Parallelization of Sphere-Decoding methods

Rafael A. Trujillo[1][2], Antonio M. Vidal[1], Víctor M. García[1], and Alberto González[3]

[1] Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, España
e-mail:{rtrujillo, avidal, vmgarcia}@dsic.upv.es
[2] Departamento de Técnicas de Programación,
Universidad de las Ciencias Informáticas,
Carretera a San Antonio de los Baños Km 2 s/n, La Habana, Cuba
e-mail:trujillo@uci.cu
[3] Departamento de Comunicaciones
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, España
e-mail:agonzal@dcom.upv.es

Abstract. *Sphere-Decoding* (SD) methods are branch-and-bound-like techniques used for optimal detection of digital communications signals over in wireless MIMO (Multiple input Multiple Output) channels. These methods look for the optimal solution in a tree of partial solutions; the size of the tree depends on the parameters of the problem (dimension of the channel matrix, cardinality of the alphabet), and such search can be much more expensive depending on these parameters. This search often has to be carried out in real time. This paper presents parallel versions of the *Sphere-Decoding* method for different parallel architectures with the goal of reducing the computation time.

1 Introduction

Let us consider the following minimum squares problem:

$$\min_{s\in\mathcal{D}^m} \|x - Hs\|^2 \tag{1}$$

where $\mathcal{D}$ is a set of discrete values that can be finite or infinite. Since $\mathcal{D}$ is discrete, this can be considered as an Integer Programming problem.

The simplest method to solve this problem would be to solve the unconstrained minimum square problem

$$\min_{s} \|x - Hs\|^2 \tag{2}$$

and then "truncate" the solution to its closest value in $\mathcal{D}^m$. However, for many problems (specially when H is bad conditioned) this gives wrong results. Therefore, special methods have to be applied.

Usually, this problem is described in terms of *Lattices*. Given $v_1, \cdots, v_m$ a set of linearly independent vectors, a lattice is the set of vectors

$$\lambda_1 v_1 + \cdots + \lambda_m v_m \quad \lambda_1, \cdots, \lambda_m \in \mathbb{Z} \tag{3}$$

If the elements of $\mathcal{D}$ are equally spaced (as is the case in communication problems) the set $\mathcal{D}^m$ forms a rectangular lattice like the displayed in Figure 1(a). When the elements of $\mathcal{D}^m$ are multiplied by the channel matrix H, they will form a skewed lattice (see Figure 1(b)).

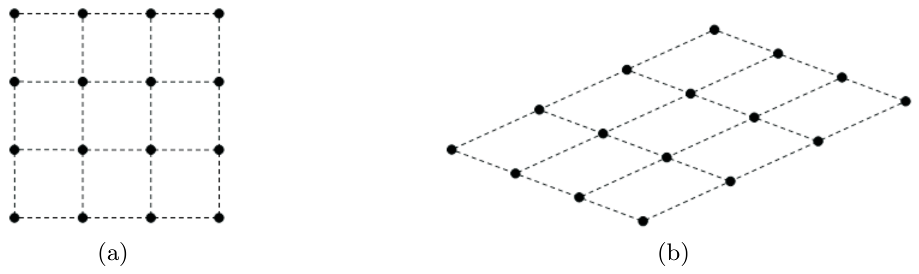

Fig. 1. (a)Rectangular Lattice; (b)Skewed Lattice.

The problem (1) is equivalent to the problem of finding the closest point (to a given point x) in a lattice with generating matrix H. This problem is known to be NP-Complete. It appears in the wireless communications field, where digital communications signals sent through systems with multiple send and receive antennas (*multiple input - multiple output* or *MIMO* systems) must be correctly "decoded" [4], [11].

