Acceleration of Integer Programming by Siblings' Exchange of Pruning Thresholds

Iaakov Exman* and **Daniel Citron**
Yanex - Computing Spaces
POB 8687, Jerusalem 91086
Israel

Extended Abstract

Concise Summary

An integer programming problem which is solved by the enumerative approach was parallelized using PVM [1]. The global cost function calculation for each feasible configuration solution is independent of the others, thus amenable to straightforward parallelization. On the other hand, the enumeration of nodes of the tree being searched grows exponentially. A simplified heuristic cost, depending only on the current node is a must for efficient pruning, but not enough for all practical purposes.

This paper shows that exchange of pruning information by sibling processes working in parallel, dramatically shortens the whole calculation, while preserving the quality of the resulting configuration.

1 The Optimization Problem

Given a set of electric power sources, the problem is to schedule their maintenance outages along the year, obeying a set of constraints. Typical constraints are seasonal consumption peaks, reserve tolerances, geographical distribution, order of maintenance. The solution of choice optimizes a global cost function.

The approach adopted is to state the constraints as a system of inequalities and to search for solutions by implicit enumeration (see e.g. [4]; for more recent literature see e.g. [3]) One visits nodes of a tree, where each tree level corresponds to a resource and each node stands for the assignment of a certain week for the outage start of the resource in question.

*also at: Institute of Computer Science, Hebrew University of Jerusalem, Jerusalem 91904 Israel. Email: ianex@cs.huji.ac.il

Since the tree is very large, one must have recourse to a *local* cost function, i.e. computed in each node and much cheaper than the global one. It serves to empirically prune the tree to manageable dimensions. An adaptive pruning threshold vector, where each element corresponds to a level in the tree, is maintained. This vector gets tighter and tighter at regular intervals. One assumes that the local function, reasonably correlates with the global one.

Integer programming yields a series of feasible *configurations*. Each configuration is a weekly schedule for outage starts of all resources in a whole year. Configurations are sorted by the resultant global cost. The best solutions are finally reported.

This approach works for problems with moderate degrees of freedom for maintenance outage starts. When the windows for the beginning of maintenances are larger, sequential computation may be intolerably long, and parallelization is called for.

2 Configuration Costs Speedup

The calculation of the global cost function per configuration is quite expensive. But this amount of computation is essentially constant, and it is independent of other configurations.

Thus parallelization is simply done by a master process dividing configurations among the available slaves.

In order to verify that parallelization was done properly, we estimated speedup in two circumstances. With the cluster of workstations under normal use - in competition with other users - and with a dedicated cluster with no other users.

We wished to measure effective speedup of the parallelized code. Therefore time is measured by wall clock of the whole computation T_w, from which one subtracts the average measured time spent by the master alone $(T_m)_{aver}$.

$(T_m)_{aver}$ is independent of the cluster size to a very good approximation. It reflects very well the code sections which were not parallelized, with a small overhead due to packing and multicasting data to the slaves. It is comparable to the corresponding part of the sequential computation.

Speedup is the ratio of these quantities for the sequential and parallel cases:

$$speedup = \frac{T_{w_{seq}} - (T_m)_{aver}}{T_{w_{paral}} - (T_m)_{aver}} \tag{1}$$

Fig. 1 enables comparison of speedup in a cluster in the regular multi-user situation, with a dedicated cluster. The latter cluster size for which speedup was still achieved is larger and the slope steeper than in the cluster with competition with other users. These results are comparable to other published applications of PVM (see e.g. [2]).

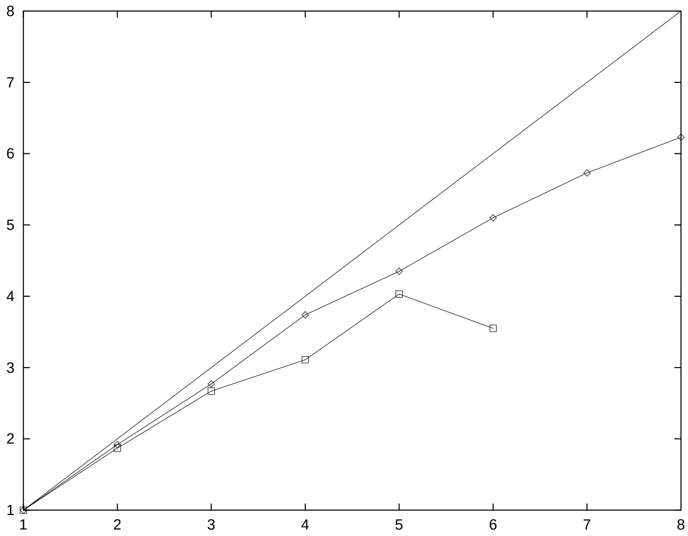

Figure 1: *Speedup against number of processors. PVM3.0 for clusters containing 1 to 8 DEC-Stations (Ultrix). Lower: cluster with multi-user competition; Middle: dedicated cluster; Upper: Ideal linear speedup.*

3 Pruning Acceleration

Parallelization of the Integer Programming part of the calculation, leading to the set of feasible configurations, is much more interesting.

In the sequential computation, the pruning threshold slowly improves as the computation progresses.

Exchange of threshold update information among children tasks, causes faster pruning of all participants, leading to dramatic acceleration of the overall computation.

Some fine points of the parallellization are as follows:

- Child tasks multicast at their own initiative their threshold vectors. When a task comes to update his own vector, it checks for incoming vectors from its siblings. Threshold updates should be done elementwise. If whole vectors were exchanged, neither configurations of the receiver, nor of the sender would satisfy them, which would cause the collapse of the whole tree.

- Pruning is controlled by the length of intervals between consecutive updates. Shorter intervals causes sharper pruning.

- Assignment of sub-trees to children processes, as well as the final sorting is done by the master process.

computation type	measured time
sequential	119.
2 siblings in 1 machine	44.
4 siblings in 1 machine	17.
2 siblings in 2 machines	24.
4 siblings in 2 machines	9.

Table 1: *Acceleration of Integer Programming. Overall wall clock times in minutes. PVM3.3 in DEC-Alpha (OSF-1) stations.*

In table 1 are results of timing of a representative test case. The measurements are straight wall clock times of the overall calculation, without further elaboration. There is no linear speedup ceiling, as parallelization implies a change of heuristics, thus a change of the algorithm, even though modifications of the sequential code were of a delicate surgical nature.

One clearly observes from results in the table, that acceleration of pruning is a consequence of very existence of more than one process, and it is significant even for a few processes in a single machine.

In conclusion, for integer programming, where one must have recourse to pruning heuristics to make computation of practical value, parallelization of sequential code by PVM is feasible by a reasonable amount of coding investement and obtains dramatic results in terms acceleration of the computation.

Acknowledgment

The sequential Integer Programming algorithm was developed by Dr. V. Gurevich at the R & D Division of the Israel Electric Corporation.

References

[1] Beguelin, A. et al. - "Users' Guide to PVM Parallel Virtual Machine " - Technical Report - Oak Ridge National Laboratory - Version 3.3 - May 1994.

[2] Ewing, R.E. et al. - "Distributed Computation of Wave Propagation Models Using PVM" - IEEE Parallel Dist. Technology, **2**, 26-31 (Spring 1994).

[3] Pardalos, P.M. (ed.) - "Advances in Optimization and Parallel Computing" - Elsevier - Amsterdam - 1992.

[4] Taha, H. A. - "Integer Programming - Theory, Applications and Computations" - Academic Press - NY - 1975.