Proceedings of
The Fourth National Congress on Fluids Engineering
August 23-25, 2006, Kyungju, Korea

근거리 힘 계산의 새로운 고속화 방법

이상환* · 안철오**

A New Fast Algorithm for Short Range Force Calculation

SangHwan Lee*, Cheol-O Ahn**

Keywords : *Molecular Dynamics(분자동역학),Tree Structure(트리구조),Fast Algorithm(고속화 알고리즘), Short Range Force(근거리 힘), Domain Decomposition(영역분할)*

Abstract

In this study, we propose a new fast algorithm for calculating short range forces in molecular dynamics. This algorithm uses a new hierarchical tree data structure which has a high adaptiveness to the particle distribution. It can divide a parent cell into k daughter cells and the tree structure is independent of the coordinate system and particle distribution. We investigated the characteristics and the performance of the tree structure according to k. For parallel computation, we used orthogonal recursive bisection method for domain decomposition to distribute particles to each processor, and the numerical experiments were performed on a 32-node Linux cluster. We compared the performance of the oct-tree and developed new algorithm according to the particle distributions, problem sizes and the number of processors. The comparison was performed using tree-independent method and the results are independent of computing platform, parallelization, or programming language. It was found that the new algorithm can reduce computing cost for a large problem which has a short search range compared to the computational domain. But there are only small differences in wall-clock time because the proposed algorithm requires much time to construct tree structure than the oct-tree and the performance gain is small compared to the time for single time step calculation.

1. 서 론

해석의 대상이 되는 계를 다수의 입자(particle)로 모델링 하여 다루게 되는 다체계 문제는 천체물리, 유체역학(입자와법), 분자동역학, 입자물리, 플라즈마 해석 등에 사용되고 있다[1,2] 이러한 문제는 기본적으로 모든 개개의 입자가 그 자신을 제외한 다른 모든 입자와 가지는 상호 작용(포텐셜, 힘 혹은 유기속도)을 연산해야 하기 때문에, 입자의 수(N)가 증가함에 따라 연산량은 그 제곱에 비례(~$O(N^2)$)한다는데 그 어려움이 있다. 이러한 다체계 문제를 흔히 N-body문제라고 부르며 이러한 문제의 연산 고속화에 가장 많은 연구가 진행된 분야는 은하, 성운 및 성단의 전산모사를 다루는 천체물리학 분야이다.

이들이 제안하였던 방법 중 가장 널리 사용되는 것은 이러한 점을 극복하기 위한 방법 중 하나는 공간에 분포하고 있는 입자들을 그 위치정보에 근거하여 트리 구조로 분할하고, 이를 연결 리스트로 만들어, 임의 입자와 충분히 멀리 떨어져 그 상호작용이 미미한 입자들의 집합을 하나의 입자로 간주하여 연산을 간략화 하는 것으로, 입자계 연산에 있어서 연산량을 $O(N^2)$에서 O(NlogN)으로 줄이는 것과 같은 상당한 고속화를 이룰 수 있다.[1,2] 이와 같은 방법은 계층적 구조를 가지는 tree데이터 구조를 사용하므로 treecode라고 부르는데, 이는 원거리에서 입자들의 덩어리의 효과가 적절한 급수의 적은 수의 초기 항에 의해 근사화될 수 있다는 아이디어를 이용한다. 이는 공간상에 위치하는 질량체들은 Newton의 중력법칙에 지배를 받기 때문에 공간에 존재하는 모든 입자들 간의 상호작용을 연산해야 한다는 특징을 가진다. 이와 같은 종류의 문제를 원거리 힘 계산(long range force calculation)이라고 구분하는 반면, 분자동역학에 있어서 분자들간의 힘을 계산하기 위한 Lennard-Jones포텐셜의 경우 다음의 식(1)과 같기 때문에, 상호분자의 거리가 일정한 값 이상이 되면 상호작용이 사라지게 되어 이와 같은 일정한 탐색반경 안에 위치하는 분자들만을 탐색하여 힘 계산을 한다는 특징을 가지므로 이를 근거리 힘 계산(short range force calculation)이라고 부른다.[3]

