Parallelization of Radiation Therapy Treatment Planning(RTTP) : A Case Study*

V. Chaudhary, C. Xu, S. Roy and S. Jia
Dept. of Electrical & Computer Engg.
Wayne State University, Detroit MI 48202

G. A. Ezzell and C. Kota
Dept. of Radiation Oncology
Wayne State University, Detroit, MI 48202

Abstract

Radiation therapy uses ionizing radiation to treat cancerous tumors. This paper reports our experiences with the parallelization of a real-world 3-D radiation therapy treatment planning (RTTP) system on a wide range of platforms, including SMP servers, Cray J916 vector machines, and clusters of SMPs. The RTTP system is a meta-problem, comprising two major loosely-coupled components: dose calculation and genetic optimization. To accelerate the planning process, we integrated the two components and parallelized the system on a wide range of platforms using vendors' native optimization tools and Stanford SUIF parallelizing compiler. For comparison, we also manually parallelized the system using multithreading, message-passing, and distributed shared memory tools. The experimental results showed that none of the automatic parallelization tools were capable of handling the real-world application, although plenty of parallelism exists in the codes. The difficulty is due to RTTP's dynamic and irregular computational behaviors.

1 Introduction

Radiation therapy using external photon beams is an integral part of the treatment of the majority of cancerous tumors [7]. As shown in Figure 1, its practice works in a way that beams of photons are directed at the tumor in a patient from different directions, thereby concentrating radiation dose in the tumor. The maximum dose that can be delivered to the tumor must be subject to its neighboring normal organs' tolerance to radiation damage. Due to the time constraints in clinical practice, computerized treatment planning was limited for many years to producing dose distribution on one or a few planes of the patient's body. The beams were assumed to be coplanar with the patient planes. A 3-D treatment planning process is too time consuming to be employed in clinical workstations. The objective of this work is to accelerate the therapy planning process by using parallel computing techniques.

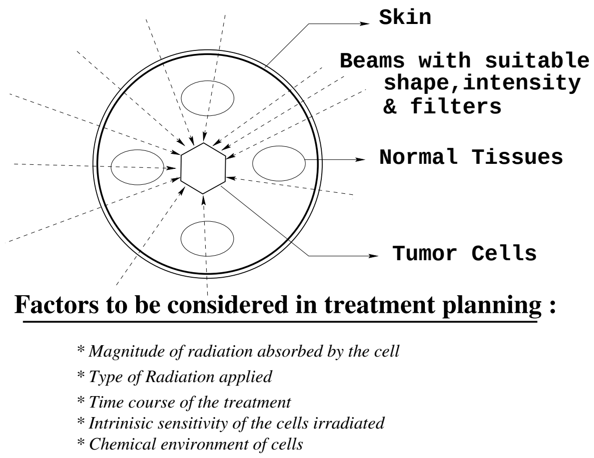

Figure 1: Radiation therapy treatment planning

We experiment with a radiation therapy treatment planning system (RTTP), which is routinely used to plan treatments for a majority of patients at the Gershenson Radiation Oncology Centers at Harper Hospital in Detroit [4]. The system is a meta-problem, comprising two components: dose calculation (VRSplan) and genetic optimization (GENplan). The GENplan code is to select a set of beams towards the tumor, each with a corresponding relative intensity weight. A beam is a description of a particular radiation field, its orientation in space, energy, shape, wedge, etc. Each beam will produce a particular dose distribution within the patient body. A plan is a total dose distribution obtained by summing, for each point being considered, the dose from each selected beam in proportion to its selected weight. The quality of a plan is judged by evaluating this dose distribution in relation to the particular constraints imposed by the physician. The sampling algorithm of the GENplan code is given as follows:

GENplan:

*This research was supported in part by a grant from Cray Research, Inc. and NSF Grant EIA-9729828

Dose calculation (VRSplan)
While not convergent
Create the initial population
Evaluate each member
Transform score into fitness
Make a new generation
Mark some to die,
Clone the best (elitism)
Clone the best with mutated indices
Clone the best with mutated weights
Sexually mate, with some mutations
Randomly create a few

The GENplan process relies on a mechanism for the definition of radiation beams, provision of patient information and dose constraints, and dose calculation. VRSplan is a broadly capable software package for virtual simulation and three-dimensional dose calculation. It works in the way as follows.

