İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ

YÜKSEK BOYUTLU MODEL GÖSTERİLİMİ İLE VERİ BÖLÜNTÜLEME YÖNTEMİNİN KOŞUTLAŞTIRILMASI

DOKTORA TEZİ

M. Engin KANAL

Bilişim Enstitüsü

Hesaplamalı Bilim ve Mühendislik Programı

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ

YÜKSEK BOYUTLU MODEL GÖSTERİLİMİ İLE VERİ BÖLÜNTÜLEME YÖNTEMİNİN KOŞUTLAŞTIRILMASI

DOKTORA TEZİ

M. Engin KANAL
(702032006)

Bilişim Enstitüsü

Hesaplamalı Bilim ve Mühendislik Programı

Tez Danışmanı: Prof. Dr. Metin DEMİRALP

İTÜ, Bilişim Enstitüsü'nün 702032006 numaralı Doktora Öğrencisi **M. Engin KANAL**, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"YÜKSEK BOYUTLU MODEL GÖSTERİLİMİ İLE VERİ BÖLÜNTÜLEME YÖNTEMİNİN KOŞUTLAŞTIRILMASI"** başlıklı tezini aşağıdaki imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Prof. Dr. Metin DEMİRALP** İstanbul Teknik Üniversitesi	.....
Jüri Üyeleri :	**Prof. Dr. Metin DEMİRALP** İstanbul Teknik Üniversitesi	.....
	Prof. Dr. M.Serdar ÇELEBİ İstanbul Teknik Üniversitesi	.....
	Prof. Dr. Ulviye BAŞER İstanbul Teknik Üniversitesi	.....

Teslim Tarihi :
Savunma Tarihi :

ÖNSÖZ

.....

M. Engin KANAL
(Analist)

İÇİNDEKİLER

Sayfa

KISALTMALAR

ALU	:	Arithmetic Logic Unit
CPU	:	Central processing unit
CUDA	:	Compute Unified Device Architecture
ÇYBMG	:	Çarpımsal Yüksek Boyutlu Model Gösterilim
GB	:	Giga byte
GPU	:	Graphics processing unit
KB	:	Kilo byte
MB	:	Mega byte
MPI	:	Message Passing Inferface
MYBMG	:	Melez Yüksek Boyutlu Model Gösterilim
SP	:	Streaming Processor
SM	:	Streaming Multiprocessor
TB	:	Tera byte
YBMG	:	Yüksek Boyutlu Model Gösterilim

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

SEMBOL LİSTESİ

f_0	:	YBMG'nin değişmez terimi
σ_0	:	birinci basamaktan değişmezlik ölçeni
σ_k	:	k. basamaktan toplamsallık ölçeni
W	:	Ağırlık işlevi
δ	:	Kronecker Delta simgesi

YÜKSEK BOYUTLU MODEL GÖSTERİLİMİ İLE VERİ BÖLÜNTÜLEME YÖNTEMİNİN KOŞUTLAŞTIRILMASI

ÖZET

Yalnızca uzaydaki belli noktalardaki değerleri verilmiş çok değişkenli bir $f(x_1, x_2, \cdots, x_N)$ işlevine alışılagelmiş yöntemlerle içdeğerbiçim işlemi yapılması boyut sayısı arttığında birer başbelası durumuna gelir. Bu tür işlevler için doğrudan bilgisayar programcılığı ile çözüm aramak yerine ilk olarak bu işlevleri bilgisayar programlaması açısından daha kolay ele alınacak, matematiksel olarak, etkili bir yapıya getirmek gerekir. Bu amaçla bu işleve yaklaştırım yapan bir böl–ve–yönet algoritması geliştirilmiştir. Bu yaklaştırım sayesinde çok değişkenli f işlevi çok daha düşük boyutlu terimlerle ifade edilebilmektedir. Bu yaklaştırıma Yüksek Boyutlu Model Gösterilimi (YBMG) adı verilmektedir. Bu yöntem çeşitli çalışmalarla başarılı bir şekilde uygulanmıştır. Fakat bu yöntem bu haliyle büyük veri hacmine sahip problemler üzerinde uygulanamaz. Problemdeki boyut sayısı ve boyutlardaki düğüm noktaları sayıları arttığında veri hacmi öyle büyür ki alışılagelmiş PC'ler verinin gereksinim duyduğu yüksek RAM sığasını karşılayamaz. Diğer bir önemli problem de YBMG terimlerini hesaplamakta kullanılan eşitliklerin yapılarıdır. Eşitlikler için yazılmış algoritmadaki döngü sayıları problemdeki boyut sayısına bağımlıdır. Bu çalışmada ilk olarak YBMG terimlerini hesaplamakta kullanılan eşitlikler iyileştirilmiştir. Bu iyileştirme sonucunda eşitliklerin problemdeki boyut sayısına bağımlılığı ortadan kaldırılmıştır. İyileştirilmiş eşitlikler sayesinde yöntem koşutlaştırmaya uygun bir hale getirilmiştir. Son olarak da yöntemin koşutlaştırmasının başarımı çözümlenmiştir.

DATA PARTITIONING VIA HIGH DIMENSIONAL MODEL REPRESENTATION BY USING PARALEL COMPUTING

SUMMARY

If the values of a multivariate function $f(x_1, x_2, \cdots, x_N)$ are given at only a finite number of points in the space of its arguments and an interpolation which employs continuous functions is considered standard multivariate routines may become cumbersome as the dimensionality grows. This urges us to develop a divide–and–conquer algorithm which approximates the function. The given multivariate data is partitioned into low-variate data. This approach is called High Dimensional Model Representation (HDMR). However the method in its current form is not applicable to problems having huge volumes of data. With the increasing dimension number and the number of the corresponding nodes, the volume of data in question reaches such a high level that it is beyond the capacity of any individual PC because huge volume of data requires much higher RAM capacity. Another aspect is that the structure of equalities used in the calculation of HDMR terms varies according to the dimension number of the problem. The number of loops in the algorithm increases with the increasing dimension number. In this work, as a first step, the equations used are modified in such a way that their structure does not depend on the dimension number. With the newly obtained equalities, the method becomes appropriate for parallelization. Due to the parallelization, the RAM problem arising from problems with high volume of data is solved. Finally, the performance of the parallelized method is analyzed.

1. GİRİŞ

YBMG yöntemi ilk olarak 1993 yılında I.M.Sobol tarafından Monte Carlo ve quasi-Monte Carlo algoritmalarını ele alarak farklı değişkenler ya da değişken gruplarına göre bir $f(x_1, x_2, \cdots, x_N)$ işlevinin duyarlılığını kestirmek için geliştirilmiştir[11]. Bu duyarlılık belirlenmesi, doğrudan f işlevi üzerinden gerçekleştirilmemiştir. YBMG yöntemi temel olarak Sobol Teoremi'nde değinilen ve integrali alınabilir bir işlevin, kendisini oluşturan değişkenlerin kendileri ve birbirleri ile olan etkileşimlerinin birleşimine eşit olduğu düşüncesine dayanır. Bu şekilde N değişkenli bir işlev için Taylor açılımına benzer yapıda, ilk olarak değişmez (sabit) bileşen, bir değişkenli bileşenler, iki değişkenli bileşenler ve en sonunda bir adet N değişkenli bileşenin birleşimi olarak yazılır. Burada değinilen HDMR bileşenleri, içerdikleri değişkenlerin birbirleri ile etkileşimlerini göstermektedir.

Sobol'un çalışmasının ardından YBMG yöntemi, Hershel Rabitz ve grubu tarafından daha kapsamlı bir hale getirilmiştir[1-4]. YBMG ve YBMG bazlı diğer algoritmalar atmosferik modelleme, atmosfer dinamikleri ve source-sinks of trace gase, quantitative risk assessment, atmosferik ölçümlerinin inversiyonu, mali ve ekonomik uygulamalar ve stratosferik kimyasal devinim bilimi gibi farklı mühendislik alanları için geliştirilmiştir[1-6].

Bu çalışmada koşutlaştırılan YBMG yöntemi çok değişkenli içdeğerbiçim problemlerine çözüm için geliştirilmiştir[7-10]. Fakat burada içdeğerbiçim uygulanacak işlevin analitik yapısı belli değildir, bunun yerine belli noktalardaki değerleri elde bulunmaktadır. Bu çokdeğişkenli veri yöntem yardımıyla daha az değişkenle anlatılabilecek bir yapıya dönüştürülür ve bu yapı üzerinden içdeğerbiçim işlemi yapılır. Burada anafikir verinin en fazla birli değişkenlerden oluşan bir yapı ile yaklaştırım yapılmasını sağlamaktır.

Bu noktada yöntemin gerçek dünya problemlerinde uygulanması için neden koşutlaştırmaya gereksinim duyuldugu açıklanmalıdır. Koşutlaştırma gereksinimi temel bir soruna çözüm bulacaktır. Bu sorun yüksek veri hacmine sahip problemler üzerinde

yöntemin uygulanmasıdır. Gerçek dünya problemlerinde bir olayı etkileyen çok sayıda değişken vardır. Bu değişkenlerin ve değişkenlerdeki bileşen sayısı arttıkça olayın gerçekleştiği uzay da genişler. YBMG, N değişkenli bir f işlevini daha düşük boyutlardaki terimlerde ifade eder. Fakat bunun için eldeki tüm veriyi kullanması gerekir. Veri kümesi değişkenlerdeki bileşen sayılarının çarpımıdır. İster düşük ister yüksek sayıda boyut kullanılsın, veri kümesini oluşturan her bir değişkene ait bileşen sayıları belli bir sayının üzerine çıktıkça, milyonlara hatta milyarlara varan noktalardan oluşmuş veri setleri kolaylıkla ortaya çıkar. Örnek olarak 3 değişkenli bir uzayda her bir değişkende 10.000 adet bileşen olduğu düşünülürse ve bu bileşenlerin oluşturduğu nokta sayısı $10.000^3 = 1 \cdot 10^{12}$ adet olacaktır. Tüm noktalar $float$ türünde tutulsa veri kümesinin hacmi $4 \cdot 10^{12}$ $byte$ olacaktır. Bu da $terabyte$ hacmine sahip bir veri seti ile çalışıldığı anlamına gelmektedir. Bu kadar büyük hacimlerle alışılagelmiş bir PC üzerinde uygulama çalıştırılamaz. Yöntemin koşutlaştırılması için yöntemde kullanılacak eşitliklerin koşutlaştırılması gerekmektedir. Burada da diğer önemli problem ortaya çıkar. Yöntemde kullanılan eşitliklerin yapıları problemdeki boyut sayısına göre değişkenlik göstermektedir. Bu problemi aşmak için ilk olarak bu eşitlikler üzerinde iyileştirme çalışması yapılmıştır. İyileştirme çalışması sonucunda eşitlikler boyut sayılarından bağımsız bir yapıya kavuşturulmuştur. Bu aşamadan sonra yöntem koşutlaştırılmıştır. Koşutlaştırma yöntemi olarak MPI (Message Passing Interface) ve CUDA (Compute Unified Device Architecture) kitaplıkları kullanılmıştır. MPI, supercomputer altyapısı olan kullanıcılar için tasarlanmış bir kitaplıktır. CUDA kitaplığı ise son yıllarda çok tanınmış olan ve ekran kartı üzerinde bulunan "Performance per watt" olarak çok daha etkili GPU (Graphical Processing Unit) kullanan bir kitaplıktır. Böylece hem supercomputing altyapısı taşıyan, hem de GPU kullanabilen bir bilgisayar altyapısına sahip kullanıcılar için koşutlaştırma seçenekleri sunulmuştur ve başarımları incelenmiştir.

Tezin 2. kısmında YBMG yöntemi hakkında bilgi verilmiştir.

Tezin 3. kısmında veri bölüntüleme ve içdeğerbiçim üzerinde durulmuştur.

Tezin 4. kısmında yöntemde kullanılan eşitliklerin iyileştirilme aşamaları üzerinde durulmuştur.

Tezin 5. kısmında yöntemin MPI ile koşutlaştırılma aşamaları anlatılmıştır.

Tezin 6. kısmında yöntemin CUDA ile koşutlaştırılma aşamaları anlatılmıştır.

Tezin sonuç kısmında koşutlaştırma süreçlerinden çıkan sonuçlar tartışılmıştır.

2. TEZDE KOŞUTLAŞTIRILACAK YÖNTEMLER

Bu bölümde tez çalışması içerisinde koşutlaştırılacak olan Yüksek Boyutlu Model Gösterilimi (YBMG) yöntemiyle ilgili kısaca bilgi verilmesi amaçlanmıştır. Bu amaçla ilk olarak bu yöntemin çeşitleri üzerinde durulacak ve tarihsel gelişim süreciyle ilgili bilgi verilecektir.

2.1 Yüksek Boyutlu Model Gösterilimi (YBMG) Yöntemleri

YBMG konusunda yapılan ilk çalışma olan Sobol Kanıtsavına (teoremine) göre tümlevlenebilir (integrali alınabilir) her $f(x_1,...,x_N)$ işlevi aşağıdaki anlatımla yazılabilir.

$$f(x_1,...,x_N) = f_0 + \sum_{i_1=1}^{N} f_{i_1}(x_{i_1}) + \sum_{\substack{i_1,i_2=1 \\ i_1<i_2}}^{N} f_{i_1 i_2}(x_{i_1},x_{i_2}) + \cdots + f_{12\ldots N}(x_1,...,x_N) \tag{2.1}$$

Açılım, görüldüğü gibi, sonlu bir toplamdır ve değişmez terim, tek değişkenli terimler, iki değişkenli terimler sırasında ilerlemektedir. Burada, önemli olan, sağ yandaki f'leri tanımlamaktır. Kanıtsavdaki öngörüme göre

$$\int_0^1 dx_s f_{i_1,i_2,\ldots i_k}(x_{i_1},\ldots,x_{i_k}) = 0 \qquad s = i_1,\ldots,i_k \tag{2.2}$$

(**2.1**) eşitliğinin sağ yanındaki terimler için diklik koşulu geçerlidir.

$$(f_{i_1 i_2\ldots i_k}, f_{i_1 i_2\ldots i_l}) = 0, \quad \{i_1,i_2,\ldots,i_k\} \not\equiv \{i_1,i_2,\ldots,i_l\}, \quad 1 \le k,l \le N \tag{2.3}$$

$$\begin{aligned} \left(f_{i_1,\ldots i_k}, f_{j_1,\ldots j_l}\right) \equiv & \int_{a_1}^{b_1} dx_1 \cdots \int_{a_1}^{b_1} dx_N W(x_1,...,x_N) f_{i_1,\ldots i_k}(x_{i_1},\ldots,x_{i_k}) \\ & \times f_{j_1,\ldots j_l}\left(x_{j_1},\ldots,x_{j_l}\right), \qquad 1 \le i_1 < \ldots < i_k \le N, \end{aligned}$$

$$1 \le j_1 < \ldots < j_l \le N, \qquad 1 \le k,l \le N \tag{2.4}$$

Yukarıda tanımlanan eşitliğe ve diklik koşuluna uygun olarak (**2.1**)'de değişmez terim olarak gösterilen f_0'ı bulmak için (**2.1**) eşitliğinin her iki yanının N kez tümlevini alırız.

$$\int_0^1 \cdots \int_0^1 dx_1 \ldots dx_N f(x_1,\ldots,x_N) = f_0 \tag{2.5}$$

Benzer biçimde, (**2.1**) eşitliğinin her iki yanının x_i değişkeni dışlanarak N-1 kez tümlevi alınırsa aşağıdaki eşitlik elde edilir ve bu eşitlikten $f_i(x_i)$ değerleri üretilebilir.

$$\underbrace{\int_0^1 \cdots \int_0^1}_{N-1\ kat} dx_1 \ldots dx_{i-1} dx_{i+1} \ldots dx_N f(x_1,\ldots,x_N) = f_0 + f_i(x_i) \qquad \forall i = 1,\ldots,N \quad \textbf{(2.6)}$$

(**2.1**) eşitliğindeki tüm f'ler bu biçimde elde edilir. Sobol'ca bulunan yöntem Rabitz Grubu'nun çalışmaları ile genişletilmiş bir yapıya kavuşturulmuş ve kapsamlı uygulama alanları bulunmuştur[1-4]. Sobol'un yazısında 1 alınan ağırlık işlevi ve [0,1] aralığındaki tümlev yerine genişletilmiş yapı için aşağıdaki koşul verilmiştir.

$$\int_{a_1}^{b_1} dx_1 \cdots \int_{a_N}^{b_N} dx_N W(x_1,...,x_N) f_i(x_i) = 0, \quad 1 \le i \le N \qquad \textbf{(2.7)}$$

Burada verilen $W(x_1,...,x_N)$ ağırlık işlevi aşağıdaki biçimde tanımlanmaktadır.

$$W(x_1,\ldots,x_N) \equiv \prod_{j=1}^{N} W_j(x_j), \qquad x_j \in \left[a_j, b_j\right], \qquad 1 \le j \le N \qquad \textbf{(2.8)}$$

Ayrıca ağırlık işlevi aşağıdaki boylandırım (ing: normalization) koşulunu da sağlamalıdır.

$$\int_{a_j}^{b_j} dx_j W_j(x_j) = 1, \qquad 1 \le j \le N \qquad \textbf{(2.9)}$$

Sobol Kanıtsavı'na benzer biçimde değişmez, tek değişkenli ve iki değişkenli YBMG terimleri $W(x_1,...,x_N)$ ağırlığı altında aşağıdaki biçimde bulunur.

$$f_0 = \int_{a_1}^{b_1} dx_1 \cdots \int_{a_1}^{b_1} dx_N W(x_1,...,x_N) f(x_1,\ldots,x_N) \qquad \textbf{(2.10)}$$

$$\begin{aligned} f_k(x_k) \;=\; & \int_{a_1}^{b_1} dx_1 W_1(x_1) \cdots \int_{a_{k-1}}^{b_{k-1}} dx_{k-1} W_{k-1}(x_{k-1}) \\ & \times \int_{a_{k+1}}^{b_{k+1}} dx_{k+1} W_{k+1}(x_{k+1}) \cdots \int_{a_N}^{b_N} dx_N W_N(x_N) \\ & \left[f_0 + \sum_{i_1=1}^{N} f_{i_1}(x_{i_1}) + \cdots + f_{12\ldots N}(x_1,...,x_N) \right] - f_0, \qquad 1 \le k \le N \end{aligned} \qquad \textbf{(2.11)}$$

$$\begin{aligned} f_{k_1 k_2}(x_1,...,x_N) \;=\; & \int_{a_1}^{b_1} dx_1 W_1(x_1) \cdots \int_{a_{k_1-1}}^{b_{k_1-1}} dx_{k_1-1} W_{k_1-1}(x_{k_1-1}) \\ & \times \int_{a_{k_1+1}}^{b_{k_1+1}} dx_{k_1+1} W_{k_1+1}(x_{k_1+1}) \cdots \int_{a_{k_2-1}}^{b_{k_2-1}} dx_{k_2-1} W_{k_2-1}(x_{k_2-1}) \\ & \times \int_{a_{k_2+1}}^{b_{k_2+1}} dx_{k_2+1} W_{k_2+1}(x_{k_2+1}) \cdots \int_{a_N}^{b_N} dx_N W_N(x_N) \\ & \times \left[f_0 + \sum_{i_1=1}^{N} f_{i_1}(x_{i_1}) + \cdots + f_{12\ldots N}(x_1,...,x_N) \right] \\ & - f_{k_1}(x_{k_1}) - f_{k_2}(x_{k_2}) - f_0, \qquad 1 \le k_1 < k_2 \le N \end{aligned} \qquad \textbf{(2.12)}$$

Amacımız çok değişkenli işlevin kesin değerini üretmek olmadığından, aksine, belli bir kesme yaklaştırımı ile yetinilmek istendiğinden, geri kalan YBMG terimleri gözönüne alınmamaktadır. YBMG terimlerini belirlemekte kullanılan tümlev işleminde bulunan diklik koşulu üzerinde özenle durulması gereklidir. Diklik koşulu altında aşağıdaki eşitlik yazılabilir.

$$\int_{a_1}^{b_1} dx_1 \cdots \int_{a_1}^{b_1} dx_N W(x_1,...,x_N) f_{i_1,...i_k}(x_{i_1},\ldots,x_{i_k}) f_{j_1,...j_l}\left(x_{j_1},\ldots,x_{j_l}\right) = 0$$
$$i_1,\ldots,i_k \neq j_1,\ldots,j_l, \qquad 1 \leq i_1 < \ldots < i_k \leq N,$$
$$1 \leq j_1 < \ldots < j_l \leq N, \qquad 1 \leq k,l \leq N \tag{2.13}$$

Bu da bizi aşağıdaki iççarpım tanımını kullanmaya zorlar.

$$\begin{aligned}\left(f_{i_1,...i_k}, f_{j_1,...j_l}\right) \equiv & \int_{a_1}^{b_1} dx_1 \cdots \int_{a_1}^{b_1} dx_N W\left(x_1,...,x_N\right) f_{i_1,...i_k}\left(x_{i_1},\ldots,x_{i_k}\right) \\ & \times f_{j_1,...j_l}\left(x_{j_1},\ldots,x_{j_l}\right), \qquad 1 \leq i_1 < \ldots < i_k \leq N,\end{aligned}$$

$$1 \leq j_1 < \ldots < j_l \leq N, \qquad 1 \leq k,l \leq N \tag{2.14}$$

Bu bizim aşağıdaki boy tanımını kullanmamıza olanak verir.

$$\|f_{i_1,...i_k}\| \equiv (f_{i_1,...i_k}, f_{i_1,...i_k}), \qquad 1 \leq i_1 < \ldots < i_k \leq N, \qquad 1 \leq k \leq N \tag{2.15}$$

YBMG terimlerinin boy tanımını ve diklik koşulunu kullanarak (**2.1**) eşitliği için aşağıdaki anlatım yazılabilir.

$$\|f\|^2 = \|f_0\|^2 + \sum_{i_1=1}^{N} \|f_{i_1}\|^2 + \sum_{\substack{i_1,i_2=1 \\ i_1<i_2}}^{N} \|f_{i_1 i_2}\|^2 + \cdots + \|f_{12...N}\|^2 \tag{2.16}$$

Bu eşitlik bize kesme uygulanmış YBMG yaklaştırımının niteliğini ölçmemizi sağlar. Eğer YBMG değişmez terimden sonra kesilirse yaklaştırıma "Değişmez Yaklaştırım", değişmez ve birli terimlerinden sonra kesilirse "Birli Yaklaştırım", değişmez, birli ve ikili YBMG terimlerinden sonra kesilirse "İkili Yaklaştırım"adları verilir. Bu yaklaştı-

rım adlandırmaları aşağıdaki toplamsallık ölçenlerinin tanımlanmasına olanak sağlar.

$$\begin{aligned}
\sigma_0 &\equiv \frac{1}{\|f\|^2}\|f_0\|^2 \\
\sigma_1 &\equiv \frac{1}{\|f\|^2}\sum_{i=1}^{N}\|f_i\|^2 + \sigma_0 \\
&\cdots \\
\sigma_N &\equiv \frac{1}{\|f\|^2}\|f_{12\ldots N}\|^2 + \sigma_{N-1}
\end{aligned} \tag{2.17}$$

Burada σ_0 "Değişmezlik Ölçeni"olarak adlandırılmaktadır. Diğer σ_k terimleri de, k değişken içeren terimlerle ilgili olan "k. Toplamsallık Ölçeni"olacak biçimde adlandırılırlar. Bu ölçenler 0 ile 1 değerleri arasında sıralı olarak aşağıdaki anlatımı sağlarlar.

$$1 \leq \sigma_0 \leq \sigma_1 \leq \cdots \leq \sigma_N = 1 \tag{2.18}$$

YBMG yaklaştırımındaki kesme yaklaştıranları (ing: approximant) daha ayrıntılı bir biçimde aşağıda verilmektedir.

$$\begin{aligned}
s_0(x_1,x_2,\cdots,x_N) &= f_0 \\
s_1(x_1,x_2,\cdots,x_N) &= s_0(x_1,x_2,\cdots,x_N) + \sum_{i=1}^{N} f_i(x_i) \\
&\vdots \\
s_k(x_1,x_2,\cdots,x_N) &= s_{k-1}(x_1,x_2,\cdots,x_N) + \sum_{\substack{i_1,\cdots,i_k=1 \\ i_1<\cdots<i_2}}^{N} f_{i_1,\cdots,i_k}(x_{i_1},\cdots,x_{i_k}) \\
&\qquad 1 \leq k \leq N
\end{aligned} \tag{2.19}$$

Bu tanımları gözönüne alarak (**2.1**)'de verilmiş bulunan $f(x_1,\cdots,x_N)$ işlevine k. basamaktan YBMG yaklaştırımı olan $s_k(x_1,\cdots,x_N)$ aşağıdaki anlatımla verilebilir.

$$f(x_1,\cdots,x_N) \approx s_k(x_1,\cdots,x_N) \tag{2.20}$$

Bu yaklaştırımın boyu da aşağıdaki ilişkiyi sağlar.

$$\|s_k\| = \sigma_k\|f\|, \qquad 0 \leq k \leq N \tag{2.21}$$

İşlev (**2.1**)'de verilen anlatımla açılır ve $f(x_1,x_2,\ldots,x_N)$ benimsenebilir bir k değeri için "Toplamsal"(YBMG'inde salt değişmez ve birli terim barındıran, diğer bileşenleri özdeş olarak 0 olan) bir işlev ise 1. basamaktan YBMG yaklaştırımı, işlevi, kesin

olarak anlatmaya yeterli olacaktır. Bu nedenle, 1. basamaktan YBMG yaklaştırımına arıtoplamsal (ing: pure additive) yaklaştırım adı da verilmektedir. Verilen herhangi bir çok değişkenli işlevin bu özelliği taşıması beklenemeyeceğinden ve dolayısıyla YBMG ile yeterince duyarlılık elde edilemeyeceğinden ya yüksek basamaktan yaklaştırımlara çıkmak gerekmekte (ki bu durumda bilgisayım karmaşıklığının çok çok büyümesi söz konusu olabilmektedir) ya da başka bir yöntem geliştirme gereksinimi doğmaktadır.

2.1.1 Çarpımsallaştırılmış YBMG (ÇYBMG)

Eğer bir an için $f(x_1,...,x_N)$ işlevinin arıçarpımsal bir yapıda olduğu yani

$$f(x_1,...,x_N) = \prod_{i=1}^{N} u_i(x_i) \tag{2.22}$$

yazılabildiği varsayılacak olursa, YBMG f_0 değişmez terimini bulmak için (**2.10**) eşitliği kullanılırsa

$$f_0 = \int_{a_1}^{b_1} dx_1 W_1(x_1) u_1(x_1) \cdots \int_{a_N}^{b_N} dx_N W_N(x_N) u_N(x_N) \tag{2.23}$$

yazılabilir. Burada $\overline{u}_m$

$$\overline{u}_m = \int_{a_m}^{b_m} dx_m W_m(x_m) u_m(x_m), \qquad 1 \leq m \leq N \tag{2.24}$$

ile tanımlanırsa değişmez terim için yazılan (**2.23**) eşitliği aşağıdaki biçimde yeniden yazılabilir.

$$f_0 = \overline{u}_1 \overline{u}_2 \cdots \overline{u}_N \tag{2.25}$$

(**2.1**) eşitliğinden yola çıkarak

$$f_0 + f_j(x_j) = (\overline{u}_1 \overline{u}_2 \cdots \overline{u}_N) \frac{u_j(x_j)}{\overline{u}_j} = f_0 \frac{u_j(x_j)}{\overline{u}_j}, \tag{2.26}$$

$$f_j(x_j) = f_0 \left(\frac{u_j(x_j)}{\overline{u}_j} - 1 \right), \tag{2.27}$$

$$u_j(x_j) = \overline{u}_j \left(\frac{f_j(x_j)}{f_0} + 1 \right), \qquad 1 \leq j \leq N \tag{2.28}$$

sonuçları elde edilir.

ÇYBMG ikili terimleri de, benzer biçimde, aşağıdaki eşitliklerle verilen yapılarda elde edilebilirler.

$$\begin{aligned} f_0 + f_j(x_j) + f_k(x_k) + f_{j,k}(x_j,x_k) &= (\overline{u}_1\overline{u}_2\cdots\overline{u}_N)\left(\frac{u_j(x_j)}{\overline{u}_j}\right)\left(\frac{u_k(x_k)}{\overline{u}_k}\right) \\ f_{j,k}(x_j,x_k) &= f_0\left(\frac{u_j(x_j)}{\overline{u}_j}\right)\left(\frac{u_k(x_k)}{\overline{u}_k}\right) - f_0\left(\frac{u_j(x_j)}{\overline{u}_j}-1\right) \\ &\quad - f_0\left(\frac{u_k(x_k)}{\overline{u}_k}-1\right) - f_0 \\ f_{j,k}(x_j,x_k) &= f_0\left(\frac{u_{x_j}(x_j)}{\overline{u}_j}-1\right)\left(\frac{u_k(x_k)}{\overline{u}_k}-1\right), \qquad 1\le j<k\le N \end{aligned} \tag{2.29}$$

Elde edilen terimlere ait eşitlikler aşağıdaki biçimde yeniden düzenlenebilir.

$$\begin{aligned} &f_0 = \prod_{i=1}^{N}\overline{u_i}, \qquad \overline{u}_i \equiv \int_{a_i}^{b_i} dx_i W_i(x_i)\,u_i(x_i), \qquad 1\le i\le N \\ &f_i(x_i) = f_0\left(\frac{u_i(x_i)}{\overline{u}_i}-1\right), \qquad 1\le i\le N \\ &f_{i_1,i_2}(x_{i_1},x_{i_2}) = f_0\left(\frac{u_{i_1}(x_{i_1})}{\overline{u_{i_1}}}-1\right)\left(\frac{u_{i_1}(x_{i_2})}{\overline{u_{i_2}}}-1\right), \qquad 1\le i_1<i_2\le N \\ &\cdots\cdots \end{aligned} \tag{2.30}$$

Bunun anlamı $f(x_1,...,x_N)$ işlevinin arıçarpımsal olması durumunda, **2.1**'deki arıtoplamsal açılımın da arıçarpımsal bir yapının açılımı durumuna dönüştüğü doğrultusundadır. $f(x_1,...,x_N)$ işlevinin çarpımsal olmaması ve **(2.30)** bize, aşağıdaki eşitliği yazmamıza, olanak verir.

$$\begin{aligned} f_{i_1\ldots i_k}(x_{i_1},...,x_{i_N}) \equiv \frac{1}{f_0^{k-1}} f_{i_1}(x_{i_1})\ldots f_{i_k}(x_{i_k}), \\ 1\le i_1<\cdots<i_k\le N, \qquad 1\le k\le N \end{aligned} \tag{2.31}$$

Buradaki yapı $f(x_1,...,x_N)$ işlevinin $f_{YBMG}(x_1,...,x_N)$ ile simgeleyeceğimiz YBMG açılımında yerine konulur ve ortaya çıkan sonlu toplam özenle incelenirse

$$f_{YBMG}(x_1,...,x_N) = f_0\prod_{i=1}^{N}\left(1+\frac{f_i(x_i)}{f_0}\right) \tag{2.32}$$

yazılabileceği anlaşılır. Bunun doğruluğunu sınamak için birli terimlerin u'lar türünden anlatımları bu eşitlikte kullanılırsa

$$f_{YBMG}(x_1,...,x_N) = f_0\prod_{i=1}^{N}\left(1+\left[\frac{u_i(x_i)}{u_i}-1\right]\right) = f_0\left(\prod_{i=1}^{N}\frac{u_i(x_i)}{f_0}\right) \equiv f(x_1,...,x_N) \tag{2.33}$$

elde edilir. Yani, ilgilenilen çok değişkenli işlevin arıçarpımsal olması durumunda, onun YBMG açılımı (**2.32**) anlatımıyla yazılabilen arıçarpımsal bir yapıya dönüştürülebilmektedir.

Eğer ilgilenilen işlev arıçarpımsal değilse (**2.32**) geçerliliğini, daha doğrusu yeterliliğini koruyamaz. Bunun nedeni, bu anlatımda salt bir değişkenli çarpanlar kullanılmış olmasıdır. Dolayısıyla, ilgilenilen işlev ne olursa olsun, geçerliliğini koruyacak ve YBMG gibi yeterli olacak bir anlatımda değişmez bir çarpanla onu izleyen N sayıda birli çarpanla yetinilmemeli bunları sırasıyla, ikili, üçlü ve sayıları tekdüze artan ve sonunda N olan çok değişkenli çarpanlar izlemelidir. Yani, tıpkı YBMG'de olduğu gibi 2^N sayıda çarpan içerilmelidir. Dolayısıyla, ÇYBMG için önerilecek yapı, altsırasayılı r'lerle simgelenen büyüklükler bu an için bilinmeyen ama belirlenmesi istenen ögeler olmak üzere

$$\begin{aligned} f(x_1,...,x_N) \;\equiv\; & r_0 \left[\prod_{i=1}^{N}(1+r_i(x_i))\right]\left[\prod_{\substack{i_1,i_2=1\\ i_1<i_2}}^{N}(1+r_{i_1 i_2}(x_{i_1},x_{i_2}))\right]\times \\ & \cdots \times r_{12..N}(x_{i_1},...,x_{i_N}) \end{aligned} \tag{2.34}$$

anlatımıyla verilmelidir. Bu anlatımdaki r'lerin belirlenmesi için anlatımın çarpma işlemlerini gerçekleştirerek toplamsal bir yapıya açılması ve ortaya çıkan terimlerin oluşturduğu YBMG'sel yapının YBMG'in değişmez, birli, ve diğer terimleriyle, terim terim karşılaştırma yapılması gerekir. Bu işlemin ara ayrıntılarına burada girmeyeceğiz. Ancak, değişmez, birli, ve de ikili çarpanların bu işlem sonucunda elde edilecek olan, YBMG terimleri türünden anlatımlarını vermekle yetineceğiz.

$$r_0 = f_0, \tag{2.35}$$

$$r_i(x_i) = \frac{f_i(x_i)}{f_0}, \qquad 1 \le i \le N \tag{2.36}$$

$$r_{i_1 i_2}(x_{i_1},x_{i_2}) = \frac{f_{i_1 i_2}(x_{i_1},x_{i_2})}{f_0} - \frac{f_{i_1}(x_{i_1})}{f_0}\frac{f_{i_2}(x_{i_2})}{f_0}, \qquad 1 \le i_1 < i_2 \tag{2.37}$$

Burada son terim ikili terimlerin (**2.32**)'i saglamasi durumunda, yani ilgilenilen asıl işlevin arıçarpımsal olması durumunda, değeri sıfırlanır. Bu terim, bir anlamda, tıpkı $f_{i_1 i_2}(x_{i_1},x_{i_2})$ gibi, ikili bağlılaşımlara (ing: correlation) karşılık gelmektedir. Dolayısıyla, bu terim ikili bağlılaşımların arıçarpımsallığını ölçmektedir. Daha çok bağımsız değişken içeren çarpanlar için de benzer yorumlar yapılabilir. Ancak, burada bu seviyeye kadar bilgilendirim ile yetinilecektir.