The systems we are interested in, are composed of M transmit antennas and N receive antennas, through which a signal $\bar{s} = [\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_M]^T \in \mathbb{C}^M$ is transmitted. Real and imaginary parts of each component of the transmitted signal belong to a discrete set $\mathcal{D}$, finite ($|\mathcal{D}| = L$) and named *constellation* or *symbol alphabet*. The received signal $\bar{x} \in \mathbb{C}^N$ is a linear combination of the transmitted signal $\mathbf{s}$ plus a white gaussian noise term $\bar{v} \in \mathbb{C}^N$, with variance σ^2

$$\bar{x} = \bar{H}\bar{s} + \bar{v}. \tag{4}$$

Here, the channel matrix $\bar{H}$ is a general complex-valued matrix with N rows and M columns that models the MIMO channel response.

To simplify the programming details, usually the complex model (4) is transformed in a real model, where the vector s of dimension $m = 2M$, and the vectors x and v of dimensions $n = 2N$ are defined as:

$$s = \left[\mathcal{R}(\bar{s})^T \, \mathcal{I}(\bar{s})^T\right]^T, \; x = \left[\mathcal{R}(\bar{x})^T \, \mathcal{I}(\bar{x})^T\right]^T, \; v = \left[\mathcal{R}(\bar{v})^T \, \mathcal{I}(\bar{v})^T\right]^T,$$

and the matrix H of dimensions $n \times m$ as:

$$H = \begin{bmatrix} \mathcal{R}\left(\bar{H}\right) & \mathcal{I}\left(\bar{H}\right) \\ -\mathcal{I}\left(\bar{H}\right) & \mathcal{R}\left(\bar{H}\right) \end{bmatrix}$$

Thus, the real model equivalent to (4) is given by:

$$x = Hs + v. \tag{5}$$

The search within the whole lattice looking for the optimal solution of (1) where $|\mathcal{D}| = L$ would require an exponential time. However, there are better methods that take into account the special characteristic of problem (1). Examples of such methods are Kannan's algorithm [8], (where the search is limited to a rectangular parallelepiped), and the algorithms proposed by Fincke and Pohst [3] and improved by Schnorr and Euchner [9],[1] known as *sphere-decoding*, where the search is limited to an hyper-sphere of a given radius and with center in the point x.

In any case, *Sphere-Decoding* methods can be quite costly in time and memory when the problems grow in complexity; either by an increase in the number of transmitting-receiving antennas, by using larger alphabets, or by an increase in the variance of the noise.

In this paper several variants of the sphere-decoding algoritm are presented, for use in different parallel architectures. Other algorithms were developed, but, only those that gave better results are discussed.

A description of the *Sphere-Decoding* method is given in the next section, along with a formal description of the method following a *branch-and-bound* scheme. In the second section the parallelization of *Sphere-Decoding* for shared memory, distributed memory, and hybrid environments, is discussed. Finally, the experimental results and the conclusions will be presented.

2 Sphere-Decoding

The main idea in the *Sphere-Decoding* algorithm is to limit the search to the points in the lattice located into the sphere with center at the given vector x and radius r (see Fig. 2). Clearly, the closest point to x into the sphere shall be the closest point to x in the whole lattice. Since the search space is reduced, so is the computational complexity.

Mathematically speaking, SD algorithms find all the vectors $s \in \mathcal{D}^m$ such that

$$r^2 \geq \|x - Hs\|^2 \tag{6}$$

and then select the one minimizing the goal function. Using the QR decomposition of the matrix H and the orthogonality of the matrix Q, the condition (6) can be written as:

$$r^2 \geq \left\| x - \begin{bmatrix} Q_1 \; Q_2 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix} s \right\|^2 = \left\| Q_1^T x - Rs \right\|^2 + \left\| Q_2^T x \right\|^2$$