$$\phi(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \tag{1}$$

원거리 힘 계산에 있어서의 목적은 제어 가능한 만큼의 오차 하에서 가능한 많이 연산을 줄이는 것이고 근거리 힘 계산에 있어서의 목적은 가능한 신속하게 한 입자로부터 정해진 거리 이내에 있는 모든 입자를 구분해 내는 것이다.[3] 하지만 이러한 모두는 트리 구조를 이용함으로써 고속화가 가능하며, 따라서 다체계 연산에 있어서 효율 좋은 트리 구조를 이용한 고속화 알고리즘의 개발과 이의 효율적인 병렬화 혹은 벡터화가 중요한 개발목표가 된다.

2. 고속화 방법

2.1 Oct-tree

* 한양대학교 기계공학과, shlee@hanyang.ac.kr
** 한양대학교 대학원, coahn@hanyang.ac.kr

Barnes에 의해 개발된 treecode는 주어진 입자계를 계층형태의 조직화된 데이터 구조로 표현하기 위해서 입자가 분포한 공간을 하나의 cell에 하나 이하의 입자가 포함될 때까지 재귀적인 방법으로 각 방향으로 2분할 하여 원래의 정육면체 공간을 8개로 분할하는 방법을 사용한다. 이 방법은 트리의 분기비가 8이므로 oct-tree라고 불린다. Oct-tree는 좌표계 의존적인데 왜냐하면 정육면체 cell들은 그들의 면들이 좌표계 축에 평행하기 때문이다. Oct-tree의 경우 매 분할마다 8개의 동일한 cell이 생성되므로 정육면체의 공간에 일정한 밀도로 입자가 분포한 경우 알고리즘이 최고의 효율을 가진다고 할 수 있다. 분할이 이루어지게 되면서 입자는 일정한 규칙에 의하여 주소가 부여되고 분할이 완료되면서 입자들의 모든 정보는 트리 구조화 된다. Oct-tree는 단순하면서도 빠르고, 매우 효율이 좋은 알고리즘이다. Barnes에 의해 중력장 입자계를 고속으로 연산하기 위한 계층화된 트리구조 데이터를 사용하는 treecode가 개발된 이후, 이와 유사한 다양한 기법들이 개발되어 왔는데, 대부분의 treecode는 oct-tree에 기초를 두고 있다. Clarke에 의하여 제안된 binary tree는 입자 분포에 대한 적응성이 개선되었으나 트리의 노드수가 많아지고 깊이가 깊은 트리를 만들게 되어 결과적으로 연산시간을 더 필요로 하였다. Benz는 최 근접 이웃 알고리즘을 이용하여, 하나의 입자와 가장 근접한 입자를 node로 만드는, bottom-up방식의 트리를 사용하였는데 이는 기존과는 달리 좌표계에 완전히 독립적이며 입자분포에 높은 적응성을 가진다. 그러나 트리의 생성에 너무나 많은 시간을 필요로 하였다[2]. Waltz등은 binary tree의 고도의 적응성이라는 장점을 살리면서 동시에 Barnes의 treecode와 유사한 분기비(branching ratio)를 가지도록 하였는데, 높은 적응성에도 불구하고 원거리 힘 계산에서의 성능은 Barnes와 비슷하였고 근거리 힘 계산에서의 성능은 약간 우세함을 보였다.[3]

2.2 k-tree

본 연구에서는 기존의 oct-tree에 비하여 입자의 분포에 맞추어 적응 분할하도록 하는 새로운 고속화 알고리즘을 제안한다. 개발된 트리는 top-down 방식에 따라서 입자가 포함되는 공간을 매 분할마다 일정한 임의의 k개의 자식셀로 나누는 알고리즘을 이용한다. 분할에 의하여 생성된 cell은 좌표계에 독립적이며 이때의 k는 하나의 부모노드가 분할될 때 생성되는 자식노드의 수, 즉 트리의 분기비를 의미한다. 그러므로 oct-tree의 경우 이 k가 8인 특별한 경우로, binary tree는 k=2인 경우로 생각할 수 있다. 우리는 새로운 트리를 k-tree라고 부른다. 이러한 k값에 따른 특징들은 어떻게 분기비가 treecode의 성능에 영향을 미치는지 검토할 수 있도록 한다.