Tıpkı YBMG'de olduğu gibi ÇYBMG'de de bazı ölçenler tanımlanabilir. Bu amaçla, eğer,

$$\varphi_i(x) \equiv 1 + \frac{f_i(x_i)}{f_0}, \qquad 1 \leq i \leq N \tag{2.38}$$

tanımı yapılırsa

$$\pi_0 \equiv \frac{\|f_0\|^2}{\|f\|^2} \tag{2.39}$$

$$\pi_1 \equiv \frac{\left\| f_0 \prod_{i=1}^{N} \varphi_i(x_i) \right\|^2}{\|f_0\|^2} \tag{2.40}$$

anlatımında iki "Çarpımsallık Ölçeni"tanımlanabilir. Bunlardan ilkini "ÇYBMG Değişmezlik Ölçeni"olarak adlandırmak olanaklıdır. Bu ölçen (π_0) YBMG'nin Değişmezlik Ölçeni'ne (σ_0) eşdeğerdir. Buradaki diğer olçen, π_1 ise YBMG'nin Arıtoplamsallık Ölçeni'ni çağrıştırır ve "ÇYBMG Arıçarpımsallık Ölçeni"olarak adlandırılabilir. $\pi_0 = 1$ durumu Değişmezliğe, $\pi_1 = 1$ durumu ise Arıçarpımsallığa karşılık gelir. Ancak, YBMG Ölçenleri durumundaki tekdüze artma ve üstten sınırlı olma durumu buradaki ÇYBMG Ölçenleri için yoktur. Salt bu özelliği ÇYBMG'nin etkinliğini çok önemli ölçüde azaltmakta ve ilgilenilen işlevin yapısal olarak değil de verisel olarak verilmesi durumunda güvenilirliğini sağlayamamaktadır. Ancak, yine de, ilgilenilen işlevin arıçarpımsallığının çok baskın olduğunun bilindiği durumlarda, ÇYBMG'den yararlanmak, onu YBMG'ye göre öngörüme çıkarmak olanaklıdır.

2.1.2 Melez YBMG (MYBMG)

Bu kesimde çok değişkenli işlevlerde kullanılan YBMG yöntemi için sıradan YBMG ve ÇYBMG'den değişik bir türü anlatılacaktır. Bu tür YBMG için verilen çok boyutlu veriler YBMG'ye uygun olarak ne arı ya da baskın toplamsal ne de ÇYBMG'ye uygun olarak arı ya da baskın çarpımsal bir yapı içerirler. Bu tür için verilen bir veriye karşılık YBMG ve ÇYBMG gösterim türlerini birlikte içeren bir gösterilim biçimi olan melez bir yapı ortaya konacaktır. Bu yapı aşağıdaki anlatımla verilebilir.

$$\begin{aligned} f(x_1,...,x_N) &= \gamma \left(f_0 + \sum_{i_1=1}^{N} f_{i_1}(x_{i_1}) + \cdots \right) \\ &+ (1-\gamma) \left(r_0 \left[\prod_{i_1=1}^{N} (1 + r_{i_1}(x_{i_1})) \right] \times \cdots \right) \end{aligned} \tag{2.41}$$

Aslında, burada, sağ yanda, birbirine özdeş ve biri toplamsal diğeri çarpımsal nitelikli olan iki anlatım ağırlıklı olarak devreye sokulmaktadır. Bu durum, γ'nın $[0,1]$

aralığında bulunması gerektiği anlamına gelmektedir ve onun değeri ilgilenilen işlevin toplamsallık yüzdesini, 1'e olan uzaklığı ise çarpımsallık yüzdesini gösterecektir. Bu anlatım, YBMG ile ÇYBMG birleştirimi niteliğindedir.

Burada, işlevin sayısal veriler üzerinden verilmesi durumunda, veriler kartezyen çarpımla oluşturulan bir ızgaranın (ing: grid) tüm düğüm noktalarında işlev değerlerini belirli kılıyorsa ve işlev buradaki gibi melez yapıda ise MYBMG'den yararlanılabilir.

(**2.41**)'i kullanarak MYBMG için kesme yaklaştırımları üretilebilir. Bunun için (**2.41**)'de YBMG ve ÇYBMG anlatımları yerine onların değişik kertelerden (mertebe) olabilen kesme yaklaştıranları kullanılabilir. Böylece aşağıdaki yaklaştıranlar üretilebilir.

$$f(x_1,...,x_N) \approx h_{j,k}(x_1,...,x_N;\gamma) \equiv \gamma s_j(x_1,...,x_N) + (1-\gamma)\pi_k(x_1,...,x_N), \quad 1 \le j,k \le N \tag{2.42}$$

Bu yaklaştıran "(j,k). Kerte'den Melez YBMG Yaklaştıranı"olarak adlandırılabilir. Burada gözüken s_j ve π_k işlevleri daha önceki kesimlerde tanımlanmış j'ninci YBMG yaklaştırım büyüklükleridir. Melez YBMG kesme yaklaştırımlarında kullanılan yaklaştıranlar aşağıdaki gibi bir çizelge yapısında gösterilebilirler.

$$\begin{matrix} h_{0,0} & h_{0,1} & \cdots & h_{0,N} \\ h_{1,0} & h_{1,1} & \cdots & h_{1,N} \\ \vdots & \vdots & \ddots & \vdots \\ h_{N,0} & h_{N,1} & \cdots & h_{N,N} \end{matrix} \tag{2.43}$$

Buradaki en önemli adım γ melezlik değiştirgesinin (ing: parameter) saptanmasıdır. Bu amaç için açağıdaki biçimde bir işlev tanımlayabiliriz.

$$F(x_1,...,x_N) \equiv \|f\|^2 - \|f_{MYBMG}(\gamma)\|^2 \tag{2.44}$$

Burada f ve $f_{MYBMG}(\gamma)$ sırasıyla verilen işlev ile MYBMG'nin bu işleve yaklaştıranını göstermekte olup, olması gerektiği gibi, boylardaki işlevlerde bağımsız değişkene olan bağımlılık açık olarak gösterilmemektedir. γ değerini saptamak için bu eşitliğin sol yanındaki işlevin boy değerinin en küçük kılınması gerekmektedir.

$$\frac{\partial F}{\partial \gamma} = 0 \tag{2.45}$$

Bu γ'ya göre eniyilenen işlevin oluşturumunda olsun YBMG ile ÇYBMG yaklaştıranlarının oluşturumunda olsun ayrı ayrı değişik ağırlık işlevleri kullanmayı engelleyen bir

kısıtlama söz konusu değildir. Hangi ağırlık işlevinin nerede kullanılacağına ilgilenilen sorunun yapısına göre karar verilir. Sorun YBMG'si baskın ise YBMG ağırlık işlevi, diğer durumda ise, EYBMG ağırlık işlevi kullanılmalıdır. Ama, en iyisi, hepsi için ortak bir ağırlık kullanmaktır. Bu ortak ağırlık, işlevin yapısal olarak verilmesi durumunda, ilgili tanımbölgesinin her noktasında işlev değeri bilineceğinden, ağırlık işlevini tüm tanımbölgesi içinde sürekli seçmek olanaklıdır. Ancak, buradaki amacın, daha çok, tanımbölgesi içinde sonlu sayıda yinelemesiz noktada işlev değerlerinin veri olarak verilmesi ve başka bir bilginin elde bulunmadığı durumlarla ilgilenmek olmasından dolayı ağırlık işlevini de, tümlevleme altında, salt bu noktalarda işlev değeri üretecek yapıda yani ayrık yapılı seçmek gerekmektedir. Bir değişkenli ağırlık çarpanlarının YBMG için gerekli yapıyı üretebilmesi için aşağıdaki anlatımla verilen Dirac delta işlevleri doğrusal birleştirimi olarak seçilmesi gerekir.

$$W_i(x_i) \equiv \sum_j^{n_i} \alpha_j^{(i)} \delta\left(x_i - \xi_i^{(j)}\right), \qquad 1 \le i \le N \tag{2.46}$$

Burada, $\xi_i^{(j)}$ x_i doğrultusundaki, artan yönde sıralanmış toplam n_i sayıda düğüm noktasından j. konumda bulunanı simgelenmekte olup $\alpha_j^{(i)}$ bu düğüm noktasının hangi ağırlıkla gündeme getirildiğini simgeleyen değiştirge durumundadır. Bu değiştirgeler bu ağırlık çarpanının tümlev birimlendiriminin gerçekleştirilmesi amacıyla aşağıdaki 1'e toplanırlık özelliğini taşımalıdır.

$$\sum_j^{n_i} \alpha_j^{(i)} = 1, \qquad 1 \le i \le N \tag{2.47}$$

Son iki bağıntıdan yararlanarak

$$F(x_1,...,x_N;\gamma) = \sum_{j_1=1}^{n_1} \cdots \sum_{j_N=1}^{n_N} \left[\prod_{i_1=1}^{N} \alpha_{j_{i_1}}^{(i_1)}\right] \times \left[f\left(\xi_1^{(j_1)},...,\xi_N^{(j_N)}\right) - \gamma s_j\left(\xi_1^{(j_1)},...,\xi_N^{(j_N)}\right) - (1-\gamma)\varphi_k\left(\xi_1^{(j_1)},...,\xi_N^{(j_N)}\right)\right]^2,$$

$$0 \le j,k \le N \tag{2.48}$$

Bu eşitlikle verilen işlev daha doğrusu eşitliğin sağ yanı enküçüklenirse aranan γ değeri elde edilir.

$$\gamma_{j,k}^{(ek)} = \frac{A_2 + A_3 - A_4 - A_5}{A_1 + A_2 - 2A_5}, \qquad 1 \le j,k \le N \tag{2.49}$$

Bu eşitliğin sağ yanında görünen değiştirgelerin açık anlatımları aşağıda verilmektedir.

$$
\begin{aligned}
A_1 &= \sum_{j_1=1}^{n_1} \cdots \sum_{j_N=1}^{n_N} \left[\prod_{i_1=1}^{N} \alpha_{j_{i_1}}^{(i_1)} \right] s_j \left(\xi_1^{(j_1)}, ..., \xi_N^{(j_N)} \right)^2, \\
A_2 &= \sum_{j_1=1}^{n_1} \cdots \sum_{j_N=1}^{n_N} \left[\prod_{i_1=1}^{N} \alpha_{j_{i_1}}^{(i_1)} \right] \varphi_k \left(\xi_1^{(j_1)}, ..., \xi_N^{(j_N)} \right)^2, \\
A_3 &= \sum_{j_1=1}^{n_1} \cdots \sum_{j_N=1}^{n_N} \left[\prod_{i_1=1}^{N} \alpha_{j_{i_1}}^{(i_1)} \right] f \left(\xi_1^{(j_1)}, ..., \xi_N^{(j_N)} \right) s_j \left(\xi_1^{(j_1)}, ..., \xi_N^{(j_N)} \right), \\
A_4 &= \sum_{j_1=1}^{n_1} \cdots \sum_{j_N=1}^{n_N} \left[\prod_{i_1=1}^{N} \alpha_{j_{i_1}}^{(i_1)} \right] f \left(\xi_1^{(j_1)}, ..., \xi_N^{(j_N)} \right) \varphi_k \left(\xi_1^{(j_1)}, ..., \xi_N^{(j_N)} \right), \\
A_5 &= \sum_{j_1=1}^{n_1} \cdots \sum_{j_N=1}^{n_N} \left[\prod_{i_1=1}^{N} \alpha_{j_{i_1}}^{(i_1)} \right] s_j \left(\xi_1^{(j_1)}, ..., \xi_N^{(j_N)} \right) \varphi_k \left(\xi_1^{(j_1)}, ..., \xi_N^{(j_N)} \right)
\end{aligned}
$$

$$
0 \leq j,k \leq N \tag{2.50}
$$

Böylece, her bir MYBMG yaklaştıranı için değişik olan γ değerleri elde edilebilir ve bunların kullanımıyla (**2.43**) çizelgesi yerine

$$
\begin{matrix}
h_{0,0}\left(\mathbf{x}, \gamma_{0,0}^{(ek)}\right) & h_{0,1}\left(\mathbf{x}, \gamma_{0,1}^{(ek)}\right) & \cdots & h_{0,N}\left(\mathbf{x}, \gamma_{0,N}^{(ek)}\right) \\
h_{1,0}\left(\mathbf{x}, \gamma_{1,0}^{(ek)}\right) & h_{1,1}\left(\mathbf{x}, \gamma_{1,1}^{(ek)}\right) & \cdots & h_{1,N}\left(\mathbf{x}, \gamma_{1,N}^{(ek)}\right) \\
\vdots & \vdots & \ddots & \vdots \\
h_{N,0}\left(\mathbf{x}, \gamma_{N,0}^{(ek)}\right) & h_{N,1}\left(\mathbf{x}, \gamma_{N,1}^{(ek)}\right) & \cdots & h_{N,N}\left(\mathbf{x}, \gamma_{N,N}^{(ek)}\right)
\end{matrix} \tag{2.51}
$$

olarak verilen “Eniyilenmiş MYBMG Yaklaştıranları Çizelgesi”elde edilir. Kuşkusuz, bu çizelgenin en altta ve en sağda bulunan ögesi kesin sonucu verecektir. Diğer ögelerin ürettiği yaklaştırımların niteliği bu yaklaştırana yaklaşıldıkça azalır ama buna karşın bilgisayım karmaşıklığı artar. Bu nedenle, yaklaştıranlardan altsırasayıları en çok 2 olanlarla yetinmek istenebilir ve bu istem akılcıdır. Yani, çizelgenin sol üst köşedeki (3×3) yapısındaki kesiştirimin en kullanışlı yaklaştıranları içerdiği rahatlıkla söylenebilir.

3. YBMG YÖNTEMİ İLE VERİ BÖLÜNTÜLEME

Sonlu sayıda değişik nokta üzerinde çok boyutlu içdeğerbiçim yürütülmesi gerektiğinden YBMG terimlerinden oluşan çok boyutlu bölgeyi, herhangi bir ek koşul ortaya koymadan, tüm uzaya genişletebiliriz. Böylece, her bir bağımsız değişkenin aralığını $(-\infty,\infty)$ olarak alabiliriz. Burada, içdeğerbiçim uygulanacak $f(x_1,...,x_N)$ işlevinin yapısının analitik olarak verilmediği durumlara odaklanılmaktadır. Bu durumda, bağımsız değişkenler üzerinden tanımlanmış Euclid uzayından sonlu sayıda nokta alınarak bu noktalardaki işlev değerlerinin verildiği varsayılmaktadır. Bu noktalar her bir bağımsız değişkenin örttüğü aralıkların kartezyen çarpımı olarak alınmaktadır. İşlevin değerlerinin ise bir ızgara üzerindeki düğüm noktalarının tümünde verildiği varsayılmakta, her bir boyuttaki noktasal değer kümelerinin tanımları aşağıda sunulmaktadır.

$$\mathscr{D}_j \equiv \left\{ \xi_j^{(k_j)} \right\}_{k_j=1}^{k_j=n_j} = \left\{ \xi_j^{(1)},...,\xi_j^{(n_j)} \right\}, \qquad 1 \le j \le N \tag{3.1}$$

Bunlardan kartezyen çarpımla oluşturulan noktalar kümesi

$$\mathscr{D} \equiv \mathscr{D}_1 \times \mathscr{D}_2 \times \cdots \times \mathscr{D}_N \tag{3.2}$$

eşitliğiyle tanımlamaktadır. Kartezyen çarpım ile oluşturulan $\mathscr{D}$ kümesinin açık anlatımı aşağıdaki gibidir:

$$\mathscr{D} \equiv \left\{ \tau | \tau = (x_1,x_2,...,x_N), x_j \in \mathscr{D}_j, 1 \le j \le N \right\} \tag{3.3}$$

İçdeğerbiçim için oluşturulması gereken yapının $\mathscr{D}$ kümesinde yer alan noktalardaki $f(x_1,...,x_N)$ işlevinin aldığı değerleri içermesi gereklidir. Bu yapı, uygun bir ağırlık seçimiyle elde edilebilir. Bu amaçla gerekli olan eylem, MYBMG'de yaptığımız gibi, çeşitli Dirac Delta işlevlerinin doğrusal birleştirimlerinden oluşan bir ağırlık işlevi tanımlamaktır. Bunun için aşağıdaki ağırlık işlevi seçilebilir.

$$W_j(x_j) \equiv \sum_{k_j=1}^{n_j} \alpha_{k_j}^{(j)} \delta\left(x_j - \xi_j^{(k_j)}\right), \qquad x_j \in \left[a_j, b_j\right], \qquad 1 \le j \le N \tag{3.4}$$

Burada tanımlanan ağırlık işlevinin, tümlev birimlendirim koşulunu (**2.9**) sağladığı aşağıdaki gibi gösterilebilir.

$$\int_{a_j}^{b_j} dx_j \sum_{k_j=1}^{n_j} \alpha_{k_j}^{(j)} \delta\left(x_j - \xi_j^{(k_j)}\right) = \sum_{k_j=1}^{n_j} \alpha_{k_j}^{(j)}, \qquad 1 \le j \le N \tag{3.5}$$

Bu da aşağıdaki ilişkiyi, Dirac Delta işlevlerinin doğrusal birleştirim katsayılarındaki bir koşul olarak, bize verir.

$$\sum_{k_j=1}^{n_j} \alpha_{k_j}^{(j)} = 1, \qquad 1 \le j \le N \tag{3.6}$$

(**2.10**) ile elde edilen YBMG değişmez terimi tanımlayan eşitlikte (**3.4**) kullanılırsa aşağıdaki denklem elde edilir.

$$\begin{aligned} f_0 &= \int_{a_1}^{b_1} dx_1 \sum_{k_1=1}^{n_1} \alpha_{k_1}^{(1)} \delta\left(x_1 - \xi_1^{(k_1)}\right) \cdots \times \\ &\quad \times \int_{a_N}^{b_N} dx_N \sum_{k_N=1}^{n_N} \alpha_{k_N}^{(N)} \delta\left(x_N - \xi_N^{(k_N)}\right) f(x_1, ..., x_N) \end{aligned} \tag{3.7}$$

Dirac Delta işlevinin özelliğini de kullanarak tümlevleme işlemi devreye sokulursa YBMG değişmez terimi için aşağıdaki cebirsel ilişki elde edilir

$$f_0 = \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} \cdots \sum_{k_N=1}^{n_N} \left(\prod_{i_1=1}^{N} \alpha_{k_{i_1}}^{(i_1)} \right) f\left(\xi_1^{(k_1)}, ..., \xi_N^{(k_N)}\right) \tag{3.8}$$

Bu ilişki aşağıdaki yapıda yeniden düzenlenebilir

$$f_0 \equiv \sum_{\tau \in \mathscr{D}} \zeta(\tau) f(\tau) \tag{3.9}$$

Burada

$$\begin{aligned} \tau &= \left(\xi_1^{(k_1)}, ..., \xi_N^{(k_N)}\right), \qquad \zeta(\tau) = \alpha_1^{(k_1)} \cdots \alpha_N^{(k_N)}, \\ &\qquad 1 \le k_j \le n_j, \qquad 1 \le j \le N \end{aligned} \tag{3.10}$$

tanımları geçerlidir.

$f_m(x_m)$ birli YBMG terimini belirlemek için (**2.11**) tanımından yararlanabiliriz.

$$\begin{aligned} f_m\left(\xi_m^{(k_m)}\right) &= \int_{a_1}^{b_1} dx_1 W(x_1) \cdots \int_{a_{m-1}}^{b_{m-1}} dx_{m-1} W(x_{m-1}) \times \\ &\quad \times \int_{a_{m+1}}^{b_{m+1}} dx_{m+1} W(x_{m+1}) \cdots \int_{a_N}^{b_N} dx_N W(x_N) f(x_1, ..., x_N) - f_0 \end{aligned} \tag{3.11}$$

Eşitliğin sağ yanı aşağıdaki biçimde $N-1$ kat tümlev yapılı olarak yeniden yazılabilir.

$$\begin{aligned} f_m\left(\xi_m^{(k_m)}\right) &= \int_{a_1}^{b_1} dx_1 \sum_{k_1=1}^{n_1} \alpha_{k_1}^{(1)} \delta\left(x_1 - \xi_1^{(k_1)}\right) \cdots \times \\ &\times \int_{a_{m-1}}^{b_{m-1}} dx_{m-1} \sum_{k_{m-1}=1}^{n_{m-1}} \alpha_{k_{m-1}}^{(m-1)} \delta\left(x_{m-1} - \xi_{m-1}^{(k_{m-1})}\right) \times \\ &\times \int_{a_{m+1}}^{b_{m+1}} dx_{m+1} \sum_{k_{m+1}=1}^{n_{m+1}} \alpha_{k_{m+1}}^{(m+1)} \delta\left(x_{m+1} - \xi_{m+1}^{(k_{m+1})}\right) \cdots \times \\ &\times \int_{a_N}^{b_N} dx_N \sum_{k_N=1}^{n_N} \alpha_{k_N}^{(N)} \delta\left(x_N - \xi_N^{(k_N)}\right) f(x_1,...,x_N) - f_0 \end{aligned} \quad \textbf{(3.12)}$$

(**3.9**)'da verilen kapalı yapıya benzer olarak yukarıda bulunan anlatımı aşağıdaki biçimde yeniden yazabiliriz.

$$f_m\left(\xi_m^{(k_m)}\right) = \sum_{\tau_m \in \mathscr{D}^{(m)}} \zeta_m(\tau_m) f(\tau_m, \xi_m^{(k_m)}) - \sum_{\tau \in \mathscr{D}} \zeta(\tau) f(\tau) \quad \textbf{(3.13)}$$

Burada

$$\begin{aligned} \mathscr{D}^{(m)} &\equiv \left\{ \tau_m | \tau_m = (x_1,...,x_{m-1},x_{m+1},...,x_N), x_j \in \mathscr{D}_j, 1 \le j \le N, j \ne m \right\}, \\ \tau_m &= \left(\xi_1^{(k_1)},...,\xi_{m-1}^{(k_{m-1})},\xi_{m+1}^{(k_{m+1})},...,\xi_N^{(k_N)}\right), \\ \zeta_m(\tau_m) &= \alpha_1^{(k_1)} \cdots \alpha_{m-1}^{(k_{m-1})} \alpha_{m+1}^{(k_{m+1})} \cdots \alpha_N^{(k_N)}, \\ \xi_m^{(k_m)} &\in \mathscr{D}_m, \qquad 1 \le k_m \le n_m, \qquad 1 \le m \le N \end{aligned} \quad \textbf{(3.14)}$$

tanımları geçerlidir.

Elde ettiğimiz eşitlikle birlikte N sıralı terimlerden oluşan tekli (ing:univariate) çizelge üretilmiş bulunmaktadır. $f_m(x_m)$ için elde edilen m'inci çizelgede n_m $(1 \le m \le N)$ sayıda sıralı terim içerilmektedir.

$f_{m_1 m_2}(x_{m_1}, x_{m_2})$ yapısındaki YBMG ikili terimleri de benzer biçimde elde edilir. (**2.12**) eşitliğinden ve Dirac Delta işlevinin özelliğinden yararlanarak benzer anlatımlar

aşağıdaki gibi elde edilir.

$$
\begin{aligned}
f_{m_1m_2}(\xi_{m_1}^{(k_{m_1})},\xi_{m_2}^{(k_{m_2})}) &= \sum_{\tau_{m_1m_2}\in\mathscr{D}^{(m_1m_2)}} \zeta_{m_1m_2}(\tau_{m_1m_2}) f\left(\tau_{m_1m_2},\xi_{m_1}^{(k_{m_1})},\xi_{m_2}^{(k_{m_2})}\right) \\
&\quad - \sum_{\tau_{m_1}\in\mathscr{D}^{(m_1)}} \zeta_{m_1}(\tau_{m_1}) f\left(\tau_{m_1},\xi_{m_1}^{(k_{m_1})}\right) \\
&\quad - \sum_{\tau_{m_2}\in\mathscr{D}^{(m_2)}} \zeta_{m_2}(\tau_{m_2}) f\left(\tau_{m_2},\xi_{m_2}^{(k_{m_2})}\right) \\
&\quad + \sum_{\tau\in\mathscr{D}} \zeta(\tau) f(\tau) \qquad \textbf{(3.15)}
\end{aligned}
$$

Burada

$$
\begin{aligned}
\mathscr{D}^{(m_1m_2)} &\equiv \left\{\tau_m|\tau_m=(x_1,...,x_{m_1-1},x_{m_1+1},...,x_{m_2-1},x_{m_2+1},...,x_N),\right. \\
&\qquad \left. x_j\in\mathscr{D}_j, 1\le j\le N, j\ne m_1,m_2\right\}, \\
\mathscr{D}^{(m_1)} &\equiv \left\{\tau_{m_1}|\tau_{m_1}=(x_1,\ldots,x_{m_1-1},x_{m_1+1},...,x_N),\right. \\
&\qquad \left. x_j\in\mathscr{D}_j, 1\le j\le N, j\ne m_1\right\}, \\
\mathscr{D}^{(m_2)} &\equiv \left\{\tau_{m_2}|\tau_{m_2}=(x_1,...,x_{m_2-1},x_{m_2+1},...,x_N),\right. \\
&\qquad \left. x_j\in\mathscr{D}_j, 1\le j\le N, j\ne m_2\right\}, \\
\tau_{m_1m_2} &= \left(\xi_1^{(k_1)},...,\xi_{m_1-1}^{(k_{m_1-1})},\xi_{m_1+1}^{(k_{m_1+1})},...,\xi_{m_2-1}^{(k_{m_2-1})},\xi_{m_2+1}^{(k_{m_2+1})},...,\xi_N^{(k_N)}\right), \\
\tau_{m_1} &= \left(\xi_1^{(k_1)},...,\xi_{m_1-1}^{(k_{m_1-1})},\xi_{m_1+1}^{(k_{m_1+1})},...,\xi_N^{(k_N)}\right), \\
\tau_{m_2} &= \left(\xi_1^{(k_1)},...,\xi_{m_2-1}^{(k_{m_2-1})},\xi_{m_2+1}^{(k_{m_2+1})},...,\xi_N^{(k_N)}\right),
\end{aligned}
$$

$$\begin{aligned}
\zeta_{m_1 m_2}(\tau_{m_1 m_2}) &= \alpha_1^{(k_1)} \cdots \alpha_{m_1-1}^{(k_{m_1-1})} \alpha_{m_1+1}^{(k_{m_1+1})} \cdots \alpha_{m_2-1}^{(k_{m_2-1})} \alpha_{m_2+1}^{(k_{m_2+1})} \cdots \alpha_N^{(k_N)}, \\
\zeta_{m_1}(\tau_{m_1}) &= \alpha_1^{(k_1)} \cdots \alpha_{m_1-1}^{(k_{m_1-1})} \alpha_{m_1+1}^{(k_{m_1+1})} \cdots \alpha_N^{(k_N)}, \\
\zeta_{m_2}(\tau_{m_2}) &= \alpha_1^{(k_1)} \cdots \alpha_{m_2-1}^{(k_{m_2-1})} \alpha_{m_2+1}^{(k_{m_2+1})} \cdots \alpha_N^{(k_N)}, \\
\xi_{m_1}^{(k_{m_1})} &\in \mathscr{D}_{m_1}, \quad \xi_{m_2}^{(k_{m_2})} \in \mathscr{D}_{m_2},
\end{aligned}$$

$$1 \le k_{m_1} \le n_{m_1}, \quad 1 \le k_{m_2} \le n_{m_2}, \quad 1 \le m_1, m_2 \le N \qquad \textbf{(3.16)}$$

tanımları geçerlidir. Böylece (**3.9**) ve (**3.13**)'i verilmiş veriler üzerinde kullanarak çok değişkenli işlevin analitik yapısını elde edecek içdeğerbiçim için gerekli birli ve ikili çizelgeler elde edilmiş olmaktadır.

3.0.3 İçdeğerbiçim (ing:Interpolation)

YBMG yardımıyla verilmiş veriyi bölüntülere ayırmakla $f_m(x_m)$ için analitik bir yapı yerine n_m sıralı ikilisinden (ing: ordered pair) oluşmuş bir çizelge saptanmıştır. Bu çizelge, işlev için varsayılan bir yapı altında $f_m(x_m)$ terimini saptamayı olanaklı kılar. Bu da içdeğerbiçim yaklaştırımıdır (ing: interpolation approximation). Bu yolla, tek bir çok değişkenli içdeğerbiçim, N sayıda tek değiskenli içdeğerbiçim kümesine indirgenerek yaklaştırım yapılır. Bu tür bir yaklaştırım için işlev değeri olarak YBMG birli terimleri üzerinde, genel eğilimi izleyip çokterimli yapısı varsayarak, Lagrange İçdeğerbiçimi kullanılabilir.

$$p_m(x_m) = \sum_{k_m=1}^{n_m} L_{k_m}(x_m) f_m\left(\xi_m^{(k_m)}\right), \qquad \xi_m^{(k_m)} \in \mathscr{D}_m, \qquad 1 \le m \le N \qquad \textbf{(3.17)}$$

Buradaki $L_{k_m}(x_m)$'ler Lagrange çokterimli katsayılarıdır. Bu çokterimli aşağıdaki biçimde tanımlanır.

$$L_{k_m}(x_m) \equiv \prod_{\substack{j=1 \\ j \ne k_m}}^{n_m} \frac{\left(x_m - \xi_m^{(j)}\right)}{\left(\xi_m^{(k_m)} - \xi_m^{(j)}\right)}, \qquad \xi_m^{(k_m)} \in \mathscr{D}_m,$$

$$1 \le k_m \le n_m, \quad 1 \le m \le N \qquad \textbf{(3.18)}$$

YMBG terimleri kullanılarak oluşturulan (**3.17**) ve (**3.18**) eşitlikleri kullanılarak çok değişkenli bir $f(x_1,...,x_N)$ işlevine aşağıdaki gibi bir yaklaştırım oluşturulabilir.

$$f(x_1,...,x_N) \approx f_0 + \sum_{m=1}^{N} p_m(x_m) \tag{3.19}$$

$$\begin{aligned} p_{m_1 m_2}(x_{m_1},x_{m_2}) \quad = \quad & \sum_{k_{m_1}=1}^{n_{m_1}} \sum_{k_{m_2}=1}^{n_{m_2}} L_{k_{m_1}}(x_{m_1}) L_{k_{m_2}}(x_{m_2}) f_{m_1 m_2}(\xi_{m_1}^{(k_{m_1})}, \xi_{m_2}^{(k_{m_2})}), \\ & \xi_{m_1}^{(k_{m_1})} \in \mathscr{D}_{m_1}, \quad \xi_{m_2}^{(k_{m_2})} \in \mathscr{D}_{m_2}, \quad 1 \le m_1, m_2 \le N \end{aligned} \tag{3.20}$$

$$f(x_1,...,x_N) \approx f_0 + \sum_{m=1}^{N} p_m(x_m) + \sum_{\substack{m_1,m_2=1 \\ m_1<m_2}}^{N} p_{m_1,m_2}(x_{m_1},x_{m_2}) \tag{3.21}$$

Böylece $f(x_1,...,x_N)$ çokdeğişkenli işlevi için içdeğerbiçim yaklaştırımı elde edilmiş olur.

4. KOŞUTLAŞTIRMA SÜRECİ İÇİN YBMG YÖNTEMİNİN İYİLEŞTİRİLMESİ

Bir önceki bölümde değinilen YBMG türlerinden ister YBMG ister ÇYBMG isterse de MYBMG veri bölüntüleme yöntemi kullanılsın asıl işlem yüküne sahip olan eşitlikler toplamsal YBMG'nin değişmez ve birli terimlerini bulmada kullanılan (**3.9**) ve (**3.13**) eşitlikleridir. Diğer tüm bilinmeyenler bu terimler yoluyla elde edildiğinden ve işlem ederleri düşük olduğundan koşutlaştırma süreci dışında tutulmuştur. Uygulamalarda esas olarak YBMG değişmez ve birli terimler kullanıldığından bu çalışmada esas olarak (**3.9**) ve (**3.13**) eşitlikleri koşutlaştırılacaktır.

Koşutlaştırma işlemine başlamadan önce YBMG değişmez ve birli terimleri hesaplayan (**3.9**) ve (**3.13**) eşitlikleri incelenecek olursa, eşitliklerin yapıları ele alınan problemdeki boyut sayısına göre değişkenlik gösterdigi görülür. YBMG terimlerini bulmada kullanılan algoritmadaki döngü sayısı ele alınan problemdeki boyut sayısına eşittir. Koşutlaştırma işlemine geçmeden önce bu problemi aşmak için ele alınan eşitliklerin matematiksel olarak ne anlattıkları ortaya konulup bunlar üzerinde iyileştirme yapılmalıdır. Bu amaçla ilk olarak bu eşitliklerin matematiksel yapısını ortaya çıkarmada kullanılan kavramlara değinilmelidir.

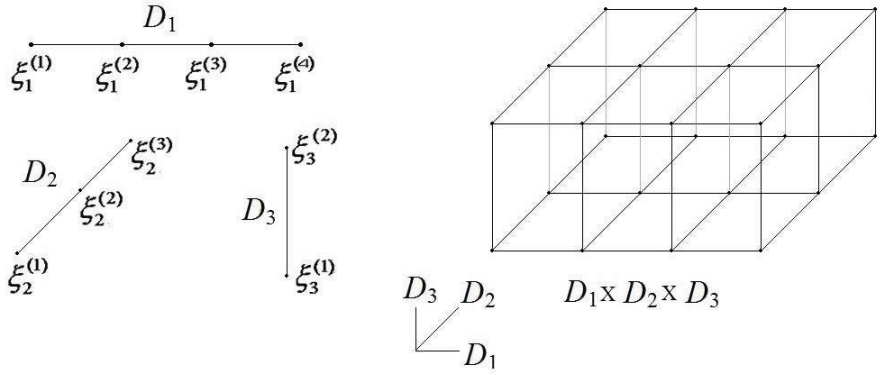
4.1 Kullanılan Matematiksel Kavramlar

(**3.9**) ve (**3.13**) eşitliklerinin matematiksel yapılarını açıklamak için örnek bir noktalar kümesi ele alınmıştır. Örnek noktalar kümesi 3 boyutlu olarak seçilmiştir ve (**3.1**), (**3.2**) ve (**3.3**) tanımlarından yararlanarak

$$\mathscr{D} = \mathscr{D}_1 \times \mathscr{D}_2 \times \mathscr{D}_3 = \left\{\xi_1^{(1)}, \xi_1^{(2)}, \xi_1^{(3)}, \xi_1^{(4)}\right\} \times \left\{\xi_2^{(1)}, \xi_2^{(2)}, \xi_2^{(3)}\right\} \times \left\{\xi_3^{(1)}, \xi_3^{(2)}\right\} \quad \textbf{(4.1)}$$

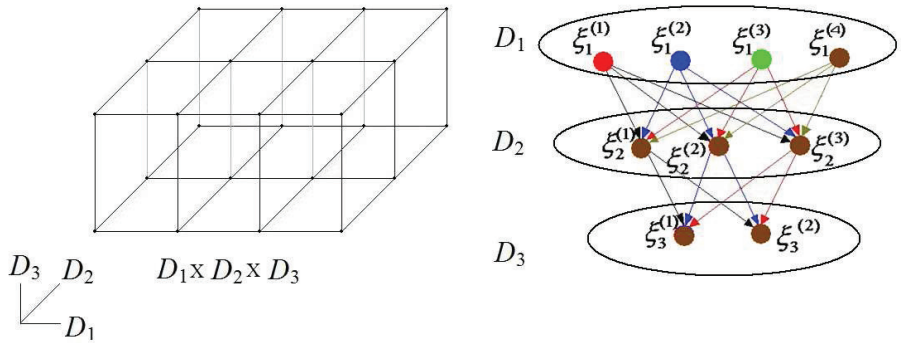
şeklinde gösterilir. Dolayısıyla birinci boyutu 4 düğüm noktasından, ikinci boyutu 3 düğüm noktasından ve üçüncü boyutu 2 düğüm noktasından oluşan bir örnek noktalar kümesi alınmıştır. örnek olarak alınan noktalar kümesi üzerinden çizge kanıtsavının (ing:*graph theory*) 3 kavramından bahsedilecektir. Bu kavramların birincisi Ağ

Kartezyen Çarpım (ing:*Grid Cartesian Product*) kavramıdır[13]. Bu matematiksel kavram yardımıyla örnek olarak aldığımız noktalar kümesi bir ağ kartezyen çarpım olarak aşağıdaki şekilde gösterilebilir.



