Parallel Pencil-Beam Redefinition Algorithm

Paul Alderson[1], Mark Wright[1], Amit Jain[1], and Richard Boyd[2]

[1] Department of Computer Science
Boise State University
Boise, Idaho 83725, USA
{mwright,aalderso}@onyx.boisestate.edu, amit@cs.boisestate.edu

[2] MD Anderson Cancer Center
University of Texas
1515 Holcombe Blvd, Houston, TX 77030, USA
rboyd@mdanderson.org

Abstract. The growing sophistication in radiation treatment strategies requires the utilization of increasingly accurate, but computationally intense, dose algorithms, such as the electron pencil-beam redefinition algorithm (PBRA). The sequential implementation of the PBRA is in production use at the MD Anderson Cancer center. The PBRA is difficult to parallelize because of the large amounts of data involved that is accessed in an irregular pattern taking varying amounts of time in each iteration. A case study of the parallelization of the PBRA code on a Beowulf cluster using PVM and PThreads is presented. The solution uses a non-trivial way of exchanging minimal amount of data between processes to allow a natural partitioning to work. Multi-threading is used to cut down on the communication times between CPUs in the same box. Finally, an adaptive load-balancing technique is used to further improve the speedup.

1 Introduction

Radiation therapy was one of the first medical disciplines where computers were used to aid the process of planning treatments; the first paper on the use of computers to calculate radiation dose distributions appeared almost 50 years ago [1]. Today, computers are involved in practically all areas of radiation therapy. Treatment planning functions include image-based tumor localization, image segmentation, virtual therapy simulation, dose calculation, and optimization. Treatment delivery functions include controlling delivery systems and treatment verification. The growing sophistication in radiation treatment strategies requires the utilization of increasingly accurate, but computationally intense, dose algorithms, such as the electron pencil-beam redefinition algorithm (PBRA) [2]. The demanding pace of the radiation therapy clinic requires the employment of these advanced dose algorithms under the most optimum conditions available in terms of computation speed. Parallel processing using multi-computer clusters or multi-processor platforms are now being introduced to the clinic for this

purpose, and the PBRA, with its extensive use of multi-dimensional arrays, is a good candidate for parallel processing.

In Section 2, we present an analysis of the PBRA sequential code. In Section 3, we describe the parallelization of the PBRA code as a series of refinements that were implemented along with the results obtained after each refinement. Section 4 further discusses some properties of the current parallel implementation. Finally, in Section 5, some conclusions are presented.

2 Sequential Code

The PBRA algorithm was implemented in FORTRAN by Robert Boyd in 1999. It is in production use at the University of Texas MD Andersen Cancer Center. We will highlight some features of the PBRA code that are relevant to the parallel implementation.

The PBRA code uses 16 three-dimensional arrays and several other lower dimensional arrays. The size of the arrays is about 45MB. The inner core of the code is the function `pencil_beam_redefinition()`, which has a triply-nested loop that is iterated several times. Profiling shows that this function takes up about 99.8% of the total execution time. The following code sketch shows the structure of the `pencil_beam_redefinition()` function.

```
kz = 0;
while (!stop_pbra && kz <= beam.nz)
{
  kz++;
  /* some initialization code here */

  /* the beam grid loop */
  for (int ix=1; ix <=beam.nx; ix++) {
    for (int jy=1; jy <= beam.ny; jy++) {
      for (int ne=1; ne <= beam.nebin; ne++) {
        ...
        /* calculate angular distribution in x direction */
        pbr_kernel(...);
        /* calculate angular distribution in y direction */
        pbr_kernel(...);
        /* bin electrons to temp parameter arrays */
        pbr_bin(...);
        ...
      }
    }
  } /* end of the beam grid loop */
  /* redefine pencil beam parameters and calculate dose */
  pbr_redefine(...);
}
```

The main loop of `pencil_beam_redefinition()` also contains some additional straight-line code that takes constant time and some function calls that perform binary search. However these take an insignificant amount of time and are hence omitted from the sketch shown above. Currently, the code uses `beam.nx = 91`, `beam.ny = 91`, `beam.nz = 25`, and `nebin = 25`. Profiling on sample data shows that the functions `pbr_kernel()` and `pbr_bin()` take up about 97.6% of the total execution time.