VRSplan:
Read patient anastruct data
Read point description data of rational organs
Do for each beam
Read beam description data
Calculate the dose of each pont of relational organs
Write the dose into a file which is to be sent to GENplan

The GENplan and VRSplan were initially loosely coupled. They were run on a DEC 5000 workstation and an SGI Indigo workstation, respectively. For compatibility, data were communicated in ASCII format via off-line file exchanges in ASCII text format. As shown in Figure 2, once the desired beams and points were created on the SGI system, their description files were transferred to the DEC machine. The routing opt_VRSplan is then invoked and the doses to the sampling points are then calculated for each beam. The results are then written to output files that are then transferred back to the SGI computer for optimization studies.

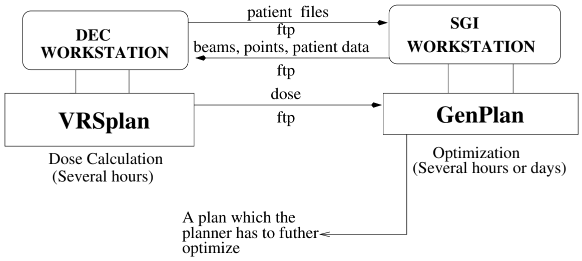

Figure 2: The initial system environment

Studies with clinical cases demonstrate the ability of GENplan to produce optimization results which compare favorably to both simulated annealing and conventional planning. Geometric selection of beam orientation is possible and leads to significant reduction in the number of beams to be considered. GENplan is seen to be a successful optimizer of different evaluation functions, and so the clinical relevance of the optimization process depends critically on the function being optimized. On the other hand, on a purely practical level, the current code is not an integrated system and has not been engineered for speed. Calculating 100-200 beams takes three hours on VRSplan, and ten runs of GENplan on a population of 50 takes about three hours on the Indigo. In order to move the code from the lab to the clinic, different platforms and parallel computing techniques will be necessary.

In the subsequent sections, we will relate our experiences with the parallelization of the RTTP on various platforms, including SUN Enterprise SMPs, a Cray J916 vector machine, and a cluster of SMPs, based on SUN and Cray's native parallelizing tools and Stanford SUIF compilers. The results from automatic parallelization tools will be compared with those from manually parallelized codes.

2 Parallel RTTP on SMP Servers

Symmetric shared-memory multiprocessors, built out of the latest microprocessors, are now a widely available class of powerful machines. Our first experiment was conducted on a SUN E3000 with four 250MHz-UltraSPARCs processors and 512MB memory. We employed two approaches to parallelize the RTTP system: the native optimization tool (SUN Workshop Compiler C4.2) and the Stanford SUIF parallelizing compiler.

2.1 Performance profiling for RTTP

We first profiled the sequential RTTP on the E3000 without using any compiler optimizations. Table 1 presents the results generated by a SUN Workshop performance profiling tool. The RTTP calling graph are given in Figure 3. The numbers along the graph edge represent the calling times of sub-routines.

Table 1: Initial profiling data of RTTP

%Time	Seconds	Calls	msec/call	Name
99.9	1727.90	1		main
61.4	1060.92	49403	21.48	sum_beams
12.9	222.83	22572	9.84	scatter_point_dose
8.1	139.96	49565	2.82	evaluate_plan_dose
7.6	131.32	148016	0.89	hpsort
5.6	96.76	49743	1.95	evaluate_plan_prob