입자가 분포하고 있는 공간을 k개로 분할하기 위해서, k개의 가상의 cell 중심점을 입자가 분포해 있는 공간에 적절히 위치시킨다. 그 다음, 부모셀에 속한 전체 입자와 각각의 가상의 cell 중심점들과의 거리를 구한 후, 각각의 입자에는 가장 가까운 가상의 cell 중심점을 지시하는 식별자를 저장한다. 같은 식별자를 가지는 입자들의 중심점을 구하고, 이 값을 가상의 cell 중심점으로 갱신한다. 이상의 과정을 k개의 가상의 cell 중심점들이 수렴할 때까지 반복하고, 동일한 식별자를 가지는 입자들을 새로운 cell로 만들어 트리를 구성한다. 완전한 트리 구조를 만들기 위해서는 이와 같은 분할이 재귀적으로 각각의 cell에 하나의 입자만이 존재할 때까지 반복한다. 이렇게 얻어진 트리 구조는 입자분포에 적응적으로 각각의 cell중심이 이동하여 생성 되기 때문에 oct-tree에 비하여 공간적 효율성이 보다 우수한 반면, 초기에 적절한 cell의 중심을 결정하는 것이 수렴속도에 크게 영향을 미치고, oct-tree에 비하여 트리 구조를 구성하는데 많은 시간이 소요된다는 단점이 있다.

2.3 Tree Walking

동일한 입자데이터로부터 생성되는 트리는 전체 트리의 구조와 크기는 다르지만 oct-tree와 k-tree모두 기본적으로는 동일하게 동작될 수 있는 트리 구조를 만들게 되므로 트리를 순환하며 입자간 연산을 수행하는 방법은 동일하다. 입자간의 상호작용을 구하기 위하여 작성된 트리의 최상위 구조에 위치한 cell로부터 트리를 순환한다. 만일 탐색반경과 cell반경의 합이 cell 중심과 입자와의 거리와 비교하여 작다면 입자와 cell이 충분히 분리되어 있으므로 그 cell의 다음 리스트를 탐색한다. 만일 탐색반경 내에 cell의 일부가 위치한다고 하면 단일 입자를 포함하는 자식cell을 탐색하여 상호작용을 계산한다.

트리 구조의 성능을 결정하는 지표는 입자와 cell과의 거리를 구하고 cell의 크기와 비교하는 연산 횟수(N_E, Number Node of Evaluation)와 실제로 입자와 cell간의 상호작용을 계산하는 횟수(N_I, Number of Interaction)로 생각할 수 있다. 근거리 힘 계산의 경우 어떠한 방법을 사용하든지 입자간 상호작용 연산 횟수는 동일하므로 어떠한 오차도 발생하지 않으며, 트리알고리즘을 이용하는 것은 전체 입자로부터 이웃한 이웃 입자를 가능한 빠르게 검색하도록 한다. 따라서 근거리 연산의 성능은 각각의 알고리즘에 대해 단일 타임스텝의 연산을 실시하고 이때 N_E를 측정하여 이 값이 낮을수록 우수한 것으로 생각할 수 있다.[3]

원거리 힘 계산의 경우 입자간 상호작용의 연산에 많은 시간이 소모되어 트리를 구성하는데 소요되는 시간은 전체 연산시간에 비해서 충분히 작기 때문에 무시할 수 있지만 분자동역학과 같은 근거리 힘 계산에 있어서는 탐색 반경에 따라 다르다고 하더라도 상호작용을 계산하는데 소요되는 시간이 비교적 짧고, 보통 1fs(femtosecond, 10^{-15})정도의 매우 작은 타임스텝을 사용하여 매우 많은 반복계산이 이루어진다는 입장에서 보면, 원거리 힘 계산과는 달리 트리를 구성하는데 소요되는 시간은 적지 않은 의미를 가진다고 할 수 있다.