Şekil 4.1: $G_{4,3,2}$, $\mathscr{D}_1 \times \mathscr{D}_2 \times \mathscr{D}_3$ için Ağ Kartezyen Çarpımı

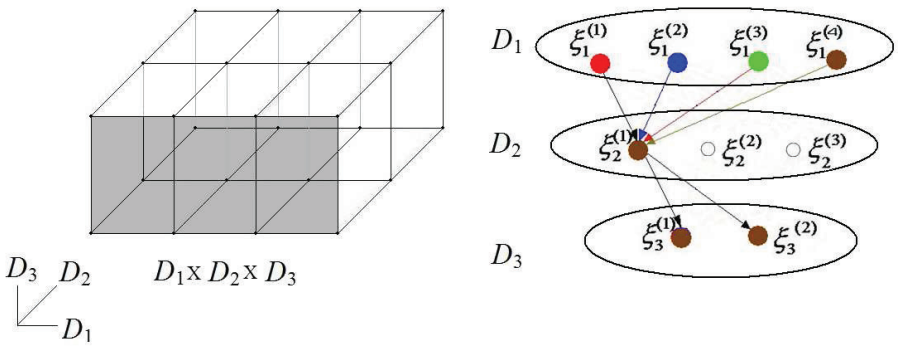
İkinci kavram Tam k-Ayrıştırılmış Çizgedir (ing:*Complete $k-$Partite Graph*)[14]. Bir k-tam ayrıştırılmış çizge k adet ayrı kümeden oluşmaktadır öyle ki küme içerisinde bulunan düğüm noktaları birbiri ile herhangi bir eşleşmeleri yoktur. Buna karşın sıralı halde bulunan bu kümelerdeki düğüm noktaları komşu kümelerdeki düğüm noktaları ile gerçekleşebilecek tüm eşleşmeleri yapmışlardır. Bu çalışmada düğüm noktaları arasında yönlü eşleşmeler (ing:*directed edges*) kullanılmıştır. Böylece herhangi bir ağ kartezyen çarpım bir tam ayrıştırılmış çizge olarak gösterilebilir. Şekil 4.2'de örnek olarak alınan kümenin ağ kartezyen çarpımının 3-Tam Ayrıştırılmış Çizge olarak görülmektedir.



Şekil 4.2: $G_{4,3,2}$ ağ kartezyen çarpımının $K_{4,3,2}$ Yönlü 3-Tam Ayrıştırılmış Çizge olarak anlatımı

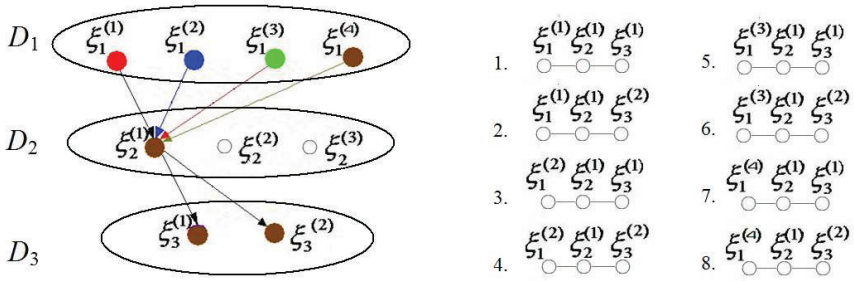
Eğer yukarıda bahsedilen ve düğüm noktaları arasındaki eşleşmelerin tümü mevcut değil ise bu kavram da k-Ayrıştırılmış Çizge (ing:$k-$*Partite Graph*) olarak tanımlanmaktadır [15]. Kartezyen çarpım ile oluşan noktalardan yalnızca ikinci boyutun birinci

düğüm noktasını $\xi_2^{(1)}$ içerenler Şekil 4.3'ün sol yanında işaretlenmiştir. Bunun çizelge gösterimi Şekil 4.3'un sağ yanındaki gibidir. Görüldüğü gibi $\mathscr{D}_2$ boyutunda yalnızca $\xi_2^{(1)}$ düğüm noktası ile yapılan yönlü eşleşmeler gösterilmiştir.



Şekil 4.3: $K_{4,1,2}$, $G_{4,1,2}$'nin Yönlü 3-Ayrıştırılmış Çizgesi

Ele alınacak son matematiksel kavram Yönlü Döngüsüz Çizgedir (ing:*Directed Acyclic Graph*). Adından da anlaşılacağı gibi çizge herhangi bir kesintisiz döngü içermiyor ise DAG olarak sınıflandırılır [16]. Sonlu bir DAG en azından bir kaynak (ing:*source*) ve bir bitim (ing:*sink*) noktasına sahip olmalıdır. Kaynak kendi üzerine herhangi bir yönlü eşleşmenin olmadığı, bitim ise kendinden herhangi bir noktaya herhangi bir yönlü eşleşmenin olmadığı noktalardır. Şekil 4.4'ün sol yanında dikkat edilecek olursa bu çizgenin 4 adet kaynak ve 2 adet bitim noktasına sahip olduğu görülür. Ayrıca bu çizge kesintisiz bir döngü sağlayacak yönlü eşleşmelere sahip değildir. Dolayısıyla bu bir DAG'dır. Ek olarak çizge eşit uzunlukta yönlü erişim yollarına (ing:*path*) sahiptir. Her bir erişim yolu $(\xi_1^{(1)}, \xi_2^{(1)}, \xi_3^{(1)})$ gibi örnek noktalar kümesi elemanlarını içeren sıralı üçlülerden oluşur. Her DAG, içerdiği noktalar arasında doğrusal bir topolojik sıralamaya sahiptir öyle ki $i < j$ durumunda v_i noktası v_j noktasından önce gelir. Fakat örnek olarak alınan noktalar kümesinde her bir nokta alt ve üst olmak üzere iki sırasayıya sahiptir. Bu nedenle ek bir sıralamaya daha gereksinim duyulmaktadır; $k < t$ olmak üzere $v_i^{(k)}$ noktası $v_i^{(t)}$ noktasından önce gelmektedir. Bu sıralama yardımıyla Şekil 4.4'deki DAG sıralı bir noktalar dizisi olarak gösterilebilir. Şekil 4.4'ün sağ yanında $G_{4,1,2}$'ın sıralı bir dizi olarak görülmektedir. Şekil 4.4'de görüldüğü gibi birinci erişim yolu hem alt hem de üst sırasayıları önceliğe sahip olduğundan $\xi_1^{(1)}$ ile başlar. $\xi_1^{(1)}$ in ikinci boyutta eşleştiği tek nokta $\xi_2^{(1)}$ dir. $\xi_1^{(1)}$'in de üçüncü boyutta eşleştiği iki nokta bulunmaktadır, $\xi_3^{(1)}$ ve $\xi_3^{(2)}$. Bu iki noktanın alt sırasayıları aynı küme içerisinde oldukları için aynıdır. Bu



Şekil 4.4: $G_{4,1,2}$'ın Yönlü Ayrıştırılmış Çizgesi ve bu çizgeyi anlatan sıralı üçlüler kümesi

nedenle sıralama için üst sırasayılara bakılır. Böylece $\xi_1^{(1)}$ ile başlayan $(\xi_1^{(1)}, \xi_2^{(1)}, \xi_3^{(1)})$, $(\xi_1^{(1)}, \xi_2^{(1)}, \xi_3^{(2)})$ sıralı üçlülerinin oluşturduğu düğüm noktaları sırasıyla dizinin ilk iki öğeleri olurlar. İkinci boyutta eşleme yapan tek nokta $\xi_2^{(1)}$ olduğundan yine birinci boyuttaki üst sırasayı nedeniyle ikinci öncelige sahip $\xi_1^{(2)}$ ile erişim yolu oluşturulmaya devam eder ve Şekil 4.4'de sağ yanda görülen biçimde sıralanırlar.

4.2 YBMG Değişmez ve Birli Terimlerin Veri Kullanım Şemaları

Yukarıdaki bölümde anlatılan matematiksel kavramlardan yola çıkarak herhangi sayıda boyuta ve boyut düğüm noktalarına sahip herhangi bir kartezyen çarpımın oluşturduğu çok boyutlu noktalar kümesinin tamamı ya da bir kesimi yönlü paylaştırılmış çizge olarak edilebilir. Buradan kartezyen çarpım kümesi öğeleri eşsiz bir sırada dizi gösterilebilir. Bu gösterimden yola çıkarak 3 boyutlu olarak aldığımız örnek noktalar kümesinin kartezyen çarpımı (**4.1**) sıralı dizi olarak eşsiz bir şekilde aşağıdaki gibi gösterilebilir.

$$
\begin{aligned}
\mathscr{D} &= \mathscr{D}_1 \times \mathscr{D}_2 \times \mathscr{D}_3 \\
&= \{(\xi_1^{(1)}, \xi_2^{(1)}, \xi_3^{(1)}), (\xi_1^{(1)}, \xi_2^{(1)}, \xi_3^{(2)}), (\xi_1^{(1)}, \xi_2^{(2)}, \xi_3^{(1)}), (\xi_1^{(1)}, \xi_2^{(2)}, \xi_3^{(2)}), \\
&\quad (\xi_1^{(1)}, \xi_2^{(3)}, \xi_3^{(1)}), (\xi_1^{(1)}, \xi_2^{(3)}, \xi_3^{(2)}), (\xi_1^{(2)}, \xi_2^{(1)}, \xi_3^{(1)}), (\xi_1^{(2)}, \xi_2^{(1)}, \xi_3^{(2)}), \\
&\quad (\xi_1^{(2)}, \xi_2^{(2)}, \xi_3^{(1)}), (\xi_1^{(2)}, \xi_2^{(2)}, \xi_3^{(2)}), (\xi_1^{(2)}, \xi_2^{(3)}, \xi_3^{(1)}), (\xi_1^{(2)}, \xi_2^{(3)}, \xi_3^{(2)}), \\
&\quad (\xi_1^{(3)}, \xi_2^{(1)}, \xi_3^{(1)}), (\xi_1^{(3)}, \xi_2^{(1)}, \xi_3^{(2)}), (\xi_1^{(3)}, \xi_2^{(2)}, \xi_3^{(1)}), (\xi_1^{(3)}, \xi_2^{(2)}, \xi_3^{(2)}), \\
&\quad (\xi_1^{(3)}, \xi_2^{(3)}, \xi_3^{(1)}), (\xi_1^{(3)}, \xi_2^{(3)}, \xi_3^{(2)}), (\xi_1^{(4)}, \xi_2^{(1)}, \xi_3^{(1)}), (\xi_1^{(4)}, \xi_2^{(1)}, \xi_3^{(2)}), \\
&\quad (\xi_1^{(4)}, \xi_2^{(2)}, \xi_3^{(1)}), (\xi_1^{(4)}, \xi_2^{(2)}, \xi_3^{(2)}), (\xi_1^{(4)}, \xi_2^{(3)}, \xi_3^{(1)}), (\xi_1^{(4)}, \xi_2^{(3)}, \xi_3^{(2)})\} \quad \textbf{(4.2)}
\end{aligned}
$$

Kartezyen çarpım sonucunda oluşan bu dizide 24 öğe bulunmaktadır. Bu öğelerden her biri, kartezyen çarpım sonucunda oluşan çok boyutlu noktaların koordinatlarına karşılık gelmektedir. Bu aşamadan sonra anlatım kolaylığı açısından çok boyutlu $f(x_1,x_2,x_3)$ işlevinin bu noktalardaki değerleri sırasıyla 1'den 24'e kadar giden sayılar şeklinde gösterilecektir. Bu gösterim biçimini kullanarak örnek uzaydaki noktalarda sayısal değerleri bilinen $f(x_1,x_2,x_3)$ çok boyutlu işlevinin YBMG terimleri hesaplansın.

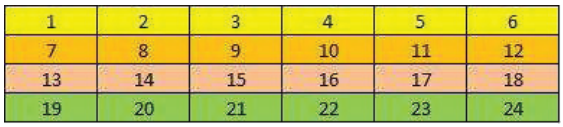
YBMG değişmez terimleri hesaplamakta kullanılan (**3.9**) eşitliği incelendiğinde bu gösterimin, $f(x_1,x_2,x_3)$ işlevinin (**4.2**)'de gösterilmiş 24 noktadaki değerlerinin ağırlıklar altında toplamına eşit olduğu görülür. (**3.6**) eşitliğinden ağırlıklar her bir boyut için aynı olduğundan burada işlev değerlerinin toplamı önem kazanmaktadır. Şekil 4.5'de YBMG değişmez terimleri f_0 hesaplanırken $f(x_1,x_2,x_3)$'in tüm noktalardaki sayısal değerlerinin kullandığı görülmektedir.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24

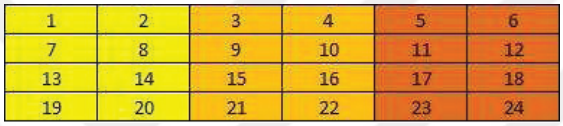
Şekil 4.5: YBMG değişmez terimi olan f_0'ı hesaplamak için kullanılan veri kullanım çizemi

Esas problem (**3.13**) eşitliği ile bulunan birli YBMG terimler hesaplanırken kullandıkları işlev değerlerinin çizemini çıkarmaktır. Fonksiyon değerleri kullanım çizemleri, seçilen boyut ve boyutlardaki düğüm noktası sayısına göre değişmektedir. YBMG terimlerinin matematiksel yapısını anlatmak için örnek olarak ikinci boyutun birinci düğüm noktasının YBMG terimi olan $f_2(\xi_2^{(1)})$ hesaplansın. $f_2(\xi_2^{(1)})$ terimi hesaplanırken kullanılan noktalar, kartezyen çarpım kümesinde ikinci boyutta yalnızca $\xi_2^{(1)}$ düğüm noktasını içeren noktalardır. Bu noktalarin seçimi Şekil 4.3'de sağ yanda bulunan ayrıştırılmış çizgede görülmektedir. Ayrıştırılmış çizgede $f_2(\xi_2^{(1)})$ hesaplanırken yalnızca $\xi_2^{(1)}$ içeren noktalardaki işlev değerleri kullanılır. Bu değerler de Şekil 4.6'daki sıralı işlev değer dizisinde 7. noktadan 12. noktalara kadar olan değerleri içeren dizi parçasına denk gelir. Diğer YBMG birli terimler bulunurken de strateji aynıdır; hangi boyutun düğüm noktasının YBMG terim değeri bulunmak isteniyorsa yalnızca o düğüm noktasını içeren noktalardaki işlev değeri kullanılır. Şekil 4.6, Şekil 4.7 ve Şekil 4.8 de birinci, ikinci ve üçüncü boyutlardaki düğüm noktalarının YBMG terim değerleri hesaplanırken kullanılan işlev değer kullanım çizemleri görülmektedir. Buradan çıkarılacak önemli bir sonuç da aynı boyutta bulunan farklı düğüm noktalarının

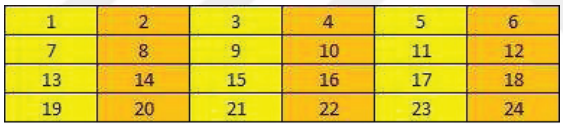
YBMG terim değerleri hesaplanırken farklı işlev değerleri kullanılır, birinin kullandığı veriyi diğeri kullanmaz. Şekil 4.6, Şekil 4.7 ve Şekil 4.8'de sırasiyla 1. boyuttaki, 2. boyuttaki ve 3. boyuttaki düğüm noktalarının YBMG terim değerleri hesaplanırken kullanılan işlev değerlerinin çizemi görülmektedir. Şekillerde görülen her bir renk grubu, farklı düğüm noktalarının hesabında kullanılan işlev değerlerini göstermektedir.



Şekil 4.6: Birinci boyuttaki düğüm noktalarına ait YBMG birli terimleri olan $f_1(\xi_1^{(n_1)})$'leri hesaplamak için kullanılan veri kullanım çizemi



Şekil 4.7: İkinci boyuttaki düğüm noktalarına ait YBMG birli terimleri olan $f_2(\xi_2^{(n_2)})$'leri hesaplamak için kullanılan veri kullanım çizemi



Şekil 4.8: Üçüncü boyuttaki düğüm noktalarına ait YBMG birli terimleri olan $f_3(\xi_3^{(n_3)})$'leri hesaplamak için kullanılan veri kullanım çizemi

Buraya kadar gelinen süreç özetlenirse, (**3.9**) ve (**3.13**) eşitlikleri ile hesaplanan değişmez ve birli YBMG terimlerinin matematiksel yapıları çeşitli matematiksel kavramlar yardımıyla belirlenmiştir. Bu yapıya göre terimler hesaplanırken aslında belirli noktalardaki işlev değerlerinin toplamı kullanılmaktadır. Her bir YBMG terimi, veri dizisinin çeşitli parçalarındaki değerleri toplayarak hesaplanır. Bu veri setinde hangi değerleri kullandıklarını gösteren çizemler yukarıda belirtilmiştir. Fakat bu çizemler bu haliyle yeterli değildir. Herhangi bir boyuttaki herhangi bir düğüm noktasının YBMG değeri hesaplanırken eldeki veri dizisinde hangi verilerin kullandığı özyineli bir ilişki ile verilmelidir. Yukarıdaki şekillerde gösterilen ve belirli noktalardaki işlev değerlerini içeren 24 öğeli veri dizisi göz önüne alınsın. Her bir YBMG terimi hesaplanırken bu dizi içerisinde belirli öğelerin değerlerinin kullanıldığı belirtilmiştir. Özyineli ilişkilerin çıkarımına 1. boyuta ait düğüm noktalarının YBMG hesabı ile başlanabilir. İlk boyutta 5 düğüm noktası olduğundan her bir düğüm noktası için bir

YBMG terimi bulunacaktır. Şekil 4.6'da her bir rengin farklı bir YBMG teriminin hesabında kullanılan veriler olduğu belirtilmişti. Şekil 4.6'da ilk satırda yer alan değerler $f_1(\xi_1^{(1)})$ hesabında kullanılan verilerdir. Bu veriler hep beraber bir veri dizi parçası olarak düşünülebilir. Şekil 4.6'da görüldüğü gibi her bir YBMG değeri hesaplanırken 1 tane dizi parçası kullanılmaktadır. Bu bilgi daha sonra kullanılacaktır. Her bir dizi parçası 6 adet veri içermektedir. Bu değer aslında bundan sonraki boyutlarda bulunan düğüm noktalarının sayılarının çarpımıdır. 2. ve 3. boyutlarda sırasıyla 3 ve 2 adet düğüm noktası vardı. Bunların çarpımı ile YBMG terimleri hesaplanan boyutta kullanılan dizi parçalarının uzunluğu elde edilmiş olur. Dizi parçasının baslangıcının hangi öğeden başlanacağının belirlenmesi daha kolaydır. Boyutun hangi düğüm noktasının YBMG değeri hesaplanıyorsa boyutun dizi parçalarının uzunluğu ile düğüm noktası sırasayısının çarpımı YBMG teriminin kullanacağı dizi parçasının başlangıcını belirler. Bu şekilde 1. boyuta ait düğüm noktalarının YBMG değerleri hesaplanır. 2. ve sonraki boyutlarda YBMG terimleri hesabı başka özyineli ilişkileri gerektirir. Bu ilişkilerin çıkarımı için 2. boyuta ait YBMG terimleri hesaplanırken kullanılan veri kullanım çizemi Şekil 4.7 gözönüne alınmalıdır. Dikkat edilecek olursa bu boyuttaki YBMG terimleri hesaplanırken tek bir dizi parçası kullanılmamaktadır. Bu boyutta bulunan 3 düğüm noktası için ayrı ayrı 4 adet veri parçası kullanılmıştır. Ayrıca bu veri parçaları arasında sabit bir uzaklık vardır. Dolayısıyla YBMG terimleri hesaplanırken dizi parçaları sayısı ve bu parçalar arasındaki uzaklığın özyineli olarak bulunması gerekmektedir. Dizi parçaları sayısı 1. boyut için 1 olarak alınmıştı. Herhangi bir boyuttaki bir düğüm noktasının YBMG değeri hesabında kullanılacak dizi sayısı, ondan önceki boyutlardaki düğüm noktaları sayılarının çarpımıdır. 2. boyut için dizi parçaları sayısı 1. boyuttaki düğüm noktası sayısına eşittir ve 4'dür. Bu sonuç Şekil 4.7'den de görülmektedir. Aynı zamanda 3. boyutta hesaplanan YBMG değerleri için de aynı hesaplama yapılacak olursa, 1. ve 2. boyuttaki düğüm noktaları sayıları çarpımından 12 sayısı elde edilir. Şekil 4.8'den de görüldüğü gibi her bir düğüm noktalasının YBMG değeri hesabında 12 adet veri parçası kullanılmıştır. Son olarak herhangi bir düğüm noktasının YBMG değeri hesabında kullanılan veri parçaları arasındaki uzaklığın özyineli ilişkisi elde edilmelidir. Herhangi bir boyuttaki düğüm noktalasa ait YBMG değeri hesabında kullanılan dizi parçaları arasındaki uzaklık, dizinin toplam uzunluğunun ondan önceki boyutlardaki düğüm noktaları sayıları çarpımından elde edilen sayıya bölümüyle bulunur. Bölen değer 1. boyut için özel olarak 1 olarak

tanımlanmıştır. 2. boyuttaki herhangi bir düğüm noktası için bu hesap kolayca yapılabilir. Dizideki toplam öğe sayısı 24 idi. 1. boyutta 4 düğüm noktası vardı. 24/4 den iki dizi parçası baslangıçları arası uzaklık 6 olarak bulunur. Şekil 4.7 incelenecek olursa $f_2(\xi_2^{(1)})$ hesap edilirken kullanılan ilk dizi parçası 1. öğeden başlamaktadır. Aynı YBMG teriminin hesabında kullanılan ikinci dizi parçasının başlangıcı 7. öğedir. Dolayısıyla aralarındaki uzaklık 6'dır. Benzer hesap 3. boyut için de yapılabilir. Bu boyutta bir düğüm noktasının YBMG değeri hesabında kullanılan dizi parçaları başlangıç uzaklıkları $24/(4 \cdot 3) = 2$ olarak bulunur. Gerçekten de Şekil 4.8 incelenirse kurulan özyineli ilişkinin doğru olduğu görülür.

Şimdiye kadar elde edilen YBMG terimlerin veri kullanım çizemleri ve çıkarılan özyineli ilişkiler gözönüne alındığında (**3.9**) ve (**3.13**) eşitlikleri kolayca hesaplanabilir. Her iki eşitlik de $O(N^2)$ hesaplama karmaşıklığına sahip olacak şekilde daha verimli bir biçimde yeniden yazılabilir. Her iki eşitlik de iç içe 2 döngü içerir. Dıştaki döngü toplanması gereken veri parçalarının başlangıç noktalarını denetlerken iç döngü veri parçaları içindeki öğelerin toplanmasından sorumludur. Bu şekilde tüm YBMG terimleri elde edilmiş olur.

5. YBMG Yönteminin MPI ile Koşutlaştırılması

Önceki bölümde HDMR terimlerini hesaplamakta kullanılan (**3.9**) ve (**3.13**) eşitlikleri koşutlaştırma amacıyla iyileştirilmişti. Bu aşamadan sonra koşutlaştırmada üzerinde durulacak süreç, hesaplamalarda kullanılan ve belli bir özyineli ilişki içeren dizi parçaları içindeki sayısal değerlerin başarımlı bir şekilde toplanmasını sağlamaktır. Bu süreç anlatılmadan önce verinin hesaplama düğümlerine (ing:node) paylaştırma stratejisi anlatılmalıdır.

Şekil 4.6, Şekil 4.7 ve Şekil 4.8 incelenecek olursa veri dizisinin neden 6'şarlı parçalar halinde satır satır bölündüğü açıklanmıştı. Dikkat edilecek olursa dizi parçaları bu biçimde bölünürse her bir parçanın YBMG terimleri hesabında veri kullanım çizemi tıpatıp aynıdır. Dolayısıyla her bir parçaya düşen iş yükü eşittir. Bu bölüm şekli rastlantısal değildir. Herhangi bir veri dizisi 1. boyuttaki düğüm noktası sayısı kadar eşit parçaya ayrıldığında her zaman bu sonuç elde edilir. Elde edilen veri parçalarının YBMG terimleri hesabında kullandığı veri çizemi ve iş yükü aynı olacaktır. Veri dizisini bu şekilde bölmek eşsiz değildir. Sıranın korunması koşulu ile boyutlardaki düğüm noktası sayılarının çarpımından elde edilen sayılar da veri dizisini aynı özelliklerde böler. Örnek veri dizisinde 1. boyuttaki düğüm noktası sayısı 4 idi. 2. boyuttaki düğüm noktası sayısı olan 3 ile çarpıldığında 12 sayısı elde edilir. Veri dizisi 12 eşit satıra bölünürse aynı şekilde her bir parçanın YBMG terimleri hesabında veri kullanım çizemi ve iş yükü aynı olur.

Yukarıda anlatıldığı şekilde veri dizisi eşit parçalara ayrılır ve hesaplama düğümlerine olabildiğince eşit sayida paylaştırılır. Böylece her bir hesaplama düğümlerine düşen veri hacmi ve iş yükü eşit olur. Dolayısıyla load balancing elde edilmiş olur. Bu tür paylaştırımda dikkat edilecek olursa herhangi bir hesaplama düğümü tek başına herhangi bir YBMG terimini hesaplayamaz. Tek bir YBMG teriminin hesabı tüm hesaplama düğümlerine bağlı olarak gerçekleşir. Bu durumda her bir hesaplama düğümünün hesapladığı değeri bir araya getirecek bir yapıya gereksinim vardır. Bu gereksinime karşılık veren *MPI_Reduce*() işlevidir[17]. Bu işlev her bir hesaplama düğümündeki sonucu alır ve seçilen matematiksel işlemi uygulayarak

Sunucu Hesaplama Düğümü:	**İstemci Hesaplama Düğümleri:**
1. Kendi üzerine düşen veri satır sayısını hesapla **2.** Her bir boyut için dizi parçası sayısı, genişliği ve dizi parçalarının birbirlerinden olan uzaklıkları hesapla **3.** MPI I/O ile kendine düşen veriyi çek **4.** Çektiği verilerden HDMR değişmez ve birli terimleri hesapla **5.** MPI_Reduce() kullanarak istemci hesaplama düğümlerinde hesaplanan HDMR terimleri değerlerini toplayarak birleştir.	**1.** Kendi üzerine düşen veri satır sayısını hesapla **2.** Her bir boyut için dizi parçası sayısı, genişliği ve dizi parçalarının birbirlerinden olan uzaklıkları hesapla **3.** MPI I/O ile kendine düşen veriyi çek **4.** Çektiği verilerden HDMR değişmez ve birli terimleri hesapla **5.** MPI_Reduce() ile hesaplanan HDMR terimleri değerlerini sunucu hesaplama düğümüne yolla.

Çizelge 5.1: Yöntemi MPI kitaplığı ile koşutlaştırma süreci

istenilen hesaplama düğümüne sonucu aktarır. Burada seçilen matematiksel işlem toplama işlemidir. Her hesaplama düğümü, belirli bir YBMG teriminin kendisine düşen toplamını hesaplar. *MPI_Reduce*(*MPI_SUM*) ile sonuçlar toplanır ve sunucu hesaplama düğümüne aktarılır. Böylece YBMG terimlerinin değerini veren sonuç elde edilmiş olur.

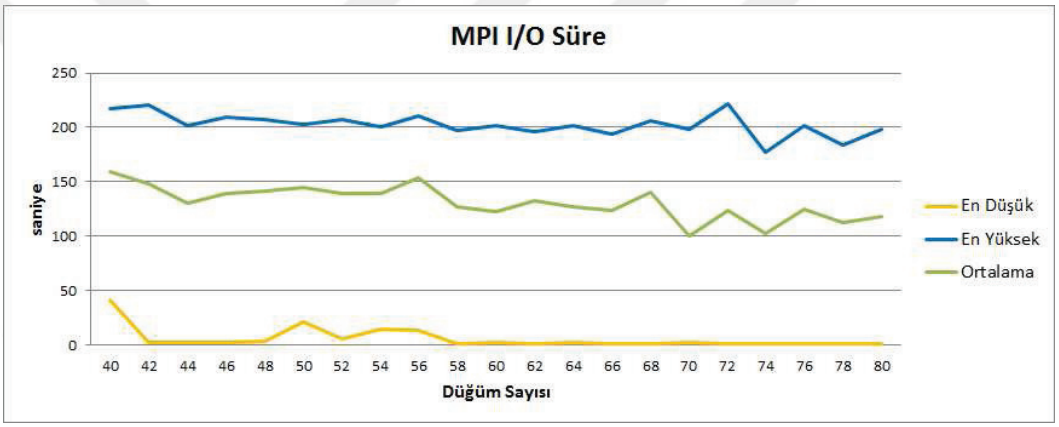
Çizelge 5.1'de adım adım açıklanan koşutlaştırma sürecinde sunucu ve istemci hesaplama düğümlerindeki iş adımları görülmektedir. İlk olarak tüm hesaplama düğümleri kendi üzerlerine düşen dizi satır sayılarını hesaplar. Daha sonra her bir boyut için hesaplanması gereken dizi parçaları uzunlukları, sayıları ve birbirlerinden olan uzaklıklar hesaplanır. Sonrasında MPI Parallel I/O kullanılarak veriler hesaplama düğümlerine aynı anda aktarılır. Sonrasında her bir hesaplama düğümü kendisine düşen veri parçasından YBMG terimlerini hesaplar. Bu işlem tamamlandıktan sonra *MPI_Reduce*(*sum*) sayesinde tüm sonuçlar toplanarak nihai sonuç sunucu hesaplama düğümüne aktarılır. Değişmez ve birli YBMG terimlerin hesapları bu şekilde tamamlanmış olur.

5.1 Oluşturulan MPI Algoritmasının Başarım Analizi

Yapılan başarım analizi her biri Intel(R) Xeon(TM) 2.66 GHz CPU'ya sahip, dağıtık bellek mimarisine sahip ve her bir düğümde 8 core ve 16GB belleğe sahip InfiniBand 20 Gbps bağlantılı bir sistem üzerinde yapılmıştır. Başarımı analiz etmede her bir test adımı en az 5 kez yapılmıştır. Bu testler sonucunda düğümlerde harcanan süre *En Düşük*, *En Yüksek* ve *Ortalama* olarak sınıflandırılmıştır. *En Düşük*, düğüm noktalarında

ilgili hesaplama için harcanan en kısa süreyi, *En Yüksek* en uzun süreyi, *Ortalama* da tüm düğüm noktalarında harcanan sürenin ortalamasını ifade etmektedir. Analiz için kullanılan tüm çizimlerde sarı renk *En Düşük*, mavi renk *En Yüksek* ve yeşil renk te *Ortalama* sınıflandırmaları için kullanılmıştır. Başarım analizi için yaklaşık 70GB'lık hacme sahip 6 boyutlu bir veri kullanılmıştır. Analizin düğüm başlangıç sayısı, verinin hacmi gözönüne alınarak bellek aşımı hatasına karşın 40 olarak belirlenmiştir.

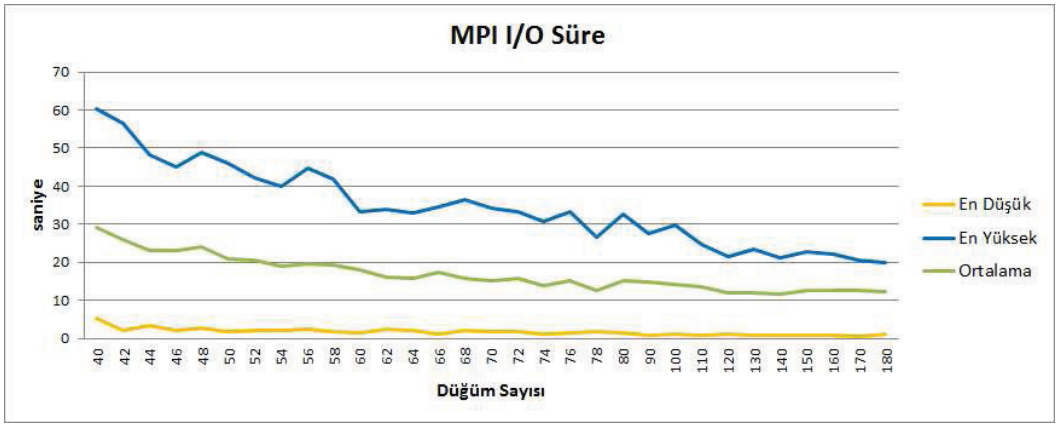
Başarım analizi için öncelikli olarak MPI I/O incelenecektir. Bu kısımda yaşanabilecek gecikmeler tüm algoritmanın başarımını etkileyecek düzeyde olacaktır. Analiz yapılan dosyanın MPI I/O ile düğümler tarafından okunma süreleri aşağıdaki Şekil 5.1'de gösterilmektedir.



Şekil 5.1: MPI algoritmasının MPI I/O süreleri

Şekilden görülebileceği gibi düğümlerde harcanan *En Düşük* ve *En Yüksek* süreler arasında çok büyük farklar vardır. Ek olarak, düğüm noktası artmasına rağmen dosyanın tüm düğüm noktaları tarafından okunma süresi azalmamakta ve sabit kalmaktadır. Bunun nedeni kullanılan sistemde varsayılan dosya parçalama sayısının (ing:*stripe*) 1 olarak ayarlanmış olmasıdır. Dosya tek bir parça üzerinde olduğundan dosya sunucusu ancak bir adet istemciye cevap verebilmektedir. Dolayısıyla düğüm noktası ne kadar artarsa artsın MPI I/O süresi sabit kalmaktadır ve okuma 200 saniye civarında seri olarak gerçekleşmektedir. Dosya, test edilen sistem tarafından izin verilen en yüksek sayı olan 9 parçaya bölünmüştür. Bu parçalama sonrası düğümlerde geçen MPI I/O süresi Şekil 5.2'deki gibidir.

