Application of the V–Ray Technology for Optimization of the TRFD and FLO52 Perfect Club Benchmarks to CRAY Y–MP and CRAY T3D Supercomputers

Alexander S. Antonov, Vladimir V. Voevodin
Moscow State University, Russia
e-mail: voevodin@vvv.srcc.msu.su

Abstract

The paper shows an application of the so-called V-Ray Technology for optimizing the TRFD and FLO52 Perfect Club Benchmarks to CRAY Y–MP and CRAY T3D supercomputers. We also discuss briefly the process of the determination of the potential parallelism of programs within V–Ray since this part of the technology played the key role for the successful optimization of the codes.

1. Introduction

Although the problem of optimizing programs to parallel computers exists for more than twenty years its satisfactory solution is not found so far. Several different approaches were suggested, however, none of them is able now to cover the whole scope of problems arising in optimization of industrial applications to vector/parallel as well as massively parallel supercomputers.

In this paper we would like to show an application of a principally new strictly mathematical approach based on the so called V–Ray technology. This technology incorporates recent advanced results from the graph theory and the theory of static analysis of programs [3, 4] and allows one to perform an exhaustive analysis of all required properties of programs starting from the commonly adopted data dependency and the control flow analysis up to the optimization of data distribution and data locality.

Depending on goals, this technology makes possible either to detect and describe parallel properties of a program or to transform source code in accordance with peculiarities of a particular parallel computer architecture.

It should be emphasized that the V–Ray technology provides a mathematical guarantee of detection of all required properties within a class of programs (for example, the whole resource of parallelism) and ensures that all suggested transformations are strictly equivalent even taking into account round-off errors.

As a numerical example we consider application of the V–Ray technology for optimizing the TRFD and FLO52 Perfect Club Benchmarks to CRAY Y–MP and CRAY T3D computers. We chose the TRFD code initially because of it's poor performance on both the vector and parallel architecture computers. We chose the FLO52 code because it is a very difficult program to parallelize and, to our knowledge, there has not been much success in this area at all.

The current state of implementation of the V–Ray technology in software tools enabled us to perform the most important part of the analysis of the codes by means of the V–Ray system, whereas its forthcoming versions should significantly facilitate transformation of source codes into parallel programs.

The original version of TRFD delivers the near 90 Mflop/s performance on the CRAY Y–MP C90 computer independently of the actual number of processors available (we remind that the peak performance of the CRAY Y–MP C90 computer equals 960 Mflop/s per processor). The best hand optimization of the code enables one to speedup execution up to 140 Mflop/s for a one processor. It is clear that the original TRFD code is not only poorly vectorized, but even hand transformations do not lead to any considerable improvement of the performance.

Is it possible at all to improve performance of this benchmark using only equivalent transformations of the code? Running a little ahead we give the answer right now: the V–Ray technology enables us, first, to improve performance of this code for one processor of CRAY Y–MP C90 computer up to 579 Mflop/s, second, to keep significant growth of performance for multiprocessor Y–MP configurations, and at last to find a way to a really scalable program with performance 1.7 Gflop/s on 256 processors of massively parallel CRAY T3D computer.

This paper is organized as follows. Section 2 gives a flavor of the V–Ray mathematical technology. Sections 3.1 and 3.2 present a brief description of the TRDF and FLO52 codes and the results of optimization for the vector parallel (CRAY Y–MP M90/C90) and massively parallel (CRAY T3D) supercomputers.

2. Some Mathematical Issues of the V–Ray Technology

Detection of the parallel structure of programs is the central part in the optimization process. In this section we describe briefly only the steps related to the determination of the potential parallelism of programs leaving other questions like searching for data distributions or program transformations aside.

Step 1. First of all we need to detect all types of dependences for an analyzed program. Many existing approaches use dependences between points of iteration space to analyze and describe structure of programs. Usually a subset of points is associated with each point of iteration space in accordance with exactness of method in use for discovering dependences. However we need to know more exact information about dependences to determine the real structure of the program. In particular, for each point A of iteration space we should find the only point B generating dependence. This point is either lexicographical maximum or minimum of just mentioned subset depending on the type of analyzed dependence. Hence we have to deal with oriented graphs where nodes are points of iteration space and arcs connect two lexicographically nearest points referenced the same data. Graphs are similar to each other but distinguished by the definition of the start and end points of arcs and by the memory access mode (read/write). As a rule arcs of these graphs represent lexicographical ascending or descending within iteration space.