3. 연산결과

3.1 비교방법

다체계 연산의 고속화 성능 비교에 있어서 문제점 중 하나는 대상이 되는 개체의 분포상태에 의존적이라는 점이다. Waltz등에 의한 선행연구와 마찬가지로, 공정하게 알고리즘간의 성능을 분석하기 위하여 입자간 연산은 동일한 트리 순환 프로그램에서 비교하였다. 그러므로 트리순환에 있어서 얻어지는 N_E의 비교는 트리에 독립적이며 컴퓨팅 플랫폼이나 병렬화 기법, 컴퓨팅 언어에 독립적이다. 이러한 접근방법은 측정되는 성능에 영향을 미칠 수 있는 프로그래밍 차이의 가능성을 제거한다.[3]

분자동역학과 같은 근거리 힘 계산에 있어서 단일 타임스텝에서의 연산시간 ΔT 은 트리 알고리즘에 상관 없이 동일한 N_I를 가지므로 1회의 N_I에 소요되는 비용 α 와 1회의 N_E에 소요되는 비용 β, 트리를 구성하는데 소요되는 비용 γ 및 기타 오버헤드 ε 에 의해 다음과 같이 표현된다.

$$\Delta T \sim N\left[\alpha \times N_I + \beta(k) \times N_E\right] + \gamma(N,k) + \varepsilon(N) \qquad (2)$$

이때, α, ε 은 트리 알고리즘에 독립적이며, β, γ 는 트리 알고리즘에 의존적이다.

비교의 대상으로 정육면체 내부에 균일하게 입자가 분포하는 경우(case A), 한쪽 방향의 길이가 다른 두 방향의 길이의 4배인 직육면체에 균일하게 입자가 분포하는 경우(case B), 구 내부에 균일하게 입자가 분포하는 경우(case C)의 세 가지 유형을 택하였다. 이들 각각은 최대 길이척도의 0.05%, 0.1%, 0.5%, 1%를 탐색반경으로 하도록 하였으며 입자의 수는 16만개, 64만개, 128만개로 택하였다.

3.2 결과비교

본 연구는 대규모 분자동역학 문제의 연산의 고속화에 관한 것으로서 MPI기법에 의한 병렬화 된 프로그램을 구동하기 위하여 32대의 노드(AMD Athlon XP 3000+)와 1GB switching hub로 구성된 Linux Cluster에서 계산을 수행하였다. 병렬연산에 있어서 프로세서 별로 적절히 입자를 분배하기 위해 몇 가지 영역 분할기법(Domain Decomposition)이 제안되고 있는데 직교 재귀 이분할법(Orthogonal Recursive Bisection method)[6], Morton공간 채움 곡선을 이용하는 방법[6], Peano-Hilbert의 공간 채움 곡선[4,5]을 이용하는 방법 등 세가지 영역분할기법의 차이를 비교하였으나 뚜렷한 차이점은 발견할 수 없었다. 따라서 연산은 영역 분할기법으로 직교 재귀 이분할법을 선택하였다.

Table 1은 16만개의 입자로 이루어진 Case A에 대해서 병렬화만을 적용한 직접 연산의 경우(direct)와 병렬화된 트리알고리즘(oct-tree)을 적용한 경우를 프로세서 수와 탐색반경에 대해 비교한 것이다. 병렬화만을 적용한 경우, 모든 입자 상호간에 거리 연산이 필요하므로 이 경우 N_E는 곧 입자의 수에 해당하여 별도로 표시하지 않았다. 트리알고리즘을 사용하는 경우 실제 단일 타임스텝에서의 연산시간은 프로세서의 수에 크게 영향을 받지 않고 있는데, 이는 트리알고리즘을 적용함으로써 연산량이 획기적으로 줄게 되어 한 타임스텝에서 N_E와 N_I를 연산하는데 소모되는 시간의 비중이 낮아지고 상대적으로 프로세서간 통신과 기타 오버헤드의 비중이 증가했기 때문이다. 직접연산의 경우와 비교할 때, 동일한 N_I를 수행하고, 동일한 횟수와 양의 통신을 필요로 하므로 실제 연산시간의 차이는 대부분 N_E의 차이에 기인한다고 볼 수 있다.