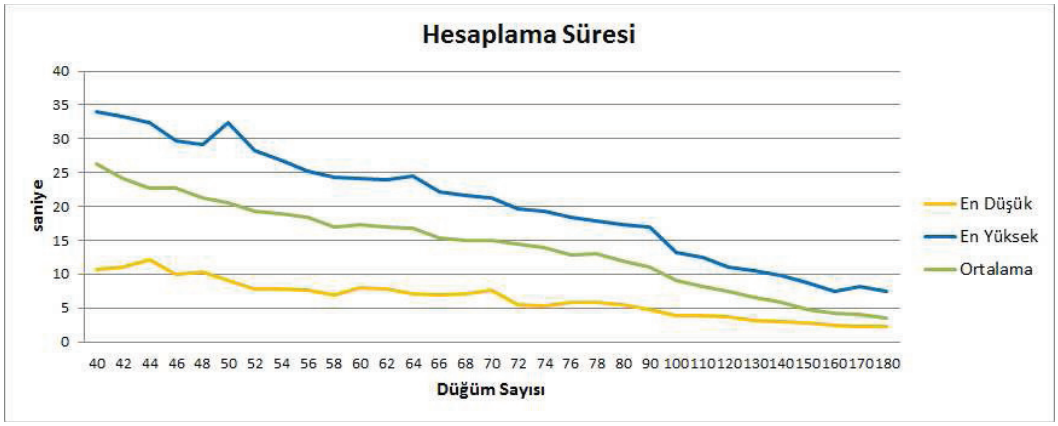
Şekil 5.2'den görüldüğü gibi *En Düşük* ile *En Yüksek* süreler arasında fark olmasına karşın önceki süreler kadar yüksek değildir. Asıl önemli ilerleme, düğüm sayıları arttıkça MPI I/O için geçen süre artık gerektiği gibi azalmaktadır. MPI I/O sürecinde



Şekil 5.2: MPI algoritmasının iyileştirme sonrası MPI I/O süreleri

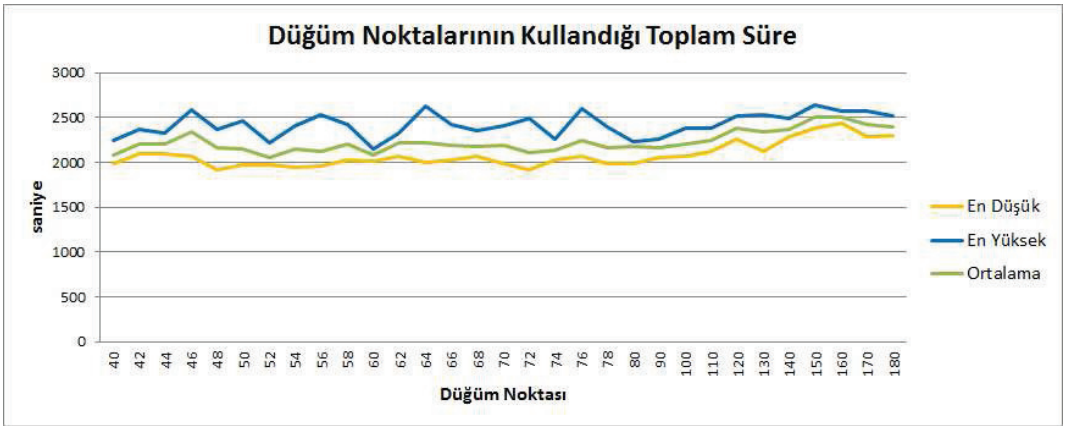
geçen zaman, dosya fiziksel olarak daha hızlı okunamayacak seviyeye inene kadar sürekli bir azalma göstermiştir. Dolayısıyla MPI I/O'dan kaynaklanabilecek başarım düşüşleri bu şekilde engellenmiştir.

MPI I/O süresi ile ilgili sorun aşıldıktan sonra değişen düğüm sayısına göre Şekil 5.3'de hesaplama sırasında geçen zamanı, Şekil 5.4'de tüm düğüm noktalarının her bir testte MPI algoritması sırasında harcadığı toplam zamanı, Şekil 5.5'de algoritmanın WallClock zamanlarını ve Şekil 5.6'de Parallel Efficiency çizimi görülmektedir.

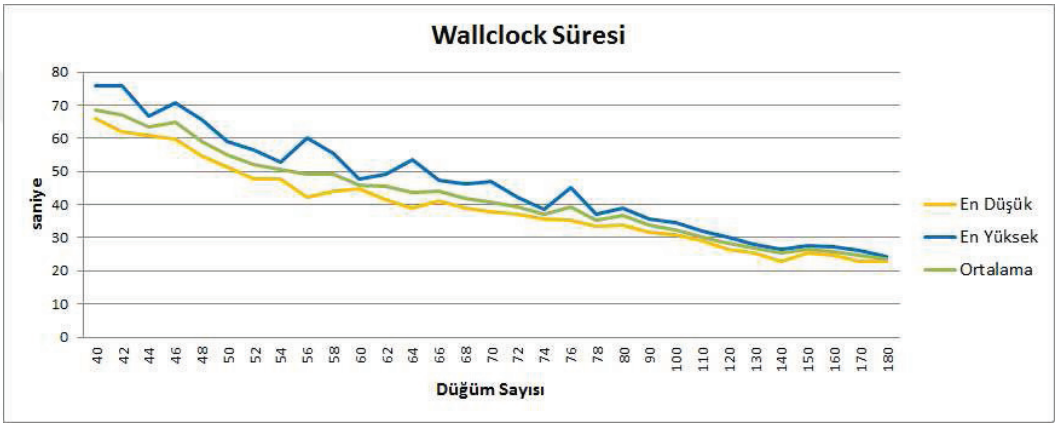


Şekil 5.3: MPI algoritması sırasında hesaplama için harcanan zaman

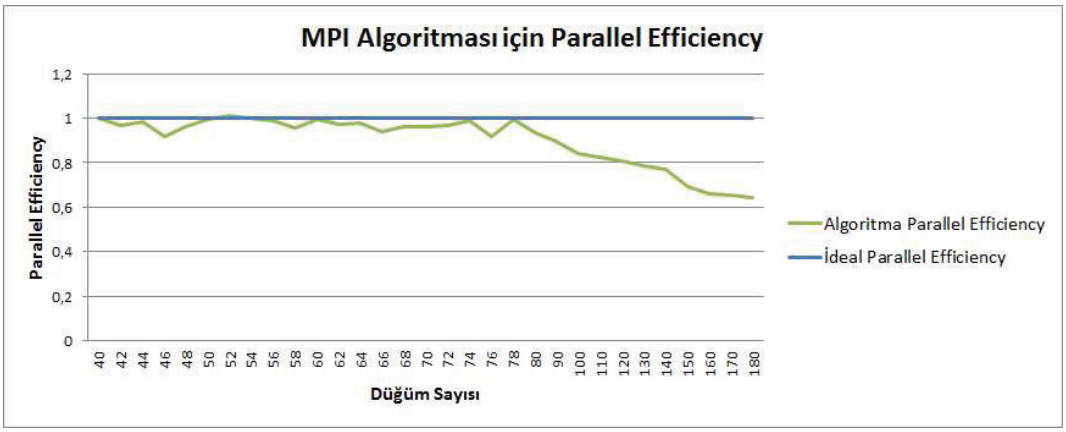
Şekil 5.3'de görüldüğü gibi düğüm noktası arttıkça hesaplama zamanı sürekli bir azalma davranışı göstermektedir. *En Düşük* ile *En Yüksek* süreler arasında fark olmasına karşın bu durum MPI I/O süresindeki durumdan farklıdır. MPI I/O'da dağılım tekdüze (ing:*uniform*) dağılım göstermektedir. Hesaplama süresi ölçümünde kullanılan testlerde ise dağılım normal dağılımdır ve tepe noktası ortalamayı işaret etmektedir. Dolayısıyla hesaplama süreleri her bir test içerisinde tutarlıdır. Düğüm noktalarında harcanan toplam zamanı gösteren Şekil 5.4 incelenirse düğüm sayısının atmasına



Şekil 5.4: Tüm düğümleri MPI algoritması sırasında harcadığı toplam zaman



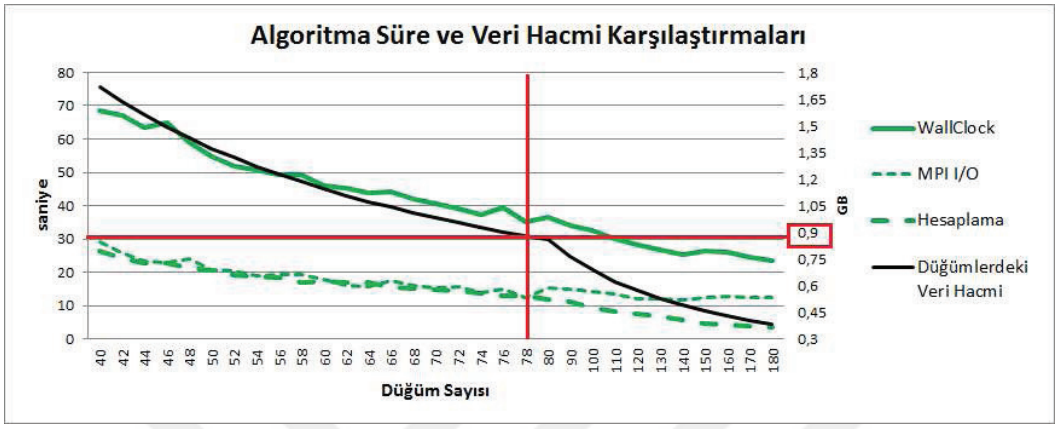
Şekil 5.5: MPI algoritmasının WallClock zamanı



Şekil 5.6: MPI algoritmasının Parallel Efficiency çizimi

rağmen başarımını koruyarak yatay konumunu koruduğu görülmektedir. Şekil 5.5'deki algoritmanın baştan sona işleyiş zamanı olan WallClock süresi ve Şekil 5.6'deki başarım çizimi incelendiğinde de benzer sonuçlar görülmektedir. Algoritma, girdi dosyası fiziksel olarak daha hızlı okunamayacak seviyeye inene kadar başarımını kaybetmeden sürdürmüş, bu sınır aşıldıktan sonra başarım yitimi başlamıştır. Dolayısıyla MPI algoritması yüksek başarıma sahiptir.

Şekil 5.7, MPI algoritmasının WallClock, MPI I/O ve Hesaplama süreleri ile birlikte her bir düğüme düşen veri hacmini göstermektedir. Sürelerin indeksi sol tarafta saniye cinsinden, veri hacminin indeksi de sağ tarafta GB cinsinden gösterilmiştir. Kullanılan tüm süreler ortalamalar üzerinden verilmiştir.



Şekil 5.7: MPI algoritmasının süre ve her bir düğüm noktasına düşen veri hacmi karşılaştırması

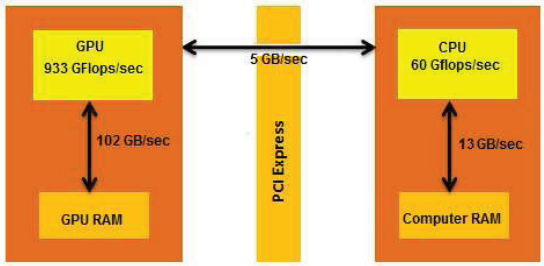
Şekilden görüldüğü gibi algoritma wallclock süresi üzerinde MPI I/O ve hesaplama süreleri aynı ağırlıkta etki etmektedir. Bu etki düğüm başına düşen veri hacmi 0.9GB'a kadar düşene de değişmemektedir. Bu noktadan sonra düğüm satısının artması fiziksel olarak girdi dosyasının daha hızlı aktarımını sağlamadığından MPI I/O süresi sabit kalmış, buna karşın düğüm sayısı arttıkça hesaplama süresi beklendiği gibi azalmaya devam etmiştir. Fakat MPI I/O süresinden dolayı algortimanın başarımı bu noktadan sonra azalmaya başlamıştır. Buradan çıkarılacak sonuç bu sistemde en uygun düğüm sayısı, her bir düğüme 0.9GB'dan daha az veri hacmi düşmeyecek şekilde bir seçim yapılmasıdır.

6. YBMG Yönteminin CUDA ile Koşutlaştırılması

Son birkaç yılda "watt başına başarım" gücü yüksek olan GPU işlemcileri kullanan CUDA kitaplığı oldukça popüler bir hale gelmiştir. Bilimsel yazında diğer koşutlaştırma kitaplıklarına göre oldukça yeni olan CUDA'nın bu kadar başarımlı çalışmasının arkasında yatan etmenler donanımdan yazılıma kadar bu bölümde açıklanacaktır. Daha sonra CUDA kitaplığını kullanarak YBMG yönteminin koşutlaştırma adımları gösterilecektir.

6.0.1 GPU Donanımının Temel Özellikleri:

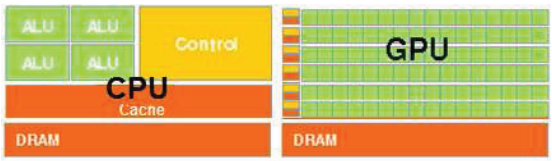
CUDA kitaplığına değinmeden önce bu kitaplığın kullandığı GPU işlemcisi ve bağlı olduğu donanım özelliklerine değinilecektir. GPU işlemciler ekran kartı üzerinde bulunur. Bu nedenden dolayı ekran kartı ile yönetici makinanın (ing:*host machine*) ile ilişkisi irdelenmelidir. Şekil 6.1'de sol yanda bir ekran kartını temsili, sağ yanda ise ekran kartının bağlı olduğu ve CPU işlemci kullanan yönetici makinanın temsili görülmektedir.



Şekil 6.1: CPU ve GPU sistemlerin veriyolu başarım yapılmalıdırlıkları

Ekran kartı ile yönetici makina arasındaki veri transferi PCI express veri yolu ile sağlanır. Bu veriyolu saniyede 5GB aktarım hızına sahiptir. Şekil 6.1'de her iki sistemi kabaca karşılaştıracak olursak günümüz CPU'ların 60 Gflops/sn. başarıma ulaştığı görülmektedir. Buna karşın test için kullanılan Nvidia Tesla C1060 kartı ile 933 Gflops/sn. başarım elde edilebilmektedir. Hem GPU hem de CPU'nun verileri işleyebilmesi için makinanın belleğinden verileri çekmeleri gerekmektedir. Bu veri akışı CPU'da 13GB/sn. hıza sahipken GPU'da bu hız 102GB/sn. olmaktadır. Temsili

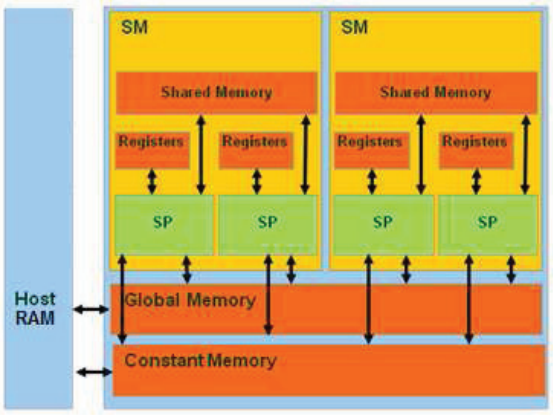
şekillerden başarım farkının nasıl oluştuğu temel olarak görülmektedir. GPU'nun CPU'ya göre hesaplama başarımının bu kadar yüksek olmasının farkını görmek için yonga (ing:*chipset*) mimarisi arasındaki farklara bakmak gerekmektedir. Bu farklar Şekil 6.2'de görülmektedir.



Şekil 6.2: CPU ve GPU'nun yonga mimarileri farkları

CPU'nun en önemli görevi, işletim sistemi ile donanım arasında köprü oluşturan programları çalıştırmaktır. Doğal olarak bu iş yüküne cevap vermek için sınırlı ALU (Arithmetic Logic Unit) sayısına karşın geniş ve oldukça hızlı bir yonga üzeri hafıza (ing:*onchip memory*) kullanır. Gereksinim duyduğunda bellek adı verilen ve büyük olmasına karşın başarımı düşük sistem belleğini kullanır.

GPU ise kullanım amacına uygun bir biçimde, CPU'nun görevleri dışında kalan görüntüleme işlerinin gereksinim duyduğu yüksek hesaplama yükünü karşılamak için üretilmiştir. Dolayısıyla bu yüksek hesaplama maliyetini karşılamak için çok sayıda ALU biriminden yararlanır. GPU'daki her bir ALU birimine SP (Streaming Processor) denilmektedir. GPU çok sayıda ALU ve daha sınırlı büyüklükte GPU yonga üzeri hafıza sahiptir. Gereksinim duyduğunda ekran kartın üzerinde bulunan bellekleri de kullanır. Bu bellekler aynı zamanda host belleği ile veri alışverişi yapabildiği kısmıdır. GPU bileşenlerinin ekran kartı üzerinde bulunan hafıza türlerini kullanım biçimi Şekil 6.3'de incelenebilir.



Şekil 6.3: GPU'nun hafıza kullanım Şeması

GPU üzerinde bulunan her bir SP genel hafızaya (ing: *Global Memory*) erişip yazma yetkisine sahiptir. Bu hafıza türü, ekran kartı üzerindeki en kapasiteli hafızadır. Fakat kaşelenemediği (ing:*caching*) için erişim hızı diğer hafızalara göre oldukça düşüktür. Sabit hafıza (ing: *constant Memory*) kaşelenebilir, fakat SP'ler sabit hafıza içindeki veriyi sadece okuyabilir, yazamaz. Ayrıca kapasitesi de çok daha küçüktür. SP'ler belirli sayıda gruplar halinde SM (Streaming Multiprocessor) içinde bulunurlar. Her bir SM içindeki SP'ler kendi bulunduklari SM üzerinde bulunan ve paylaşılan hafıza (ing: *Shared Memory*) adı verilen ve GPU üzerinde bulunan hafıza türünü kullanırlar. Bu hafıza türü kaşelenebilmektedir. Aynı SM içinde bulunan SP'ler bu hafıza türü aracılığı ile veri paylaşımı yapabilirler. Ayrıca SP'ler bu hafıza üzerinde veri okuma ve yazma işlemleri yapabilirler. Fakat bu hafızanın kapasitesi genel bellek yanında KB'lar seviyesindedir. En küçük kapasite ve en hızlı işleme sahip hafıza türü ise her bir SP'ye özel olarak atanmış yazmaçtır (ing:*register*). Bu hafıza türüne yazılabilecek değişken sayısı sınırlıdır ve her bir yazmaç kendi SP'si tarafından kullanılabilir.

6.1 CUDA Programlama Yapısı

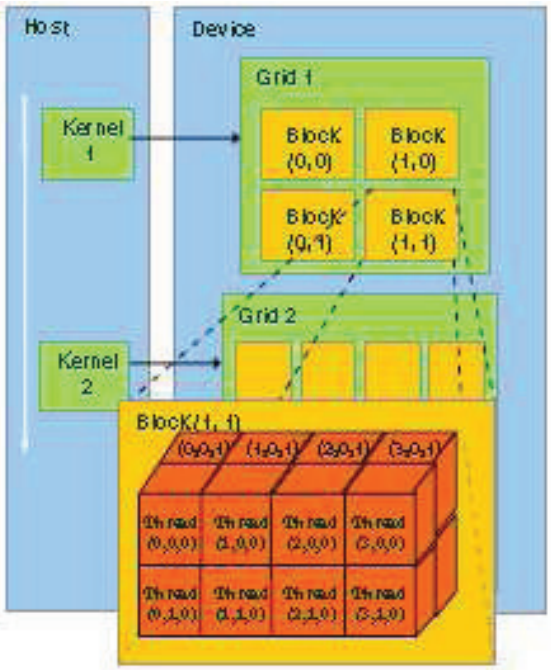
GPU temel olarak SIMD (Single Instruction Multiple Data) yapısına sahip ve işyükünü iş parçacıklarına (ing:*thread*) bölerek koşutlaştırmaktadır. İşyükünün başarımlı bir biçimde dağıtılabilmesi için kullanılan programlama yapısının bilinmesi zorunludur.

NVIDIA C Derleyici (ing: NVIDIA C Compiler) kısaca NVCC ikiye ayrılabilir. Sistem üzerinde çalışan C/Fortran kodudur ve sistemin C/Fortran derleyicisi tarafından derlenir. Ekran kartında GPU üzerinde çalışan kısım ise çekirdek (ing:*kernel*) adı verilen koşut fonksiyonlarından oluşur. Çekirdekler verinin koşutlaştırılabilmesi için iş parçacıkları adı verilen ve belli kurallarla oluşturulabilinen yapılar üretir. Bu aşamada çekirdekten başlayarak iş parçacıklarına doğru giden yapı incelenmelidir.

6.1.1 CUDA Programlamada İş Parçacıkları Yapısı

Şekil 6.4'de CUDA iş parçacıkları düzenleme yapısı gösterilmektedir.

Çalıştırılan bir CUDA programı içinde birden çok çekirdek olabilir, fakat sırayla çalışabilir, aynı anda birden çok çekirdek çalıştırılamaz. Çekirdekler de kendi içinde işyükünün iş parçacıklarına bölünebilmesi için ızgara (ing:*grid*) adı verilen bir yapı



Şekil 6.4: CUDA İş parçacığı düzenleme yapısı

kullanırlar. Benzer bir biçimde her bir çekirdek yalnızca bir adet ızgara yapısı içerebilirler. Izgara, 2 boyutlu bir yapıya sahiptir ve ızgaranın her bir boyutu en fazla 65,535 adet blok (ing:*block*) adı verilen yapıları barındırabilirler. Şekil 6.4'de Izgara1 4 adet bloğa sahiptir ve her bir boyutta 2 adet iş parçacığı vardır. Bloklar ise 3 boyutlu bir yapıya sahiptir. G80 GPU işlemci için ibr blok toplam olarak en fazla 512 adet iş parçacığı içerebilir. Aynı işlemci için her bir blok iki boyutun her birinde maksimum 512 adet iş parçacığı içerebilir, üçüncü boyutta ise bu sayı 64 ile sınırlıdır. Bir ızgaradaki her bloğun boyut uzunlukları aynı olmalıdır. Şekil 6.4'de ele alınan Block(1,1) içinde 16 adet iş parçacığı vardır ve bloklar (4,2,2) biçiminde düzenlenmiştir.

Örnek olarak 1000x1000 boyutunda iki matrisin çarpımı ele alınsın. Burada hesaplanacak matrisin eleman sayısı 1.000.000 olacaktır. Hesaplanacak her bir eleman için 1 adet iş parçacığı yaratılacağından bu problem için toplam 1.000.000 adet iş parçacığı yaratılacaktır. Daha önceden değinildiği gibi bu sayıda iş parçacığı oluşturmak, yapısı itibariyle GPU için oldukça düşük bir maliyettir. Bu problem için blok yapısı (16,16,1) olacak şekilde oluşturulabilir. (32,32,1) blok yapısı geçersiz bir yapıdır, çünkü bloktaki eleman sayısı toplamı 512 sınırını aşarak 1024 olmaktadır. Bloğun ilk iki boyut uzunluğu 16 olduğundan 1000/16 = 62,5 olur. Bu nedenle ızgaradaki blok yapısının (63,63) şeklinde düzenlenmesi gerekmektedir.

6.1.2 İş Parçacığı Yapısının Donanımsal Kısıtları

G80 işlemcisine ait her bir SM'de 768 iş parçacığı aynı anda çalışabilmektedir. Bu da G80'nin 12.000'den çok iş parçacığını aynı anda çalıştırabildiğini gösterir. Intel tabanlı CPU'larda bu rakam her bir çekirdek için en yüksek sayı 8'dir. Üstelik iş parçacığı yaratma işleminin GPU açısından maliyeti ihmal edilecek kadar azdır. Tüm bu farklılıklar bir araya geldiğinde GPU'ların başarımı CPU'ya göre bazı uygulamalarda 200 kat çok olabilmektedir.

Şekil 6.3'deki bellek yapısı gözönüne alınarak donanım ile iş parçacığı arasındaki özellikler şu şekilde gösterilebilir:

Her iş parçacığı, çalıştığı SM içindeki yazmaç belleğe yazıp erişebilir, diğer SM'lerinkine erişemez.

Her blok, çalıştığı SM içindeki paylaşılan belleğe yazıp erişebilir, diğer SM'lerinkine erişemez.

Her ızgara, genel belleğe yazıp erişebilir, sabit belleğe sadece erişebilir.

Koşullardan da anlaşılacağı gibi her bir blok tek bir SM içinde çalıştırılabilir. G80 için aynı anda her bir SM'de sadece 8 blok çalışır. Toplamda 128 blok aynı anda G80 işlemcisinde çalışabilir.

6.1.3 İş Parçacığı Çalıştırım Sırası

Bir SM, çalıştırması gereken iş parçacıklarını 32'lik birimlere ayırır. Her bir birime WARP adı verilir. Aslında bu tip bir ayırma CUDA programlamasında yoktur, bu GPU işlemcisinin kendi mimarisinden kaynaklanmaktadır. Fakat programlama açısından önemlidir. Eğer bir SM içerisindeki herhangi bir WARP bir sonraki WARP verisine gereksinim duyarsa, o ana kadar çalışmış WARP bekler ve diğer WARP gereksinim duyulan veriye kadar çalışır. Bu da gecikmeye ve başarım yitimine neden olur.

İş parçacığı düzenlemesinde diğer önemli konu da işleme alınacak iş parçacıklarının sırasıdır. Örnek olarak 8x8'lik bir blok gözönüne alınsın. Toplamda bu blok 2 WARP'tan oluşacaktır. İlk WARP $T_{0,0}$'dan başlar $T_{3,7}$' a kadar devam eder, son WARP ise $T_{4,0}$'dan başlar $T_{7,7}$' ye kadar devam eder. 3 boyutlu 4x8x2 blokta ise ilk WARP $T_{0,0,0}$'dan başlar $T_{3,7,0}$' a kadar devam eder, son WARP da $T_{0,0,1}$'dan başlar $T_{3,7,1}$' a kadar devam eder.

İş parçacıklarının işlem sırası özellikle verinin genel bellekten paylaşılan ya da sabit belleğe alınması sürecinde etkilidir.

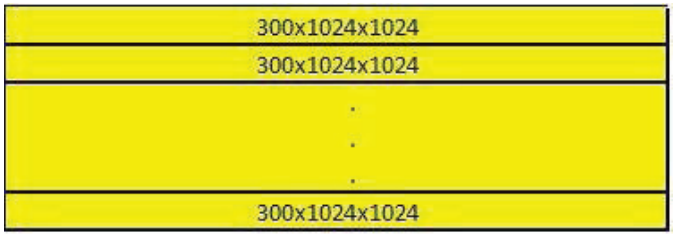
6.1.4 YBMG Terimleri Hesabının CUDA ile Koşutlaştırılması

Daha önceden değinildiği gibi YBMG, hesaplama maliyeti verinin boyut ve bileşen sayısı ile orantılı olarak artan bir yöntemdir. Gerçek bir hesaplama maliyeti yüksek hacimli verilerde ortaya çıkmaktadır. Buna karşın günümüzde GPU işlemcilerin kullanabildiği genel hafiza en fazla 6GB'dır. Bu nedenle yüksek hacimli problemleri GPU ile çözmek, verinin ekran kartının genel hafizaya parça parça gönderilmesi ile mümkündür. Bu parçalama öyle bir şekilde yapılmalıdır ki CUDA programlamaya uygun olan SIMD koşutlaştırma türünde olsun. Özetlemek gerekirse, veriler parça parça ekran kartının genel hafizasına uygun parçalar halinde gönderilmeli, öyle ki her bir SP aynı işlemi farklı verileri kullanarak hesaplama yapabilsin.

Bu durumu daha iyi açıklamak için yeni bir örnek noktalar kümesi alınacaktır. Bu kümede 1. boyuta ait bileşen sayısı 300, 2. ve 3. boyutlara ait bileşen sayısı 1024, 4. boyuta ait bileşen sayısı 300 alınmıştır. Boyutlardaki bileşen sayıları rastlantısal alınmamıştır. MPI algoritmasında kullanılan veri bölüntüleme stratejisi burada da kullanılacaktır. Bölüntüleme sonrasında çıkan her bir satır GPU genel hafizasına aktarılabilecek sığadadır. 1024 adet bileşen, işyüklerini 32'lik parçalar halinde çalıştıran CUDA programlama yapısı için uygun bir sayıdır. 300 adet bileşen ise CUDA işyükü paylaşımı açısından çok uygun bir sayı değildir. Eğer tek bir veri satırı için YBMG bileşenleri hesabı CUDA ile yapilabilirse, diger satırlar için de aynı algoritma çalıştırılıp nihai sonuçlar elde edilir. Bu nedenle tek bir veri satırı için yapılan YBMG terimleri hesabı üzerinden algoritma oluşturulacaktır. Algoritma oluşturulurken örnek kümenin 2. boyutuna ait bileşenlerin YBMG terimleri hesabından başlanacaktır. İkinci boyuta ait YBMG terimlerinin hesabında kullanılan tüm toplamlar aslında hem YBMG değişmez hem de YBMG birli terimleri hesabında kullanılan toplamlardır. Dolayısıyla bu terimlerin hesabı pratik bir şekilde yapılabilir.

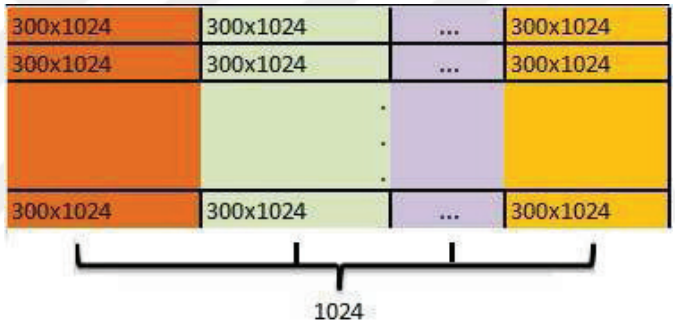
6.1.4.1 2. Boyuta Ait YBMG Terimleri Hesabının CUDA ile Koşutlaştırılması

Programlamanın nasıl yapıldığına geçmeden önce YBMG değişmez, 1. ve 2. boyutlardaki YBMG terimleri hesabında kullanılan veri çizemleri gösterilmelidir.



Şekil 6.5: YBMG değişmez terimi f_0'ın veri kullanım çizemi

Şekil 6.6: 1. boyuta ait düğüm noktalarının YBMG birli terimlerinin veri kullanım çizemi



Şekil 6.7: 2. boyuta ait düğüm noktalarının YBMG birli terimlerinin veri kullanım çizemi

Önceden değinildiği gibi, Şekil 6.7'deki 2. boyuta ait bileşenlerin YBMG terimleri elde etmede kullanılan toplamlar ile Şekil 6.5 ve Şekil 6.6'da gösterilen YBMG değişmez ve 1. boyuta ait YBMG birli terimleri hesaplanabilir. 2. boyuttaki bileşen sayısı 1024 tanedir. Dolayısıyla bu boyutta hesaplanacak 1024 adet YBMG terimi vardır. İlk satırdaki verilerin GPU genel hafizaya aktarıldığında ilk problem ortaya çıkar. İlk satırda 300x1024x1024 = 314.572.800 eleman vardır. Önceki bilgilerimizden tek bir boyutta GPU'nun en fazla 65.535 adet bloğa izin verdiğini bilmekteyiz. Her bir blokta 512 iş parçacığı kullanılsa bile 65.535x512 = 33.553.920 işparçacığı tek bir boyutta kullanılabilmektedir. Bu sorunu ortadan kaldırmak için veri iki boyutlu hale getirilebilir. Fakat bu durumda da hesaplanması gereken ikinci bir indeks ve hesaplama maliyeti ortaya çıkmaktadır. Bu tür problemlerde *CUDAReduction* [**?**] algoritmasında kullanılan metod önem taşımaktadır. Algoritmanın CUDA çekirdek kodu aşağıdaki gibidir.

```
__genel__ void reduce6(int *g_idata, int *g_odata, unsigned int n)
{
   extern __shared__ int sdata[];
   unsigned int tid = threadIdx.x;
   unsigned int i = blockIdx.x*(blockSize*2) + tid;
   unsigned int gridSize = blockSize*2*gridDim.x;
   sdata[tid] = 0;
   while (i < n) { sdata[tid] += g_idata[i] + g_idata[i+blockSize]; i += gridSize; }
   __syncthreads();

   // do reduction in shared mem
```

```
    if (blockSize >= 512) { if (tid < 256) { sdata[tid] += sdata[tid + 256]; } __syncthreads(); }
    if (blockSize >= 256) { if (tid < 128) { sdata[tid] += sdata[tid + 128]; } __syncthreads(); }
    if (blockSize >= 128) { if (tid <  64) { sdata[tid] += sdata[tid +  64]; } __syncthreads(); }

    #ifndef __DEVICE_EMULATION__
    if (tid < 32)
    #endif
    {
        if (blockSize >=  64) { sdata[tid] += sdata[tid + 32]; EMUSYNC; }
        if (blockSize >=  32) { sdata[tid] += sdata[tid + 16]; EMUSYNC; }
        if (blockSize >=  16) { sdata[tid] += sdata[tid +  8]; EMUSYNC; }
        if (blockSize >=   8) { sdata[tid] += sdata[tid +  4]; EMUSYNC; }
        if (blockSize >=   4) { sdata[tid] += sdata[tid +  2]; EMUSYNC; }
        if (blockSize >=   2) { sdata[tid] += sdata[tid +  1]; EMUSYNC; }
    }

    // write result for this block to genel mem
    if (tid == 0) odata[blockIdx.x] = sdata[0];
}
```

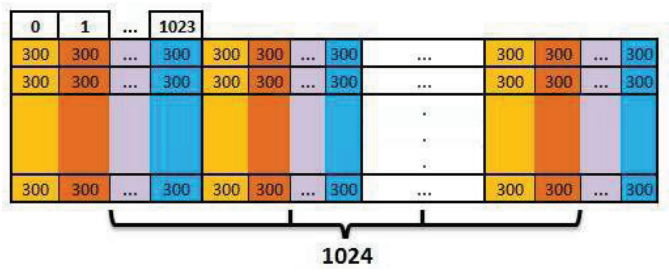
Algoritmada uygulanan ilk iş, tek boyutlu bir ağ oluşturmak ve ağ içerisindeki elemanları toplamaktır. Bu toplam bittikten sonra ağ kendi uzunluğu kadar sağa kaydırılır ve kaydırılan ağ içinde kalan elemanlar toplama eklenir. Veri dizisi sonuna kadar işlem tekrarlanarak tüm elemanlar toplanır. Uygulanan algoritma etkin olmasına karşın YBMG yöntemi için bu haliyle uygun değildir. *Reduction* algoritması tüm veri dizisini toplamaktadır. Oysa yöntem için veri parçalarının ayrı ayrı toplanması gerekmektedir. Burada *Reduction* algoritması şu şekilde iyileştirilmiştir: Ağ içinde hesaplanacak YBMG terimleri kadar blok sayısı içeren bir ağ oluşturulur. Her bir blok tek bir YBMG teriminin hesaplanmasından sorumlu olacağından 1024 adet blok yaratılmalıdır. Buradaki kritik nokta, blokların sadece sorumlu oldukları veri parçaları içerisindeki elemanların toplamasını sağlamasıdır. *Reduction* algoritmasından farklı olarak bu sefer ağ değil ağ içinde bloklar kaydırılmaktadır. Bloklar kendilerine düşen veri dizisi bitimine kadar kaydırılarak içinde bulunan elemanlar toplanır. Bu amaçla tanımlı *gridDim*, *blockIdx* ve *threadIdx* değişkenleri kullanarak hesaplamalar yapılmalıdır. İlk olarak veri satırındaki her bir YBMG terimi hesaplanmasında kullanılacak veri parçası uzunluğunun saptanması gereklidir. Bu hesaplama $VirtualBlockSize = n/gridDim.x$ ile belirlenir. Burada n, veri satırının uzunluğunu temsil etmektedir. Örneğimizde $VirtualBlockSize = n/gridDim.x$
$= 300x1024x1024/1024 = 300x1024$ olarak hesaplanır. Bu da her bir YBMG terimini hesaplamada kullanılacak toplamlardaki eleman sayısına eşittir. $tid = threadIdx.x$ atamasını paylaşılan belleğin indeksini kontrol için, $i = blockIdx.x * VirtualBlockSize + tid$ atamasını ise verinin genel hafizadaki indeks kontrolünde kullanılacaktır. İndeks ötelemelerinin 300x1024 eleman içinde sınırlı kalması için $finish = (blockIdx.x + 1) * VirtualBlockSize$ değişkeni tanımlanmıştır. Block içinde kalan verilerin toplama işlemi

için *CUDA Parallel Reduction* algoritması kullanılmıştır. Böylece her bir *block* aynı anda çalışarak toplama işlemlerini tamamlar. Elde edilen sonuçlar her bir veri satırı için toplanıp ağırlıkları ile çarpıldığında 2. boyuta ait YBMG terimleri elde edilmiş olur. Elde edilen CUDA çekirdeği kodu aşağıdaki gibidir:

```
__genel__ void SumArray(float *data, float *odata, unsigned int n)
{
   unsigned int VirtualBlockSize = n/gridDim.x;
   unsigned int tid = threadIdx.x;
   unsigned int i = blockIdx.x * VirtualBlockSize + tid;
   unsigned int finish = (blockIdx.x + 1) * VirtualBlockSize;
   unsigned int step = blockDim.x;
   unsigned int blockSize = blockDim.x;

   __shared__ float sdata[256];

   sdata[tid] = 0;

   while(i<finish)
   {
      sdata[tid] += data[i];
      i += step;
   }
    __syncthreads();


   // do reduction in shared mem
   if (blockSize >= 512) { if (tid < 256) { sdata[tid] += sdata[tid + 256]; } __syncthreads(); }
   if (blockSize >= 256) { if (tid < 128) { sdata[tid] += sdata[tid + 128]; } __syncthreads(); }
   if (blockSize >= 128) { if (tid <  64) { sdata[tid] += sdata[tid +  64]; } __syncthreads(); }

   #ifndef __DEVICE_EMULATION__
   if (tid < 32)
   #endif
   {
      if (blockSize >=  64) { sdata[tid] += sdata[tid + 32]; EMUSYNC; }
      if (blockSize >=  32) { sdata[tid] += sdata[tid + 16]; EMUSYNC; }
      if (blockSize >=  16) { sdata[tid] += sdata[tid +  8]; EMUSYNC; }
      if (blockSize >=   8) { sdata[tid] += sdata[tid +  4]; EMUSYNC; }
      if (blockSize >=   4) { sdata[tid] += sdata[tid +  2]; EMUSYNC; }
      if (blockSize >=   2) { sdata[tid] += sdata[tid +  1]; EMUSYNC; }
   }
   // write result for this block to genel mem
   if (tid == 0) odata[blockIdx.x] = sdata[0];

}
```

6.1.4.2 3. Boyuta Ait YBMG Terimleri Hesabının CUDA ile Koşutlaştırılması

Üçüncü boyuta ait düğüm noktalarının YBMG terimleri hesabı için kullanılan veri kullanımı Şekil 6.8'de görülmektedir.



