Parallel Pencil Beam Redefinition Algorithm

Paul Alderson
Mark Wright
Amit Jain
Robert Boyd

Problem Definition

- Radiation Therapy
  - Pencil Beam Redefinition Algorithm (PBRA) calculates radiation dose distributions.
- PBRA, with its extensive use of multi-dimensional arrays, is a good candidate for parallel processing.
- The sequential implementation of the PBRA is in production use at the MD Anderson Cancer center, University of Texas.

Sequential Code

- The PBRA code uses 16 three-dimensional arrays and several other lower dimensional arrays.
- The total size of the arrays is about 45 MB.
- The core functions take about 99.8% of the total execution time.
  - Contains a triply-nested loop that is iterated several times.
- Sequential Time – 2050 Seconds

Sequential PBRA Pseudo code

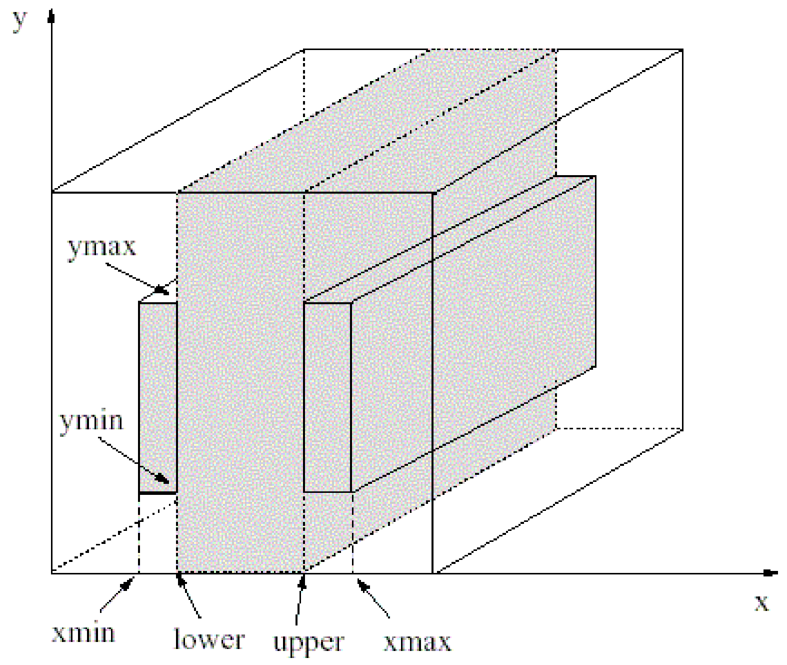
```
kz = 0;
while (!stop_pbra && kz <= beam.nz)
{
  kz++;
  /* some initialization code here */

  /* 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(...);
}
```

Experimental Setup

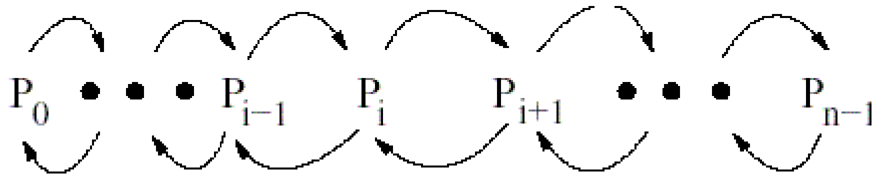
- A Beowulf-cluster was used for demonstrating the viability of parallel PBRA code.
- PVM version 3.4.3 and XPVM version 1.2.5 are being used.
- For threads, the native POSIX threads library in Linux is being used.

Initial PVM Implementation



Each process works on the x-axis slice of the main three-dimensional array.

Beam Spreading in Initial Implementation



- The processes exchange partial amounts of data at the end of each iteration.
- The amount of data exchanged is dependent upon how much the beam scatters.
- The initial implementation yielded a speedup of 3.12
  (runtime of 657 secs with 12 processes)

Pthreads

- Each thread runs the entire triply-nested for loop
- To obtain a better load-balance the threads are assigned iterations in a round-robin fashion.

```
/* 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++) {
            ...
            pbr_kernel(...); <use semaphore to update parameters in critical
section>
            pbr_kernel(...); <use semaphore to update parameters in critical
section>

            <semaphore_down to protect access to pbr_bin */
            pbr_bin(...);
            <semaphore_up to release access to pbr_bin */
            ...
         }
      }
   }
```

- Two CPU in one machine time was 1434 secs, with a speedup of 1.43.
- Overall runtime with 12 processes was 550 secs, speedup improved to 3.73.