Table 1. Comparison between treecode and direct calculation

No. Processors	Search Radius	oct-tree		direct
		NE	Time(s)	
4	0.05%	100.95	0.71	109.79
	0.10%	101.97	0.72	110.07
	0.50%	111.48	0.74	109.68
	1.00%	127.16	0.78	109.76
8	0.05%	118.25	0.64	55.25
	0.10%	119.17	0.65	55.43
	0.50%	127.94	0.65	55.16
	1.00%	142.72	0.66	55.37
16	0.05%	182.97	0.63	16.13
	0.10%	183.93	0.64	16.37
	0.50%	192.97	0.64	17.33
	1.00%	208.00	0.64	17.25
32	0.05%	311.45	0.65	8.02
	0.10%	312.47	0.65	8.06
	0.50%	322.01	0.65	8.21
	1.00%	337.83	0.64	8.03

Table 2는 세가지 크기의 Case A에 대해서 32개의 프로세서를 사용한 경우, k값에 따른 트리의 노드 수와 트리 구성에 소요된 시간을 비교한 결과이다. 입자의 분포에 무관하게 트리를 구성하는 노드의 수는 k값이 증가함에 따라 감소하는 경향을 보이는데, k=2인 경우 binary tree에서 지적되었던 바와 같이 트리의 깊이가 지나치게 깊어져 트리의 구조가 비대해지고, k=4이상이 되면 oct-tree보다 작은 값을 가진다. 트리를 만드는데 소요되는 시간은 k=2인 경우 약 1.5배 정도 이며, k값이 증가할수록 감소하여 k=8인 경우 oct-tree의 트리분할에 소요되는 시간의 2배 미만으로 감소한다. 병렬화 프로그램의 경우, 입자데이터를 영역 분할기법에 의해 적절히 프로세서 별로 할당한 후, 각각의 프로세서가 독자적으로 트리를 만들게 되므로, 트리를 구성하는데 걸리는 시간은 병렬계산에 동원된 프로세서의 수가 많을수록 감소한다. 따라서 k-tree가 트리구성에 드는 비용이 크다는 단점은 프로세서의 수가 증가함에 따라 감소한다. 병렬계산에 동원된 프로세서의 수가 2배 증가할수록 트리구성에 소요되는 시간은 1/2이 되므로 oct-tree와 k-tree의 구성에 소요되는 시간에서 나타나는 차이는 프로세서의 수가 증가할수록 점차로 감소하게 되어 전체 연산시간에서 이 차이가 미치는 영향 역시 감소하게 된다.

연산 결과는 입자의 분포에 별로 영향을 받지 않고 유사한 경향을 보였으며 k값에 따른 변화도 거의 일정한 경향을 가진다. Fig.1은 64만개 입자에서 탐색반경을 0.1%로 한 경우, 프로세서 수에 따른 N_E값을 비교한 것이다. 프로세서의 수가 증가함에 따라 oct-tree의 N_E가 더욱 가파르게 상승하였는데 이는 병렬화 효율이 좋지 않음을 의미한다. 반면 k=2인 경우 매우 더디게 증가하여, 대규모 병렬계산에서는 k-tree가 보다 유리함을 알 수 있다. Fig.2는 32개 프로세서에서 탐색반경을 0.1%로 한 경우 문제 크기에 따른 입자당 N_E를 비교한 것이다. 여전히 k=2인 경우 N_E가 가장 우수한 성능을 보이고 있으며, 문제의 크기가 증가하여도 N_E의 증가는 심각하지 않았다. Fig.3은 32개의 프로세서를 사용한 경우 64만개의 입자를 탐색반경에 따라 비교한 것이다. 탐색반경이 작을수록 k-tree에 의해 N_E가 감소하므로 최대 길이척도에 대비하여 탐색반경이 작인 경우, 고속화에 유리할 것으로 생각된다.