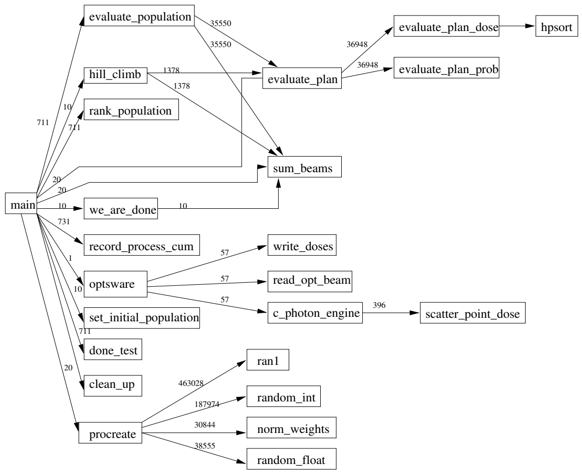

Figure 3: Calling graph of RTTP program

Table 1 shows that of the 1727.90 seconds total execution time, more than 60% was spent by a kernel function "sum_beams". It was executed 49403 times with 21.48 seconds each invocation. The function is used in GENplan for the evaluation of each plan. The main loop of sum_beam is listed as follows:

```
for (iorg=0; iorg<(*plan).norgs; iorg++)
for (ipt=0; ipt<(*plan).organ[iorg].npts; ipt++)
   (*plan).organ[iorg].dose[ipt] = 0;
for (iorg=0; iorg<(*plan).norgs; iorg++)
for (ibm=0; ibm<(*test).nbeams; ibm++)
   for (ipt=0; ipt<(*plan).organ[iorg].npts; ipt++)
      (*plan).organ[iorg].dose[ipt] = (*plan).organ[iorg].dose[ipt] +
         (*plan).beam[(*test).index[ibm]].organ[iorg].dose[ipt] *
         (*test).weight[ibm] * (*goals).isocenter_dose /
         (*plan).beam[(*test).index[ibm]].iso_dose;
```

The loop features multilevel indirect access to *dose* arrays. Of the VRSplan, the most important routine is "scatter_point_dose". It was executed for 22572 times with 9.83 seconds each run. Since the total execution time was dominated by these two routines, our major efforts had been on the parallelization of these routines.

2.2 Optimizing RTTP with Compiler Optimizations

We employed SUN Workshop Compiler C4.2 to optimize the RTTP code on the SUN 4-way E3000. We tried a combination of flags *-fast -native -xO5 -dalign*, which was expected to yield the best result for most sequential programs. Table 2 gives a profile of the RTTP when it was run under the corresponding optimization options. The table shows a speedup of 2.3 times over the *main* execution time in Table 1. The improvement is mostly due to the kernel loop of *sum_beam*. Recall that the sum_beam loop involves many multilevel indirect data references. The combination of optimization options reduces the loop overhead by minimizing the depth of indirect references.

Table 2: Profiling data due to sequential optimization

Function Name	Time (sec)	%Time
main	750.44	99.9
sum_beam	255.88	34.10
evaluate_plan_prob	177.82	23.70
scatter_point_dose	163.96	21.85
evaluate_plan_dose	64.99	8.66
hpsort	43.66	5.82

Next, we experimented with automatic parallelizing option *-xparallel*. It is a macro of a combination of *-xautopar -xdepend -xexplicitpar* flags. Columns two and three in Table 3 shows a profile of the RTTP due to the *-xparallel* option, together with the sequential optimization flags. From the table, we can not see any performance improvement from using the *-xparallel* flag. It is because RTTP's complex data structures and indirect access patterns as shown in the sum_beam loop are beyond the capability of the option [12].

Table 3: Profiling data due to parallelization

	-xparallel		SUIF	
Function Name	Time(s)	%Time	Time(s)	%Time
main	1298.32	99.9	1313.20	99.9
evaluate_plan_prob	415.39	32.00	449.70	34.24
scatter_point_dose	197.25	15.19	104.63	7.97
evaluate_plan_dose	178.90	13.78	227.83	17.35
hpsort	129.65	10.00	168.00	12.79

2.3 Parallel RTTP with Stanford SUIF

As an alternative to SUN's native compiler, we also experimented with Stanford SUIF [8] compiler to parallelize the RTTP system. The SUIF translates sequential programs into shared addressing space parallel codes. The code is run in a single-program, multiple-data (SPMD) paradigm, which contains calls to a portable run-time library. A profile of the RTTP generated from SUIF presetned in the right two columns of Table 3. The table shows SUIF cannot parallelize the loops either. Since the RTTP compiled by SUIf would generate wrong results when sequential optimization flags were enabled, the profile was obtained without using sequential optimization flags.