From mathematical point of view all types of dependence graphs are identical therefore we apply one and the same algorithm for their construction. The final formal description of these graphs may be treated as a finite set of triples (F_i, Δ_i, N). N is a vector of unknown input integer variables of a program, the last entry of the vector is 1. Δ_i is a linear polyhedron within iteration space whose size and location depend on the vector N. The polyhedron is defined by a system of linear inequalities. The function F_i is linear and has the form $F_i = J_i x + \Phi_i N$. The matrices J_i, Φ_i are constant. The domain of the function F_i is the polyhedron Δ_i. The polyhedrons Δ_i may be intersected and may not cover the entire iteration space. The values of the functions F_i are points of iteration space. The points $J_i x + \Phi_i N$ and x are the start and end points of an arc respectively.

We have developed an efficient and fast algorithm to construct the complete set of triples by a source code, i. e. to compute coefficients of linear inequalities describing the polyhedrons Δ_i and coefficients of the matrices J_i, Φ_i. Unlike many other algorithms it does not apply time-consuming integer programming technique. The key feature is using an important characteristic property of lexicographical order that we call L-property [3].

Dependence graphs of some programs might be more complicated. In the general case domain of the functions F_i is an intersection of polyhedrons Δ_i with some regular lattices. Parameters of the lattices can be determined also by source code in accordance with general theory (the actual version of the V–Ray System does not support lattice analysis because in practice we have met them only twice and found a way to process them by much simpler methods). It should be noted that if due to some reasons a dependence graph can not be expressed as a set of triples, than we use triples for its minimal extension keeping the further analysis without changes.

The result of the first step is explicit symbolic representation of all dependences. Exactly this form of representation enables us to solve many problems necessary for efficient exploration of parallel structure of algorithms and programs.

Step 2. The next step is detection of ParDo loops. Taking into account explicit formulas for data dependences this problem can be solved easily. To examine ParDo property one may use a few criteria (sufficient as well as sufficient and necessary) testing some joint properties of matrices J_i and Φ_i. The actual number of loops and nesting structure do not affect seriously complexity of the criteria. For example,

Statement: *for a loop to be a ParDo loop it is necessary and sufficient that for any triple of an algorithm graph of this loop the inclusion $\Delta_i \subset G_i$ be true, where*
— *Δ_i is a polyhedron from a triple,*
— $G_i = \left\{ \begin{smallmatrix} f_1 \le i_1 \\ i_1 \le f_1 \end{smallmatrix} \right.$,
— *i_1 is a parameter of the loop under analysis,*
— *f_1 is the first component of the vector function F_i*

Innermost ParDo loops are better suited for vector processing while outermost ones for parallel execution on multiprocessor computers. Hence sometimes for optimal mapping of programs onto a particular computer we would like to use loop interchange. To test whether it is possible to apply this technique or not we make use a statement requiring quite trivial analysis of triples:

Statement: *in order to interchange two neighbouring loops in a perfect nest of loops it is necessary and sufficient that for any triple of an algorithm graph of this nest the inclusion $\Delta_i \subset G_i$ be true, where*
— *Δ_i is a polyhedron from a triple,*
— *$G_i = \{f_2 \leq i_2$,*
— *i_2 is a parameter of the inner loop,*
— *f_2 is the second component of the function F_i*

The second step mostly deals with general properties of programs which on the one hand do not depend on a computer architecture (the output of this stage might be a marked original code), but on the other hand can be easily detected after simple analysis of the explicit formulas representing data dependences. An interesting observation: for the large part of realworld applications this information suffices for the efficient target optimization. Notice, that advanced analysis of dependences should be continued on the third stage if we want to adjust well program structure with peculiarities of a particular computer and compiler.

Step 3. ParDo loops describe coordinate parallelism within iteration space. However resource of coordinate parallelism could be not sufficient therefore the main purpose of this step is discovering "skew" parallelism based on the notion of *schedule*.

Let an oriented graph be given in iteration space. A strict (generalized) schedule of the graph is a real function defined on iteration space and strictly increasing (nondecreasing) along arcs of the graph. Schedules have numerous very important properties. Suppose, for example, the graph be an algorithm graph. Let us take any schedule and examine its level surfaces. These surfaces split iteration space into nonintersecting sets, where each set corresponds to one level. The schedule gives equal value for all points of one set. Since the schedule does not decrease along arcs of the algorithm graph one may execute operations set-by-set according to growth of schedule values. If the schedule is strict then points of any set can not be linked by arcs and hence these operations can be run simultaneously. In other words it means that search for computational parallelism in programs we reduce to the investigation of schedules for its algorithm graph.

