Accelerating the Requirement Space Exploration through Coarse-Grained Parallel Execution

Zhongwei Lin and Yiping Yao

School of Computer, National University of Defense Tecnology
{zwlin,ypyao}@nudt.edu.cn

Abstract. The design and analysis of complex systems need to determine suitable configurations for meeting requirement constraints. The Monotonic Indices Space (MIS) method is a useful approach for monotonic requirement space exploration. However, the method is highly time and memory-Consuming. Aiming to the problem of low efficiency of sequential MIS method, this paper introduces a coarse-grained parallel execution mechanism to the MIS method for accelerating the process of requirement space exploration. The task pool model is used to receive and deploy hyperboxes for work balancing. To validate our approach, the speedup is estimated by a mathematical analysis and then an experiment is conducted in a PC cluster environment. The results show that high speedup and efficiency is achieved through our approach.

Keywords: requirement space exploration, coarse-Grained parallel execution, task pool model.

1 Introduction

A lot of practical problems, such as getting the system capability requirement indices [1, 2, 3, and 4], getting the effective light intensity space [5], and getting the effective radar coverage [5], can be transformed into the problem of requirement space exploration. These problems have a common feature: the requirement measure of these systems is related with several factors (indices), and their combination forms a huge parameter space in which we should explore to make the measurement satisfying requirement constraints. For example, the response time (requirement measure) of a web server is affected by several factors, including the CPU's frequency, memory, network, operating system, and current load, so we are anxious about what configurations of these factors which can satisfy the demand on the response time.

The problem of requirement space exploration can be defined formally as follows.

Definition 1. *Requirement space: suppose R^n is n-Dimensional Euclidean space, P is non-Empty subset of R^n, then $P \subset R^n$ and $P \neq \emptyset$, there exists a function f defined in P, $f : P \rightarrow R$, the requirement space S satisfies: $S \subset P$ and $\forall p \in S, f(p) \geq \alpha$, where α is the threshold.*

E. Altman and W. Shi (Eds.): NPC 2011, LNCS 6985, pp. 59–70, 2011.

Due to the complexity and uncertainty of some practical problems, it is very difficult or even impossible to get the analytic form of the function f, thus we need to determine the require space in other ways, rather than analytic way.

A few classic methods have been used to solve the requirement space determining problem. The system effectiveness analysis method [1] compares the system capabilities and the mission requirements in a common attribute space, while attaining the system mission requirements locus is difficult. ADC model [6] is a very strict mathematical method, but the calculation will increase exponentially with the increase of system state dimensions. In multi-Attribute analysis method [7], the design of the weights of each attribute should be very skillful, but very difficult. The Monotonic Indices Space (MIS) method [4] introduced by Hu Jianwen takes the typical complex system characteristics into account, and turns out an efficacious way.

Coarse-Grained Multicomputer (CGM) model is a well-known parallel method for multi-Replication simulations. Frank Dehne presented the method [8], described and proofed the time cost of the method, and employed it to solve Hausdorff Voronoi Diagrams problem [9]. CGM has been widely used, and turned out effective. Thierry Garcia and David Sem' employed CGM to solve the Longest Repeated Suffix Ending problem [10], Albert Chan introduced CGM to the next element search problem [11]. According to their results, CGM is highly effective.

The paper is organized as follows. In section 2, we introduce the MIS method in detail. The coarse-Grained approach is described in section 3. We analyze the speedup of our approach formally, and then present some experimental results in section 4 and 5. Future work is arranged in section 6.

2 Monotonic Indices Space (MIS) Method

The main idea of the MIS method is divide-and-conquer strategy: the whole parameter space is partitioned into a lot of hierarchical sub-spaces which are isomorphic with their parental space. Any sub-space is called a hyperbox which is an important concept of MIS. A hyperbox can be imaged as a hypercube, and it is a description of parameter space.

2.1 MIS Method

The MIS method is mainly applied in monotonic vector space. In monotonic vector space P, f should be n-Dimensional monotonic function, that means $\forall p(x_1, x_2, \cdots, u, \cdots, x_n), q(x_1, x_2, \cdots, v, \cdots, x_n) \in P$, if $u \geq v$, then $f(p) \geq f(q)$ or $f(p) \leq f(q)$. For example: $P = [0, 2] \times [0, 2] \subset R^2$, $f(x, y) = x^2 + y^2$, the requirement space $S = \{p : p \in P, f(p) \leq 4\}$. In this case, the initial hyperbox is P, and the partition point is $p^*(\sqrt{2}, \sqrt{2})$ which is on the diagonal of the current hyperbox, and then it produces 4 sub-hyperboxes, as illustrated by Fig. 1.