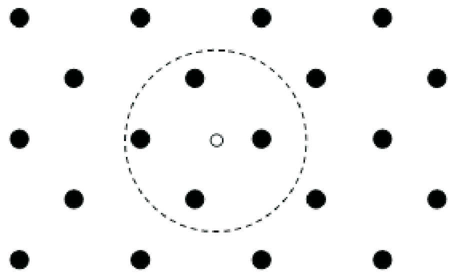

Fig. 2. Idea behind the sphere-decoding

or, equivalently:

$$r^2 - \left\|Q_2^T x\right\|^2 \geq \left\|Q_1^T x - Rs\right\|^2 \tag{7}$$

Defining $y = Q_1^T x$ and $r'^2 = r^2 - \left\|Q_2^T x\right\|^2$ (7) can be rewritten as

$$r'^2 \geq \|y - Rs\|^2 \tag{8}$$

Since R is upper triangular, equation (8) can be written as

$$r'^2 \geq (y_m - R_{m,m} s_m)^2 + \|y_{1:m-1} - R_{1:m-1,1:m} s_{1:m}\|^2 \tag{9}$$

From this last condition, it can be deduced that a necessary condition for Hs to be located inside the sphere is that $r'^2 \geq (y_m - R_{m,m} s_m)^2$, or, equivalently, that the component s_m belongs to the interval

$$\left\lceil \frac{-r' + y_m}{R_{m,m}} \right\rceil \leq s_m \leq \left\lfloor \frac{r' + y_m}{R_{m,m}} \right\rfloor \tag{10}$$

where $\lceil \xi \rceil$ is the smallest element of the constellation greater or equal than ξ, and $\lfloor \xi \rfloor$ is the greatest element of the constellation smaller or equal than ξ. Therefore, for each value of s_m inside the interval (10), it is possible to determine the interval where the values of s_{m-1} will lie

$$\left\lceil \frac{-r'_{m-1} + y_{m-1|m}}{R_{m-1,m-1}} \right\rceil \leq s_{m-1} \leq \left\lfloor \frac{r'_{m-1} + y_{m-1|m}}{R_{m-1,m-1}} \right\rfloor \tag{11}$$

where $r'^2_{m-1} = r'^2 - (y_m - R_{m,m} s_m)^2$ and $y_{m-1|m} = y_{m-1} - R_{m-1,m} s_m$.

The algorithm continues following the same procedure to determine $s_{m-2}, s_{m-1}, \cdots, s_1$.

If no solution is found, the radius r must be increased and the algortihm must be executed again. A more detailed description of the *Sphere-Decoding* method, as well as an analysis of its computational complexity, can be found in [7].

The search in *Sphere-Decoding* belongs to the Branch–and–Bound general class; next, an algorithm implementing a general Branch-and-Bound search is displayed:

Algorithm 1 General Branch-and-Bound Algorithm to find the best solution

```
Variables:
S: LinearDataStructure;
N, solution : Node;
childs : ARRAY [1..MAXCHILDS] OF Node;
nChilds : INTEGER;
solutionValue : REAL;
 1: solution ← NoSolution(); solutionValue ← MAX
 2: Generate a initial node $N_0$
 3: Add(S, $N_0$) {Add $N_0$ to the Linear Data Structure $S$}
 4: while S is not empty do
 5:   N ← Extract(S) {A node of $S$ is extracted and assigned to $N$}
 6:   [childs, nChilds] ← Branch(N);
 7:   for i = 1 to nChilds do
 8:     if IsAcceptable(childs[i]) then
 9:       if IsSolution(childs[i]) then
10:         if Value(childs[i]) < solutionValue then
11:           solutionValue ← Value(childs[i])
12:           solution ← childs[i]
13:         end if
14:       else
15:         Add(S, N) {Add the node $N$ to the Linear Data Structure $S$}
16:       end if
17:     end if
18:   end for
19: end while
20: return solution
```

In the *Sphere-Decoding* method, the k-level nodes are the lattice points inside the sphere with radius r'_{m-k+1} and dimension $m-k+1$. The leaves of the tree would be the solutions of (6).