Adaptive Load Balancing

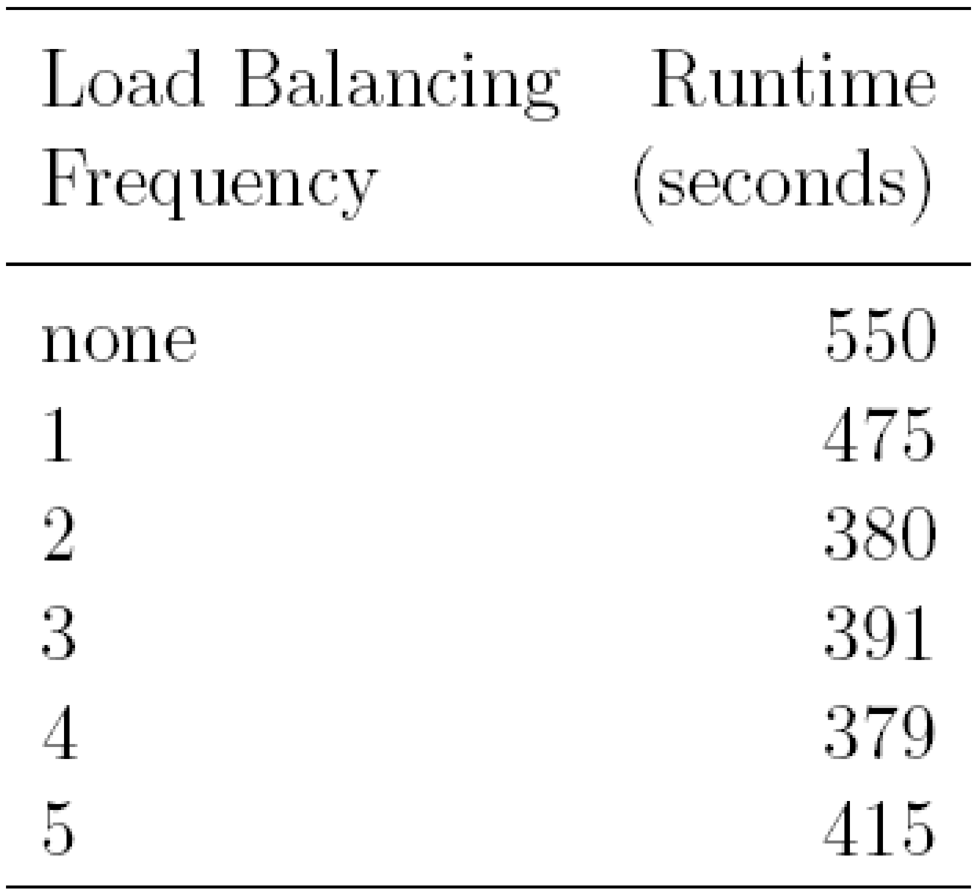
- Although each process had an equal amount of data, the amount of time required was not distributed equally.
- The uneven distribution had an irregular pattern that varies with each outer iteration.
- The variation from the average time was used to predict the times for the next iteration and to vary the work load of each slave.
- A customizable slackness factor was also incorporated.

Load Balancing Pseudo Code

- The following pseudo code shows a sketch of the main function for the slave processes after incorporating the load-balancing.