The main operation of the MIS method is resolving hyperbox, and the resolving includes two procedures: search and partition:

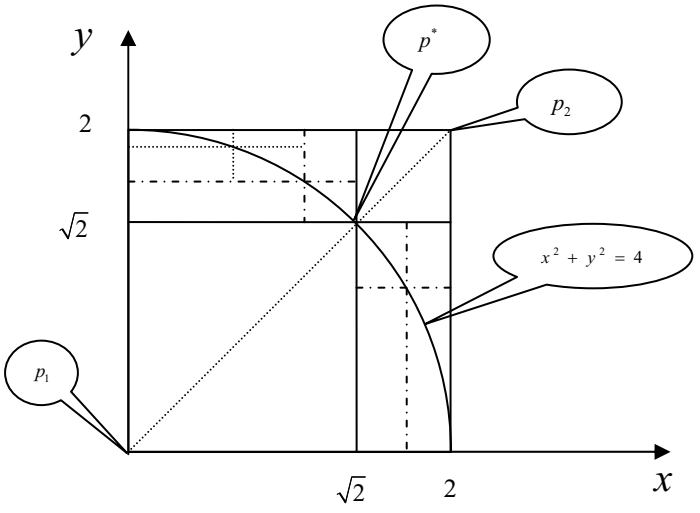

Fig. 1. An example of the MIS method. The root hyperbox is determined by point p_1 and p_2, and the partition point p^* ($f(p^*) = \alpha$) is on the diagonal. The partition point cuts the current hyperbox into 4 sub-hyperboxes, one of them (determined by p_1 and p^*) belongs to requirement space, one (determined by p^* and p_2) of them doesn't, and the other two should be inspected again.

1. Search: search the partition point along the diagonal of the hyperbox. As shown in Fig. 1, the current hyperbox is determined by point p_1 and p_2, let $a_0 = p_1$, $b_0 = p_2$, $c_0 = (p_1 + p_2)/2$, if $f(c_i) \leq \alpha$, then $a_{i+1} = c_i, b_{i+1} = b_i, c_{i+1} = (a_{i+1} + b_{i+1})/2$, if $f(c_i \geq \alpha)$, then $a_{i+1} = a_i, b_{i+1} = c_i, c_{i+1} = (a_{i+1} + b_{i+1})/2$, repeat this operation. Since there must be at least one partition point on the diagonal[4], thus the point c_i can be considered as the partition point when $\|a_i - b_i\| \leq \theta l_0$, where θ is a parameter called cut rate, and l_0 stands for the length of the diagonal of the root hyperbox.
2. Partition: each component of the partition point will divide the same dimension of the current hyperbox into two segments, and the combination of all the segments will produce 2^n sub-hyperboxes, see Fig. 1, and the undetermined $2^n - 2$ sub-hyperboxes should be inspected again. In the partition procedure, sub-hyperboxes that are too little to affect the requirement space (or affect a little bit) will be abandoned directly. Suppose V is the volume of some sub-hyperbox, we can abandon this subhyperbox when $V \leq \gamma V_0$, where γ is a parameter called stop rate, and V_0 stands for the volume of the root hyperbox.

During the above two procedures, we can get a hyperbox tree, see Fig. 2:

After the above two procedures , we can get a requirement-Satisfied hyperbox s_i (i stands for the identifier of the hyperbox on the hyperbox tree) determined by p_1 and p^*, see Fig. 1, obviously $s_i \subset S$, then the requirement space can be determined as follows:

