cambridge, MA. *The Connection Machine CM-5 Reference Manual*, 1992.

[92] P. L. Vaughan, A. Skjellum, D. S. Reese, and F. Cheng. Migrating from PVM to MPI, Part I: The UNIFY System. Technical report, Mississippi State University, July 1994. ftp://aurora.cs.msstate.edu/pub/reports/Message-Passing/-unify_frontiers95.ps.Z.

[93] T. von Eichen, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active Messages: a Mechanism for Integrated Communication and Computation. In *Proceedings of the 19'th International Symposium on Computer Architecture*, Gold Coast, Australia, May 1992.

[94] J. C. Wang, T. H. Lin, and S. Ranka. NICE: Non-uniform Irregular Communication Exchange on Distributed Memory Systems. Technical Report SU-CIS-93, Syracuse University, June 1993.

[95] J. C. Wang, T. H. Lin, and S. Ranka. Distributed Scheduling of Unstructured Collective Communication on the CM5. *Parallel Processing Letters*, 5(4):pp. 647–658, 1995.

[96] M. C. Zerner. Semiempirical Molecular Orbital Methods. In K. B. Lipkowitz and D. B. Boyd, editors, *Reviews in Computational Chemistry II*, chapter 8, pages pp. 313–366. VCH Publishers, Inc., 1991.

[97] H. Zima and B. Chapman. *Supercompilers for Parallel and Vector Computers*. ACM Press, New York, NY, 1991.

Biographical Data

Name:	Tseng-Hui Lin
Date and Place of Birth:	January 8, 1964 Hsin-Chu City, Taiwan, R.O.C.
Degrees Awarded:	Master of Science, 1999 Syracuse University Syracuse, NY, U.S.A.
	Master of Engineer, 1988 National Central University Chung-Li, Tao-Yuan, Taiwan, R.O.C.
	Bachelor of Engineer, 1986 National Chiao-Tung University Hsin-Chu, Taiwan, R.O.C.
Professional Experience:	Software Developer IBM Scalable POWERparallel Systems Poughkeepsie, NY, U.S.A., 1998 -
	Engineer Education and Training Division Institute of Information Industry Chung-Li, Tao-Yuan, Taiwan, R.O.C., 1988 - 1990