To fit the *Sphere-Decoding* method into a *Branch and Bound* scheme, each node should keep the following information:

– Tree level: $m-k+1$
– Value of $y_{k|k+1}$
– Value of r'_k
– Components of the vector $\hat{s}$ determined up to this moment: $\hat{s}_m, \ldots, \hat{s}_k$

where $k = m, m-1, \ldots, 1$. The generation of branches of a node of level $m-k+1$ (given in the algorithm 1 by the routine `Branch`) should generate as many nodes as elements has the alphabet or constellation, and the Bounding (given in the algorithm 1 by the routine `IsAcceptable`) should be carried out accepting only the nodes whose component $\hat{s}_{k-1}$ falls within the interval

$$\left[\left\lceil \frac{-r'_{k-1} + y_{k-1|k}}{R_{k-1,k-1}} \right\rceil, \left\lfloor \frac{r'_{k-1} + y_{k-1|k}}{R_{k-1,k-1}} \right\rfloor\right] \tag{12}$$

All nodes whose level is m are solution nodes. The routine `Value` is defined for these nodes, (points inside the hyper-sphere), which would return the value of $\|x - H\hat{s}\|^2$.

Since the *Sphere-Decoding* performs a depth-first search, looking for the best of all possible solutions, the nodes are stored using a LIFO (*Last-In First-Out*) Heap. This data structure is initialized using a special node N_0 whose level in the tree shall be 0, the value of $y_{m+1|m+2}$ shall be the last component of the vector y, the value of r_{m+1} might be the initial radius and the vector $\hat{s}$ would not have any defined component.

3 Sphere-Decoding Paralellization

In this section several possibilities for parallelization of the *Sphere-Decoding* method shall be discussed. The models considered shall be the shared memory model, distributed memory model and a hybrid model where several shared memory multiprocessors are interconnected. A study about parallelization of *Branch-and-Bound* methods can be found in [5], where different techniques are considered for distribution of workload among processors. In this section, it is described the adaptation of these techniques, for parallelization of *Sphere-Decoding*, using specific characteristics of the problem. including as well several novel proposals oriented to minimize the communications and to correctly balance the workload.

3.1 Distributed memory Parallelization of Sphere-Decoding

The main issue in parallelizing *Sphere-Decoding* in a message-passing environment is the distribution of the tree among processors, so that the number of nodes (and the workload) is distributed as evenly as possible.

The parallel algorithm has the following structure: First, the *root* processor creates an initial node, and from this node starts a sequential execution of *Sphere-Decoding*, until enough nodes have been generated to be distributed among the rest of processors (at least one per processor). These nodes are distributed cyclically, so that each processor has its own data structure (the distribution is cyclic to avoid that some processors receive only nodes from the last levels, while others receive nodes from the first levels; the nodes from the last levels should generate much less work than the nodes from the first levels). See figure 3.

Then, in each iteration, each processor expands a fixed number of nodes in its structure. When these nodes are expanded, there will be a synchronization point so that all processors broadcast the state of its heap (empty or not) to the other processors. The processors whose heap is empty select randomly a processor to send a nodes *request* message. If it receives a negative (message *reject*) answer, it would choose another processor, until it receives a positive answer or has received a negative answer from all processors.

After expanding the pre-fixed number of nodes, the processors must check their queue of messages, to see whether they have some *request* message. In such

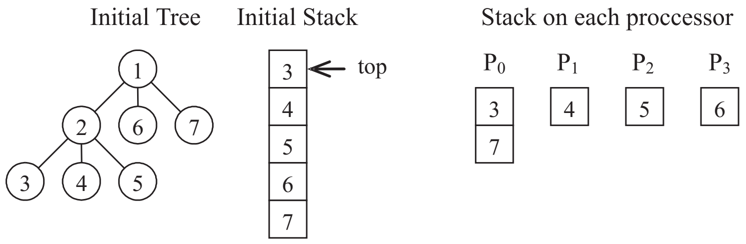

Fig. 3. Initial Expansion and distribution of nodes among processors

case, they answer with *reject* message if their structure is empty or *accept* if it has nodes to send. If a processor is going to send nodes to other processor, it will send a fixed percentage of nodes from its structure, chosen cyclically.