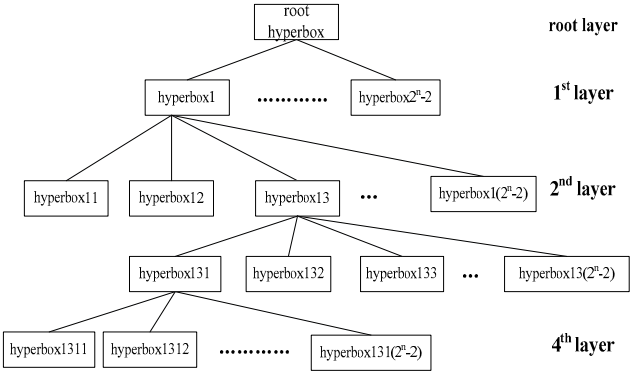

Fig. 2. Hyperbox tree. Resolving the current hyperbox will produce $2^n - 2$ undetermined sub-hyperboxes, and the current hyperbox is the parent of these sub-hyperboxes, then all the hyperboxes form a tree structure.

$$S = \bigcup_{i=0}^{\infty} s_i \tag{1}$$

2.2 Time and Memory Costs of the MIS Method

The MIS method is highly recursive, thus it is very suitable to implement it by recursive program. However the depth of the recursion is the key for precision. We can't get the precise requirement space with low depth. While deep depth will bring in a very large amount of calculation and memory need (as shown in Fig. 2, the hyperbox scale is $O\left((2^n-2)^k\right)$, and the memory need is $O\left(n(2^n-2)^k\right)$, where n is the dimension of the requirement space, and k stands for the maximal depth of recursion.) which exceed a single computer's capability in a certain period of time. Additionally, extra cost brought by recursion to operating system makes the execution ineffective.

Most practical problems are related with high-Dimensional model, they need much more calculation and memory, compared to low-Dimensional model. In some cases, the calculation and memory need are so large that it makes a single computer exhausted or even halted. Thus we come to the edge that we must process the calculation and satisfy the memory need in other way.

Above all, our task becomes accelerating the execution of the MIS method, promising a certain depth of recursion.

3 Coarse-Grained Approach

Reviewing the MIS method, we can find that the algorithm has immanent parallelism: the resolving of hyperboxes is all independent. Any hyperbox determined by the initial parameters can be resolved without communicating with other hyperboxes, thus the hyperboxes can be resolved concurrently.

A Coarse-Grained Multicomputer model: p processors solving a problem on n data items, each processor has $O(n/p) \gg O(1)$ local memory, and all the processors are connected via some arbitrary interconnection network [8], can be represented as $CGM(n, p)$. In the MIS method, each hyperbox is one data item, thus the key is to assign the hyperboxes to the processors, and keep load-Balanced for each processor.

In most cases, the requirement space is not evenly distributed in the parameter space, thus it is not suitable to partition the items (hyperboxes) according to their location in the parameter space. While it is very hard or impossible to predict the requirement space, thus we almost can't partition the items and assign them to the processors before the execution, and we have to do this work during resolving dynamically. New hyperboxes will come out during resolving their paternal hyperbox, and the hyperboxes can be stored into a container (usually called pool), and then a hyperbox will be taken out and assigned to each idle processer if there exists idle processers.

3.1 Parallelized Resolving in C/S Mode

There are two core operation of the MIS method: resolving hyperbox and maintaining the hyperbox tree. Generally speaking, resolving hyperbox is a kind of compute-intensive procedure which needs a large amount of calculation, while maintaining the hyperbox tree is a kind of communicate-intensive procedure which needs to communicate with others frequently. Therefore, the two operation should be implemented separately and execute on separate computers.

Basing on the analysis before, the computer which executes resolving hyperbox is called calculative node, and it is in charge of resolving hyperbox. The computer which executes maintaining hyperbox tree is called control node, and it is in charge of receiving and deploying hyperbox. Therefore the calculative nodes and the control node form a client-Server structure.

Resolving Hyperbox on Client. The MIS method will generate a lot of hyperboxes, and the faster the generated hyperboxes resolved the faster the experiment completes, thus we always assign a few calculative nodes in the system. A procedure called clientProcedure is running on each calculative node, and it will process the fundamental tasks of calculative node: resolving the hyperbox and interacting with the control node. The action of calculative node is driven by the commands from the control node, and the command can be packed into a message, such as a socket message, thus the messages can be classified and identified according to the command type. The main body of clientProcedure is to operate according to the message received from the control node.

```
clientProcedure:
    Initialize;
    Connect to the Server;
    Listen message from Server;
    switch(message)
```

```
        case information of experiment:
            save the imformation;
        case hyperBox:
            call resolveProcedure;
        case collectResults:
            collect the local results and send them to
                server;
```