The third step is completely intended to determining and analysis of schedules represented similar to graph description as a finite set of triples (f_i, ∇_i, N). N is a vector of input integer variables of a program, ∇_i is a linear polyhedron within iteration space whose size and location depend on the vector N. The linear functional f_i sets a schedule and has the form $f_i = < j_i, x > + < \varphi_i, N >$, where $< *, * >$ means a dot product and the vectors j_i, φ_i are constant. The domain of the functional f_i is polyhedron ∇_i. Unlike triples of dependence graphs polyhedrons ∇_i are nonintersecting and cover the entire iteration space.

There is certain freedom in determining of the polyhedrons ∇_i. In particular, we may consider them identical to domains of loop parameters or take into account that arcs of dependence are actually piecewise and so on. Assume that we fix polyhedrons ∇_i somehow. A condition of nondecreasing of schedules along arcs is equivalent to inequality $At \leq 0$, where t is a vector of direct sum of all vectors j_i and φ_i. The matrix A is computed unambiguously by triples of the graph, polyhedrons ∇_i and domains of the loop parameters. Hence the schedule search problem we reduce to searching solutions of the system of linear inequalities $At \leq 0$. We call this system as a system of informational closure.

At the end of the third step we should find quite representative set of solutions of the system $At \leq 0$ in view of one more property of schedules. If do not go into details it can be stated as: if we have found s independent schedules than by means of these schedules we can transform program to a new form where each nest of loops with depth no less than s will have at least $s - 1$ ParDo loops.

3. Sample V–Ray Problems

As our sample problems we have chosen two application codes from the Perfect Benchmarks. The Perfect Club suite has been used for many years for evaluating of real performance of parallel computers on industrial applications. By now there are published a lot of papers (for example, [1, 2]) describing structure of each code. However, just that high popularity of the suite seemed attractive to us: it is much more interesting to produce a nontrivial result in the field where many researchers have worked very hard.

3.1. Optimization of the TRFD code for Y-MP

Analysis of TRFD involved the following steps:

1. First, it was necessary to reconstruct TRFD's multidimensional arrays from their compact one dimensional form. After completion of this step it became possible to describe TRFD in the terms of a linear model as is needed by the V–Ray Technology.

2. The most important part of the optimization process is the determination of the exact information-

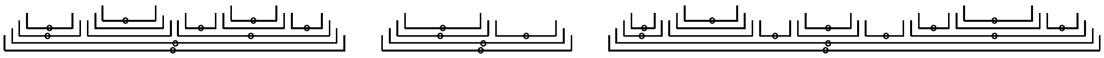

Figure 1. Marked loop profile of TRFD.

M90 CPUs	Peak perform. Mflop/s	Baseline Mflop/s	Manual opt. Mflop/s	V–Ray opt. Mflop/s
1	333	56.19	81.72	247
4	1332	54.86	261.64	822
8	2664	54.34	481.03	954
C90 CPUs	**Peak perform. Mflop/s**	**Baseline Mflop/s**	**Manual opt. Mflop/s**	**V–Ray opt. Mflop/s**
1	960	89.5	139.71	579.7
8	7680	89.6	962.68	2440.5

Table 1. Results for CRAY Y-MP supercomputers.

al structure of the program. Conversion of the program to this analyzable form enables us to determine all necessary parallel properties, for example, the existence or absence of dependency between statements and of separate parts of the program, locality of data references, the possibility for array privatization, and so on.

3. Once we know the exact "informational structure" of the code we can easily extract the whole resource of parallelism. In particular, the total resource of the so-called coordinate parallelism of the subroutine OLDA which is similar to the Par-Do property of loops can be illustrated by the marked loop profile (loops with independent iterations are marked by dots at fig. 1).

   Note that at least one important conclusion immediately follows from this picture: if one relies only on the structure of the innermost and outermost loops of the routine then the considerable part of its resource of parallelism will remain hidden and unusable.

4. From the previous step, we discovered that TRFD possesses a large reserve of coordinate parallelism therefore for this program we could use quite simple loop transformations to increase vector length, to perform vectorization along the leading dimension of arrays, to remove dependencies from loops, and to remove superfluous synchronization barriers.