At the end of each iteration the processors must broadcast their best solution (to use it for comparison with solutions found in further iterations) and the state of their heap, to determine the end of the algorithm (which will happen when all heaps are empty).

An important issue is the choice of the number of nodes to expand by each processor. To avoid load imbalance, this number must be the same for each processor. This number should not be neither too small (it would increase the communications) nor too large (some processors might finish their search soon, and should wait a long time for the other processors to finish)

It is proposed by the authors that the estimation of the number of nodes to expand in a given iteration is carried out at the end of the former iteration; each processor estimates the number of nodes that might expand in the next iteration, broadcast the estimates to all processors, and the number of nodes to expand would be the median of all estimates.

The authors propose as well that each processor carries out its relative estimation taking into account the number of nodes in its heap, the levels of each node and the cardinal of the constellation $\mathcal{D}$. If the structure has l_1 nodes with level $m-1$ and l_2 nodes with level less than $m-1$, then the estimate of nodes that the processor might expand in the next iteration is

$$l_1 L + l_2 L^2 \tag{13}$$

In the parallel version of *Sphere-Decoding* implemented in this work, both proposals were included.

3.2 Shared memory Parallelization of Sphere-Decoding

One of the drawbacks of the distributed memory parallelization is that, as shown above, it is necessary to fix the number of nodes to be expanded in each iteration, and check the state of the heap of each processor (empty or not). This number of nodes to expand must be carefully estimated, to avoid extra communications or load imbalance. In a shared memory environment, this problem would not exist.

A first design for shared memory implementation would be that all processors share the heap, so that in each iteration each processor extracts a node, expands it and adds the generated new nodes to the heap. The algorithm would finish when the heap is empty.

The main problem of this design is that the operations of extraction and addition of nodes must be carried out in a critical section, that is, in a given time point only a single processor can add or extract nodes from the heap. It has been checked that this creates a severe bottleneck; therefore, to share the heap is not a good solution, and this means that each processor must keep and handle its own heap.

The algorithm proposed is, therefore, similar to the one described in the section 3.1. The main difference would be that there exist a global variable which controls whether any processor has an empty queue. This variable can be updated (within a critical section) by the first processor that becomes idle, and then by the other processors when a redistribution of the nodes is carried out. Therefore, all the processors search through their heaps until these become empty or until the global variable indicates that some processor finished its job. Then, all remaining nodes are collected in a global structure and redistributed cyclically over all processors.

This variant, which can be applied only in shared memory machines, decreases sensibly the likelihood of a workload imbalance.

3.3 Hybrid Parallelization of Sphere-Decoding

A different, hybrid, parallel architecture is formed by networks of shared memory multiprocessors (the memory of the global system is distributed, but the multiprocessors share a local memory). Of course, the message passing library MPI [10], can be used, although the message passing model may not be the most suitable to take full advantage of the architecture.

Keeping in mind the parallel SD algorithms for shared memory and for distributed memory, is easy to obtain an algorithm for the hybrid case. The algorithm would have the same parallel distribution that the distributed memory algorithm described in 3.1, while the search caried out in each multiprocessor would use the shared memory algorithm described in 3.2.

It must be considered that, in each iteration of the algorithm, each multiprocessor must select from its heap a fixed number of nodes to expand. Hence, the shared memory algorithm to be executed by the processors within a multiprocessor is slightly different from the algorithm described above. Apart from sharing a variable which indicates if a processor emptied its heap, they would share as well a variable with the number of nodes to be expanded in that iteration by the processor. Such variable should be decremented (in a critical region) by each processor every time that a node is expanded, and if this variable becomes zero all processors must stop to, through a reduction, determine the best solution found and rebuild the heap with the nodes not yet processed.