The function `pbr_kernel()` takes linear time ($O($`beam.nx`$)$ or $O($`beam.ny`$)$). The `pbr_kernel()` function computes new ranges in x and y direction that are used in the `pbr_bin()` function to update the main three -dimensional arrays. The execution time for `pbr_bin()` is dominated by a doubly-nested loop that is $O((\texttt{xmax} - \texttt{xmin} + 1) \times (\texttt{ymax} - \texttt{ymin} + 1))$, where the parameters `xmin, xmax, ymin, ymax` are computed in the two invocations of the `pbr_kernel()` function. The function `pbr_bin()` function updates the main three-dimensional arrays in an irregular manner, which makes it difficult to come up with a simple partitioning scheme for parallel implementation.

3 Parallelization of PBRA

Initially, the PBRA code was rewritten in C/C++. The C/C++ version is used as the basis for comparing performance. During this section, we will discuss timing results that are all generated using the same data set. For this data set, the **sequential PBRA code ran in 2050 seconds**. In production use, the PBRA code may be used to run on several data sets in a sequence. In the next section, we will present results for different data sets.

The sequential timing skips the initialization of data structures and the final writing of the dosage data to disk. These two take insignificant amount of time relative to the total running time. Before we describe the parallel implementation, we need to describe the experimental setup.

3.1 Experimental Setup

A Beowulf-cluster was used for demonstrating the viability of parallel PBRA code. The cluster has 6 dual-processor 166MHz Pentium PCs, each with 64 MB of memory, connected via a 100 MBits/s Ethernet hub.

The PCs are running RedHat Linux 7.1 with a version 2.4.2-2 SMP kernel. The C/C++ compiler used is the GNU C/C++ compiler version 2.96 with the optimizer option enabled. PVM version 3.4.3 and XPVM version 1.2.5 [3] are being used. For threads, the native POSIX threads library in Linux is being used.

3.2 Initial PVM implementation

The parallel implementation uses the PVM Master/Slave model. The master process initializes some data structures, spawns off the PVM daemon, adds machines to PVM, spawns off slave processes, multicasts some data structures to

the slave processes and then waits to get the dosage arrays back from the slave processes. Embedding PVM makes it transparent to the users.

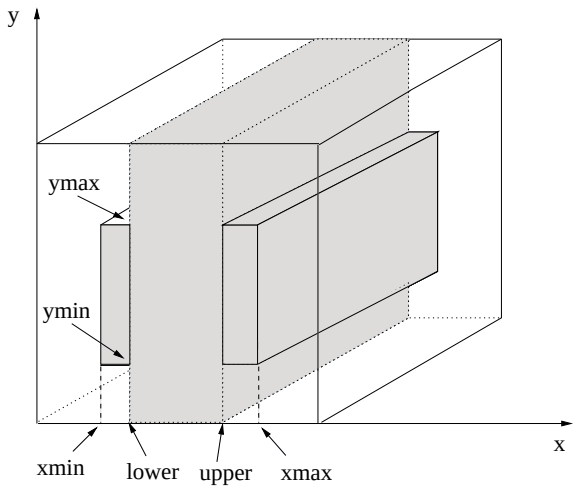

Fig. 1. Spreading of a beam being processed in one slice by one process

The main loop in `pencil_beam_redefinition` is being run by each slave process. However, each process works on one slice of the main three-dimensional arrays. The slicing was chosen to be done in the X-axis. If we have p processes and n is the size in the X dimension, then each process is responsible for a three-dimensional slice of about n/p width in the X axis. However, as each process simulates the beam through its slice, the beam may scatter to other slices. See Figure 1 for an illustration. Most of the time the scattering is to adjoining slices but the scattering could be throughout. One effect of this observation is that each slave process has to fully allocate some of the three-dimensional arrays even though they only work on one slice.