A procedure called resolveProcedure is in charge of searching the partition point and generating sub-hyperboxes and it will be called after the calculative node receiving a message telling a hyperbox from the control node. Two n-Dimensional points can determine a hyperbox uniquely, see Fig. 1, and they also determine the diagonal on which the partition point locates. Thus we can search the partition point along the diagonal of the hyperbox. The model is used to calculate the requirement measure, and it is always implemented as a program whose input is a single point. So the search can be divided into two iterative sub-courses: calculating the input point, and calling the model program. Once the partition point has been determined, $2^n - 2$ sub-hyperboxes will be produced and filtered. Any hyperbox can be determined by two n-Dimensional points, and the two points can be packed into a message, and then the message will be sent to the control node, telling a hyperbox. Sub-hyperbox is filtered according to its volume, and the one that is too little to affect the requirement space will be abandon directly. After the resolving completed, a message will be sent to the control node to tell that this calculative node has become idle.

Maintain the Hyperbox Tree on Server. Maintaining the hyperbox tree is the main task of the control node, and it includes two fundamental operations: receiving the subhyperboxes generated by the calculative nodes and deploying hyperbox to them. A procedure called serverProcedure is running on the control node, and it will process the above two operations.

```
serverProcedure:
    Initialize;
    the user choose an experiment configuration file;
    deploy the necessary files to the connected clients;
    start the experiment;
    Listen message from the connected clients;
    switch(message)
        case finish://means client completes resolving
            put the client from which the message came
                into idle clients queue;
            if(the hyperbox queue is empty && all the
                clients are idle)
                call resultProcessProcedure;
            else
                call deployProcedure;
        case hyperBox:
```

```
        receive the hyperbox and put it into the
            hyperbox queue
    case resultFile:
        receive the result file
```

3.2 Task Pool Work Balancing

Task pool is one of implementations of work stealing [12], and it can be used for load balancing. In this paper, the hyperboxes generated by the calculative nodes will be put into a hyperbox queue which is assigned on the control node and plays the task pool. Once a message telling a hyperbox has been received by the control node, the serverProcedure will unpack the message to get the two n-Dimensional points, and then construct a hyperbox according to the two points. The constructed hyperbox will be put into the hyperbox queue.

The control node maintains a list to record the information of the calculative nodes. In the beginning, the whole parameter space will be constructed as the first hyperbox, and then the hyperbox will be put into the hyperbox queue. After starting the execution , the front hyperbox of the hyperbox queue will be taken out and assigned to the front idle calculative node of the calculative nodes list. To use the calculative nodes fully, we hope that the nodes are keeping resolving hyperboxes if there are unresolved hyperboxes in the hyperbox queue. Once the control node finds an idle calculative node, a hyperbox will be taken out and assigned to the idle node. To decrease the overhead of preparing the message of assigning a task to the idle calculative node, the server always gets the front hyperbox of the hyperbox queue out(time consuming $O(1)$), and then packs it into the assigning message. Therefore, the calculative nodes are keeping resolving hyperbox controlled by the calculative node. The terminal condition of the whole execution is that the hyperbox queue on the control node is empty and all the calculative nodes are idle.

4 Speedup Analysis

For the sequential MIS method, the total execution time is the summary of the time of resolving each sub-hyperbox, and suppose T_s is the period of time which the sequential MIS method costs, then

$$T_s = \sum_{i=1}^{N} t_i \geq \sum_{i=1}^{N} t_{min} \tag{2}$$

where N is the total number of hyperboxes in the experiment, $t_i$is the period of time which resolving ith hyperbox costs, and $t_{min} = \min\{t_i\}$. T_p is the period of time which the parallel algorithm costs, then

$$T_p = \sum_{j=1}^{\lceil N/m \rceil} t_j \leq \sum_{j=1}^{\lceil N/m \rceil} t_{max} \tag{3}$$

where m is the number of calculative nodes, and $t_{max} = \max\{t_j\}$, then the speedup is the ratio

$$speedup = \frac{T_s}{T_p} \geq \frac{t_{min}}{t_{max}} \cdot \frac{N}{\lceil N/m \rceil} \approx \frac{t_{min}}{t_{max}} \cdot m \qquad (N \gg m) \tag{4}$$