Şekil 6.8: 3. boyuta ait düğüm noktalarının YBMG birli terimlerin veri kullanım çizemi

Üçüncü boyuta ait bileşenlerin CUDA programlaması, son boyuta kadar tüm boyutların bileşenlerinin GPU programlaması ile aynıdır. Aralarındaki farklar elemanları toplanan dizi parçaları, toplanması gereken dizi parçaları arasındaki uzaklık ve dizi parçalarının sayısıdır. Dolayısıyla burada açıklanacak yöntem, ikinci ve sonuncu boyutlar arasındaki tüm boyutların YBMG terimleri hesabında kullanılabilir.

3. boyutta da 1024 adet bileşen olduğundan 1024 adet YBMG terim değeri hesaplanmalıdır. Şekil 6.8'de görüleceği gibi ilk 300x1024 veri için kullanılan çizim, veri dizisi sonuna kadar 1024 kez tekrarlanmaktadır. Dolayısıyla burada YBMG terimlerini hesaplamak için uygulanacak yöntem, 2. boyuttaki YBMG terimlerini bulurken kullanılan yöntemin 1024 kez tekrarlanmasından ibarettir. 2. boyutta YBMG terimleri ağ içindeki blokların kaydırılması ile bulunmaktaydı. Bu kez hem ağ hem de blok kaydırma işlemi uygulanacaktır. Şimdiki sorun blok içindeki 300 adet toplamın iş parçacıklarına nasıl paylaştırılacağıdır. 300 eleman tabi ki CUDA blok yapısının izin verdigi sınırlar içerisindedir, fakat burada genel bir algoritma oluşturulmaktadır. Bu nedenle ardışık olarak toplanacak 300 elemanın blok sınırları içerisine sığmadığı varsayılır. Toplanması gereken eleman sayısı yüksek başarım için tavsiye edilen 256 iş parçacığı sayısının altında ve 300'e tam bölünebilen bir sayı seçilmelidir. Bu seçime her bir blokta 150 iş parçacığı sayısı uygundur. Farklı örneklerde ortaya çıkacak ardışık toplamların iş parçacıklarına paylaşımı bu yolla yapılır. Bu örnek için 1024 blok ve bloklarda 150 iç parçacığı seçilir. İlk blok düşünüldüğünde 300 elemanın ardışık olarak toplanması gerekmektedir. Fakat blok uzunluğu 150 seçilmişti. 150 adet eleman blok içinde toplandıktan sonra blok konumu kendisi kadar ötelenir ve kalan 150 elemanın da toplamı hesaplanır. Ağ için de benzer olarak ilk 1024x300 elemanı kapsayacak şekilde atanır. Ağ içindeki bloklar anlatıldığı biçimde işlemlerini tamamladıktan sonra ağ, bir sonraki 1024x300 elemanı kapsayacak şekilde kendi uzunluğu kadar kaydırılır. Bu şekilde tüm toplamlar elde edilir.

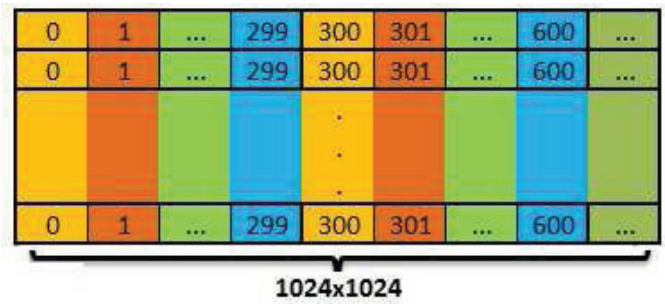
Bir diğer problem, ele alınan bloktaki iş parçacığı sayısı genel GPU *Reduction* algoritması için uygun olmamasıdır. Uygun algoritma oluşturulabilir, fakat amaç, genel kod olduğundan aynı *Reduction* algoritmasının kullanılmasıtercih sebebidir. Burada ufak bir hile uygulanır. Blokların paylaşılan hafizasındaki eleman sayısı 150 değil algoritmaya uygun olarak 256 alınır ve fazladan alınan elemanlar sıfir olarak atanır.

Böylece algoritma içindeki toplama işlemleri genel yapısını bozmadan *Reduction* algoritması ile yapılır. Oluşturulan CUDA çekirdeğinin kodu aşağıdaki gibidir:

```
__genel__ void SumArray2(float *data, float *odata, unsigned int n)
{
   unsigned int VirtualBlockSize = 300;//n/gridDim.x;
   unsigned int tid = threadIdx.x;
   unsigned int i = blockIdx.x * VirtualBlockSize + tid;
   unsigned int blockSize = 256;//blockDim.x;

   __shared__ float sdata[256];

   sdata[tid] = 0;

   for(unsigned int s=150; s<256;s++)
      sdata[s] = 0;

   while(i<n)
   {
      sdata[tid] += data[i] + data[i+150];
      i += 300 * 1024;
   }
    __syncthreads();

   // do reduction in shared mem
   if (blockSize >= 512) { if (tid < 256) { sdata[tid] += sdata[tid + 256]; } __syncthreads(); }
   if (blockSize >= 256) { if (tid < 128) { sdata[tid] += sdata[tid + 128]; } __syncthreads(); }
   if (blockSize >= 128) { if (tid <  64) { sdata[tid] += sdata[tid +  64]; } __syncthreads(); }

   #ifndef __DEVICE_EMULATION__
   if (tid < 32)
   #endif
   {
      if (blockSize >=  64) { sdata[tid] += sdata[tid + 32]; EMUSYNC; }
      if (blockSize >=  32) { sdata[tid] += sdata[tid + 16]; EMUSYNC; }
      if (blockSize >=  16) { sdata[tid] += sdata[tid +  8]; EMUSYNC; }
      if (blockSize >=   8) { sdata[tid] += sdata[tid +  4]; EMUSYNC; }
      if (blockSize >=   4) { sdata[tid] += sdata[tid +  2]; EMUSYNC; }
      if (blockSize >=   2) { sdata[tid] += sdata[tid +  1]; EMUSYNC; }
   }


   // write result for this block to genel mem
   if (tid == 0) odata[blockIdx.x] = sdata[0];

}
```

6.1.4.3 Son Boyuta Ait YBMG Terimleri Hesabının CUDA ile Koşutlaştırılması

Son boyuta ait düğüm noktalarına ait YBMG terimleri hesabı için kullanılan veri çizemi Şekil 6.9'de görülmektedir.



Şekil 6.9: Son boyuta ait düğüm noktalarının YBMG birli terimlerin veri kullanım çizemi

Özyineli ilişkilerden bilinmektedir ki son boyutun birinci düğüm noktasının YBMG terimi hesabında veri satırının ilk elemanı alınır, son boyuttaki düğüm noktası kadar ötelenir ve ötelendiği yerdeki veri ile toplanır. Bu süreç veri satırı sonuna kadar devam eder. Benzer süreç diğer düğüm noktalarının YBMG terimleri hesabında da kullanılır.

Burada CUDA açısından sorun ortadadır. Ardışık toplamlar yoktur, her bir ötelemede tek bir toplama işlemi gerçekleşmektedir. Dolayısıyla *Reduction* algoritması kullanılamaz. Burada uygulanacak yöntem şu şekildedir: blok sayısı ve her blokta yer alan iş parçacıkları sayısı şu şekilde belirlenir; Son boyuttaki düğüm noktası sayısı blok içindeki iş parçacıkları sayısına tam bölünmelidir. Ek olarak bu bölümden düğüm noktası sayısı kaç parçaya bölünüyorsa bu sayı ile önceki boyuttaki düğüm noktaları sayısının çarpımı blok sayısına tam bölünmelidir. Her bir bloktaki iş parçacığı sayısı 150 olsun. Blok sayısı da farklı olarak 64 seçilsin. Son bloktaki 300 bileşen 150 ile tam bölünebilmektedir. Ayrıca 2x1024 = 2048 sayısını blok sayısı olarak belirlenen 64 tam böler. Burada amaç ağ ötelenirken özyineli ilişkiye uygun bir şekilde doğru sıradaki elemanların birbirleri üzerine toplanmasını sağlamaktır. Verilen kısıt sıkıntı verici gibi görünse de aslında değildir, blok içindeki eleman sayısı belirlendikten sonra uygun birçok blok sayısı kolayca bulunabilir. Örneğin 300 bileşene uygun olarak her bloktaki iş parçacıklarının sayısı 100 olarak seçilsin. 3x1024=3072 sayısı ile tam bölünen herhangi bir sayı blok sayısı olabilir, 64, 96, 192, 256 bunlara örnektir.

Açıkça görülebileceği gibi son boyutta ardışık toplamlar yoktur, dolayısıyla her bir blok içindeki YBMG terimlerinin tümümü hesaplamakta kullanılan toplamlar bulunacaktır. Her bir blok kendi içinde diğerinden değer alamamaktadır. Ek olarak hatırlanacağı gibi örnek olarak aldığımız ağ 64 blok ve her blok 150 işparçacığından oluşmaktadır. Daha önceki boyutlardaki YBMG terimleri bulunurken uygulandığı gibi blok kendisi kadar kaydırılıp toplama işlemine devam edilmek istenirse yanlış veri elemanlarını birbirleri ile toplayacaktır. Sorun "odata[(blockIdx.x% (Boyut bileşenlerinin bloklara bölünme sayısı)* (Blok içindeki iş parçacığı sayısı) + tid] += sdata[tid]" ilişkisi ile çözülür. Seçimlerimize göre bu ilişki "odata[(blockIdx.x% 2)*150 + tid] += sdata[tid]" olur. Son boyuttaki düğüm noktalarının YBMG fonksiyon toplamlarını bulan kod aşağıdaki gibidir:

```
__genel__ void SumArray3(float *data, float *odata, unsigned int n)
{
   unsigned int blockSize = 150;
   unsigned int tid = threadIdx.x;
   unsigned int i = blockIdx.x * blockSize + tid;
```

```
    unsigned int gridSize = blockSize * gridDim.x;

    __shared__ float sdata[150];

    sdata[tid] = 0;

    while(i<n)
    {
       sdata[tid] += data[i];

       i += gridSize;
    }
    __syncthreads();

    odata[(blockIdx.x%2)*150 + tid] += sdata[tid];
}
```

Görüldüğü gibi diğer kodlardan farklı olarak GPU "Reduction" algoritması kullanılmamıştır.

6.1.4.4 Algoritmanın Başarım Değerlendirmesi

Başarım değerlendirmesine geçmeden önce CPU ve GPU sistem özellikleri, karşılaştırma için verilmelidir.

CPU System:

- 2 x Intel(R) Xeon QuadCore X5550 2.67GHz

- 32GB 1333Mhz DDR3 667Mhz

GPU System:

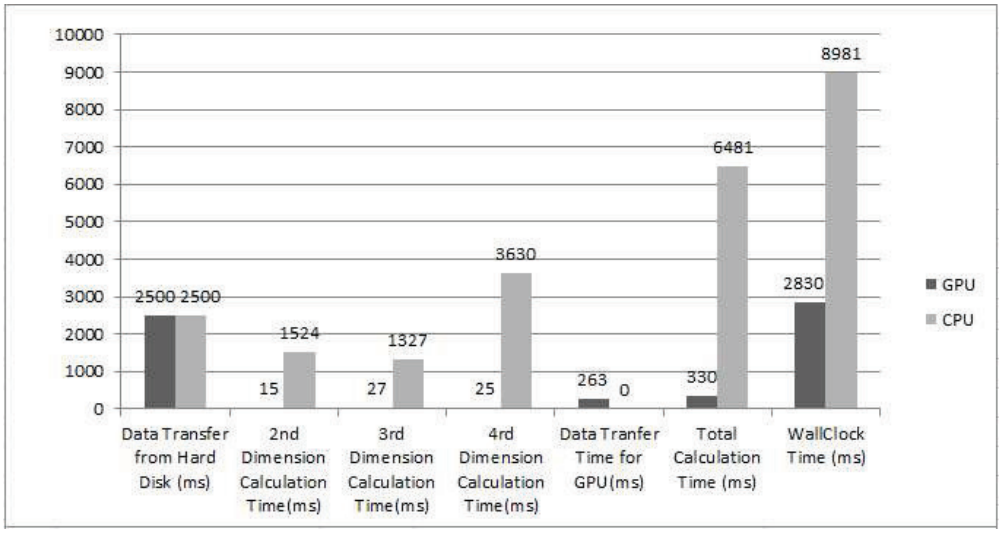
- Nvidia(R) Tesla C1060 GPU

- 240 x 1.296Ghz Stream Processors

- 4GB DDR2 667Mhz Genel Hafiza

Alınan veri setinin ilk veri satırı için performans değerlendirmesi Şekil 6.10'de görülmektedir.

Buradaki sonuçlar, genel olarak bakıldığında GPU'nun CPU'ya göre çok daha performanslı çalıştığını göstermektedir. Boyut kırılımında teker teker sonuçları inceleyelim. İkinci boyuttaki hesaplama zaman performans farkı incelensin. Hatırlanırsa 1024 adet blok ve her bir blokta 256 iş parçacığı vardı. Bu seçim GPU için en performanslı



Şekil 6.10: CPU ve GPU sistemlerinin YBMG yöntemi için başarım karşılaştırması

seçimlerden biriydi. CPU her bir YBMG terms işlemini seri şekilde yaparken GPU blokları ayrı ayrı hesaplamalarına başlayabildiler. Ayrıca seçilen 256 iş parçacığı 32'lik WARP kalıbına da uygun olduğundan çalışan herhangi bir WARP diğer WARP'ı beklememiştir. Sonucunda da yüksek performans yakalanmıştır.

Üçüncü boyut bileşenlerinin YBMG terimleri hesaplama başarımına bakılacak olursa, yine performansın yüksek olduğu, fakat bir önceki performansa göre hatırı sayılır bir düşüş olduğu gözlenebilir. Bu durum da bir önceki başarımın nedenleri ile açıklanabilir. Bloklardaki iş parçacıkları sayısı 150 seçilmiştir ki 256 olan paylaşılan bellek uzunluğu ile uyuşmaz. 32'lik WARP yapısına uyamamaktadır, dolayısıyla bazı WARP'lar bir önceki WARP'ı beklemek durumunda kalır. Fakat bu durumda bile performans literatürdeki yüksek performanslı sonuçlar içerisinde yer alır.

Son boyutta CPU performansının oldukça kötü olduğu gözlemlenir. Bunun sebebi de açıktır, hesaplamada yer alan iç içe for döngülerinde toplama işlemi ardışık değildir. Bu nedenle her bir hesaplama oldukça fazla ötelemeli ve her ötelemede sadece 1 toplama işlemi yapılarak tamamlanır. GPU programlamada, kendine özgü yapısı sayesinde bu durumda bile çok büyük bir performans kaybı yaşanmamaktadır.

Fakat burada GPU üzerinden çalıştırılan algoritmaya özgü zaman yitimleri söz konusudur. GPU'nun veriyi işleyebilmesi için öncelikle bilgisayarın RAM'inden kendi genel hafızasına aktarması gerekmektedir. Aktarım için geçen süre 263 ms'dir. Bu süre, GPU üzerinden yapılan toplam hesaplama süresinden kat ve kat yüksektir. Ek olarak verinin hem CPU hem de GPU üzerinde işlenebilmesi için öncelikle sabit diskten

bilgisayarın RAM'ine aktarılması gerekmektedir. Bu süre de ortak olarak 2500 ms'dir. Sabit diskten RAM'e aktarım süresi gözönüne alındığında GPU üzerinden çalıştırılan algoritmada bulunan hesaplama ve veri aktarım süreleri ihmal edilebilecek kadar azdır. Sonuçta GPU, CPU'ya göre toplamda 3 katın üzerinde başarım yakalamaktadır.

7. SONUÇLAR

Bu tez çalışmasında özgün olarak geliştirilen ve yukarıda da belirtilen çalışmaları maddeler şeklinde özetlersek:

- Çok boyutlu veri bölüntüleme amacı için geliştirilmiş ve uygulamaları bulunan YBMG yönteminin temel eşitlikleri koşutlaştırma amacıyla iyileştirilmiştir. Yöntemde ele alınan problemin boyut ve bileşen sayılarına göre yapıları ve hesaplama karmaşıklığı değişim gösteren bu eşitlikler iyileştirme süreci sonucunda boyut ve bileşen sayısından bağımsız bir yapıya kavuşmuştur.
- Graph teorisi kullanılarak yöntemde kullanılan eşitliklerin eşitliklerin matematiksel yapısı ortaya çıkarılmıştır.
- İyileştirilmiş yeni yapı sayesinde C/Fortran gibi makina diline daha yakın ve hızlı dillerle yöntemin kodunun yazılması olanaklı hale gelmiştir.
- Yöntemin seri C kodu yazılmış ve başarımlı bir şekilde çalışır hale getirilmiştir. Bu sayede yüksek hacimli problemlere sahip olmayan kullanıcıların da sonuçları daha hızlı bir şekilde elde etmeleri sağlanmıştır.
- Bunun yanısıra tezin amacına uygun olarak MPI kitaplığı kullanılarak yöntem başarımlı bir şekilde koşutlaştırılmıştır. Benzer şekilde CUDA kitaplığı kullanarak da koşutlaştırma başarımı elde edilmiştir. Bu koşutlaştırmalar sayesinde hem supercomputer altyapısına sahip hem de CUDA çalıştırabilecek donanıma sahip kullanıcıların yüksek hacimli problemler üzerinde ele alınan yöntemi kullanma olanağı sağlanmıştır.

KAYNAKLAR:

[1] Rabitz H. and Alis O.F. and Shorter J. and Shim K., (1999). Efficient Input-Output Model Representations, *Computer Physics Communications*, **117 (1-2)**, 11-20.

[2] Shorter J.A. and Precila C.I. and Rabitz H., (1999). An Efficient Chemical Kinetics Solver Using High Dimensional Model Representation*, J. Phys. Chem. A*, **103**, 7192-7198.

[3] Li G. and J. Hu and S.-W. Wang and Rabitz H., (2002). Practical Approaches To Construct RS-HDMR Component Functions, *Phys. Chem. A*, **106 (37)**, 8721-8733.

[4] Li G. and J. Hu and S.-W. Wang Georgopoulos P.G. and Schoendorf J. and Rabitz H., (2006). Random Sampling-High Dimensional Model Representation (RS-HDMR) and Orthogonality of Its Different Order Component Functions, *Phys. Chem. A*, **110 (7)**, 2474-2485.

[5] Tomlin A.S., (2006). The Use of Global Uncertainty Methods for the Evaluation of Combustion Mechanisms, *Reliability Engineering and System Safety*, **91**, 1213-1231.

[6] Ziehn T. and Tomlin A.S., (2008). A Global Sensitivity Study of Sulphur Chemistry in a Premixed Methane Flame Model Using HDMR, *International Journal of Chemical Kinetics*, **40 (11)**, 742-753.

[7] Tunga M.A. and Demiralp M., (2003). Data Partitioning via Generalized High Dimensional Model Representation (GHDMR) and Multivariate Interpolative Applications, *Mathematical Research,***9**, 447-462.

[8] Ho T.-S. and Rabitz H., (2003). Evaluation of city refuse compost maturity Reproducing Kernel Hilbert Space Interpolation Methods as a Paradigm of High Dimensional Model Representations: Application to Multidimensional Potential Energy Surface Construction, *Journal of Chemical Physics*, **119 (13)**, 6433-6442.

[9] Tunga M.A. and Demiralp M., (2005). A Factorized High Dimensional Model Representation on the Partitioned Random Discrete Data, Applied Numerical Analysis and Computational Mathematics, 1 (1), 231-241.

[10] Tunga M.A. and Demiralp M., (2005). A Factorized High Dimensional Model Representation on The Nodes of a Finite Hyperprismatic Regular Grid. Applied Mathematics and Computation, 164 (3), 865-883.

[11] Sobol I.M., (1993). Sensitivity Estimates for Nonlinear Mathematical Models, Mathematical Modelling and Computational Experiments, 1, 407-414.

[12] Tunga M.A. and Demiralp M., (2006). Hybrid High Dimensional Model Representation (HHDMR) on The Partitioned Data, Journal of Computational and Applied Mathematics, 185 (1), 107-132.

[13] Wolfram Math World Web Site, (Erişim Tarihi: Kasım 2010). Grid Graph, http://mathworld.wolfram.com/ GridGraph.html.

[14] Wolfram Math World Web Site, (Erişim Tarihi: Kasım 2010). Complete k-Partite Graph, http://math\-world.wolfram.com/Completek-PartiteGraph.html.

[15] Wolfram Math World Web Site, (Erişim Tarihi: Kasım 2010). k-Partite Graph, http://mathworld.wolfram.com/k-PartiteGraph.html.

[16] Wolfram Math World Web Site, (Erişim Tarihi: Kasım 2010). Acyclic Digraph, http://mathworld.wolfram.com/AcyclicDigraph.html.

[17] Argonne National Laboratory Web Site, (Erişim Tarihi: Kasım 2010). Acyclic Digraph, http://www.mcs.anl.gov/research/projects/mpi/www/www3/MPI_Reduce.html.

[18] Mark Harris, (Erişim Tarihi: Kasım 2010). Optimizing Parallel Reduction in CUDA, NVIDIA Developer Technology Whitepaper.

[19] Wilkinson B. and Allen M., (2004). Parallel Programming Techniques and Applications Using Networked Workstations and Parallel Computers 2nd ed., Pearson Education Inc.

[20] M. Alper Tunga, (2006). Data Partitioning and Multivariate Interpolation via Various High Dimensional Model Representations, İstanbul Teknik Üniversitesi Doktora Tezi.

[21] Evrim Korkmaz, (2009). Bütünleştirilmiş Küçük Ölçekli Yüksek Boyutlu Model Gösterilimi ve Çok Değişkenli İşlev Yaklaştırımında Kullanımı İstanbul Teknik Üniversitesi Yüksek Lisans Tezi.

ÖZGEÇMİŞ

Ad Soyad: Mehmet Engin Kanal

Doğum Yeri ve Tarihi: İstanbul, 23/08/1977

Lisans Üniversite: D.E.Ü., Matematik Eğitimi Bölümü (1995-1999)

Yüksek Lisans Üniversite: Marmara Ü., Fen Bilimleri Enstitüsü, Uygulamalı Matematik (1999-2002)

Yayın Listesi:

- **Kanal M.E., Baykara N.A., Demiralp M.**, 2011. , Theory And Algorithm of The Inversion Method For Pentadiagonal Matrices, Journal of Mathematical Chemistry, DOI:
- **Kanal M.E., Demiralp M.**, 2011. , A Modified Method of Calculating High Dimensional Model Representation (HDMR) Terms for Parallelization with MPI and CUDA, Journal of Supercomputing, DOI: 10.1007/s11227-011-0695-0
- **Kanal M.E.**, 2010. , Parallel Algorithm on Inversion for Adjacent Pentadiagonal Matrices with MPI, Journal of Supercomputing, DOI: 10.1007/s11227-010-0487-y
- **Kanal M. E.**, Demiralp M., 2008. Data Partitioning via High Dimensional Model Representation by Using Paralel Computing, Proceedings of the 1st WSEAS International Conference on Multivariate Analysis and its Application in Science and Engineering (MAASE 08), İstanbul, Turkey
- Üsküplü S., **Kanal M. E.**, Demiralp M., 2005. Extension of A Finite Regular Data Given To Determine A Univariate Function By Using Forward and Backward Differences, International Conference of Numerical Analysis and Applied Mathematics (ICNAAM 2005), Rhodes, Greece
- **Kanal M. E.**, Üsküplü S., Demiralp M., 2005. Precision Increased Truncated Derivative Formulae in Terms of Forward and Backward Difference Operators, International Conference of Numerical Analysis and Applied Mathematics (ICNAAM 2005), Rhodes, Greece
- **Kanal M. E.**, 2004. Parallel Algorithm on Inversion for Adjacent Pentadiagonal Matrices with MPI, International Conference of Computational Methods in Sciences and Engineering 2004 (ICCMSE 2004), Attica, Greece
- **Kanal M. E.**, Baykara N.A. and Demiralp M., 2003. A Novel Approach To The Numerical Inversion Algorithm For Adjacent Pentadiagonal Matrices, Fourth International Conference (MathTools 2003), St. Petersburg, Russia
- Baykara N.A., **Kanal M. E.**, Yaman İ. and Demiralp M., 2001. Çarpımsallaştırılmış Yüksek Boyutlu Model Gösterilimi Sınama Uygulamaları: Ü ç Köşegenli Matrislerin Determinantlarının Sayısal Hesaplamaları , XII. Ulusal Mekanik Kongresi, Konya

M.Engin Kanal 1977’de İstanbul’da doğdu. Lisans derecesini 1999 yılında Dokuz Eylül Üniversitesi, Buca Eğitim Fakültesi, Matematik Eğitimi Bölümü’nden aldı. Yüksek lisansını Marmara Üniversitesi Fen Bilimleri Enstitüsü Uygulamalı Matematik Programı’ndan 2002 yılında almıştır. 2003 yılında İstanbul Teknik Üniversitesi, Bilişim Enstitüsü, Hesaplamalı Bilim ve Mühendislik Programı’nda doktoraya başlamıştır.

İÇİNDEKİLER

Sayfa

KISALTMALAR

ALU	:	Arithmetic Logic Unit
CPU	:	Central processing unit
CUDA	:	Compute Unified Device Architecture
ÇYBMG	:	Çarpımsal Yüksek Boyutlu Model Gösterilim
GB	:	Giga byte
GPU	:	Graphics processing unit
KB	:	Kilo byte
MB	:	Mega byte
MPI	:	Message Passing Inferface
MYBMG	:	Melez Yüksek Boyutlu Model Gösterilim
SP	:	Streaming Processor
SM	:	Streaming Multiprocessor
TB	:	Tera byte
YBMG	:	Yüksek Boyutlu Model Gösterilim

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

SEMBOL LİSTESİ

f_0	:	YBMG'nin değişmez terimi
σ_0	:	birinci basamaktan değişmezlik ölçeni
σ_k	:	k. basamaktan toplamsallık ölçeni
W	:	Ağırlık işlevi
δ	:	Kronecker Delta simgesi

YÜKSEK BOYUTLU MODEL GÖSTERİLİMİ İLE VERİ BÖLÜNTÜLEME YÖNTEMİNİN KOŞUTLAŞTIRILMASI

ÖZET

Yalnızca uzaydaki belli noktalardaki değerleri verilmiş çok değişkenli bir $f(x_1, x_2, \cdots, x_N)$ işlevine alışılagelmiş yöntemlerle içdeğerbiçim işlemi yapılması boyut sayısı arttığında birer başbelası durumuna gelir. Bu tür işlevler için doğrudan bilgisayar programcılığı ile çözüm aramak yerine ilk olarak bu işlevleri bilgisayar programlaması açısından daha kolay ele alınacak, matematiksel olarak, etkili bir yapıya getirmek gerekir. Bu amaçla bu işleve yaklaştırım yapan bir böl–ve–yönet algoritması geliştirilmiştir. Bu yaklaştırım sayesinde çok değişkenli f işlevi çok daha düşük boyutlu terimlerle ifade edilebilmektedir. Bu yaklaştırıma Yüksek Boyutlu Model Gösterilimi (YBMG) adı verilmektedir. Bu yöntem çeşitli çalışmalarla başarılı bir şekilde uygulanmıştır. Fakat bu yöntem bu haliyle büyük veri hacmine sahip problemler üzerinde uygulanamaz. Problemdeki boyut sayısı ve boyutlardaki düğüm noktaları sayıları arttığında veri hacmi öyle büyür ki alışılagelmiş PC'ler verinin gereksinim duyduğu yüksek RAM sığasını karşılayamaz. Diğer bir önemli problem de YBMG terimlerini hesaplamakta kullanılan eşitliklerin yapılarıdır. Eşitlikler için yazılmış algoritmadaki döngü sayıları problemdeki boyut sayısına bağımlıdır. Bu çalışmada ilk olarak YBMG terimlerini hesaplamakta kullanılan eşitlikler iyileştirilmiştir. Bu iyileştirme sonucunda eşitliklerin problemdeki boyut sayısına bağımlılığı ortadan kaldırılmıştır. İyileştirilmiş eşitlikler sayesinde yöntem koşutlaştırmaya uygun bir hale getirilmiştir. Son olarak da yöntemin koşutlaştırmasının başarımı çözümlenmiştir.

DATA PARTITIONING VIA HIGH DIMENSIONAL MODEL REPRESENTATION BY USING PARALEL COMPUTING

SUMMARY

If the values of a multivariate function $f(x_1, x_2, \cdots, x_N)$ are given at only a finite number of points in the space of its arguments and an interpolation which employs continuous functions is considered standard multivariate routines may become cumbersome as the dimensionality grows. This urges us to develop a divide–and–conquer algorithm which approximates the function. The given multivariate data is partitioned into low-variate data. This approach is called High Dimensional Model Representation (HDMR). However the method in its current form is not applicable to problems having huge volumes of data. With the increasing dimension number and the number of the corresponding nodes, the volume of data in question reaches such a high level that it is beyond the capacity of any individual PC because huge volume of data requires much higher RAM capacity. Another aspect is that the structure of equalities used in the calculation of HDMR terms varies according to the dimension number of the problem. The number of loops in the algorithm increases with the increasing dimension number. In this work, as a first step, the equations used are modified in such a way that their structure does not depend on the dimension number. With the newly obtained equalities, the method becomes appropriate for parallelization. Due to the parallelization, the RAM problem arising from problems with high volume of data is solved. Finally, the performance of the parallelized method is analyzed.

1. GİRİŞ

YBMG yöntemi ilk olarak 1993 yılında I.M.Sobol tarafından Monte Carlo ve quasi-Monte Carlo algoritmalarını ele alarak farklı değişkenler ya da değişken gruplarına göre bir $f(x_1, x_2, \cdots, x_N)$ işlevinin duyarlılığını kestirmek için geliştirilmiştir[11]. Bu duyarlılık belirlenmesi, doğrudan f işlevi üzerinden gerçekleştirilmemiştir. YBMG yöntemi temel olarak Sobol Teoremi'nde değinilen ve integrali alınabilir bir işlevin, kendisini oluşturan değişkenlerin kendileri ve birbirleri ile olan etkileşimlerinin birleşimine eşit olduğu düşüncesine dayanır. Bu şekilde N değişkenli bir işlev için Taylor açılımına benzer yapıda, ilk olarak değişmez (sabit) bileşen, bir değişkenli bileşenler, iki değişkenli bileşenler ve en sonunda bir adet N değişkenli bileşenin birleşimi olarak yazılır. Burada değinilen HDMR bileşenleri, içerdikleri değişkenlerin birbirleri ile etkileşimlerini göstermektedir.

Sobol'un çalışmasının ardından YBMG yöntemi, Hershel Rabitz ve grubu tarafından daha kapsamlı bir hale getirilmiştir[1-4]. YBMG ve YBMG bazlı diğer algoritmalar atmosferik modelleme, atmosfer dinamikleri ve source-sinks of trace gase, quantitative risk assessment, atmosferik ölçümlerinin inversiyonu, mali ve ekonomik uygulamalar ve stratosferik kimyasal devinim bilimi gibi farklı mühendislik alanları için geliştirilmiştir[1-6].

Bu çalışmada koşutlaştırılan YBMG yöntemi çok değişkenli içdeğerbiçim problemlerine çözüm için geliştirilmiştir[7-10]. Fakat burada içdeğerbiçim uygulanacak işlevin analitik yapısı belli değildir, bunun yerine belli noktalardaki değerleri elde bulunmaktadır. Bu çokdeğişkenli veri yöntem yardımıyla daha az değişkenle anlatılabilecek bir yapıya dönüştürülür ve bu yapı üzerinden içdeğerbiçim işlemi yapılır. Burada anafikir verinin en fazla birli değişkenlerden oluşan bir yapı ile yaklaştırım yapılmasını sağlamaktır.