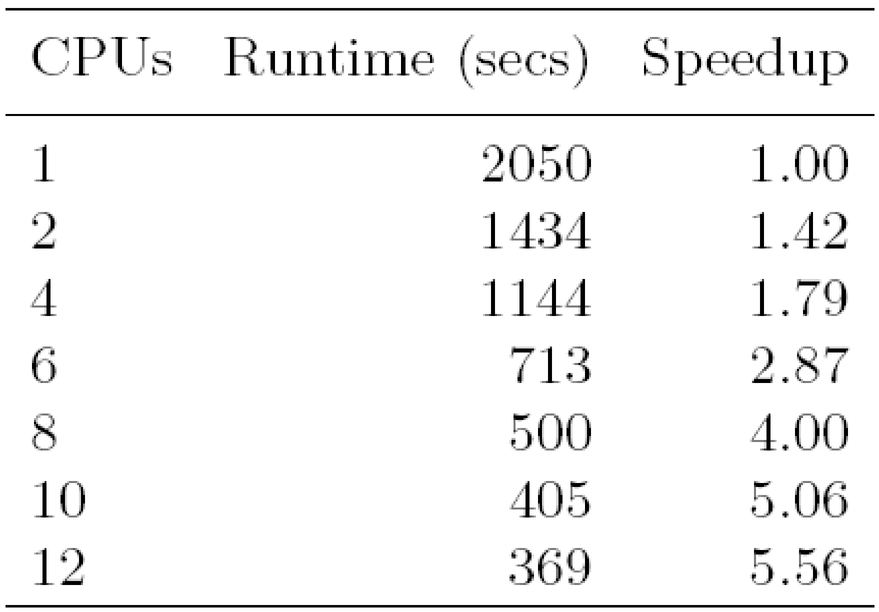
```
kz = 0;
while (!stop_pbra && kz <= beam.nz)
{
   kz++;
   for (int i=0; i<procPerMachine; i++)
      pthread_create(...,pbra_grid,...);
   for (int i=0; i<procPerMachine; i++)
      pthread_join(...);
   <send compute times for main loop to master>
   <exchange appropriate data with P(i-1) and P(i+1)>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

Load Balancing Frequency Results



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

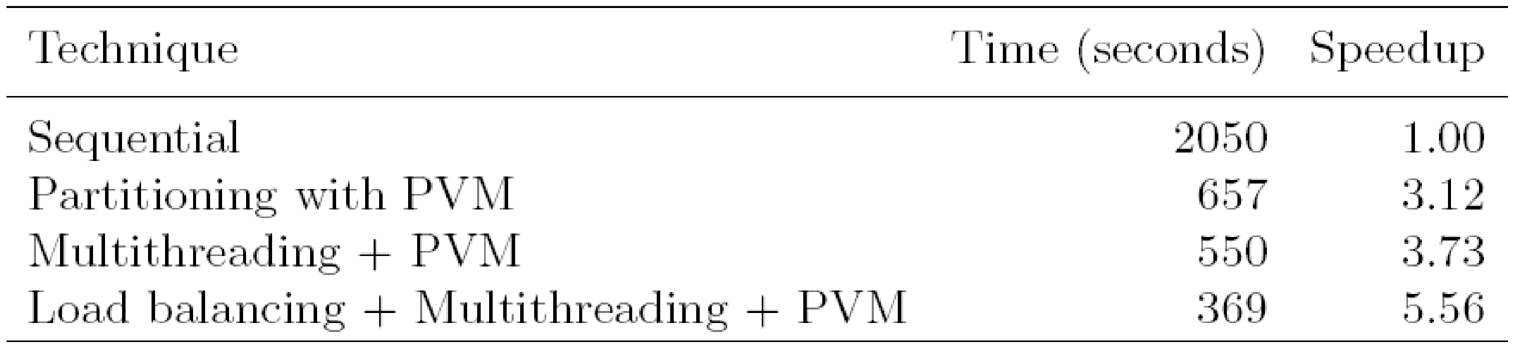
Parallel Runtimes



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

- Results calculated with a load balancing frequency of 4 and a slackness factor of 80%.

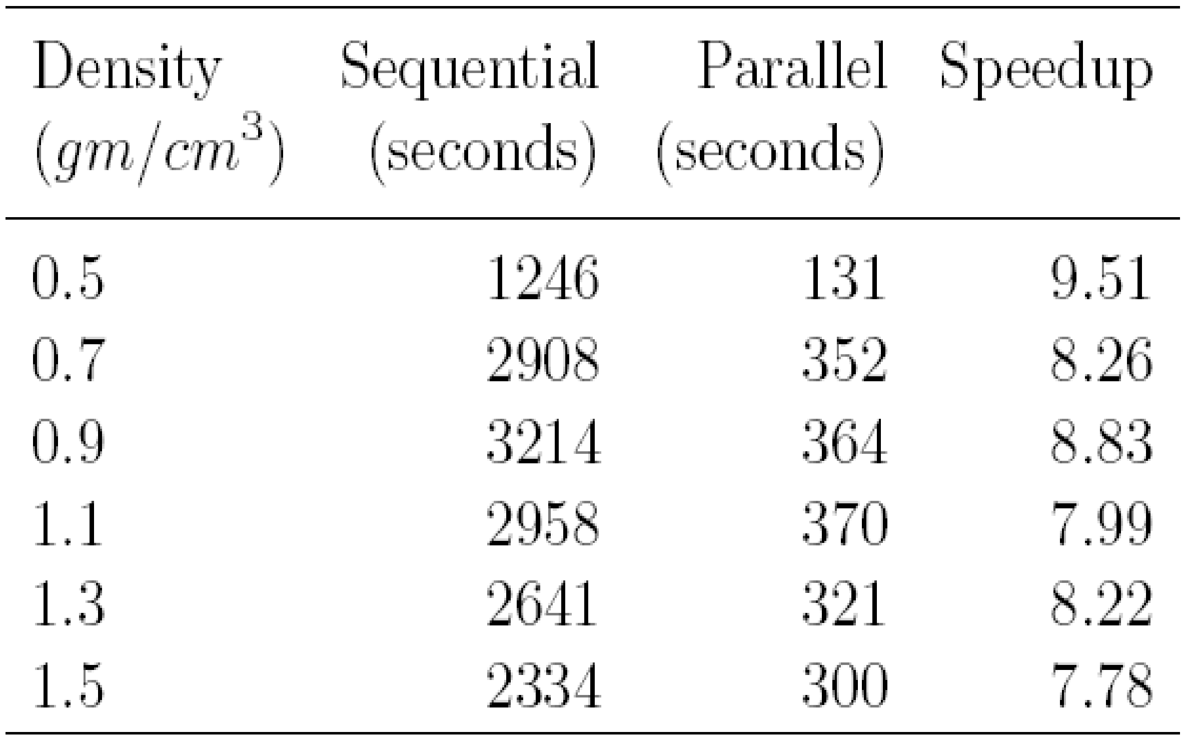
Summary of Improvements



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

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

Different Data Sets



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

- The density column shows the density of the matter through which the beam is traveling.
- All times are for 12 CPUs.