In the sequential MIS method, the period of resolving a hyperbox is mainly composed of two parts: time of model calls and the extra cost of the system (including the OS and the program itself). While in the parallel algorithm, the period is mainly composed of five parts: overhead of the server preparing message, the delay of transferring message from the server to the clients (determined by the bandwidth of the network), overhead of the client receiving message, the overhead of the client preparing message (above four are the communication cost), and time of model calls. The communication cost can be represented as following

$$t_{com} = o_s + L + o_c + (2^n - 2)o_c \tag{5}$$

then the period can be represented as

$$t_j = t_{com} + t_{call} \tag{6}$$

Suppose the period of single execution of the model f is const T, then the period of model calls t_{call} is proportional to the number of times of the model calls. Reviewing the search procedure of the MIS method , the repeat will have the length of the distance between a_i and b_i to halve again and again , then the terminal condition of binary search can be described as following

$$\frac{l_k/2^r}{l_0} = \theta \tag{7}$$

where k is the depth of the current hyperbox on the hyperbox tree, l_k is the length of the diagonal of the current hyperbox, r is the number of times of the model calls, l_0 is the length of the diagonal of the root hyperbox, and θ is the cut rate. Statistically, $l_k = 2^{-k} l_0$ is correct, then

$$r = -\log_2 \theta - k \tag{8}$$

In the partition procedure, a sub-hyperbox will be abandoned directly, if its volume is too little to affect the requirement space of the problem. Statistically, $V_k = 2^{-nk} V_0$ (n is the dimension of the parameter space)is correct, then

$$\frac{2^{-nk} V_0}{V_0} = \gamma \tag{9}$$

$$k = \frac{\log_2 \gamma}{n} \tag{10}$$

formula (10) is also the maximal depth of recursion. Let

$$h_1(\theta,\gamma,n) = \frac{t_{min}}{t_{max}} = \frac{(-\log_2\theta - k_0)T}{(-\log_2\theta - k_{max})T + \max\{t_{com}\}} \overset{k_0=0}{=} \frac{T\log_2\theta}{(\log_2\theta + \frac{\log_2\gamma}{n})T - \max\{t_{com}\}} \tag{11}$$

For a certain experiment, the dimension of the requirement space is stationary, thus the variable n in formula (11) can be considered as const, and then $h_1(\theta,\gamma,n)$ becomes

$$h_2(\theta,\gamma) = \frac{\log_2\theta}{(\log_2\theta + \frac{\log_2\gamma}{n}) - \frac{\max\{t_{com}\}}{T}} \tag{12}$$

$$speedup \geq h_2(\theta,\gamma)\cdot m \tag{13}$$

5 Experimental Results

To test our approach, we take the system mentioned by Hu Jianwen in [4] as an example. It is an air defense information system consisting of three entities: long-Distance warning radar, command and control center, and enemy's aircraft. The scenario is following: one of enemy's aircraft is flying direct to target T which is protected by us, and once the long-Distance warning system discovers the aircraft, it sends the information of the aircraft to the command and control center, and then the center sends firing order to the air defense missiles after processing the information. In this case, the requirement measure is destroying the enemy's aircraft more than 0.7 success probability, and the indices are abstracted into long-Distance warning radar detecting index, system processing delay index, and tracking index of the tracking radar.

The model f is implemented by a program, and the sequential algorithm is implemented as a recursive program. The parallel algorithm is implemented and running in a group of PCs (PC of same type) connected by 100M Ethernet with each other. In the group, one PC is assigned as control node to execute the serverProcedure, and several other PCs are assigned as calculative node to execute the clientProcedure. Test the sequential algorithm and the parallel algorithm separately, and then add up the period of their execution. We played two schemes: increase calculative nodes, keeping the parameters (θ and γ) unchanged, and change one of the parameters, keeping the other parameter and calculative nodes unchanged, the results are shown in Table 1.

In this case, the period of single execution of the model call is tens of seconds. For the PCs are connected by 100M Ethernet, then the latency of transferring message is very tiny, and the overhead of the server preparing message is also very tiny because of the dequeue operation. Compared to t_{call}, t_{com} is too tiny to affect $h_2(\theta,\gamma)$ much, then we can ignore it to estimate the speedup.