3 Optimizing RTTP on a Cray J916

3.1 Using Cray C compiler

On a Cray J916, we employed Cray vectorizing and autotasking tools to optimize the RTTP system. Using Cray *job accounting* (ja) and *hardware performance monitor* (hpm), we first present base performance (without optimization) of the RTTP code in the second column of Table 4. The performance data show that scalar operations dominated the program execution time. The big difference between the elapsed time and user CPU time also indicates that the RTTP involved I/O intensive tasks. Our optimization efforts had focused I/O processing and CPU computation managements.

Table 4: Performance on Cray J916 using various optimization techniques

	Base_Perf	I/O Opt	CPU Opt
MIPS	28.54	29.54	31.28
Floating ops/CPU second	3.41M	6.54M	13.72M
Vector int&logical ops	0.00M/cpu	0.00 M/cpu	0.04M/cpu
Vector floating point	0.00 M/cpu	0.00M/cpu	0.00M/cpu
Scalar functional unit	4.38M/cpu	4.01M/cpu	19.31M/cpu
Elapsed time	10916.43s	6212.69s	3316.12s
User CPU time	4004.13s	3107.53s	2795.75s
System CPU time	100.94s	197.94s	267.07s

We optimized the I/O performance by removing the unnecessary I/O operation; combining all the I/O operation that can be combined; changing some formatted I/O operations (*e.g.* scanf, printf) into unformatted I/O operations (*e.g.*ffread, ffwrite); eliminating some formatted I/O operations which are produced by some function for the input data of other functions by using memory-resident files or memory-resident predefined file systems. The results from the above I/O optimizations are presented in the third column of Table 4. Comparing to the base performance, we can see that the user CPU time was reduced drastically, although no vectorization was noticed.

The CPU computation performance can be optimized by several techniques: Scalar optimization improves the performance of scalar processing; Vector processing uses pipelining of arithmetic operations; Inlining optimization moves code from very small subprograms in-lined to the calling program; Autotasking allows parallel execution on multiple CPUs. In light of the fact that the RTTP code contains deep function calls and multi- level indirect data references, we enhanced the program by changing some reference values into arrays and optimized the code by using a combination of flags *-scalar3, -inline3, -vecter3*. The resulting performance listed in the third column of Table 4 shows a further improvement over the I/O optimization.

Finally, we used the autotasking tool provided by Cray to automatically parallelize the code. Unfortunately, we could not get any speedup from autotasking.

3.2 Optimizing RTTP with Stanford SUIF

As we did on SUN SMP servers, we experimented with Stanford SUIF for the optimization of the RTTP on the Cray J916. We first ported SUIF to Cray based on a Pthread runtime library. We compiled the SUIF code using the *-inline3, -vector3, -scalar3* flags of the Cray C compiler. Table 5 gives the execution results on four processors. A comparison with Table 4 clearly indicates the superiority of the Cray vectorization compiler to the SUIF.

Table 5: Performance with SUIF on Cray J916

MIPS	31.28
Floating ops/CPU second	2.01M/s
Vector integer&logical operation	1.06M/s
Vector floating point	0.91M/s
Scalar functional unit	7.30M/s
Elapsed time	4895.92s
CPU #1	3131.47s
CPU #2	1383.53s
CPU #3	356.12s
CPU #4	24.87s
System CPU time	192.71s

4 Parallel RTTP on Distributed Environments

From the optimization experience on Cray J916, we realized that the data input/output part of the RTTP system accounted for a large fraction of execution time. Since the real clinic application must be a distributed system, we employed Parallel Virtual Machine (PVM) to perform the RTTP components on different platforms. We assumed a SUN SPARCstation-5 workstation (170MHz SuperSPARC and 32MB memory) to be its front-end. We extracted the data input/output part out of the RTTP system and kept it run on the workstation. The other compute-intensive tasks, such as dose calculation and genetic optimization were run on a SUN SMP or Cray J916. We refer to the multiprocessor machines as RTTP servers. The RTTP has an input of more than 40MB floating point numbers and an output data of about 2KB floating point numbers. Since each beam dose is characterized by its independent beam name data, anastruct data, constraints data, skin data, and points data to calculate its dose, we assigned the input data onto the workstation and the functions of dose computation and optimization on a RTTP server before each beam's dose calculation was finished. We pipelined the next beam data from the RTTP server. To tolerate the network latency, we also take the advantage of *pvm_setopt* to set up direct task-to-task links for all subsequent TCP/IP communications. Each PVM process is maximumly optimized on each machine. Table 6 summarizes the timings in different configurations. The superiority of the configuration of SUN workstation and server is partly due to the fact that they are located within the same local area network.