Bu noktada yöntemin gerçek dünya problemlerinde uygulanması için neden koşutlaştırmaya gereksinim duyuldugu açıklanmalıdır. Koşutlaştırma gereksinimi temel bir soruna çözüm bulacaktır. Bu sorun yüksek veri hacmine sahip problemler üzerinde

yöntemin uygulanmasıdır. Gerçek dünya problemlerinde bir olayı etkileyen çok sayıda değişken vardır. Bu değişkenlerin ve değişkenlerdeki bileşen sayısı arttıkça olayın gerçekleştiği uzay da genişler. YBMG, N değişkenli bir f işlevini daha düşük boyutlardaki terimlerde ifade eder. Fakat bunun için eldeki tüm veriyi kullanması gerekir. Veri kümesi değişkenlerdeki bileşen sayılarının çarpımıdır. İster düşük ister yüksek sayıda boyut kullanılsın, veri kümesini oluşturan her bir değişkene ait bileşen sayıları belli bir sayının üzerine çıktıkça, milyonlara hatta milyarlara varan noktalardan oluşmuş veri setleri kolaylıkla ortaya çıkar. Örnek olarak 3 değişkenli bir uzayda her bir değişkende 10.000 adet bileşen olduğu düşünülürse ve bu bileşenlerin oluşturduğu nokta sayısı $10.000^3 = 1 \cdot 10^{12}$ adet olacaktır. Tüm noktalar $float$ türünde tutulsa veri kümesinin hacmi $4 \cdot 10^{12}$ *byte* olacaktır. Bu da *terabyte* hacmine sahip bir veri seti ile çalışıldığı anlamına gelmektedir. Bu kadar büyük hacimlerle alışılagelmiş bir PC üzerinde uygulama çalıştırılamaz. Yöntemin koşutlaştırılması için yöntemde kullanılacak eşitliklerin koşutlaştırılması gerekmektedir. Burada da diğer önemli problem ortaya çıkar. Yöntemde kullanılan eşitliklerin yapıları problemdeki boyut sayısına göre değişkenlik göstermektedir. Bu problemi aşmak için ilk olarak bu eşitlikler üzerinde iyileştirme çalışması yapılmıştır. İyileştirme çalışması sonucunda eşitlikler boyut sayılarından bağımsız bir yapıya kavuşturulmuştur. Bu aşamadan sonra yöntem koşutlaştırılmıştır. Koşutlaştırma yöntemi olarak MPI (Message Passing Interface) ve CUDA (Compute Unified Device Architecture) kitaplıkları kullanılmıştır. MPI, supercomputer altyapısı olan kullanıcılar için tasarlanmış bir kitaplıktır. CUDA kitaplığı ise son yıllarda çok tanınmış olan ve ekran kartı üzerinde bulunan "Performance per watt" olarak çok daha etkili GPU (Graphical Processing Unit) kullanan bir kitaplıktır. Böylece hem supercomputing altyapısı taşıyan, hem de GPU kullanabilen bir bilgisayar altyapısına sahip kullanıcılar için koşutlaştırma seçenekleri sunulmuştur ve başarımları incelenmiştir.

Tezin 2. kısmında YBMG yöntemi hakkında bilgi verilmiştir.

Tezin 3. kısmında veri bölüntüleme ve içdeğerbiçim üzerinde durulmuştur.

Tezin 4. kısmında yöntemde kullanılan eşitliklerin iyileştirilme aşamaları üzerinde durulmuştur.

Tezin 5. kısmında yöntemin MPI ile koşutlaştırılma aşamaları anlatılmıştır.

Tezin 6. kısmında yöntemin CUDA ile koşutlaştırılma aşamaları anlatılmıştır.

Tezin sonuç kısmında koşutlaştırma süreçlerinden çıkan sonuçlar tartışılmıştır.

2. TEZDE KOŞUTLAŞTIRILACAK YÖNTEMLER

Bu bölümde tez çalışması içerisinde koşutlaştırılacak olan Yüksek Boyutlu Model Gösterilimi (YBMG) yöntemiyle ilgili kısaca bilgi verilmesi amaçlanmıştır. Bu amaçla ilk olarak bu yöntemin çeşitleri üzerinde durulacak ve tarihsel gelişim süreciyle ilgili bilgi verilecektir.

2.1 Yüksek Boyutlu Model Gösterilimi (YBMG) Yöntemleri

YBMG konusunda yapılan ilk çalışma olan Sobol Kanıtsavına (teoremine) göre tümlevlenebilir (integrali alınabilir) her $f(x_1,...,x_N)$ işlevi aşağıdaki anlatımla yazılabilir.

$$f(x_1,...,x_N) = f_0 + \sum_{i_1=1}^{N} f_{i_1}(x_{i_1}) + \sum_{\substack{i_1,i_2=1 \\ i_1<i_2}}^{N} f_{i_1 i_2}(x_{i_1},x_{i_2}) + \cdots + f_{12\ldots N}(x_1,...,x_N) \tag{2.1}$$

Açılım, görüldüğü gibi, sonlu bir toplamdır ve değişmez terim, tek değişkenli terimler, iki değişkenli terimler sırasında ilerlemektedir. Burada, önemli olan, sağ yandaki f'leri tanımlamaktır. Kanıtsavdaki öngörüme göre

$$\int_0^1 dx_s f_{i_1,i_2,\ldots i_k}(x_{i_1},\ldots,x_{i_k}) = 0 \qquad s = i_1,\ldots,i_k \tag{2.2}$$

(**2.1**) eşitliğinin sağ yanındaki terimler için diklik koşulu geçerlidir.

$$(f_{i_1 i_2 \ldots i_k}, f_{i_1 i_2 \ldots i_l}) = 0, \quad \{i_1,i_2,\ldots,i_k\} \not\equiv \{i_1,i_2,\ldots,i_l\}, \quad 1 \le k,l \le N \tag{2.3}$$

$$\begin{aligned}\left(f_{i_1,\ldots i_k}, f_{j_1,\ldots j_l}\right) \equiv & \int_{a_1}^{b_1} dx_1 \cdots \int_{a_1}^{b_1} dx_N W(x_1,...,x_N)\, f_{i_1,\ldots i_k}(x_{i_1},\ldots,x_{i_k}) \\ & \times f_{j_1,\ldots j_l}\left(x_{j_1},\ldots,x_{j_l}\right), \qquad 1 \le i_1 < \ldots < i_k \le N,\end{aligned}$$

$$1 \le j_1 < \ldots < j_l \le N, \qquad 1 \le k,l \le N \tag{2.4}$$

Yukarıda tanımlanan eşitliğe ve diklik koşuluna uygun olarak (**2.1**)'de değişmez terim olarak gösterilen f_0'ı bulmak için (**2.1**) eşitliğinin her iki yanının N kez tümlevini alırız.

$$\int_0^1 \cdots \int_0^1 dx_1 \ldots dx_N f(x_1,\ldots,x_N) = f_0 \tag{2.5}$$

Benzer biçimde, (**2.1**) eşitliğinin her iki yanının x_i değişkeni dışlanarak N-1 kez tümlevi alınırsa aşağıdaki eşitlik elde edilir ve bu eşitlikten $f_i(x_i)$ değerleri üretilebilir.

$$\underbrace{\int_0^1 \cdots \int_0^1}_{N-1\ kat} dx_1 \ldots dx_{i-1} dx_{i+1} \ldots dx_N f(x_1, \ldots, x_N) = f_0 + f_i(x_i) \qquad \forall i = 1, \ldots, N \quad \textbf{(2.6)}$$

(**2.1**) eşitliğindeki tüm f'ler bu biçimde elde edilir. Sobol'ca bulunan yöntem Rabitz Grubu'nun çalışmaları ile genişletilmiş bir yapıya kavuşturulmuş ve kapsamlı uygulama alanları bulunmuştur[1-4]. Sobol'un yazısında 1 alınan ağırlık işlevi ve [0,1] aralığındaki tümlev yerine genişletilmiş yapı için aşağıdaki koşul verilmiştir.

$$\int_{a_1}^{b_1} dx_1 \cdots \int_{a_N}^{b_N} dx_N W(x_1, ..., x_N) f_i(x_i) = 0, \quad 1 \le i \le N \qquad \textbf{(2.7)}$$

Burada verilen $W(x_1, ..., x_N)$ ağırlık işlevi aşağıdaki biçimde tanımlanmaktadır.

$$W(x_1, \ldots, x_N) \equiv \prod_{j=1}^{N} W_j(x_j), \qquad x_j \in \left[a_j, b_j\right], \qquad 1 \le j \le N \qquad \textbf{(2.8)}$$

Ayrıca ağırlık işlevi aşağıdaki boylandırım (ing: normalization) koşulunu da sağlamalıdır.

$$\int_{a_j}^{b_j} dx_j W_j(x_j) = 1, \qquad 1 \le j \le N \qquad \textbf{(2.9)}$$

Sobol Kanıtsavı'na benzer biçimde değişmez, tek değişkenli ve iki değişkenli YBMG terimleri $W(x_1, ..., x_N)$ ağırlığı altında aşağıdaki biçimde bulunur.

$$f_0 = \int_{a_1}^{b_1} dx_1 \cdots \int_{a_1}^{b_1} dx_N W(x_1, ..., x_N) f(x_1, \ldots, x_N) \qquad \textbf{(2.10)}$$

$$\begin{aligned} f_k(x_k) \;=\; & \int_{a_1}^{b_1} dx_1 W_1(x_1) \cdots \int_{a_{k-1}}^{b_{k-1}} dx_{k-1} W_{k-1}(x_{k-1}) \\ & \times \int_{a_{k+1}}^{b_{k+1}} dx_{k+1} W_{k+1}(x_{k+1}) \cdots \int_{a_N}^{b_N} dx_N W_N(x_N) \\ & \left[f_0 + \sum_{i_1=1}^{N} f_{i_1}(x_{i_1}) + \cdots + f_{12\ldots N}(x_1, ..., x_N) \right] - f_0, \qquad 1 \le k \le N \end{aligned} \qquad \textbf{(2.11)}$$

$$\begin{aligned} f_{k_1 k_2}(x_1, ..., x_N) \;=\; & \int_{a_1}^{b_1} dx_1 W_1(x_1) \cdots \int_{a_{k_1-1}}^{b_{k_1-1}} dx_{k_1-1} W_{k_1-1}(x_{k_1-1}) \\ & \times \int_{a_{k_1+1}}^{b_{k_1+1}} dx_{k_1+1} W_{k_1+1}(x_{k_1+1}) \cdots \int_{a_{k_2-1}}^{b_{k_2-1}} dx_{k_2-1} W_{k_2-1}(x_{k_2-1}) \\ & \times \int_{a_{k_2+1}}^{b_{k_2+1}} dx_{k_2+1} W_{k_2+1}(x_{k_2+1}) \cdots \int_{a_N}^{b_N} dx_N W_N(x_N) \\ & \times \left[f_0 + \sum_{i_1=1}^{N} f_{i_1}(x_{i_1}) + \cdots + f_{12\ldots N}(x_1, ..., x_N) \right] \\ & - f_{k_1}(x_{k_1}) - f_{k_2}(x_{k_2}) - f_0, \qquad 1 \le k_1 < k_2 \le N \end{aligned} \qquad \textbf{(2.12)}$$

Amacımız çok değişkenli işlevin kesin değerini üretmek olmadığından, aksine, belli bir kesme yaklaştırımı ile yetinilmek istendiğinden, geri kalan YBMG terimleri gözönüne alınmamaktadır. YBMG terimlerini belirlemekte kullanılan tümlev işleminde bulunan diklik koşulu üzerinde özenle durulması gereklidir. Diklik koşulu altında aşağıdaki eşitlik yazılabilir.

$$\int_{a_1}^{b_1} dx_1 \cdots \int_{a_1}^{b_1} dx_N W(x_1,...,x_N)\, f_{i_1,...i_k}(x_{i_1},\ldots,x_{i_k})\, f_{j_1,...j_l}\left(x_{j_1},\ldots,x_{j_l}\right) = 0$$
$$i_1,\ldots,i_k \neq j_1,\ldots,j_l, \qquad 1 \leq i_1 < \ldots < i_k \leq N,$$
$$1 \leq j_1 < \ldots < j_l \leq N, \qquad 1 \leq k,l \leq N \tag{2.13}$$

Bu da bizi aşağıdaki iççarpım tanımını kullanmaya zorlar.

$$\begin{aligned}\left(f_{i_1,...i_k}, f_{j_1,...j_l}\right) \equiv & \int_{a_1}^{b_1} dx_1 \cdots \int_{a_1}^{b_1} dx_N W\left(x_1,...,x_N\right) f_{i_1,...i_k}\left(x_{i_1},\ldots,x_{i_k}\right) \\ & \times f_{j_1,...j_l}\left(x_{j_1},\ldots,x_{j_l}\right), \qquad 1 \leq i_1 < \ldots < i_k \leq N,\end{aligned}$$

$$1 \leq j_1 < \ldots < j_l \leq N, \qquad 1 \leq k,l \leq N \tag{2.14}$$

Bu bizim aşağıdaki boy tanımını kullanmamıza olanak verir.

$$\| f_{i_1,...i_k} \| \equiv (f_{i_1,...i_k}, f_{i_1,...i_k}), \qquad 1 \leq i_1 < \ldots < i_k \leq N, \qquad 1 \leq k \leq N \tag{2.15}$$

YBMG terimlerinin boy tanımını ve diklik koşulunu kullanarak (**2.1**) eşitliği için aşağıdaki anlatım yazılabilir.

$$\| f \|^2 = \| f_0 \|^2 + \sum_{i_1=1}^{N} \| f_{i_1} \|^2 + \sum_{\substack{i_1,i_2=1 \\ i_1<i_2}}^{N} \| f_{i_1 i_2} \|^2 + \cdots + \| f_{12...N} \|^2 \tag{2.16}$$

Bu eşitlik bize kesme uygulanmış YBMG yaklaştırımının niteliğini ölçmemizi sağlar. Eğer YBMG değişmez terimden sonra kesilirse yaklaştırıma “Değişmez Yaklaştırım”, değişmez ve birli terimlerinden sonra kesilirse “Birli Yaklaştırım”, değişmez, birli ve ikili YBMG terimlerinden sonra kesilirse “İkili Yaklaştırım”adları verilir. Bu yaklaştı-

rım adlandırmaları aşağıdaki toplamsallık ölçenlerinin tanımlanmasına olanak sağlar.

$$\begin{aligned}
\sigma_0 &\equiv \frac{1}{\|f\|^2}\|f_0\|^2 \\
\sigma_1 &\equiv \frac{1}{\|f\|^2}\sum_{i=1}^{N}\|f_i\|^2 + \sigma_0 \\
&\cdots \\
\sigma_N &\equiv \frac{1}{\|f\|^2}\|f_{12\ldots N}\|^2 + \sigma_{N-1}
\end{aligned} \tag{2.17}$$

Burada σ_0 "Değişmezlik Ölçeni"olarak adlandırılmaktadır. Diğer σ_k terimleri de, k değişken içeren terimlerle ilgili olan "k. Toplamsallık Ölçeni"olacak biçimde adlandırılırlar. Bu ölçenler 0 ile 1 değerleri arasında sıralı olarak aşağıdaki anlatımı sağlarlar.

$$1 \le \sigma_0 \le \sigma_1 \le \cdots \le \sigma_N = 1 \tag{2.18}$$

YBMG yaklaştırımındaki kesme yaklaştıranları (ing: approximant) daha ayrıntılı bir biçimde aşağıda verilmektedir.

$$\begin{aligned}
s_0(x_1,x_2,\cdots,x_N) &= f_0 \\
s_1(x_1,x_2,\cdots,x_N) &= s_0(x_1,x_2,\cdots,x_N) + \sum_{i=1}^{N} f_i(x_i) \\
&\vdots \\
s_k(x_1,x_2,\cdots,x_N) &= s_{k-1}(x_1,x_2,\cdots,x_N) + \sum_{\substack{i_1,\cdots,i_k=1 \\ i_1<\cdots<i_2}}^{N} f_{i_1,\cdots,i_k}(x_{i_1},\cdots,x_{i_k}) \\
& \qquad 1 \le k \le N
\end{aligned} \tag{2.19}$$

Bu tanımları gözönüne alarak (**2.1**)'de verilmiş bulunan $f(x_1,\cdots,x_N)$ işlevine k. basamaktan YBMG yaklaştırımı olan $s_k(x_1,\cdots,x_N)$ aşağıdaki anlatımla verilebilir.

$$f(x_1,\cdots,x_N) \approx s_k(x_1,\cdots,x_N) \tag{2.20}$$

Bu yaklaştırımın boyu da aşağıdaki ilişkiyi sağlar.

$$\|s_k\| = \sigma_k\|f\|, \qquad 0 \le k \le N \tag{2.21}$$

İşlev (**2.1**)'de verilen anlatımla açılır ve $f(x_1,x_2,\ldots,x_N)$ benimsenebilir bir k değeri için "Toplamsal"(YBMG'inde salt değişmez ve birli terim barındıran, diğer bileşenleri özdeş olarak 0 olan) bir işlev ise 1. basamaktan YBMG yaklaştırımı, işlevi, kesin

olarak anlatmaya yeterli olacaktır. Bu nedenle, 1. basamaktan YBMG yaklaştırımına arıtoplamsal (ing: pure additive) yaklaştırım adı da verilmektedir. Verilen herhangi bir çok değişkenli işlevin bu özelliği taşıması beklenemeyeceğinden ve dolayısıyla YBMG ile yeterince duyarlılık elde edilemeyeceğinden ya yüksek basamaktan yaklaştırımlara çıkmak gerekmekte (ki bu durumda bilgisayım karmaşıklığının çok çok büyümesi söz konusu olabilmektedir) ya da başka bir yöntem geliştirme gereksinimi doğmaktadır.

2.1.1 Çarpımsallaştırılmış YBMG (ÇYBMG)

Eğer bir an için $f(x_1,...,x_N)$ işlevinin arıçarpımsal bir yapıda olduğu yani

$$f(x_1,...,x_N) = \prod_{i=1}^{N} u_i(x_i) \tag{2.22}$$

yazılabildiği varsayılacak olursa, YBMG f_0 değişmez terimini bulmak için (**2.10**) eşitliği kullanılırsa

$$f_0 = \int_{a_1}^{b_1} dx_1 W_1(x_1)u_1(x_1) \cdots \int_{a_N}^{b_N} dx_N W_N(x_N)u_N(x_N) \tag{2.23}$$

yazılabilir. Burada $\overline{u}_m$

$$\overline{u}_m = \int_{a_m}^{b_m} dx_m W_m(x_m)u_m(x_m), \qquad 1 \leq m \leq N \tag{2.24}$$

ile tanımlanırsa değişmez terim için yazılan (**2.23**) eşitliği aşağıdaki biçimde yeniden yazılabilir.

$$f_0 = \overline{u}_1\overline{u}_2 \cdots \overline{u}_N \tag{2.25}$$

(**2.1**) eşitliğinden yola çıkarak

$$f_0 + f_j(x_j) = (\overline{u}_1\overline{u}_2 \cdots \overline{u}_N)\frac{u_j(x_j)}{\overline{u}_j} = f_0\frac{u_j(x_j)}{\overline{u}_j}, \tag{2.26}$$

$$f_j(x_j) = f_0\left(\frac{u_j(x_j)}{\overline{u}_j} - 1\right), \tag{2.27}$$

$$u_j(x_j) = \overline{u}_j\left(\frac{f_j(x_j)}{f_0} + 1\right), \qquad 1 \leq j \leq N \tag{2.28}$$

sonuçları elde edilir.

ÇYBMG ikili terimleri de, benzer biçimde, aşağıdaki eşitliklerle verilen yapılarda elde edilebilirler.

$$\begin{aligned} f_0 + f_j(x_j) + f_k(x_k) + f_{j,k}(x_j,x_k) &= (\overline{u}_1\overline{u}_2\cdots\overline{u}_N)\left(\frac{u_j(x_j)}{\overline{u}_j}\right)\left(\frac{u_k(x_k)}{\overline{u}_k}\right) \\ f_{j,k}(x_j,x_k) &= f_0\left(\frac{u_j(x_j)}{\overline{u}_j}\right)\left(\frac{u_k(x_k)}{\overline{u}_k}\right) - f_0\left(\frac{u_j(x_j)}{\overline{u}_j}-1\right) \\ &\quad - f_0\left(\frac{u_k(x_k)}{\overline{u}_k}-1\right) - f_0 \\ f_{j,k}(x_j,x_k) &= f_0\left(\frac{u_{x_j}(x_j)}{\overline{u}_j}-1\right)\left(\frac{u_k(x_k)}{\overline{u}_k}-1\right), \qquad 1\le j<k\le N \end{aligned} \tag{2.29}$$

Elde edilen terimlere ait eşitlikler aşağıdaki biçimde yeniden düzenlenebilir.

$$\begin{aligned} &f_0 = \prod_{i=1}^{N}\overline{u}_i, \qquad \overline{u}_i \equiv \int_{a_i}^{b_i} dx_i W_i(x_i)\,u_i(x_i), \qquad 1\le i\le N \\ &f_i(x_i) = f_0\left(\frac{u_i(x_i)}{\overline{u}_i}-1\right), \qquad 1\le i\le N \\ &f_{i_1,i_2}(x_{i_1},x_{i_2}) = f_0\left(\frac{u_{i_1}(x_{i_1})}{\overline{u_{i_1}}}-1\right)\left(\frac{u_{i_1}(x_{i_2})}{\overline{u_{i_2}}}-1\right), \qquad 1\le i_1<i_2\le N \\ &\cdots\cdots \end{aligned} \tag{2.30}$$

Bunun anlamı $f(x_1,...,x_N)$ işlevinin arıçarpımsal olması durumunda, **2.1**'deki arıtoplamsal açılımın da arıçarpımsal bir yapının açılımı durumuna dönüştüğü doğrultusundadır. $f(x_1,...,x_N)$ işlevinin çarpımsal olmaması ve **(2.30)** bize, aşağıdaki eşitliği yazmamıza, olanak verir.

$$\begin{aligned} f_{i_1\ldots i_k}(x_{i_1},...,x_{i_N}) \equiv \frac{1}{f_0^{k-1}} f_{i_1}(x_{i_1})\ldots f_{i_k}(x_{i_k}), \\ 1\le i_1<\cdots<i_k\le N, \qquad 1\le k\le N \end{aligned} \tag{2.31}$$

Buradaki yapı $f(x_1,...,x_N)$ işlevinin $f_{YBMG}(x_1,...,x_N)$ ile simgeleyeceğimiz YBMG açılımında yerine konulur ve ortaya çıkan sonlu toplam özenle incelenirse

$$f_{YBMG}(x_1,...,x_N) = f_0\prod_{i=1}^{N}\left(1+\frac{f_i(x_i)}{f_0}\right) \tag{2.32}$$

yazılabileceği anlaşılır. Bunun doğruluğunu sınamak için birli terimlerin u'lar türünden anlatımları bu eşitlikte kullanılırsa

$$f_{YBMG}(x_1,...,x_N) = f_0\prod_{i=1}^{N}\left(1+\left[\frac{u_i(x_i)}{u_i}-1\right]\right) = f_0\left(\prod_{i=1}^{N}\frac{u_i(x_i)}{f_0}\right) \equiv f(x_1,...,x_N) \tag{2.33}$$

elde edilir. Yani, ilgilenilen çok değişkenli işlevin arıçarpımsal olması durumunda, onun YBMG açılımı (**2.32**) anlatımıyla yazılabilen arıçarpımsal bir yapıya dönüştürülebilmektedir.

Eğer ilgilenilen işlev arıçarpımsal değilse (**2.32**) geçerliliğini, daha doğrusu yeterliliğini koruyamaz. Bunun nedeni, bu anlatımda salt bir değişkenli çarpanlar kullanılmış olmasıdır. Dolayısıyla, ilgilenilen işlev ne olursa olsun, geçerliliğini koruyacak ve YBMG gibi yeterli olacak bir anlatımda değişmez bir çarpanla onu izleyen N sayıda birli çarpanla yetinilmemeli bunları sırasıyla, ikili, üçlü ve sayıları tekdüze artan ve sonunda N olan çok değişkenli çarpanlar izlemelidir. Yani, tıpkı YBMG'de olduğu gibi 2^N sayıda çarpan içerilmelidir. Dolayısıyla, ÇYBMG için önerilecek yapı, altsırasayılı r'lerle simgelenen büyüklükler bu an için bilinmeyen ama belirlenmesi istenen ögeler olmak üzere

$$\begin{aligned} f(x_1,...,x_N) \equiv\ & r_0 \left[\prod_{i=1}^{N}(1+r_i(x_i))\right] \left[\prod_{\substack{i_1,i_2=1 \\ i_1<i_2}}^{N}(1+r_{i_1 i_2}(x_{i_1},x_{i_2}))\right] \times \\ & \cdots \times r_{12..N}(x_{i_1},...,x_{i_N}) \end{aligned} \tag{2.34}$$

anlatımıyla verilmelidir. Bu anlatımdaki r'lerin belirlenmesi için anlatımın çarpma işlemlerini gerçekleştirerek toplamsal bir yapıya açılması ve ortaya çıkan terimlerin oluşturduğu YBMG'sel yapının YBMG'in değişmez, birli, ve diğer terimleriyle, terim terim karşılaştırma yapılması gerekir. Bu işlemin ara ayrıntılarına burada girmeyeceğiz. Ancak, değişmez, birli, ve de ikili çarpanların bu işlem sonucunda elde edilecek olan, YBMG terimleri türünden anlatımlarını vermekle yetineceğiz.

$$r_0 = f_0, \tag{2.35}$$

$$r_i(x_i) = \frac{f_i(x_i)}{f_0}, \qquad 1 \le i \le N \tag{2.36}$$

$$r_{i_1 i_2}(x_{i_1},x_{i_2}) = \frac{f_{i_1 i_2}(x_{i_1},x_{i_2})}{f_0} - \frac{f_{i_1}(x_{i_1})}{f_0}\frac{f_{i_2}(x_{i_2})}{f_0}, \qquad 1 \le i_1 < i_2 \tag{2.37}$$

Burada son terim ikili terimlerin (**2.32**)'i saglamasi durumunda, yani ilgilenilen asıl işlevin arıçarpımsal olması durumunda, değeri sıfırlanır. Bu terim, bir anlamda, tıpkı $f_{i_1 i_2}(x_{i_1},x_{i_2})$ gibi, ikili bağlılaşımlara (ing: correlation) karşılık gelmektedir. Dolayısıyla, bu terim ikili bağlılaşımların arıçarpımsallığını ölçmektedir. Daha çok bağımsız değişken içeren çarpanlar için de benzer yorumlar yapılabilir. Ancak, burada bu seviyeye kadar bilgilendirim ile yetinilecektir.

Tıpkı YBMG'de olduğu gibi ÇYBMG'de de bazı ölçenler tanımlanabilir. Bu amaçla, eğer,

$$\varphi_i(x) \equiv 1 + \frac{f_i(x_i)}{f_0}, \qquad 1 \leq i \leq N \tag{2.38}$$

tanımı yapılırsa

$$\pi_0 \equiv \frac{\|f_0\|^2}{\|f\|^2} \tag{2.39}$$

$$\pi_1 \equiv \frac{\left\| f_0 \prod_{i=1}^{N} \varphi_i(x_i) \right\|^2}{\|f_0\|^2} \tag{2.40}$$

anlatımında iki "Çarpımsallık Ölçeni"tanımlanabilir. Bunlardan ilkini "ÇYBMG Değişmezlik Ölçeni"olarak adlandırmak olanaklıdır. Bu ölçen (π_0) YBMG'nin Değişmezlik Ölçeni'ne (σ_0) eşdeğerdir. Buradaki diğer olçen, π_1 ise YBMG'nin Arıtoplamsallık Ölçeni'ni çağrıştırır ve "ÇYBMG Arıçarpımsallık Ölçeni"olarak adlandırılabilir. $\pi_0 = 1$ durumu Değişmezliğe, $\pi_1 = 1$ durumu ise Arıçarpımsallığa karşılık gelir. Ancak, YBMG Ölçenleri durumundaki tekdüze artma ve üstten sınırlı olma durumu buradaki ÇYBMG Ölçenleri için yoktur. Salt bu özelliği ÇYBMG'nin etkinliğini çok önemli ölçüde azaltmakta ve ilgilenilen işlevin yapısal olarak değil de verisel olarak verilmesi durumunda güvenilirliğini sağlayamamaktadır. Ancak, yine de, ilgilenilen işlevin arıçarpımsallığının çok baskın olduğunun bilindiği durumlarda, ÇYBMG'den yararlanmak, onu YBMG'ye göre öngörüme çıkarmak olanaklıdır.

2.1.2 Melez YBMG (MYBMG)

Bu kesimde çok değişkenli işlevlerde kullanılan YBMG yöntemi için sıradan YBMG ve ÇYBMG'den değişik bir türü anlatılacaktır. Bu tür YBMG için verilen çok boyutlu veriler YBMG'ye uygun olarak ne arı ya da baskın toplamsal ne de ÇYBMG'ye uygun olarak arı ya da baskın çarpımsal bir yapı içerirler. Bu tür için verilen bir veriye karşılık YBMG ve ÇYBMG gösterim türlerini birlikte içeren bir gösterilim biçimi olan melez bir yapı ortaya konacaktır. Bu yapı aşağıdaki anlatımla verilebilir.

$$\begin{aligned} f(x_1,...,x_N) &= \gamma \left(f_0 + \sum_{i_1=1}^{N} f_{i_1}(x_{i_1}) + \cdots \right) \\ &+ (1-\gamma) \left(r_0 \left[\prod_{i_1=1}^{N} (1 + r_{i_1}(x_{i_1})) \right] \times \cdots \right) \end{aligned} \tag{2.41}$$

Aslında, burada, sağ yanda, birbirine özdeş ve biri toplamsal diğeri çarpımsal nitelikli olan iki anlatım ağırlıklı olarak devreye sokulmaktadır. Bu durum, γ'nın $[0,1]$

aralığında bulunması gerektiği anlamına gelmektedir ve onun değeri ilgilenilen işlevin toplamsallık yüzdesini, 1'e olan uzaklığı ise çarpımsallık yüzdesini gösterecektir. Bu anlatım, YBMG ile ÇYBMG birleştirimi niteliğindedir.

Burada, işlevin sayısal veriler üzerinden verilmesi durumunda, veriler kartezyen çarpımla oluşturulan bir ızgaranın (ing: grid) tüm düğüm noktalarında işlev değerlerini belirli kılıyorsa ve işlev buradaki gibi melez yapıda ise MYBMG'den yararlanılabilir.

(**2.41**)'i kullanarak MYBMG için kesme yaklaştırımları üretilebilir. Bunun için (**2.41**)'de YBMG ve ÇYBMG anlatımları yerine onların değişik kertelerden (mertebe) olabilen kesme yaklaştıranları kullanılabilir. Böylece aşağıdaki yaklaştıranlar üretilebilir.

$$f(x_1,...,x_N) \approx h_{j,k}(x_1,...,x_N;\gamma) \equiv \gamma s_j(x_1,...,x_N) + (1-\gamma)\pi_k(x_1,...,x_N), \quad 1 \le j,k \le N \tag{2.42}$$

Bu yaklaştıran "(j,k). Kerte'den Melez YBMG Yaklaştıranı"olarak adlandırılabilir. Burada gözüken s_j ve π_k işlevleri daha önceki kesimlerde tanımlanmış j'ninci YBMG yaklaştırım büyüklükleridir. Melez YBMG kesme yaklaştırımlarında kullanılan yaklaştıranlar aşağıdaki gibi bir çizelge yapısında gösterilebilirler.

$$\begin{matrix} h_{0,0} & h_{0,1} & \cdots & h_{0,N} \\ h_{1,0} & h_{1,1} & \cdots & h_{1,N} \\ \vdots & \vdots & \ddots & \vdots \\ h_{N,0} & h_{N,1} & \cdots & h_{N,N} \end{matrix} \tag{2.43}$$

Buradaki en önemli adım γ melezlik değiştirgesinin (ing: parameter) saptanmasıdır. Bu amaç için açağıdaki biçimde bir işlev tanımlayabiliriz.

$$F(x_1,...,x_N) \equiv \|f\|^2 - \|f_{MYBMG}(\gamma)\|^2 \tag{2.44}$$

Burada f ve $f_{MYBMG}(\gamma)$ sırasıyla verilen işlev ile MYBMG'nin bu işleve yaklaştıranını göstermekte olup, olması gerektiği gibi, boylardaki işlevlerde bağımsız değişkene olan bağımlılık açık olarak gösterilmemektedir. γ değerini saptamak için bu eşitliğin sol yanındaki işlevin boy değerinin en küçük kılınması gerekmektedir.

$$\frac{\partial F}{\partial \gamma} = 0 \tag{2.45}$$

Bu γ'ya göre eniyilenen işlevin oluşturumunda olsun YBMG ile ÇYBMG yaklaştıranlarının oluşturumunda olsun ayrı ayrı değişik ağırlık işlevleri kullanmayı engelleyen bir

kısıtlama söz konusu değildir. Hangi ağırlık işlevinin nerede kullanılacağına ilgilenilen sorunun yapısına göre karar verilir. Sorun YBMG'si baskın ise YBMG ağırlık işlevi, diğer durumda ise, EYBMG ağırlık işlevi kullanılmalıdır. Ama, en iyisi, hepsi için ortak bir ağırlık kullanmaktır. Bu ortak ağırlık, işlevin yapısal olarak verilmesi durumunda, ilgili tanımbölgesinin her noktasında işlev değeri bilineceğinden, ağırlık işlevini tüm tanımbölgesi içinde sürekli seçmek olanaklıdır. Ancak, buradaki amacın, daha çok, tanımbölgesi içinde sonlu sayıda yinelemesiz noktada işlev değerlerinin veri olarak verilmesi ve başka bir bilginin elde bulunmadığı durumlarla ilgilenmek olmasından dolayı ağırlık işlevini de, tümlevleme altında, salt bu noktalarda işlev değeri üretecek yapıda yani ayrık yapılı seçmek gerekmektedir. Bir değişkenli ağırlık çarpanlarının YBMG için gerekli yapıyı üretebilmesi için aşağıdaki anlatımla verilen Dirac delta işlevleri doğrusal birleştirimi olarak seçilmesi gerekir.

$$W_i(x_i) \equiv \sum_{j}^{n_i} \alpha_j^{(i)} \delta\left(x_i - \xi_i^{(j)}\right), \qquad 1 \le i \le N \tag{2.46}$$

Burada, $\xi_i^{(j)}$ x_i doğrultusundaki, artan yönde sıralanmış toplam n_i sayıda düğüm noktasından j. konumda bulunanı simgelenmekte olup $\alpha_j^{(i)}$ bu düğüm noktasının hangi ağırlıkla gündeme getirildiğini simgeleyen değiştirge durumundadır. Bu değiştirgeler bu ağırlık çarpanının tümlev birimlendiriminin gerçekleştirilmesi amacıyla aşağıdaki 1'e toplanırlık özelliğini taşımalıdır.

$$\sum_{j}^{n_i} \alpha_j^{(i)} = 1, \qquad 1 \le i \le N \tag{2.47}$$

Son iki bağıntıdan yararlanarak

$$F(x_1,...,x_N;\gamma) = \sum_{j_1=1}^{n_1} \cdots \sum_{j_N=1}^{n_N} \left[\prod_{i_1=1}^{N} \alpha_{j_{i_1}}^{(i_1)}\right] \times \left[f\left(\xi_1^{(j_1)},...,\xi_N^{(j_N)}\right) - \gamma s_j\left(\xi_1^{(j_1)},...,\xi_N^{(j_N)}\right) - (1-\gamma)\varphi_k\left(\xi_1^{(j_1)},...,\xi_N^{(j_N)}\right)\right]^2,$$

$$0 \le j,k \le N \tag{2.48}$$

Bu eşitlikle verilen işlev daha doğrusu eşitliğin sağ yanı enküçüklenirse aranan γ değeri elde edilir.

$$\gamma_{j,k}^{(ek)} = \frac{A_2 + A_3 - A_4 - A_5}{A_1 + A_2 - 2A_5}, \qquad 1 \le j,k \le N \tag{2.49}$$

Bu eşitliğin sağ yanında görünen değiştirgelerin açık anlatımları aşağıda verilmektedir.

$$
\begin{aligned}
A_1 &= \sum_{j_1=1}^{n_1} \cdots \sum_{j_N=1}^{n_N} \left[\prod_{i_1=1}^{N} \alpha_{j_{i_1}}^{(i_1)} \right] s_j \left(\xi_1^{(j_1)}, ..., \xi_N^{(j_N)} \right)^2, \\
A_2 &= \sum_{j_1=1}^{n_1} \cdots \sum_{j_N=1}^{n_N} \left[\prod_{i_1=1}^{N} \alpha_{j_{i_1}}^{(i_1)} \right] \varphi_k \left(\xi_1^{(j_1)}, ..., \xi_N^{(j_N)} \right)^2, \\
A_3 &= \sum_{j_1=1}^{n_1} \cdots \sum_{j_N=1}^{n_N} \left[\prod_{i_1=1}^{N} \alpha_{j_{i_1}}^{(i_1)} \right] f \left(\xi_1^{(j_1)}, ..., \xi_N^{(j_N)} \right) s_j \left(\xi_1^{(j_1)}, ..., \xi_N^{(j_N)} \right), \\
A_4 &= \sum_{j_1=1}^{n_1} \cdots \sum_{j_N=1}^{n_N} \left[\prod_{i_1=1}^{N} \alpha_{j_{i_1}}^{(i_1)} \right] f \left(\xi_1^{(j_1)}, ..., \xi_N^{(j_N)} \right) \varphi_k \left(\xi_1^{(j_1)}, ..., \xi_N^{(j_N)} \right), \\
A_5 &= \sum_{j_1=1}^{n_1} \cdots \sum_{j_N=1}^{n_N} \left[\prod_{i_1=1}^{N} \alpha_{j_{i_1}}^{(i_1)} \right] s_j \left(\xi_1^{(j_1)}, ..., \xi_N^{(j_N)} \right) \varphi_k \left(\xi_1^{(j_1)}, ..., \xi_N^{(j_N)} \right)
\end{aligned}
$$

$$
0 \leq j,k \leq N \tag{2.50}
$$

Böylece, her bir MYBMG yaklaştıranı için değişik olan γ değerleri elde edilebilir ve bunların kullanımıyla (**2.43**) çizelgesi yerine

$$
\begin{array}{cccc}
h_{0,0}\left(\mathbf{x}, \gamma_{0,0}^{(ek)}\right) & h_{0,1}\left(\mathbf{x}, \gamma_{0,1}^{(ek)}\right) & \cdots & h_{0,N}\left(\mathbf{x}, \gamma_{0,N}^{(ek)}\right) \\
h_{1,0}\left(\mathbf{x}, \gamma_{1,0}^{(ek)}\right) & h_{1,1}\left(\mathbf{x}, \gamma_{1,1}^{(ek)}\right) & \cdots & h_{1,N}\left(\mathbf{x}, \gamma_{1,N}^{(ek)}\right) \\
\vdots & \vdots & \ddots & \vdots \\
h_{N,0}\left(\mathbf{x}, \gamma_{N,0}^{(ek)}\right) & h_{N,1}\left(\mathbf{x}, \gamma_{N,1}^{(ek)}\right) & \cdots & h_{N,N}\left(\mathbf{x}, \gamma_{N,N}^{(ek)}\right)
\end{array} \tag{2.51}
$$

olarak verilen “Eniyilenmiş MYBMG Yaklaştıranları Çizelgesi”elde edilir. Kuşkusuz, bu çizelgenin en altta ve en sağda bulunan ögesi kesin sonucu verecektir. Diğer ögelerin ürettiği yaklaştırımların niteliği bu yaklaştırana yaklaşıldıkça azalır ama buna karşın bilgisayım karmaşıklığı artar. Bu nedenle, yaklaştıranlardan altsırasayıları en çok 2 olanlarla yetinmek istenebilir ve bu istem akılcıdır. Yani, çizelgenin sol üst köşedeki (3×3) yapısındaki kesiştirimin en kullanışlı yaklaştıranları içerdiği rahatlıkla söylenebilir.