We can attain pretty speedup through our parallel algorithm, and the speedup increases almost linearly with the number of calculative node, see Fig. 3. But

Table 1. results table. T_s: time of sequential MIS, T_p: time of parallel execution, m: number of calculative nodes, θ: cut rate, γ: stop rate, *speedup*: T_s/T_p, PE: Practical Efficiency (*speedup*/m), TE: Theoretical Efficiency (formula (12)).

$T_s(s)$	$T_p(s)$	m	θ	γ	*speedup*	PE	TE
5888	3552	2	0.05	0.05	1.658	0.829	0.750
5888	2518	3	0.05	0.05	2.338	0.779	0.750
5888	1913	4	0.05	0.05	3.078	0.769	0.750
5888	1546	5	0.05	0.05	3.809	0.762	0.750
5888	1456	6	0.05	0.05	4.044	0.674	0.750
5888	1258	7	0.05	0.05	4.680	0.669	0.750
5888	1133	8	0.05	0.05	5.197	0.650	0.750
1789	1121	2	0.05	0.10	1.600	0.798	0.796
1789	601	4	0.05	0.10	2.977	0.744	0.796
7158	4567	2	0.0375	0.05	1.567	0.784	0.767
7158	2375	4	0.0375	0.05	3.014	0.753	0.767

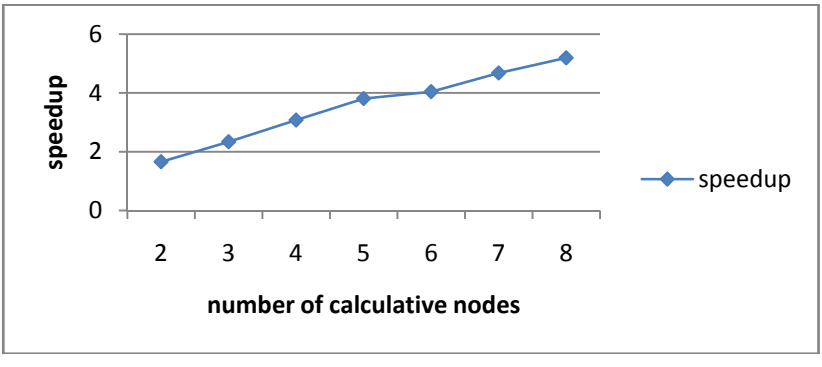

Fig. 3. Speedup. Keep the parameters(θ and γ) unchanged, and increase calculative node step by step, add up the period of sequential execution and the period of parallel execution, and then calculate speedup.

there is a trend that the speedup with more nodes (6, 7, 8 nodes) increases more slowly than that with fewer nodes (2, 3, 4, 5 nodes). The main reason for this trend: during the execution, we found that the calculative nodes were not well-Balanced, in spite of using task pool model. We found that: at some time, several nodes are busy, while the others are idle (the hyperbox queue is empty at that time), but after the busy nodes completing resolving, they will make the idle nodes busy (because the resolving will produce sub-hyperboxes, and the hyperbox queue will not still empty after resolving completed), that means some calculative node were wasted. Therefore, the more calculative nodes we have the more nodes will be idle, and the more speedup will decrease. The cause of the waste is that the deploying of hyperboxes is blindly, thus the realistic work of

the calculative nodes are not exactly symmetrical, and the faster ones have to wait for the slower ones.

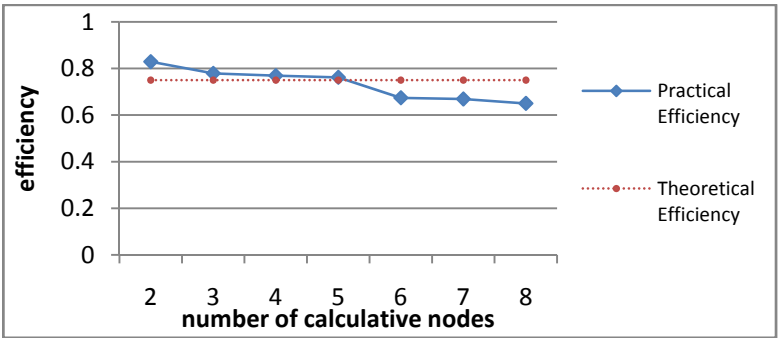

Fig. 4. efficiencies. Keep the parameters(θ and γ) unchanged, and increase calculative nodes step by step, add up the period of sequential execution and the period of parallel execution, and then calculate parallel efficiency.