4 Experimental Results

The parallel algorithms outlined above where implemented in C, using the BLAS library [6] for the operations between vectors. The version for shared memory architecture was implemented using the OpenMP library [2], while the distributed memory version was build with MPI. Both libraries were used for the hybrid version. The algorithms were tested in a cluster composed of two PRIMERGY RXI600, each one with 4 Dual-Core 1.4 GHz Intel Itanium–2®processors, sharing a 4 GB RAM memory.

The experiments were designed so that the Sphere decoding algorithm generates a large number of nodes, and, consequently, the CPU time needed is also large. The sizes of the generated trees (total number of nodes) of the test problems were $5 \cdot 10^4$, $1 \cdot 10^5$, $5 \cdot 10^5$, $1 \cdot 10^6$, $4 \cdot 10^6$, $5 \cdot 10^6$, $8 \cdot 10^6$ and $1 \cdot 10^7$. These trees were generated in problems where $n = m = 8$, so that in all cases the number of levels of the tree is 8. It must be remarked that the execution time of the sphere decoding does not depend only on the number of nodes of the tree, but also on the number of possible solutions (last level nodes, which would represent lattice points inside the hyper-sphere). Given two trees with the same number of possible solutions, the one with smaller number of last-level nodes would need less execution time, since it would insert less nodes in the heap and there would be less nodes to expand.

Figure 4 shows the speedup of the MPI parallel version for the test problems considered. In the problems with smaller size (displayed with discontinuous lines) the speedup decreases for more than 4 processors. In the larger test cases there is a better speedup, although far from the optimum theoretical speedup.

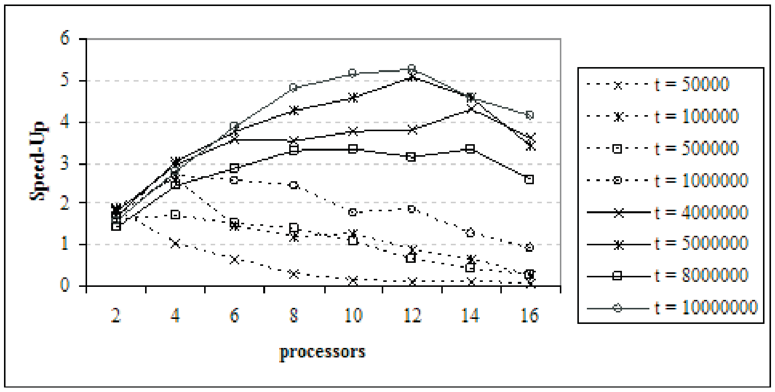

Fig. 4. SpeedUp of MPI parallel version of SD

The speedup for the OpenMP version is shown in Figure 5. This version was executed in one of the PRIMERGY RXI600 machines. It is quite remarkable that all curves present the same trend, and in all problems the optimal number of processors is 6. The speedups in this case are close to theoretical optimum, better in most cases than the obtained with the MPI version.