3. YBMG YÖNTEMİ İLE VERİ BÖLÜNTÜLEME

Sonlu sayıda değişik nokta üzerinde çok boyutlu içdeğerbiçim yürütülmesi gerektiğinden YBMG terimlerinden oluşan çok boyutlu bölgeyi, herhangi bir ek koşul ortaya koymadan, tüm uzaya genişletebiliriz. Böylece, her bir bağımsız değişkenin aralığını $(-\infty,\infty)$ olarak alabiliriz. Burada, içdeğerbiçim uygulanacak $f(x_1,...,x_N)$ işlevinin yapısının analitik olarak verilmediği durumlara odaklanılmaktadır. Bu durumda, bağımsız değişkenler üzerinden tanımlanmış Euclid uzayından sonlu sayıda nokta alınarak bu noktalardaki işlev değerlerinin verildiği varsayılmaktadır. Bu noktalar her bir bağımsız değişkenin örttüğü aralıkların kartezyen çarpımı olarak alınmaktadır. İşlevin değerlerinin ise bir ızgara üzerindeki düğüm noktalarının tümünde verildiği varsayılmakta, her bir boyuttaki noktasal değer kümelerinin tanımları aşağıda sunulmaktadır.

$$\mathscr{D}_j \equiv \left\{\xi_j^{(k_j)}\right\}_{k_j=1}^{k_j=n_j} = \left\{\xi_j^{(1)},...,\xi_j^{(n_j)}\right\}, \qquad 1 \le j \le N \tag{3.1}$$

Bunlardan kartezyen çarpımla oluşturulan noktalar kümesi

$$\mathscr{D} \equiv \mathscr{D}_1 \times \mathscr{D}_2 \times \cdots \times \mathscr{D}_N \tag{3.2}$$

eşitliğiyle tanımlamaktadır. Kartezyen çarpım ile oluşturulan $\mathscr{D}$ kümesinin açık anlatımı aşağıdaki gibidir:

$$\mathscr{D} \equiv \left\{\tau | \tau = (x_1,x_2,...,x_N), x_j \in \mathscr{D}_j, 1 \le j \le N\right\} \tag{3.3}$$

İçdeğerbiçim için oluşturulması gereken yapının $\mathscr{D}$ kümesinde yer alan noktalardaki $f(x_1,...,x_N)$ işlevinin aldığı değerleri içermesi gereklidir. Bu yapı, uygun bir ağırlık seçimiyle elde edilebilir. Bu amaçla gerekli olan eylem, MYBMG'de yaptığımız gibi, çeşitli Dirac Delta işlevlerinin doğrusal birleştirimlerinden oluşan bir ağırlık işlevi tanımlamaktır. Bunun için aşağıdaki ağırlık işlevi seçilebilir.

$$W_j(x_j) \equiv \sum_{k_j=1}^{n_j} \alpha_{k_j}^{(j)} \delta\left(x_j - \xi_j^{(k_j)}\right), \qquad x_j \in \left[a_j, b_j\right], \qquad 1 \le j \le N \tag{3.4}$$

Burada tanımlanan ağırlık işlevinin, tümlev birimlendirim koşulunu (**2.9**) sağladığı aşağıdaki gibi gösterilebilir.

$$\int_{a_j}^{b_j} dx_j \sum_{k_j=1}^{n_j} \alpha_{k_j}^{(j)} \delta\left(x_j - \xi_j^{(k_j)}\right) = \sum_{k_j=1}^{n_j} \alpha_{k_j}^{(j)}, \qquad 1 \leq j \leq N \tag{3.5}$$

Bu da aşağıdaki ilişkiyi, Dirac Delta işlevlerinin doğrusal birleştirim katsayılarındaki bir koşul olarak, bize verir.

$$\sum_{k_j=1}^{n_j} \alpha_{k_j}^{(j)} = 1, \qquad 1 \leq j \leq N \tag{3.6}$$

(**2.10**) ile elde edilen YBMG değişmez terimi tanımlayan eşitlikte (**3.4**) kullanılırsa aşağıdaki denklem elde edilir.

$$\begin{aligned} f_0 &= \int_{a_1}^{b_1} dx_1 \sum_{k_1=1}^{n_1} \alpha_{k_1}^{(1)} \delta\left(x_1 - \xi_1^{(k_1)}\right) \cdots \times \\ &\quad \times \int_{a_N}^{b_N} dx_N \sum_{k_N=1}^{n_N} \alpha_{k_N}^{(N)} \delta\left(x_N - \xi_N^{(k_N)}\right) f(x_1, ..., x_N) \end{aligned} \tag{3.7}$$

Dirac Delta işlevinin özelliğini de kullanarak tümlevleme işlemi devreye sokulursa YBMG değişmez terimi için aşağıdaki cebirsel ilişki elde edilir

$$f_0 = \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} \cdots \sum_{k_N=1}^{n_N} \left(\prod_{i_1=1}^{N} \alpha_{k_{i_1}}^{(i_1)} \right) f\left(\xi_1^{(k_1)}, ..., \xi_N^{(k_N)}\right) \tag{3.8}$$

Bu ilişki aşağıdaki yapıda yeniden düzenlenebilir

$$f_0 \equiv \sum_{\tau \in \mathscr{D}} \zeta(\tau) f(\tau) \tag{3.9}$$

Burada

$$\begin{aligned} \tau &= \left(\xi_1^{(k_1)}, ..., \xi_N^{(k_N)}\right), \qquad \zeta(\tau) = \alpha_1^{(k_1)} \cdots \alpha_N^{(k_N)}, \\ &\qquad 1 \leq k_j \leq n_j, \qquad 1 \leq j \leq N \end{aligned} \tag{3.10}$$

tanımları geçerlidir.

$f_m(x_m)$ birli YBMG terimini belirlemek için (**2.11**) tanımından yararlanabiliriz.

$$\begin{aligned} f_m\left(\xi_m^{(k_m)}\right) &= \int_{a_1}^{b_1} dx_1 W(x_1) \cdots \int_{a_{m-1}}^{b_{m-1}} dx_{m-1} W(x_{m-1}) \times \\ &\quad \times \int_{a_{m+1}}^{b_{m+1}} dx_{m+1} W(x_{m+1}) \cdots \int_{a_N}^{b_N} dx_N W(x_N) f(x_1, ..., x_N) - f_0 \end{aligned} \tag{3.11}$$

Eşitliğin sağ yanı aşağıdaki biçimde $N-1$ kat tümlev yapılı olarak yeniden yazılabilir.

$$\begin{aligned} f_m\left(\xi_m^{(k_m)}\right) &= \int_{a_1}^{b_1} dx_1 \sum_{k_1=1}^{n_1} \alpha_{k_1}^{(1)} \delta\left(x_1 - \xi_1^{(k_1)}\right) \cdots \times \\ &\times \int_{a_{m-1}}^{b_{m-1}} dx_{m-1} \sum_{k_{m-1}=1}^{n_{m-1}} \alpha_{k_{m-1}}^{(m-1)} \delta\left(x_{m-1} - \xi_{m-1}^{(k_{m-1})}\right) \times \\ &\times \int_{a_{m+1}}^{b_{m+1}} dx_{m+1} \sum_{k_{m+1}=1}^{n_{m+1}} \alpha_{k_{m+1}}^{(m+1)} \delta\left(x_{m+1} - \xi_{m+1}^{(k_{m+1})}\right) \cdots \times \\ &\times \int_{a_N}^{b_N} dx_N \sum_{k_N=1}^{n_N} \alpha_{k_N}^{(N)} \delta\left(x_N - \xi_N^{(k_N)}\right) f(x_1,...,x_N) - f_0 \end{aligned} \quad \textbf{(3.12)}$$

(**3.9**)'da verilen kapalı yapıya benzer olarak yukarıda bulunan anlatımı aşağıdaki biçimde yeniden yazabiliriz.

$$f_m\left(\xi_m^{(k_m)}\right) = \sum_{\tau_m \in \mathscr{D}^{(m)}} \zeta_m(\tau_m) f(\tau_m, \xi_m^{(k_m)}) - \sum_{\tau \in \mathscr{D}} \zeta(\tau) f(\tau) \quad \textbf{(3.13)}$$

Burada

$$\begin{aligned} \mathscr{D}^{(m)} &\equiv \left\{ \tau_m | \tau_m = (x_1,...,x_{m-1},x_{m+1},...,x_N), x_j \in \mathscr{D}_j, 1 \le j \le N, j \ne m \right\}, \\ \tau_m &= \left(\xi_1^{(k_1)},...,\xi_{m-1}^{(k_{m-1})},\xi_{m+1}^{(k_{m+1})},...,\xi_N^{(k_N)}\right), \\ \zeta_m(\tau_m) &= \alpha_1^{(k_1)} \cdots \alpha_{m-1}^{(k_{m-1})} \alpha_{m+1}^{(k_{m+1})} \cdots \alpha_N^{(k_N)}, \\ \xi_m^{(k_m)} &\in \mathscr{D}_m, \qquad 1 \le k_m \le n_m, \qquad 1 \le m \le N \end{aligned} \quad \textbf{(3.14)}$$

tanımları geçerlidir.

Elde ettiğimiz eşitlikle birlikte N sıralı terimlerden oluşan tekli (ing:univariate) çizelge üretilmiş bulunmaktadır. $f_m(x_m)$ için elde edilen m'inci çizelgede n_m $(1 \le m \le N)$ sayıda sıralı terim içerilmektedir.

$f_{m_1 m_2}(x_{m_1}, x_{m_2})$ yapısındaki YBMG ikili terimleri de benzer biçimde elde edilir. (**2.12**) eşitliğinden ve Dirac Delta işlevinin özelliğinden yararlanarak benzer anlatımlar

aşağıdaki gibi elde edilir.

$$
\begin{aligned}
f_{m_1m_2}(\xi_{m_1}^{(k_{m_1})},\xi_{m_2}^{(k_{m_2})}) &= \sum_{\tau_{m_1m_2}\in\mathscr{D}^{(m_1m_2)}} \zeta_{m_1m_2}(\tau_{m_1m_2})\, f\left(\tau_{m_1m_2},\xi_{m_1}^{(k_{m_1})},\xi_{m_2}^{(k_{m_2})}\right) \\
&\quad - \sum_{\tau_{m_1}\in\mathscr{D}^{(m_1)}} \zeta_{m_1}(\tau_{m_1})\, f\left(\tau_{m_1},\xi_{m_1}^{(k_{m_1})}\right) \\
&\quad - \sum_{\tau_{m_2}\in\mathscr{D}^{(m_2)}} \zeta_{m_2}(\tau_{m_2})\, f\left(\tau_{m_2},\xi_{m_2}^{(k_{m_2})}\right) \\
&\quad + \sum_{\tau\in\mathscr{D}} \zeta(\tau)\, f(\tau) \qquad \textbf{(3.15)}
\end{aligned}
$$

Burada

$$
\begin{aligned}
\mathscr{D}^{(m_1m_2)} &\equiv \left\{\tau_m | \tau_m = (x_1,...,x_{m_1-1},x_{m_1+1},...,x_{m_2-1},x_{m_2+1},...,x_N),\right. \\
&\qquad \left. x_j\in\mathscr{D}_j, 1\le j\le N, j\neq m_1,m_2\right\}, \\
\mathscr{D}^{(m_1)} &\equiv \left\{\tau_{m_1} | \tau_{m_1} = (x_1,\ldots,x_{m_1-1},x_{m_1+1},...,x_N),\right. \\
&\qquad \left. x_j\in\mathscr{D}_j, 1\le j\le N, j\neq m_1\right\}, \\
\mathscr{D}^{(m_2)} &\equiv \left\{\tau_{m_2} | \tau_{m_2} = (x_1,...,x_{m_2-1},x_{m_2+1},...,x_N),\right. \\
&\qquad \left. x_j\in\mathscr{D}_j, 1\le j\le N, j\neq m_2\right\}, \\
\tau_{m_1m_2} &= \left(\xi_1^{(k_1)},...,\xi_{m_1-1}^{(k_{m_1-1})},\xi_{m_1+1}^{(k_{m_1+1})},...,\xi_{m_2-1}^{(k_{m_2-1})},\xi_{m_2+1}^{(k_{m_2+1})},...,\xi_N^{(k_N)}\right), \\
\tau_{m_1} &= \left(\xi_1^{(k_1)},...,\xi_{m_1-1}^{(k_{m_1-1})},\xi_{m_1+1}^{(k_{m_1+1})},...,\xi_N^{(k_N)}\right), \\
\tau_{m_2} &= \left(\xi_1^{(k_1)},...,\xi_{m_2-1}^{(k_{m_2-1})},\xi_{m_2+1}^{(k_{m_2+1})},...,\xi_N^{(k_N)}\right),
\end{aligned}
$$

$$\begin{aligned}
\zeta_{m_1 m_2}(\tau_{m_1 m_2}) &= \alpha_1^{(k_1)} \cdots \alpha_{m_1-1}^{(k_{m_1-1})} \alpha_{m_1+1}^{(k_{m_1+1})} \cdots \alpha_{m_2-1}^{(k_{m_2-1})} \alpha_{m_2+1}^{(k_{m_2+1})} \cdots \alpha_N^{(k_N)}, \\
\zeta_{m_1}(\tau_{m_1}) &= \alpha_1^{(k_1)} \cdots \alpha_{m_1-1}^{(k_{m_1-1})} \alpha_{m_1+1}^{(k_{m_1+1})} \cdots \alpha_N^{(k_N)}, \\
\zeta_{m_2}(\tau_{m_2}) &= \alpha_1^{(k_1)} \cdots \alpha_{m_2-1}^{(k_{m_2-1})} \alpha_{m_2+1}^{(k_{m_2+1})} \cdots \alpha_N^{(k_N)}, \\
\xi_{m_1}^{(k_{m_1})} &\in \mathscr{D}_{m_1}, \quad \xi_{m_2}^{(k_{m_2})} \in \mathscr{D}_{m_2},
\end{aligned}$$

$$1 \le k_{m_1} \le n_{m_1}, \quad 1 \le k_{m_2} \le n_{m_2}, \quad 1 \le m_1, m_2 \le N \qquad \textbf{(3.16)}$$

tanımları geçerlidir. Böylece **(3.9)** ve **(3.13)**'i verilmiş veriler üzerinde kullanarak çok değişkenli işlevin analitik yapısını elde edecek içdeğerbiçim için gerekli birli ve ikili çizelgeler elde edilmiş olmaktadır.

3.0.3 İçdeğerbiçim (ing:Interpolation)

YBMG yardımıyla verilmiş veriyi bölüntülere ayırmakla $f_m(x_m)$ için analitik bir yapı yerine n_m sıralı ikilisinden (ing: ordered pair) oluşmuş bir çizelge saptanmıştır. Bu çizelge, işlev için varsayılan bir yapı altında $f_m(x_m)$ terimini saptamayı olanaklı kılar. Bu da içdeğerbiçim yaklaştırımıdır (ing: interpolation approximation). Bu yolla, tek bir çok değişkenli içdeğerbiçim, N sayıda tek değiskenli içdeğerbiçim kümesine indirgenerek yaklaştırım yapılır. Bu tür bir yaklaştırım için işlev değeri olarak YBMG birli terimleri üzerinde, genel eğilimi izleyip çokterimli yapısı varsayarak, Lagrange İçdeğerbiçimi kullanılabilir.

$$p_m(x_m) = \sum_{k_m=1}^{n_m} L_{k_m}(x_m) f_m\left(\xi_m^{(k_m)}\right), \qquad \xi_m^{(k_m)} \in \mathscr{D}_m, \qquad 1 \le m \le N \qquad \textbf{(3.17)}$$

Buradaki $L_{k_m}(x_m)$'ler Lagrange çokterimli katsayılarıdır. Bu çokterimli aşağıdaki biçimde tanımlanır.

$$\begin{aligned}
L_{k_m}(x_m) &\equiv \prod_{\substack{j=1 \\ j \ne k_m}}^{n_m} \frac{\left(x_m - \xi_m^{(j)}\right)}{\left(\xi_m^{(k_m)} - \xi_m^{(j)}\right)}, \qquad \xi_m^{(k_m)} \in \mathscr{D}_m, \\
& 1 \le k_m \le n_m, \quad 1 \le m \le N \qquad \textbf{(3.18)}
\end{aligned}$$

YMBG terimleri kullanılarak oluşturulan (**3.17**) ve (**3.18**) eşitlikleri kullanılarak çok değişkenli bir $f(x_1,...,x_N)$ işlevine aşağıdaki gibi bir yaklaştırım oluşturulabilir.

$$f(x_1,...,x_N) \approx f_0 + \sum_{m=1}^{N} p_m(x_m) \tag{3.19}$$

$$\begin{aligned} p_{m_1 m_2}(x_{m_1}, x_{m_2}) &= \sum_{k_{m_1}=1}^{n_{m_1}} \sum_{k_{m_2}=1}^{n_{m_2}} L_{k_{m_1}}(x_{m_1}) L_{k_{m_2}}(x_{m_2}) f_{m_1 m_2}(\xi_{m_1}^{(k_{m_1})}, \xi_{m_2}^{(k_{m_2})}), \\ & \xi_{m_1}^{(k_{m_1})} \in \mathscr{D}_{m_1}, \quad \xi_{m_2}^{(k_{m_2})} \in \mathscr{D}_{m_2}, \quad 1 \le m_1, m_2 \le N \end{aligned} \tag{3.20}$$

$$f(x_1,...,x_N) \approx f_0 + \sum_{m=1}^{N} p_m(x_m) + \sum_{\substack{m_1,m_2=1 \\ m_1<m_2}}^{N} p_{m_1,m_2}(x_{m_1}, x_{m_2}) \tag{3.21}$$

Böylece $f(x_1,...,x_N)$ çokdeğişkenli işlevi için içdeğerbiçim yaklaştırımı elde edilmiş olur.

4. KOŞUTLAŞTIRMA SÜRECİ İÇİN YBMG YÖNTEMİNİN İYİLEŞTİRİLMESİ

Bir önceki bölümde değinilen YBMG türlerinden ister YBMG ister ÇYBMG isterse de MYBMG veri bölüntüleme yöntemi kullanılsın asıl işlem yüküne sahip olan eşitlikler toplamsal YBMG'nin değişmez ve birli terimlerini bulmada kullanılan (**3.9**) ve (**3.13**) eşitlikleridir. Diğer tüm bilinmeyenler bu terimler yoluyla elde edildiğinden ve işlem ederleri düşük olduğundan koşutlaştırma süreci dışında tutulmuştur. Uygulamalarda esas olarak YBMG değişmez ve birli terimler kullanıldığından bu çalışmada esas olarak (**3.9**) ve (**3.13**) eşitlikleri koşutlaştırılacaktır.

Koşutlaştırma işlemine başlamadan önce YBMG değişmez ve birli terimleri hesaplayan (**3.9**) ve (**3.13**) eşitlikleri incelenecek olursa, eşitliklerin yapıları ele alınan problemdeki boyut sayısına göre değişkenlik gösterdigi görülür. YBMG terimlerini bulmada kullanılan algoritmadaki döngü sayısı ele alınan problemdeki boyut sayısına eşittir. Koşutlaştırma işlemine geçmeden önce bu problemi aşmak için ele alınan eşitliklerin matematiksel olarak ne anlattıkları ortaya konulup bunlar üzerinde iyileştirme yapılmalıdır. Bu amaçla ilk olarak bu eşitliklerin matematiksel yapısını ortaya çıkarmada kullanılan kavramlara değinilmelidir.

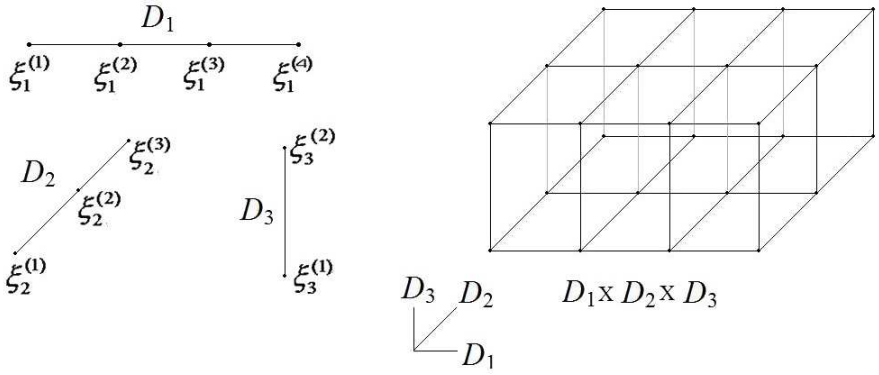
4.1 Kullanılan Matematiksel Kavramlar

(**3.9**) ve (**3.13**) eşitliklerinin matematiksel yapılarını açıklamak için örnek bir noktalar kümesi ele alınmıştır. Örnek noktalar kümesi 3 boyutlu olarak seçilmiştir ve (**3.1**), (**3.2**) ve (**3.3**) tanımlarından yararlanarak

$$\mathscr{D} = \mathscr{D}_1 \times \mathscr{D}_2 \times \mathscr{D}_3 = \left\{\xi_1^{(1)}, \xi_1^{(2)}, \xi_1^{(3)}, \xi_1^{(4)}\right\} \times \left\{\xi_2^{(1)}, \xi_2^{(2)}, \xi_2^{(3)}\right\} \times \left\{\xi_3^{(1)}, \xi_3^{(2)}\right\} \quad \textbf{(4.1)}$$

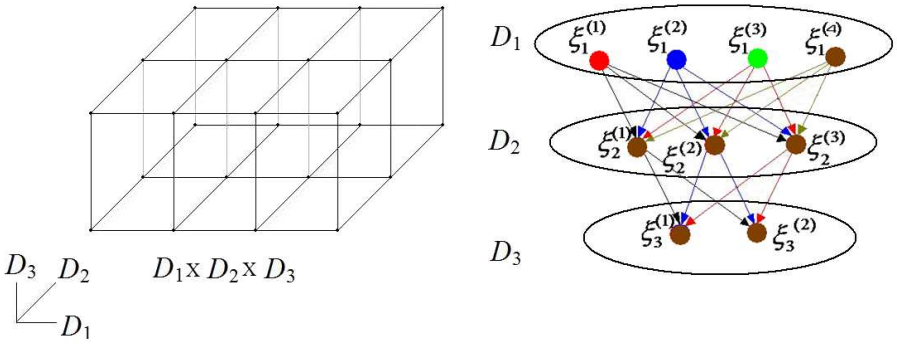
şeklinde gösterilir. Dolayısıyla birinci boyutu 4 düğüm noktasından, ikinci boyutu 3 düğüm noktasından ve üçüncü boyutu 2 düğüm noktasından oluşan bir örnek noktalar kümesi alınmıştır. örnek olarak alınan noktalar kümesi üzerinden çizge kanıtsavının (ing:*graph theory*) 3 kavramından bahsedilecektir. Bu kavramların birincisi Ağ

Kartezyen Çarpım (ing:*Grid Cartesian Product*) kavramıdır[13]. Bu matematiksel kavram yardımıyla örnek olarak aldığımız noktalar kümesi bir ağ kartezyen çarpım olarak aşağıdaki şekilde gösterilebilir.



Şekil 4.1: $G_{4,3,2}$, $\mathscr{D}_1 \times \mathscr{D}_2 \times \mathscr{D}_3$ için Ağ Kartezyen Çarpımı

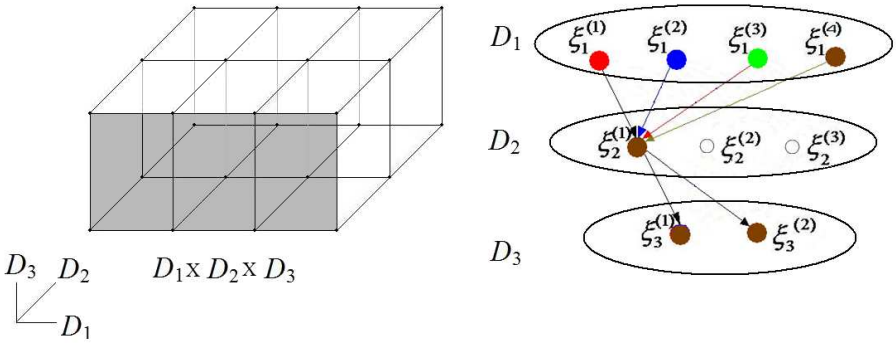
İkinci kavram Tam k-Ayrıştırılmış Çizgedir (ing:*Complete $k-$Partite Graph*)[14]. Bir *k*-tam ayrıştırılmış çizge *k* adet ayrı kümeden oluşmaktadır öyle ki küme içerisinde bulunan düğüm noktaları birbiri ile herhangi bir eşleşmeleri yoktur. Buna karşın sıralı halde bulunan bu kümelerdeki düğüm noktaları komşu kümelerdeki düğüm noktaları ile gerçekleşebilecek tüm eşleşmeleri yapmışlardır. Bu çalışmada düğüm noktaları arasında yönlü eşleşmeler (ing:*directed edges*) kullanılmıştır. Böylece herhangi bir ağ kartezyen çarpım bir tam ayrıştırılmış çizge olarak gösterilebilir. Şekil 4.2'de örnek olarak alınan kümenin ağ kartezyen çarpımının 3-Tam Ayrıştırılmış Çizge olarak görülmektedir.



Şekil 4.2: $G_{4,3,2}$ ağ kartezyen çarpımının $K_{4,3,2}$ Yönlü 3-Tam Ayrıştırılmış Çizge olarak anlatımı

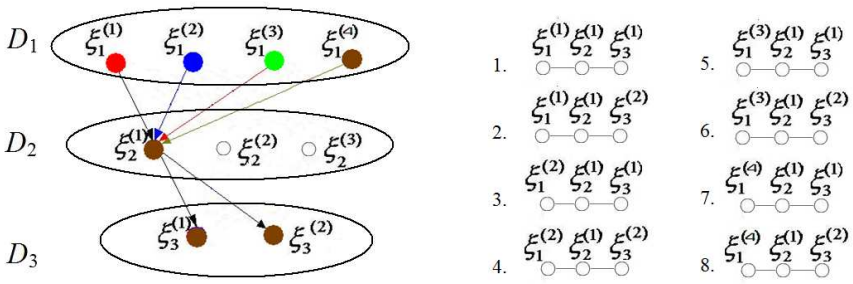
Eğer yukarıda bahsedilen ve düğüm noktaları arasındaki eşleşmelerin tümü mevcut değil ise bu kavram da *k*-Ayrıştırılmış Çizge (ing:*$k-$Partite Graph*) olarak tanımlanmaktadır [15]. Kartezyen çarpım ile oluşan noktalardan yalnızca ikinci boyutun birinci

düğüm noktasını $\xi_2^{(1)}$ içerenler Şekil 4.3'ün sol yanında işaretlenmiştir. Bunun çizelge gösterimi Şekil 4.3'un sağ yanındaki gibidir. Görüldüğü gibi $\mathscr{D}_2$ boyutunda yalnızca $\xi_2^{(1)}$ düğüm noktası ile yapılan yönlü eşleşmeler gösterilmiştir.



Şekil 4.3: $K_{4,1,2}$, $G_{4,1,2}$'nin Yönlü 3-Ayrıştırılmış Çizgesi

Ele alınacak son matematiksel kavram Yönlü Döngüsüz Çizgedir (ing:*Directed Acyclic Graph*). Adından da anlaşılacağı gibi çizge herhangi bir kesintisiz döngü içermiyor ise DAG olarak sınıflandırılır [16]. Sonlu bir DAG en azından bir kaynak (ing:*source*) ve bir bitim (ing:*sink*) noktasına sahip olmalıdır. Kaynak kendi üzerine herhangi bir yönlü eşleşmenin olmadığı, bitim ise kendinden herhangi bir noktaya herhangi bir yönlü eşleşmenin olmadığı noktalardır. Şekil 4.4'ün sol yanında dikkat edilecek olursa bu çizgenin 4 adet kaynak ve 2 adet bitim noktasına sahip olduğu görülür. Ayrıca bu çizge kesintisiz bir döngü sağlayacak yönlü eşleşmelere sahip değildir. Dolayısıyla bu bir DAG'dır. Ek olarak çizge eşit uzunlukta yönlü erişim yollarına (ing:*path*) sahiptir. Her bir erişim yolu $(\xi_1^{(1)}, \xi_2^{(1)}, \xi_3^{(1)})$ gibi örnek noktalar kümesi elemanlarını içeren sıralı üçlülerden oluşur. Her DAG, içerdiği noktalar arasında doğrusal bir topolojik sıralamaya sahiptir öyle ki $i < j$ durumunda v_i noktası v_j noktasından önce gelir. Fakat örnek olarak alınan noktalar kümesinde her bir nokta alt ve üst olmak üzere iki sırasayıya sahiptir. Bu nedenle ek bir sıralamaya daha gereksinim duyulmaktadır; $k < t$ olmak üzere $v_i^{(k)}$ noktası $v_i^{(t)}$ noktasından önce gelmektedir. Bu sıralama yardımıyla Şekil 4.4'deki DAG sıralı bir noktalar dizisi olarak gösterilebilir. Şekil 4.4'ün sağ yanında $G_{4,1,2}$'ın sıralı bir dizi olarak görülmektedir. Şekil 4.4'de görüldüğü gibi birinci erişim yolu hem alt hem de üst sırasayıları önceliğe sahip olduğundan $\xi_1^{(1)}$ ile başlar. $\xi_1^{(1)}$ in ikinci boyutta eşleştiği tek nokta $\xi_2^{(1)}$ dir. $\xi_1^{(1)}$'in de üçüncü boyutta eşleştiği iki nokta bulunmaktadır, $\xi_3^{(1)}$ ve $\xi_3^{(2)}$. Bu iki noktanın alt sırasayıları aynı küme içerisinde oldukları için aynıdır. Bu



Şekil 4.4: $G_{4,1,2}$'ın Yönlü Ayrıştırılmış Çizgesi ve bu çizgeyi anlatan sıralı üçlüler kümesi

nedenle sıralama için üst sırasayılara bakılır. Böylece $\xi_1^{(1)}$ ile başlayan $(\xi_1^{(1)},\xi_2^{(1)},\xi_3^{(1)})$, $(\xi_1^{(1)},\xi_2^{(1)},\xi_3^{(2)})$ sıralı üçlülerinin oluşturduğu düğüm noktaları sırasıyla dizinin ilk iki öğeleri olurlar. İkinci boyutta eşleme yapan tek nokta $\xi_2^{(1)}$ olduğundan yine birinci boyuttaki üst sırasayı nedeniyle ikinci öncelige sahip $\xi_1^{(2)}$ ile erişim yolu oluşturulmaya devam eder ve Şekil 4.4'de sağ yanda görülen biçimde sıralanırlar.