그러나 실제로 oct-tree에 비해서 N_E가 개선되었다고 하더라도 한 타임스텝에서의 이득은 크지 않은데, 우선 oct-tree에 의해서 N_E의 연산이 대폭 감소하였기 때문에 더욱 낮은 N_E를 가지는 k-tree의 경우라고 하더라도 이들간 NE의 차이가 미미하여 실제 시간상 별 이득이 없다는 점, 그리고 트리를 구성

Table 2. Tree size and the time for tree construction

	No. nodes of tree			Time for tree construction (s)		
	160k	640k	1280k	160k	640k	1280k
oct-tree	241k	960k	1,901k	0.004	0.021	0.045
k=2	319k	1,279k	2,559k	0.010	0.054	0.116
k=3	257k	1,028k	2,057k	0.008	0.045	0.094
k=4	235k	940k	1,881k	0.008	0.040	0.085
k=5	223k	894k	1,789k	0.007	0.036	0.078
k=6	216k	866k	1,731k	0.007	0.037	0.076
k=7	211k	846k	1,693k	0.006	0.038	0.070
k=8	208k	831k	1,664k	0.006	0.034	0.071

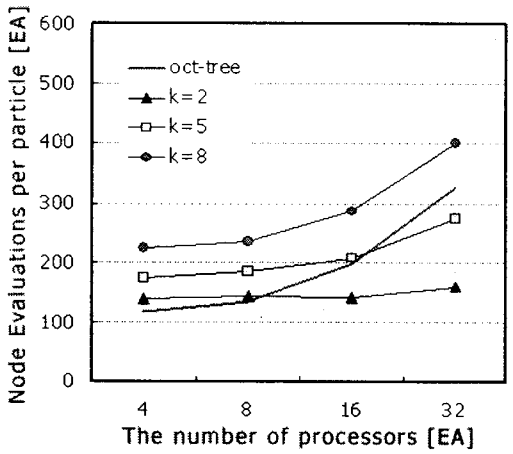

Fig.1 N_E as a function of the number of processors
(Case A, 640k particles, 0.1% search radius)

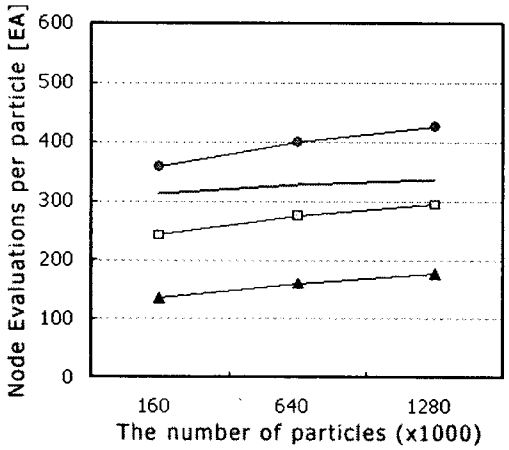

Fig.2 N_E as a function of the problem size
(Case A, 32 processors, 0.1% search radius)

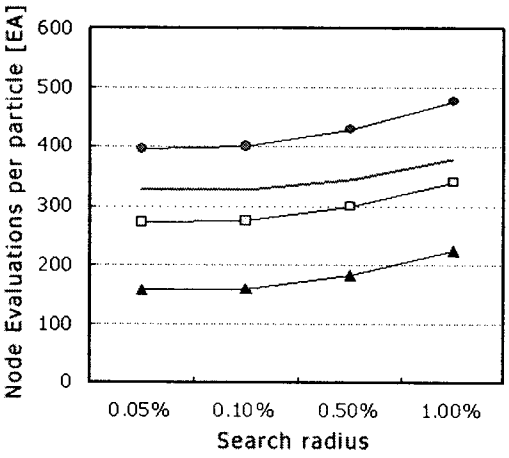

Fig.3 N_E as a function of search radius
(Case A, 32 processors, 640k particles)