5. The analysis of data locality is very important for efficient execution of programs on any kind of parallel computers. In practical use, the objective of this process is to reduce the data transfers between main memory and registers. In terms of the V–Ray technology, it requires an analysis of data locality and the expediency of temporary use of arrays. In this particular TRFD benchmark this optimization yielded a significant improvement of performance.

The final results of optimization for CRAY Y–MP M90/C90 computers are shown in tab. 1 which adopts the following notation:

- 'Peak perform.': stands for the theoretical performance of the computers;
- 'Baseline': corresponds to the performance of the original (nontransformed) code;
- 'Manual opt.': shows previously the best results of manual optimization of the code;
- 'V–Ray opt.': shows the performance after optimization of the TRFD code using our approach.

We would like to especially note that all TRFD optimizations were performed using equivalent Fortran transformations with no assemble fragments and without calls of highly optimized library routines.

3.2. Optimization of the TRFD code for T3D

The method for optimizing the TRFD code for the T3D computer differs from the methods used for vector/parallel computers. Our objectives for T3D were:

1. To identify a number of parallel computation branches that would be comparable to the number of processors available;
2. To find a data distribution strategy for the arrays of the code that would be consistent with the code's parallelism. This step is crucial for minimizing communication; and
3. To enhance the "locality of the computations" and the "locality of data" inside each processor for more effective use of the alpha-chip's cache memory.

From the work we did to vectorize TRFD, we already knew that the total potential resource of parallelism of the code is extensive. However, while investigating the set of possible data distributions among processors, we proved that TRFD possessed the following crucial properties:

1. Within each individual nest of loops, a data distribution strategy exists that supports two-dimensional parallelism. Two-dimensional parallelism implies independence of all iterations of two loops inside a nest. By using a consistent data distribution strategy within the loop nest, we can avoid the need for data transfer between iterations.
2. A data distribution strategy that could be used across all three nests that would also be consistent with the use of two-dimensional parallelism within each nest *does not exist*. Moreover, there does not exist a uniform distribution for all nests consistent with the use even of one dimensional parallelism within each nest.

Following our objective to have a highly scalable code we decided to use different data distribution strategies within each of the nests and to perform the redistribution of data between them.

The parallel version for the CRAY T3D uses Elegant Mathematics' communication library. The results of optimization are presented in tab. 2.

CRAY T3D PEs	V–Ray opt. seconds	V–Ray opt. Mflop/s
4	6.013	71.66
8	3.215	134.01
16	1.702	253.43
32	0.896	480.85
64	0.494	871.44
128	0.301	1435.96
256	0.253	1700.93

Table 2. Results for the CRAY T3D supercomputer.

3.3. Optimization of the FLO52 code for T3D

Our FLO52 example is a much more complicated code than TRFD. The original code consists of approximately 25 subroutines. 11 subroutines consume a considerable part of the total time and should be parallelized.

The final version of FLO52 contains more than 60 communication points and the total number of data transfers during run-time is more than 35,000, where 4,440 are all-to-all.

We show data for three cases (fig. 2). The small case is the small variant of the Perfect Club suite with a rectangular X by Y domain. For this case we can see that performance degradation begins immediately at the 32 PE configuration. Predictably, the 4X larger version provided in the Perfect Club Suite moves the point at which performance degradation begins. We further extended FLO52 ourselves to include a 16X larger version of the program and see the degradation start point moves as expected. Nevertheless, the 16X larger version is still not so big as it is able to be run on a single PE.

The most important property of the parallel FLO52 code becomes evident from this figure: the larger problem size, the higher efficiency of the code. The reason of the poor speedup of the small case is too small problem size and as a result data transfers completely dominate computations.

The key point for FLO52 is that, by using V–Ray, we are certain that all resources of parallelism have been exploited and that any additional parallelization efforts using the FLO52 algorithmic approach to solution of the application problem would be pointless. This is extremely valuable information, for example, for the code manager responsible for implementation of existing application programs on parallel computers.