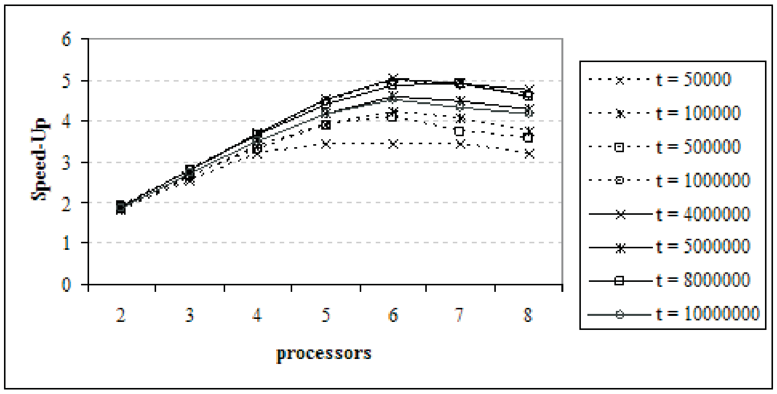

Fig. 5. SpeedUp of OpenMP parallel version of SD

Finally, the speedup of the hybrid version is displayed in Figure 6. This hybrid version was implemented in two levels, a global level implemented with MPI and a second, local level implemented with OpenMP. In some cases, the speedup obtained with this algorithm was far better than with the MPI or OpenMP versions. The problems in which the speedup was larger than 7 were those with a larger number of solution nodes in the tree (more than 70% of solution nodes). In problems where the percentage of solution nodes is small, the performance decreases, as in the case where the number of nodes is approximately $5 \cdot 10^5$. From these, only 11% were solution nodes.

5 Conclusions

Several possibilities for parallelization of the *Sphere-Decoding* method have been proposed. These cover three possible computing environments: Distribute memory, shared memory, and hybrid machines, where shared memory machines are connected. The hybrid version gave the greater speedups, although the shared memory version had more stable and consistent speedups. This was an expected behaviour, given the reduced communications and the possibility of a quite good work distribution. The workload balance is much more troublesome in distributed memory environments, so that the performance of the MPI and OpenMP+MPI versions suffers strong variations, depending on the structure of the tree generated during the search. The overall good performance of the OpenMP versions is quite relevant, since multicore processors and hybrid architectures are becoming increasingly popular.

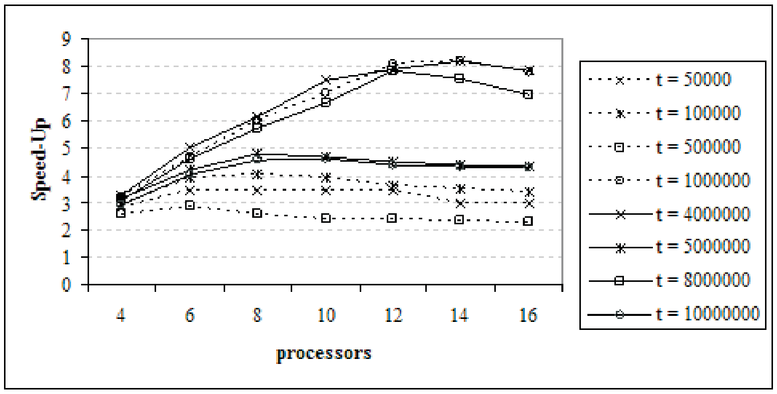

Fig. 6. SpeedUp of hybrid parallel version of SD

References

1. E. Agrell, T. Eriksson, A. Vardy, and K. Zeger. Closest point search in lattices. *IEEE Transactions on Information Theory*, 48:2201–2214, 2002.
2. R. Chandra, L. Dagun, D. Kohr, D. Maydan, J. McDonald, and R. M. *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers, 2000.
3. U. Fincke and M. Pohst. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Mathematics of Computation*, 44(170):463–471, 1985.
4. Gerard J. Foschini. Layered space-time architecture for wireless communication in a fading environment when using multi-element antennas. *Bell Labs Technical Journal*, 1:41–59, 1996.
5. A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to Parallel Computing*. Addison Wesley, 2003.
6. S. Hammarling, J. Dongarra, J. Du Croz, and R. J. Hanson. An extended set of fortran basic linear algebra subroutines. *ACM Trans. Mat. Software*, 1988.
7. B. Hassibi and H. Vikalo. On sphere decoding algorithm. i. expected complexity. *IEEE Transactions on Signal Processing*, 53:2806–2818, 2005.
8. R. Kannan. Improved algorithms for integer programming and related lattice problems. *ACM Symp. Theory of Computing*, 1983.
9. C.P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Programming*, 66:181–191, 1994.
10. M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: The Complete Reference*. MIT Press, 1996.
11. I. E. Telater. Capacity of multi-antenna gaussian channels. *Europ. Trans. Telecommun.*, pages 585–595, 1999.