하는데 더 많은 시간을 필요로 한다는 점, 또한 실제 상호작용을 계산하는 횟수(N_I)는 어느 경우에나 동일하다는 점 때문이다. 본 연구에서 연산환경과 선택한 문제의 유형과 크기에서 실제 연산시간을 측정한 결과 단일 타임스텝에서의 이득은 측정오차 범위 이내이거나 최대 0.1초에 불과한 것으로 측정되는데 대개 k가 4혹은 5의 값을 가지는 경우이다. 이 이유는 N_E가 k=2인 경우에 대비해 다소 많다고 하더라도 트리의 노드수가 oct-tree에 근사하고 k=2인 경우에 대비해서 트리를 구성하는 속도가 빠르기 때문이다.

4. 결 론

분자동역학과 같은 근거리 힘 계산을 위한 고속화 연산을 위해 트리의 분기비를 결정할 수 있으며 입자 분포에 적응적으로 트리 분할을 하는 새로운 트리 알고리즘을 제안하였다. 제안된 알고리즘(k-tree)은 병렬연산에 동원된 프로세서의 수가 많을 수록, 보다 작은 탐색반경을 가지는 경우 oct-tree에 비해 고속화가 가능한 것으로 나타났다. 그러나 이러한 이득은 트리의 생성에 소요되는 시간이 증가하게 되어 서로 상쇄되므로 단일 타임스텝에서의 실제 이득은 크게 기대할 수 없었다. 그러나 oct-tree를 이용한 고속화 결과와 비교해서 한 타임스텝에서의 연산시간의 이득이 크지 않다고 하더라도 적응적인 트리 분할을 필요로 하는 입자 분포의 경우나, 매우 많은 반복연산을 요하는 문제의 해석에 있어서는 이득을 기대할 수 있을 것이다. 최근의 슈퍼컴퓨팅의 개발 동향은 수천 개의 프로세서를 고속의 네트워크를 이용해 연결하는 거대 병렬컴퓨터라는 점을 감안할 때, 제안된 알고리즘이 대규모 분자동역학을 이용한 전산 해석에 활용될 수 있을 것으로 개대한다.

병렬화 과정에서 프로세서간 통신에 사용되는 시간이 비교적 큰 비중임을 감안한다면 가장 이상적인 연산환경은, 강력한 프로세서들을 사용하는 공유메모리 형태의 컴퓨터에서 트리알고리즘을 사용하는 경우라고 할 수 있을 것이다. 아울러 분자동역학과 같은 근거리 힘 계산 문제의 경우, 상호작용의 연산에 사용되는 시간의 비중이 그리 높지 않기 때문에 프로세서간의 이상적인 부하분산과 네트워크의 고속화, 기타의 오버헤드를 줄이는 것 또한 연산비용의 감소에 크게 기여할 수 있을 것이다

참고문헌

[1] Barnes, J. and Hut, P., 1986, "A Hierarchical O(NlogN) Force-Calculation Algorithm", Nature, Vol. 324, pp. 446-449.

[2] Makino, J., 1990, "Comparison of Two Different Tree Algorithms", Journal of Computational Physics, Vol. 88, pp.393-408

[3] Waltz, J., Page, G. L., Milder, S. D., Wallin, J. and Antunes, A., 2002, "A Performance Comparison of Two Tree Data Structures for N-Body Simulation", Journal of Computational Physics, Vol. 178, pp. 1-14

[4] Warren, M. S. and Salmon, J. K., 1992, "Astrophysical N-Body Simulations using Hierarchical Tree Data Structures", Proceedings of the 1992 ACM/IEEE conference on Supercomputing, Minneapolis, Minnesota, United States, pp. 570 - 576

[5] Warren, M. S. and Salmon, J. K., 1993, "A Parallel Hashed Oct-Tree N-Body Algorithm", Proceedings of the 1993 ACM/IEEE conference on Supercomputing, Portland, Oregon, United States, pp. 12-21

[6] Liu, P. and Bhatt, S.N., 2000, "Experiences with Parallel N-Body Simulation", IEEE Trans. On Parallel & Distributes Systems, Vol. 11, pp.1306~1323