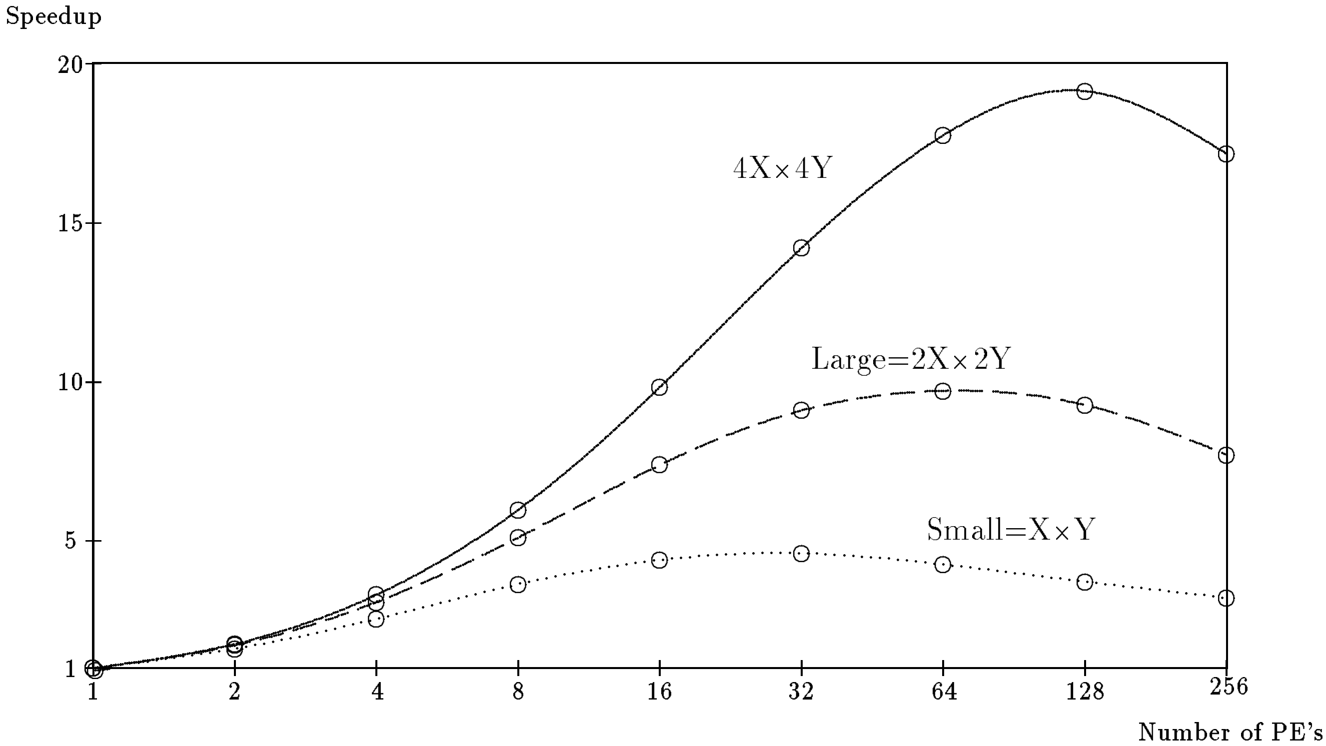

Figure 2. Speedup of FLO52.

4. Conclusions

The process of optimization of programs for parallel computers involves many difficult problems related to the structure of program to be analyzed as well as to the properties of the target computer architecture. To resolve them successfully we use the V–Ray technology that not only gives keys to their solutions but also makes possible to understand whether any further improvement of the performance is feasible. We have shown the potential of the V–Ray technology using only the TRFD and FLO52 Perfect Benchmarks and we hope to demonstrate soon its capabilities when optimizing other benchmarks/applications to the wide range of parallel computers.

5. Acknowledgments

The authors would like acknowledge CRAY Research, Inc. (USA) for providing computer resources and its specialists J. Brooks, Q. Sheikh and R. Numrich for the assistance when performing numerical experiments.

References

[1] G. Cybenko, L. Kipp, L. Pointer, and D. Kuck. Supercomputer performance evaluation and the perfect benchmarks. Technical Report 965, CSRD, University of Illinois at Urbana, 1990.

[2] C. M. Grassl. Parallel performance of applications on supercomputers. *Parallel Computing*, 17:1257–1273, 1991.

[3] V. V. Voevodin. *Mathematical Foundations of Parallel Computing*, volume 33 of *Computer Science Series*. World Sci. Publ. Co., Singapore, 1992.

[4] Vl. V. Voevodin. Theory and practice of parallelism detection in sequential programs. *Programming and Computer Software*, 18(3), 1992.