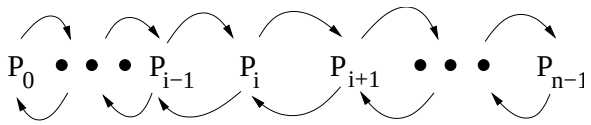

Fig. 2. Compensation for beam scattering at the end of each outer iteration

To solve the problem of beam-scattering, the processes exchange partial amounts of data at the end of each iteration (which also synchronizes the processes). The communication pattern is shown in Figure 2. Each process P_i receives appropriate parts of the three-dimensional arrays from P_{i-1} as well as

P_{i+1}. Each process P_i combines the received data with its computed data. Then P_i sends the appropriate parts of the data off to its two neighbors. The processes P_0 and P_{n-1} bootstrap the process, one in each direction. The amount of data exchanged is dependent upon how much the beam scatters.

After some more fine-tuning, the best timing with the initial implementation was 657 seconds with 12 slave processes, for a speedup of 3.12.

3.3 Using PThreads on Each Machine

Using XPVM the communication patterns of the PBRA parallel code was observed. It was realized that the communication time can be reduced by letting only one process run per machine but use two threads (since each machine was a SMP machine with 2 CPUs). The main advantage was that threads would share the memory, so no communication would be required.

Each thread is now running the entire triply-nested for loop. To obtain a better load balance, the threads are assigned iterations in a round-robin fashion. Another problem is the need to synchronize threads in between calls to `pbr_kernel` and for the call to `pbr_bin`, since during the call to `pbr_bin` the partial results from the threads were being combined. The net effect of multi-threading is that the calls to `pbr_kernel` were now happening concurrently while communication is reduced significantly. The sketch of the thread main function (`pbra_grid`) is shown below.

```
/* inside pbra_grid: main function for each thread */
  for (int ix=lower; ix <=upper; ix=ix+procPerMachine) {
    for (int jy=1; jy <= beam.ny; jy++) {
      for (int ne=1; ne <= beam.nebin; ne++) {
        ...
        /* calculate angular distribution in x direction */
        pbr_kernel(...);
        <use semaphore to update parameters in critical section>
        /* calculate angular distribution in y direction */
        pbr_kernel(...);
        <use semaphore to update parameters in critical section>
        /* bin electrons to temp parameter arrays */
        <semaphore_down to protect access to pbr_bin */
        pbr_bin(...);
        <semaphore_up to release access to pbr_bin */
        ...
      }
    }
  }
```

Compared to the sequential program, the multi-threaded version running on one machine with 2 CPUs took 1434 seconds, for a speedup of 1.43. When we ran the PVM program with multi-threading on 12 CPUs, we got a time of 550 seconds for a speedup of 3.73, about a 20% improvement.

3.4 Adaptive Load Balancing

Using XPVM and with the help of code instrumentation, it was discovered that although each process had an equal amount of data, the amount of time required is not distributed equally. Also, the uneven distribution had an irregular pattern that varies with each outer iteration. A general load balancing scheme is described below to deal with the unbalanced workloads.

Each slave process sends the time taken for the last outer iteration to the master. Suppose that these times are $t_0, t_1, t_2, \ldots, t_{n-1}$ for n processes. The master computes the average time, say t_{avg}. Based on that, process P_i needs to retain a fraction t_{avg}/t_i for the next round. We are using the past iteration to predict the times for the next iteration. We also made the frequency of load balancing be a parameter; that is, whether to load-balance every iteration or every second iteration and so on. The following code shows a sketch of the main function for the slave processes after incorporating the load-balancing.

```
kz = 0;
while (!stop_pbra && kz <= beam.nz)
{
  kz++;
  /* the pbra_grid does most of the work now */
  for (int i=0; i<procPerMachine; i++)
    pthread_create(...,pbra_grid,...);
  for (int i=0; i<procPerMachine; i++)
    pthread_join(...);
  <send compute times for main loop to master>
  <exchange appropriate data with P(i-1) and P(i+1)>
  /* redefine pencil beam parameters and calculate dose */
  pbr_redefine(...);
  <send or receive data to rebalance based on
   feedback from master and slackness factor>
}
```

Table 1 shows the effect of load balancing frequency on the parallel runtime.