In our test, the practical efficiency is very close to the theoretical efficiency which is determined by formula (12), and it indicates that formula (12) is useful to estimate the parallel efficiency in this algorithm, see Fig. 4. As same as the speedup, the practical efficiency will decrease with the increasing of number of calculative node. The main reason for the decrease includes two points: 1) the calculative nodes are not well-balanced; 2) extra work which is brought in by parallelization goes up when the number of calculative node goes up.

6 Conclusions and Future Work

Basing on the analysis of the MIS method, this paper digs the immanent parallelism of the algorithm, and then introduces a coarse-grained parallel execution mechanism to the MIS method for accelerating the process of requirement space exploration. We implement the algorithm and then test it on PC clusters. The results indicate that the parallel algorithm is much more effective than the sequential algorithm.

In spite of the high parallel effectiveness of our parallel algorithm, we find the communication delay can decrease more, and the memory pressure for the control node can decrease more. In our parallel algorithm, the hyperboxes generated by the calculative nodes should be sent to the control node, and then sent to some calculative node back. In this course, one hyperbox is double-sent, and the delay between the time of generating and resolving will increase logically. In fact, the source and destination of hyperboxes are both calculative nodes, thus a hyperbox

can be sent from one calculative node to some other calculative nodes directly or stay on itself controlled by the control node.

In our test, we find that the calculative nodes are not well load-Balanced. In fact, some feature (such as volume, bound, and so on) of hyperbox can be used to estimate the calculation which resolving the hyperbox needs, and the estimated information can be used to improve the deploy policy to adjust the load of each calculative node for load balancing.

References

1. Bouthonnier, V., Levis, A.H.: System Effectiveness Analysis of C3 Systems. IEEE Trans On Systems, Man, and Cybernetics 14(1), 48–54 (1984)
2. Levis, A.H., et al.: Effectiveness Analysis of Automotive System. Decision Syst., MIT, Cambridge (1984); LIDS-P-1383
3. Christine, B.M.: Computer Graphics for System Effectiveness Analysis.M.S.thesis, Dept.Elect.Eng.Comput.Sci, MIT, Cambridge, MA (1986)
4. Hu, J., Hu, X., Zhang, W., et al.: Monotonic Indices Space Method and Its Application in the Capability Indices effectiveness analysis of a Notional Anti-Stealth Information System. IEEE Transaction on System, Man, Cybernetics Part A 39(2), 404–413 (2009)
5. Hu, J.: Analysis and design of search for weapon system indices. National Defense Industry Press, Beijing (2009) (胡剑文: 武器装备体系能力指标的探索性分析与设计.国防工业出版社,北京(2009))
6. Prediction Measurement. New York: WSEIAC, Final Report of Task Group 2, 1,2,3 (January 1965)
7. Hwang, C.L., et al.: Multiple Attribute Decision Making. Springer, Berlin (1981)
8. Dehne, F., Fabri, A., Rau-Chaplin, A.: Scalable Parallel Computational Geometry For Coarse Grained Multicomputers. International Journal of Computational Geometry & Applications 6(3), 379–400 (1996)
9. Dehne, F., Maheshwari, A., Taylor, R.: A Coarse Grained Parallel Algorithm for Hausdorff Voronoi Diagrams. In: Proceedings of the 2006 International Conference on Parallel Processing, ICPP 2006 (2006)
10. Garcia, T., Sem', D.: A Coarse-Grained Multicomputer algorithm for the Longest Repeated Suffix Ending at Each Point in a Word. In: 11-th Euromicro Conference on Parallel Distributed and Network Based Processing (PDP 2003),
11. Chan, A., Dehne, F., Rau-Chaplin, A.: Coarse Grained Parallel Geometric Search. Journal of Parallel and Distributed Computing 57, 224–235 (1999)
12. Dinan, J., Larkins, D.B., Krishnamoorthy, S., Nieplocha, J.: Scalable Work Stealing. In: Proceeding SC 2009 Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis. ACM, New York (2009)