Table 6: Performance on distributed environments

Configurations	Time(Seconds)
SUN workstation and SUN Enterprise	812.23
SUN workstation and Cray J916 (through PVM daemons)	4231.71
SUN workstation and Cray J916 (through direct connection)	4082.29

5 Hand-coding RTTP on clusters

The preceding sections uncovered that neither the vendor compilers nor research prototypes were able to parallelize or vectorize the RTTP system. For an illustration of the potential of its parallelization, this section reports the results from our manual parallelizations on a cluster of four SMP servers. One of the servers is a 6-way SUN E4000 with 1.5GB memory and the others are 4-way SUN E3000 with 0.5 GB memory each. Of the E4000 machine, four processors were used in implementation.

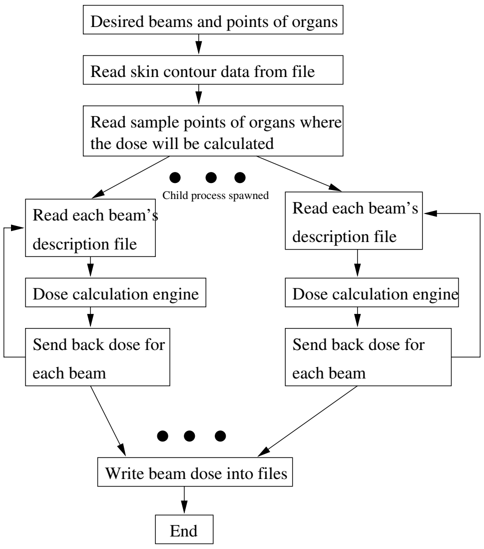

Figure 4: Message passing implementation of VRSplan

It is noticed that dose calculation for beams in the VRSplan subsystem is based on the same anastruct data and point description data and there are no dependences between beams. Accordingly, the VRSplan can be executed concurrently. Figure 4 shows a message passing VRSplan algorithm. It was implemented in both PVM and MPI libraries. Their parallel execution time, together with those from multithreading on a single SMP, are presented in Figure 5. From the figure, it can be seen that the multi-

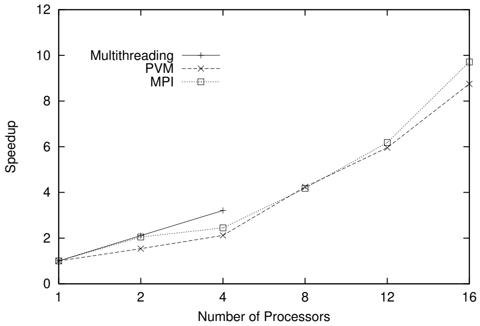

Figure 5: Speedup for VRSplan due to different programming approaches

threaded RTTP could gain a speedup of 3.21 on a 4-way SMP. The message-passing versions achieved speedups of up to 10 over a cluster of 16 processors. The moderate high speedup, in comparison with the marginal improvements from parallelizing compilers, demonstrates the weakness of the automatic parallelization tools.

For the GENplan, since the population of each generation is based on former generation except the first random generation, the main loop is essentially sequential and cannot be parallelized.. However, each function call within the main loop has many simple uniform loops which can be executed in parallel. The parallel algorithm of GENplan is shown in Figure 6. For each iteration, we parallelized the inner loops within each subroutine via three approaches: pthread, MPI, and String-based distributed shared memory (DSM) [5, 6]. The algorithm has a fork-join implementation and a large number of synchronization points. Figure 7 shows the speed-up due to different programming approaches. Again, the achieved moderate speedup, in comparison with the marginal improvements from parallelizing compilers, demonstrates the limitations of the current automatic parallelization tools.