The next improvement comes from the observation that if a process finishes very fast, it might receive too much data for the next round. The slackness factor allows us to specify a percentage to curtail this effect. If it is set to 80%, then the process receives only 80% of the data specified by the load-balancing scheme. This allows the data received to be limited but still enables receiving a fair amount to be load-balanced. We experimented with various slackness factors: 90%, 80%, 70%, 60%, 50%. A slack factor of 80% gave an additional 5% improvement in runtime.

Finally, Table 2 shows the runtime and speedup for the sample data set with all the above refinements in place. The load balancing frequency was set to 4 and the slackness factor was 80%.

Table 1. Effect of load balancing frequency on runtime for parallel PBRA code running on 6 machines with 2 CPUs each

Load Balancing Frequency	Runtime (seconds)
none	550
1	475
2	380
3	391
4	379
5	415

Table 2. Parallel runtime versus number of CPUs on a sample data set

CPUs	Runtime (secs)	Speedup
1	2050	1.00
2	1434	1.42
4	1144	1.79
6	713	2.87
8	500	4.00
10	405	5.06
12	369	5.56

4 Further Results and Discussion

Table 3. Comparison of various refinements to parallel PBRA program. All times are for 12 CPUs

Technique	Time (seconds)	Speedup
Sequential	2050	1.00
Partitioning with PVM	657	3.12
Multithreading + PVM	550	3.73
Load balancing + Multithreading + PVM	369	5.56

Table 3 summarizes the improvements obtained with the various refinements on one sample data set.

Table 4 shows the runtime and speedup for six additional data sets. The first column shows the density of the matter through which the beam is traveling. Note that when the density of the matter is high, the electron pencil-beam stops early and thus the computation time is smaller. When the density is low, then the beam goes farther but does not scatter as much, which would also tend

Table 4. Parallel runtime and speedups for different data sets. The density column shows the density of the matter through which the beam is traveling. All times are for 12 CPUs

Density (gm/cm^3)	Sequential (seconds)	Parallel (seconds)	Speedup
0.5	1246	131	9.51
0.7	2908	352	8.26
0.9	3214	364	8.83
1.1	2958	370	7.99
1.3	2641	321	8.22
1.5	2334	300	7.78

to reduce the time. Human tissue has density values close to 1.0, which is the density in the sample data set used in the previous section. The average speedup for all the data sets tested was around 8. The load balancing frequency was set to 4 and the slackness factor was 80%.

Although the speedups obtained are encouraging, further work remains to be done. We would like to test the parallel PBRA program on a larger cluster. Further tuning of the parallel code could lead to more speedups. Testing on a wider variety of data sets should allow us to further customize the load balancing frequency and the slackness factor.

5 Conclusions

We have presented a parallelization of electron Pencil Beam Redefinition Algorithm program originally written by Robert Boyd. The program was difficult to parallelize because of the large amounts of data that is accessed in an irregular pattern, requiring varying amounts of time in each iteration. We were able to obtain speedups in the range of 5.56 to 9.5 using 12 CPUs. The speedups obtained were based on three techniques: (1) using a simple partitioning scheme but compensating for the irregularity of data with an interesting data exchange technique that attempts to minimize the amount of data traffic on the network; (2) using threads in conjunction with PVM to reduce communication time for SMP machines; (3) using an adaptive load-balancing technique.

References

1. K. C. Tsien, "The application of automatic computing machines to radiation treatment," Brit. J. Radiology 28:432-439, 1955.
2. A. S. Shiu and K. R. Hogstrom, "Pencil-beam redefinition algorithm for electron dose distributions," Med. Phys. 18: 7-18, 1991.
3. PVM Home Page: `http://www.csm.ornl.gov/pvm/pvm_home.html`