6 Summary of Results

This paper has presented our extensive experience with the parallelization of a real-world 3-D radiation therapy treatment planning (RTTP) system. The RTTP system is rou-

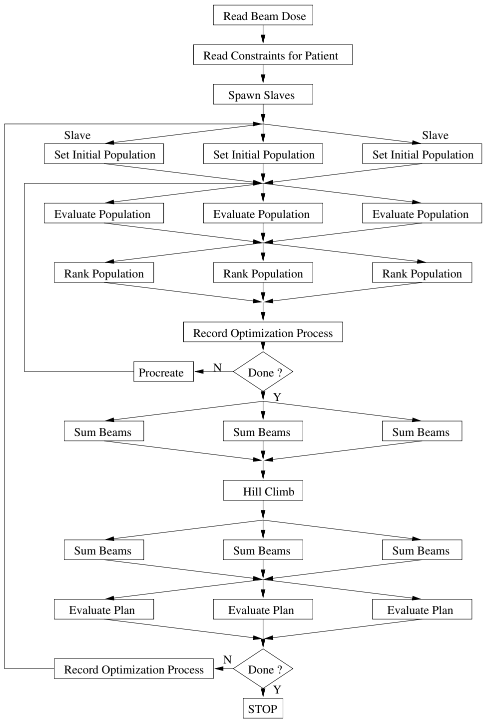

Figure 6: Message passing implementation of the GENplan

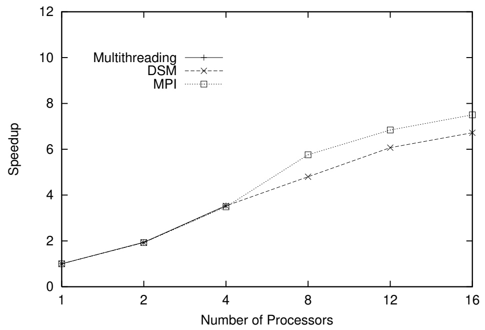

Figure 7: Speedup for GENpaln due to different programming approaches

tinely used to plan treatments for a majority of patients at the Gershenson Radiation Oncology Center at Harper Hospital in Detroit. It is a meta-problem, comprising two major components: dose calculation and genetic optimization. To accelerate the process for real-time planning in clinic, we integrated the two components and parallelized the system on a wide range of platforms, including Sun SMP servers, Cray J916 vector machines, and clusters of SMPs. We experimented with vendors' native optimization tools and Stanford SUIF parallelizing compilers for automatic parallelization of the codes. For comparison, we hand-coded parallel RTTP using multithreading, message-passing, and distributed shared memory tools. The experimental results showed that none of the automatic parallelization tools were capable of handling the real-world application, although plenty of parallelism exists in the codes. The difficulty is due to RTTP's dynamic and irregular computational behaviors.

References

[1] Cray Research, Inc. Cray C/C++ Reference Manual, SR-2179 3.0.

[2] Cray Research, Inc. Guide to Parallel Vector Applications, SG-2182 2.0.

[3] Cray Research, Inc. Optimizing Code on Cray PVM Systems, SG-2192 3.0.

[4] G. A. Ezzell, "Genetic and Geometric Optimization of Three-Dimensional Therapy Treatment Planning", Wayne State University, PhD Dissertation, 1994.

[5] S. Roy and V. Chaudhary, "Strings: A High-Performance Distributed Shared Memory for Symmetrical Multiprocessor Clusters", in *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, (Chicago, IL), pp. 90-97*, July 1998.

[6] S. Roy, V. Chaudhary, S. Jia, and P. Menon, "Application based evaluation of distributed shared memory versus message passing", in *Proc. of the ISCA 12th Int. Conf. on Parallel and Distri. Comput. Systetms*, August 1999.

[7] G. W. Sherouse, K. Novins, E. L. Chaney, "Computation of Digitally Reconstructed Radiographs for Use in Radiotherapy Treatment Design", *International Journal of Radiation Oncology, Biology, Physics*, 18:651-658 1990.

[8] B. Silson, *et al.*, "An Overview of the SUIF Compiler System", available at http://suif.stanford.edu/suif

[9] SUN Microsystems, SUN Workshop Compiler C User's Guide.

[10] SUN Microsystems, SUN Workshop Performance Profiling Tools.

[11] R. Tanese, "Distributed Genetic Algorithms for Function Optimization", PhD thesis, University of Michigan. Computer Science and Engineering, 1989.

[12] C. Xu and V. Chaudhary, "Time-Stamping Algorithms for Parallelization of Loops at Run-Time" in *Proc. of the 11th International Parallel Processing Symposium*, April 1997.