4.2 YBMG Değişmez ve Birli Terimlerin Veri Kullanım Şemaları

Yukarıdaki bölümde anlatılan matematiksel kavramlardan yola çıkarak herhangi sayıda boyuta ve boyut düğüm noktalarına sahip herhangi bir kartezyen çarpımın oluşturduğu çok boyutlu noktalar kümesinin tamamı ya da bir kesimi yönlü paylaştırılmış çizge olarak edilebilir. Buradan kartezyen çarpım kümesi öğeleri eşsiz bir sırada dizi gösterilebilir. Bu gösterimden yola çıkarak 3 boyutlu olarak aldığımız örnek noktalar kümesinin kartezyen çarpımı (**4.1**) sıralı dizi olarak eşsiz bir şekilde aşağıdaki gibi gösterilebilir.

$$
\begin{aligned}
\mathscr{D} &= \mathscr{D}_1 \times \mathscr{D}_2 \times \mathscr{D}_3 \\
&= \{(\xi_1^{(1)},\xi_2^{(1)},\xi_3^{(1)}),(\xi_1^{(1)},\xi_2^{(1)},\xi_3^{(2)}),(\xi_1^{(1)},\xi_2^{(2)},\xi_3^{(1)}),(\xi_1^{(1)},\xi_2^{(2)},\xi_3^{(2)}),\\
&\quad (\xi_1^{(1)},\xi_2^{(3)},\xi_3^{(1)}),(\xi_1^{(1)},\xi_2^{(3)},\xi_3^{(2)}),(\xi_1^{(2)},\xi_2^{(1)},\xi_3^{(1)}),(\xi_1^{(2)},\xi_2^{(1)},\xi_3^{(2)}),\\
&\quad (\xi_1^{(2)},\xi_2^{(2)},\xi_3^{(1)}),(\xi_1^{(2)},\xi_2^{(2)},\xi_3^{(2)}),(\xi_1^{(2)},\xi_2^{(3)},\xi_3^{(1)}),(\xi_1^{(2)},\xi_2^{(3)},\xi_3^{(2)}),\\
&\quad (\xi_1^{(3)},\xi_2^{(1)},\xi_3^{(1)}),(\xi_1^{(3)},\xi_2^{(1)},\xi_3^{(2)}),(\xi_1^{(3)},\xi_2^{(2)},\xi_3^{(1)}),(\xi_1^{(3)},\xi_2^{(2)},\xi_3^{(2)}),\\
&\quad (\xi_1^{(3)},\xi_2^{(3)},\xi_3^{(1)}),(\xi_1^{(3)},\xi_2^{(3)},\xi_3^{(2)}),(\xi_1^{(4)},\xi_2^{(1)},\xi_3^{(1)}),(\xi_1^{(4)},\xi_2^{(1)},\xi_3^{(2)}),\\
&\quad (\xi_1^{(4)},\xi_2^{(2)},\xi_3^{(1)}),(\xi_1^{(4)},\xi_2^{(2)},\xi_3^{(2)}),(\xi_1^{(4)},\xi_2^{(3)},\xi_3^{(1)}),(\xi_1^{(4)},\xi_2^{(3)},\xi_3^{(2)})\} \quad \textbf{(4.2)}
\end{aligned}
$$

Kartezyen çarpım sonucunda oluşan bu dizide 24 öğe bulunmaktadır. Bu öğelerden her biri, kartezyen çarpım sonucunda oluşan çok boyutlu noktaların koordinatlarına karşılık gelmektedir. Bu aşamadan sonra anlatım kolaylığı açısından çok boyutlu $f(x_1,x_2,x_3)$ işlevinin bu noktalardaki değerleri sırasıyla 1'den 24'e kadar giden sayılar şeklinde gösterilecektir. Bu gösterim biçimini kullanarak örnek uzaydaki noktalarda sayısal değerleri bilinen $f(x_1,x_2,x_3)$ çok boyutlu işlevinin YBMG terimleri hesaplansın.

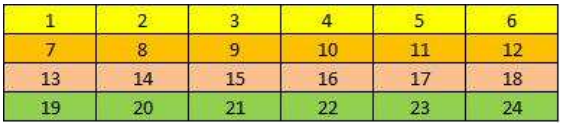
YBMG değişmez terimleri hesaplamakta kullanılan (**3.9**) eşitliği incelendiğinde bu gösterimin, $f(x_1,x_2,x_3)$ işlevinin (**4.2**)'de gösterilmiş 24 noktadaki değerlerinin ağırlıklar altında toplamına eşit olduğu görülür. (**3.6**) eşitliğinden ağırlıklar her bir boyut için aynı olduğundan burada işlev değerlerinin toplamı önem kazanmaktadır. Şekil 4.5'de YBMG değişmez terimleri f_0 hesaplanırken $f(x_1,x_2,x_3)$'in tüm noktalardaki sayısal değerlerinin kullandığı görülmektedir.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24

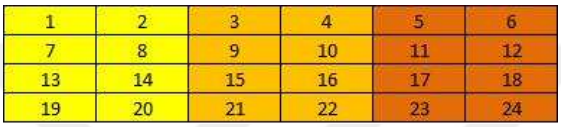
Şekil 4.5: YBMG değişmez terimi olan f_0'ı hesaplamak için kullanılan veri kullanım çizemi

Esas problem (**3.13**) eşitliği ile bulunan birli YBMG terimler hesaplanırken kullandıkları işlev değerlerinin çizemini çıkarmaktır. Fonksiyon değerleri kullanım çizemleri, seçilen boyut ve boyutlardaki düğüm noktası sayısına göre değişmektedir. YBMG terimlerinin matematiksel yapısını anlatmak için örnek olarak ikinci boyutun birinci düğüm noktasının YBMG terimi olan $f_2(\xi_2^{(1)})$ hesaplansın. $f_2(\xi_2^{(1)})$ terimi hesaplanırken kullanılan noktalar, kartezyen çarpım kümesinde ikinci boyutta yalnızca $\xi_2^{(1)}$ düğüm noktasını içeren noktalardır. Bu noktalarin seçimi Şekil 4.3'de sağ yanda bulunan ayrıştırılmış çizgede görülmektedir. Ayrıştırılmış çizgede $f_2(\xi_2^{(1)})$ hesaplanırken yalnızca $\xi_2^{(1)}$ içeren noktalardaki işlev değerleri kullanılır. Bu değerler de Şekil 4.6'daki sıralı işlev değer dizisinde 7. noktadan 12. noktalara kadar olan değerleri içeren dizi parçasına denk gelir. Diğer YBMG birli terimler bulunurken de strateji aynıdır; hangi boyutun düğüm noktasının YBMG terim değeri bulunmak isteniyorsa yalnızca o düğüm noktasını içeren noktalardaki işlev değeri kullanılır. Şekil 4.6, Şekil 4.7 ve Şekil 4.8 de birinci, ikinci ve üçüncü boyutlardaki düğüm noktalarının YBMG terim değerleri hesaplanırken kullanılan işlev değer kullanım çizemleri görülmektedir. Buradan çıkarılacak önemli bir sonuç da aynı boyutta bulunan farklı düğüm noktalarının

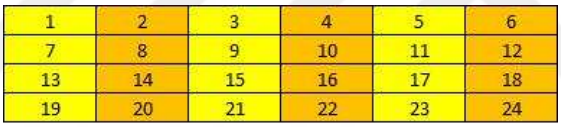
YBMG terim değerleri hesaplanırken farklı işlev değerleri kullanılır, birinin kullandığı veriyi diğeri kullanmaz. Şekil 4.6, Şekil 4.7 ve Şekil 4.8'de sırasiyla 1. boyuttaki, 2. boyuttaki ve 3. boyuttaki düğüm noktalarının YBMG terim değerleri hesaplanırken kullanılan işlev değerlerinin çizemi görülmektedir. Şekillerde görülen her bir renk grubu, farklı düğüm noktalarının hesabında kullanılan işlev değerlerini göstermektedir.



Şekil 4.6: Birinci boyuttaki düğüm noktalarına ait YBMG birli terimleri olan $f_1(\xi_1^{(n_1)})$'leri hesaplamak için kullanılan veri kullanım çizemi



Şekil 4.7: İkinci boyuttaki düğüm noktalarına ait YBMG birli terimleri olan $f_2(\xi_2^{(n_2)})$'leri hesaplamak için kullanılan veri kullanım çizemi



Şekil 4.8: Üçüncü boyuttaki düğüm noktalarına ait YBMG birli terimleri olan $f_3(\xi_3^{(n_3)})$'leri hesaplamak için kullanılan veri kullanım çizemi

Buraya kadar gelinen süreç özetlenirse, (**3.9**) ve (**3.13**) eşitlikleri ile hesaplanan değişmez ve birli YBMG terimlerinin matematiksel yapıları çeşitli matematiksel kavramlar yardımıyla belirlenmiştir. Bu yapıya göre terimler hesaplanırken aslında belirli noktalardaki işlev değerlerinin toplamı kullanılmaktadır. Her bir YBMG terimi, veri dizisinin çeşitli parçalarındaki değerleri toplayarak hesaplanır. Bu veri setinde hangi değerleri kullandıklarını gösteren çizemler yukarıda belirtilmiştir. Fakat bu çizemler bu haliyle yeterli değildir. Herhangi bir boyuttaki herhangi bir düğüm noktasının YBMG değeri hesaplanırken eldeki veri dizisinde hangi verilerin kullandığı özyineli bir ilişki ile verilmelidir. Yukarıdaki şekillerde gösterilen ve belirli noktalardaki işlev değerlerini içeren 24 öğeli veri dizisi göz önüne alınsın. Her bir YBMG terimi hesaplanırken bu dizi içerisinde belirli öğelerin değerlerinin kullanıldığı belirtilmiştir. Özyineli ilişkilerin çıkarımına 1. boyuta ait düğüm noktalarının YBMG hesabı ile başlanabilir. İlk boyutta 5 düğüm noktası olduğundan her bir düğüm noktası için bir

YBMG terimi bulunacaktır. Şekil 4.6'da her bir rengin farklı bir YBMG teriminin hesabında kullanılan veriler olduğu belirtilmişti. Şekil 4.6'da ilk satırda yer alan değerler $f_1(\xi_1^{(1)})$ hesabında kullanılan verilerdir. Bu veriler hep beraber bir veri dizi parçası olarak düşünülebilir. Şekil 4.6'da görüldüğü gibi her bir YBMG değeri hesaplanırken 1 tane dizi parçası kullanılmaktadır. Bu bilgi daha sonra kullanılacaktır. Her bir dizi parçası 6 adet veri içermektedir. Bu değer aslında bundan sonraki boyutlarda bulunan düğüm noktalarının sayılarının çarpımıdır. 2. ve 3. boyutlarda sırasıyla 3 ve 2 adet düğüm noktası vardı. Bunların çarpımı ile YBMG terimleri hesaplanan boyutta kullanılan dizi parçalarının uzunluğu elde edilmiş olur. Dizi parçasının baslangıcının hangi öğeden başlanacağının belirlenmesi daha kolaydır. Boyutun hangi düğüm noktasının YBMG değeri hesaplanıyorsa boyutun dizi parçalarının uzunluğu ile düğüm noktası sırasayısının çarpımı YBMG teriminin kullanacağı dizi parçasının başlangıcını belirler. Bu şekilde 1. boyuta ait düğüm noktalarının YBMG değerleri hesaplanır. 2. ve sonraki boyutlarda YBMG terimleri hesabı başka özyineli ilişkileri gerektirir. Bu ilişkilerin çıkarımı için 2. boyuta ait YBMG terimleri hesaplanırken kullanılan veri kullanım çizemi Şekil 4.7 gözönüne alınmalıdır. Dikkat edilecek olursa bu boyuttaki YBMG terimleri hesaplanırken tek bir dizi parçası kullanılmamaktadır. Bu boyutta bulunan 3 düğüm noktası için ayrı ayrı 4 adet veri parçası kullanılmıştır. Ayrıca bu veri parçaları arasında sabit bir uzaklık vardır. Dolayısıyla YBMG terimleri hesaplanırken dizi parçaları sayısı ve bu parçalar arasındaki uzaklığın özyineli olarak bulunması gerekmektedir. Dizi parçaları sayısı 1. boyut için 1 olarak alınmıştı. Herhangi bir boyuttaki bir düğüm noktasının YBMG değeri hesabında kullanılacak dizi sayısı, ondan önceki boyutlardaki düğüm noktaları sayılarının çarpımıdır. 2. boyut için dizi parçaları sayısı 1. boyuttaki düğüm noktası sayısına eşittir ve 4'dür. Bu sonuç Şekil 4.7'den de görülmektedir. Aynı zamanda 3. boyutta hesaplanan YBMG değerleri için de aynı hesaplama yapılacak olursa, 1. ve 2. boyuttaki düğüm noktaları sayıları çarpımından 12 sayısı elde edilir. Şekil 4.8'den de görüldüğü gibi her bir düğüm noktalasının YBMG değeri hesabında 12 adet veri parçası kullanılmıştır. Son olarak herhangi bir düğüm noktasının YBMG değeri hesabında kullanılan veri parçaları arasındaki uzaklığın özyineli ilişkisi elde edilmelidir. Herhangi bir boyuttaki düğüm noktalasa ait YBMG değeri hesabında kullanılan dizi parçaları arasındaki uzaklık, dizinin toplam uzunluğunun ondan önceki boyutlardaki düğüm noktaları sayıları çarpımından elde edilen sayıya bölümüyle bulunur. Bölen değer 1. boyut için özel olarak 1 olarak

tanımlanmıştır. 2. boyuttaki herhangi bir düğüm noktası için bu hesap kolayca yapılabilir. Dizideki toplam öğe sayısı 24 idi. 1. boyutta 4 düğüm noktası vardı. 24/4 den iki dizi parçası baslangıçları arası uzaklık 6 olarak bulunur. Şekil 4.7 incelenecek olursa $f_2(\xi_2^{(1)})$ hesap edilirken kullanılan ilk dizi parçası 1. öğeden başlamaktadır. Aynı YBMG teriminin hesabında kullanılan ikinci dizi parçasının başlangıcı 7. öğedir. Dolayısıyla aralarındaki uzaklık 6'dır. Benzer hesap 3. boyut için de yapılabilir. Bu boyutta bir düğüm noktasının YBMG değeri hesabında kullanılan dizi parçaları başlangıç uzaklıkları $24/(4 \cdot 3) = 2$ olarak bulunur. Gerçekten de Şekil 4.8 incelenirse kurulan özyineli ilişkinin doğru olduğu görülür.

Şimdiye kadar elde edilen YBMG terimlerin veri kullanım çizemleri ve çıkarılan özyineli ilişkiler gözönüne alındığında (**3.9**) ve (**3.13**) eşitlikleri kolayca hesaplanabilir. Her iki eşitlik de $O(N^2)$ hesaplama karmaşıklığına sahip olacak şekilde daha verimli bir biçimde yeniden yazılabilir. Her iki eşitlik de iç içe 2 döngü içerir. Dıştaki döngü toplanması gereken veri parçalarının başlangıç noktalarını denetlerken iç döngü veri parçaları içindeki öğelerin toplanmasından sorumludur. Bu şekilde tüm YBMG terimleri elde edilmiş olur.

5. YBMG Yönteminin MPI ile Koşutlaştırılması

Önceki bölümde HDMR terimlerini hesaplamakta kullanılan (**3.9**) ve (**3.13**) eşitlikleri koşutlaştırma amacıyla iyileştirilmişti. Bu aşamadan sonra koşutlaştırmada üzerinde durulacak süreç, hesaplamalarda kullanılan ve belli bir özyineli ilişki içeren dizi parçaları içindeki sayısal değerlerin başarımlı bir şekilde toplanmasını sağlamaktır. Bu süreç anlatılmadan önce verinin hesaplama düğümlerine (ing:node) paylaştırma stratejisi anlatılmalıdır.

Şekil 4.6, Şekil 4.7 ve Şekil 4.8 incelenecek olursa veri dizisinin neden 6'şarlı parçalar halinde satır satır bölündüğü açıklanmıştı. Dikkat edilecek olursa dizi parçaları bu biçimde bölünürse her bir parçanın YBMG terimleri hesabında veri kullanım çizemi tıpatıp aynıdır. Dolayısıyla her bir parçaya düşen iş yükü eşittir. Bu bölüm şekli rastlantısal değildir. Herhangi bir veri dizisi 1. boyuttaki düğüm noktası sayısı kadar eşit parçaya ayrıldığında her zaman bu sonuç elde edilir. Elde edilen veri parçalarının YBMG terimleri hesabında kullandığı veri çizemi ve iş yükü aynı olacaktır. Veri dizisini bu şekilde bölmek eşsiz değildir. Sıranın korunması koşulu ile boyutlardaki düğüm noktası sayılarının çarpımından elde edilen sayılar da veri dizisini aynı özelliklerde böler. Örnek veri dizisinde 1. boyuttaki düğüm noktası sayısı 4 idi. 2. boyuttaki düğüm noktası sayısı olan 3 ile çarpıldığında 12 sayısı elde edilir. Veri dizisi 12 eşit satıra bölünürse aynı şekilde her bir parçanın YBMG terimleri hesabında veri kullanım çizemi ve iş yükü aynı olur.

Yukarıda anlatıldığı şekilde veri dizisi eşit parçalara ayrılır ve hesaplama düğümlerine olabildiğince eşit sayida paylaştırılır. Böylece her bir hesaplama düğümlerine düşen veri hacmi ve iş yükü eşit olur. Dolayısıyla load balancing elde edilmiş olur. Bu tür paylaştırımda dikkat edilecek olursa herhangi bir hesaplama düğümü tek başına herhangi bir YBMG terimini hesaplayamaz. Tek bir YBMG teriminin hesabı tüm hesaplama düğümlerine bağlı olarak gerçekleşir. Bu durumda her bir hesaplama düğümünün hesapladığı değeri bir araya getirecek bir yapıya gereksinim vardır. Bu gereksinime karşılık veren *MPI_Reduce*() işlevidir[17]. Bu işlev her bir hesaplama düğümündeki sonucu alır ve seçilen matematiksel işlemi uygulayarak

Sunucu Hesaplama Düğümü:	**İstemci Hesaplama Düğümleri:**
1. Kendi üzerine düşen veri satır sayısını hesapla **2.** Her bir boyut için dizi parçası sayısı, genişliği ve dizi parçalarının birbirlerinden olan uzaklıkları hesapla **3.** MPI I/O ile kendine düşen veriyi çek **4.** Çektiği verilerden HDMR değişmez ve birli terimleri hesapla **5.** MPI_Reduce() kullanarak istemci hesaplama düğümlerinde hesaplanan HDMR terimleri değerlerini toplayarak birleştir.	**1.** Kendi üzerine düşen veri satır sayısını hesapla **2.** Her bir boyut için dizi parçası sayısı, genişliği ve dizi parçalarının birbirlerinden olan uzaklıkları hesapla **3.** MPI I/O ile kendine düşen veriyi çek **4.** Çektiği verilerden HDMR değişmez ve birli terimleri hesapla **5.** MPI_Reduce() ile hesaplanan HDMR terimleri değerlerini sunucu hesaplama düğümüne yolla.

Çizelge 5.1: Yöntemi MPI kitaplığı ile koşutlaştırma süreci

istenilen hesaplama düğümüne sonucu aktarır. Burada seçilen matematiksel işlem toplama işlemidir. Her hesaplama düğümü, belirli bir YBMG teriminin kendisine düşen toplamını hesaplar. $MPI\_Reduce(MPI\_SUM)$ ile sonuçlar toplanır ve sunucu hesaplama düğümüne aktarılır. Böylece YBMG terimlerinin değerini veren sonuç elde edilmiş olur.

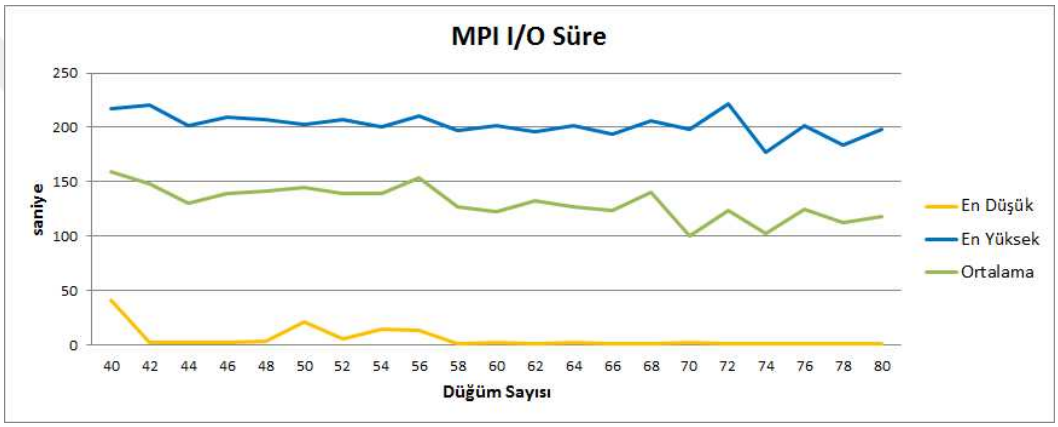
Çizelge 5.1'de adım adım açıklanan koşutlaştırma sürecinde sunucu ve istemci hesaplama düğümlerindeki iş adımları görülmektedir. İlk olarak tüm hesaplama düğümleri kendi üzerlerine düşen dizi satır sayılarını hesaplar. Daha sonra her bir boyut için hesaplanması gereken dizi parçaları uzunlukları, sayıları ve birbirlerinden olan uzaklıklar hesaplanır. Sonrasında MPI Parallel I/O kullanılarak veriler hesaplama düğümlerine aynı anda aktarılır. Sonrasında her bir hesaplama düğümü kendisine düşen veri parçasından YBMG terimlerini hesaplar. Bu işlem tamamlandıktan sonra $MPI\_Reduce(sum)$ sayesinde tüm sonuçlar toplanarak nihai sonuç sunucu hesaplama düğümüne aktarılır. Değişmez ve birli YBMG terimlerin hesapları bu şekilde tamamlanmış olur.

5.1 Oluşturulan MPI Algoritmasının Başarım Analizi

Yapılan başarım analizi her biri Intel(R) Xeon(TM) 2.66 GHz CPU'ya sahip, dağıtık bellek mimarisine sahip ve her bir düğümde 8 core ve 16GB belleğe sahip InfiniBand 20 Gbps bağlantılı bir sistem üzerinde yapılmıştır. Başarımı analiz etmede her bir test adımı en az 5 kez yapılmıştır. Bu testler sonucunda düğümlerde harcanan süre *En Düşük*, *En Yüksek* ve *Ortalama* olarak sınıflandırılmıştır. *En Düşük*, düğüm noktalarında

ilgili hesaplama için harcanan en kısa süreyi, *En Yüksek* en uzun süreyi, *Ortalama* da tüm düğüm noktalarında harcanan sürenin ortalamasını ifade etmektedir. Analiz için kullanılan tüm çizimlerde sarı renk *En Düşük*, mavi renk *En Yüksek* ve yeşil renk te *Ortalama* sınıflandırmaları için kullanılmıştır. Başarım analizi için yaklaşık 70GB'lık hacme sahip 6 boyutlu bir veri kullanılmıştır. Analizin düğüm başlangıç sayısı, verinin hacmi gözönüne alınarak bellek aşımı hatasına karşın 40 olarak belirlenmiştir.

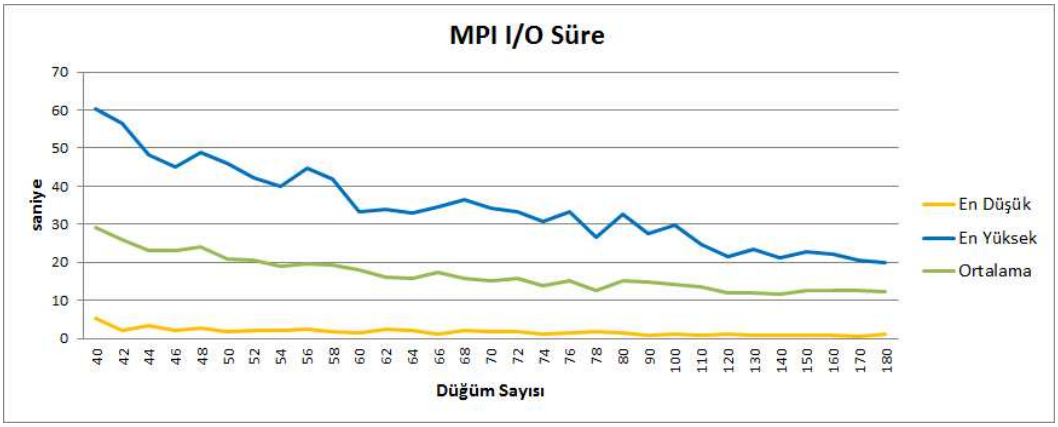
Başarım analizi için öncelikli olarak MPI I/O incelenecektir. Bu kısımda yaşanabilecek gecikmeler tüm algoritmanın başarımını etkileyecek düzeyde olacaktır. Analiz yapılan dosyanın MPI I/O ile düğümler tarafından okunma süreleri aşağıdaki Şekil 5.1'de gösterilmektedir.



Şekil 5.1: MPI algoritmasının MPI I/O süreleri

Şekilden görülebileceği gibi düğümlerde harcanan *En Düşük* ve *En Yüksek* süreler arasında çok büyük farklar vardır. Ek olarak, düğüm noktası artmasına rağmen dosyanın tüm düğüm noktaları tarafından okunma süresi azalmamakta ve sabit kalmaktadır. Bunun nedeni kullanılan sistemde varsayılan dosya parçalama sayısının (ing:*stripe*) 1 olarak ayarlanmış olmasıdır. Dosya tek bir parça üzerinde olduğundan dosya sunucusu ancak bir adet istemciye cevap verebilmektedir. Dolayısıyla düğüm noktası ne kadar artarsa artsın MPI I/O süresi sabit kalmaktadır ve okuma 200 saniye civarında seri olarak gerçekleşmektedir. Dosya, test edilen sistem tarafından izin verilen en yüksek sayı olan 9 parçaya bölünmüştür. Bu parçalama sonrası düğümlerde geçen MPI I/O süresi Şekil 5.2'deki gibidir.

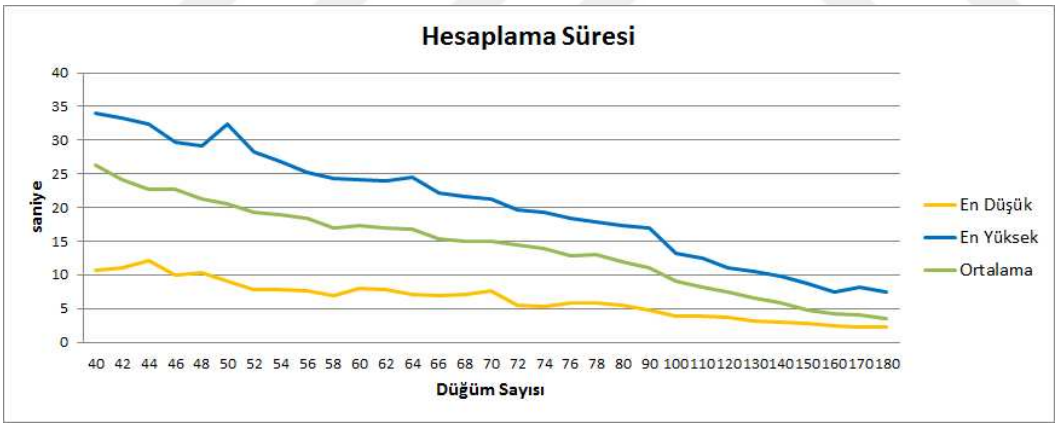
Şekil 5.2'den görüldüğü gibi *En Düşük* ile *En Yüksek* süreler arasında fark olmasına karşın önceki süreler kadar yüksek değildir. Asıl önemli ilerleme, düğüm sayıları arttıkça MPI I/O için geçen süre artık gerektiği gibi azalmaktadır. MPI I/O sürecinde



Şekil 5.2: MPI algoritmasının iyileştirme sonrası MPI I/O süreleri

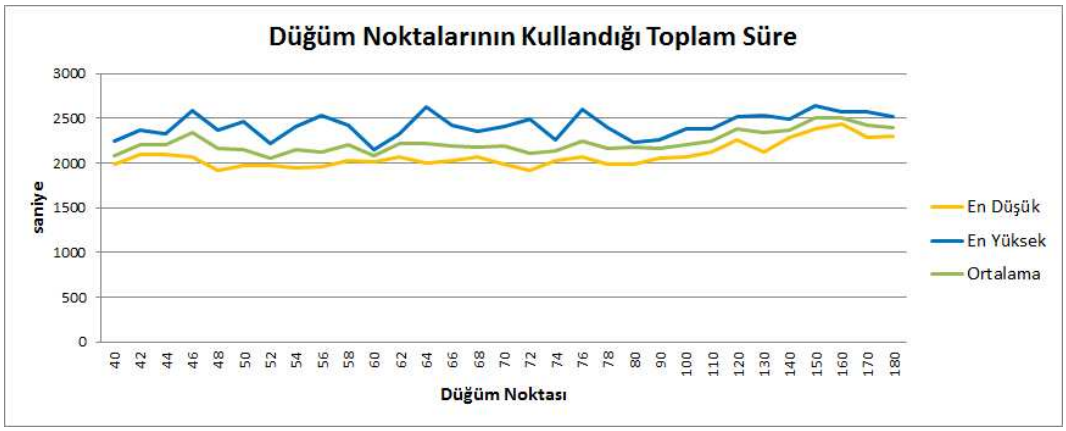
geçen zaman, dosya fiziksel olarak daha hızlı okunamayacak seviyeye inene kadar sürekli bir azalma göstermiştir. Dolayısıyla MPI I/O'dan kaynaklanabilecek başarım düşüşleri bu şekilde engellenmiştir.

MPI I/O süresi ile ilgili sorun aşıldıktan sonra değişen düğüm sayısına göre Şekil 5.3'de hesaplama sırasında geçen zamanı, Şekil 5.4'de tüm düğüm noktalarının her bir testte MPI algoritması sırasında harcadığı toplam zamanı, Şekil 5.5'de algoritmanın WallClock zamanlarını ve Şekil 5.6'de Parallel Efficiency çizimi görülmektedir.

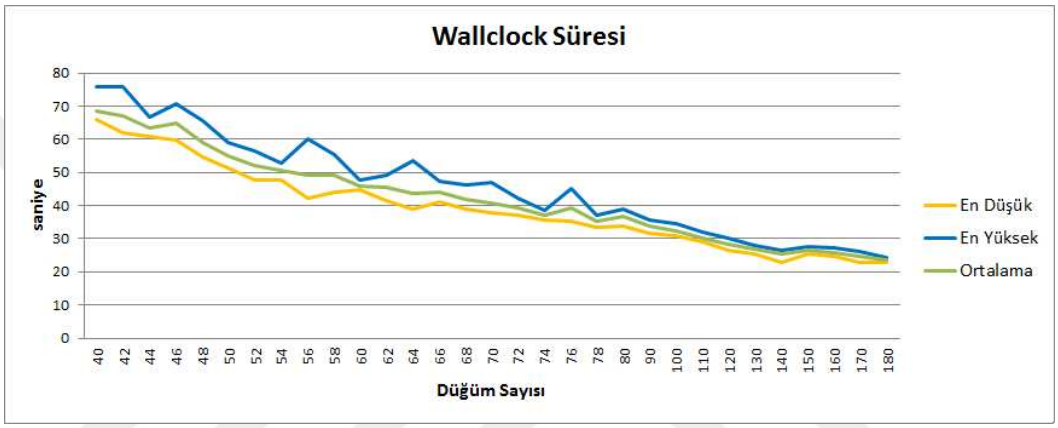


Şekil 5.3: MPI algoritması sırasında hesaplama için harcanan zaman

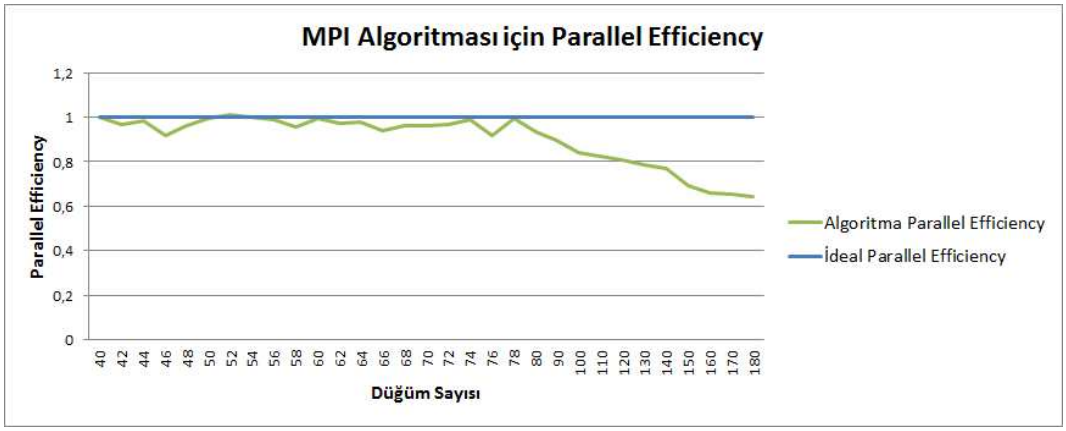
Şekil 5.3'de görüldüğü gibi düğüm noktası arttıkça hesaplama zamanı sürekli bir azalma davranışı göstermektedir. *En Düşük* ile *En Yüksek* süreler arasında fark olmasına karşın bu durum MPI I/O süresindeki durumdan farklıdır. MPI I/O'da dağılım tekdüze (ing:*uniform*) dağılım göstermektedir. Hesaplama süresi ölçümünde kullanılan testlerde ise dağılım normal dağılımdır ve tepe noktası ortalamayı işaret etmektedir. Dolayısıyla hesaplama süreleri her bir test içerisinde tutarlıdır. Düğüm noktalarında harcanan toplam zamanı gösteren Şekil 5.4 incelenirse düğüm sayısının atmasına



Şekil 5.4: Tüm düğümleri MPI algoritması sırasında harcadığı toplam zaman



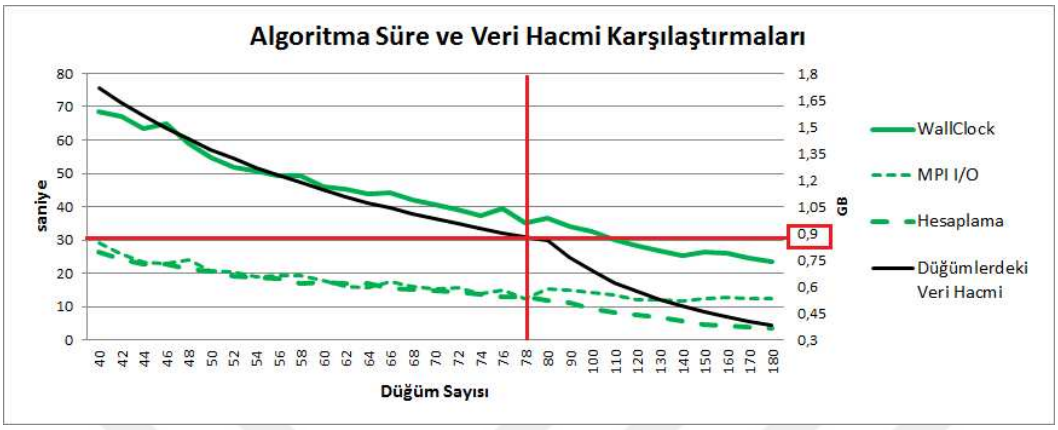
Şekil 5.5: MPI algoritmasının WallClock zamanı



Şekil 5.6: MPI algoritmasının Parallel Efficiency çizimi

rağmen başarımını koruyarak yatay konumunu koruduğu görülmektedir. Şekil 5.5'deki algoritmanın baştan sona işleyiş zamanı olan WallClock süresi ve Şekil 5.6'deki başarım çizimi incelendiğinde de benzer sonuçlar görülmektedir. Algoritma, girdi dosyası fiziksel olarak daha hızlı okunamayacak seviyeye inene kadar başarımını kaybetmeden sürdürmüş, bu sınır aşıldıktan sonra başarım yitimi başlamıştır. Dolayısıyla MPI algoritması yüksek başarıma sahiptir.

Şekil 5.7, MPI algoritmasının WallClock, MPI I/O ve Hesaplama süreleri ile birlikte her bir düğüme düşen veri hacmini göstermektedir. Sürelerin indeksi sol tarafta saniye cinsinden, veri hacminin indeksi de sağ tarafta GB cinsinden gösterilmiştir. Kullanılan tüm süreler ortalamalar üzerinden verilmiştir.



Şekil 5.7: MPI algoritmasının süre ve her bir düğüm noktasına düşen veri hacmi karşılaştırması

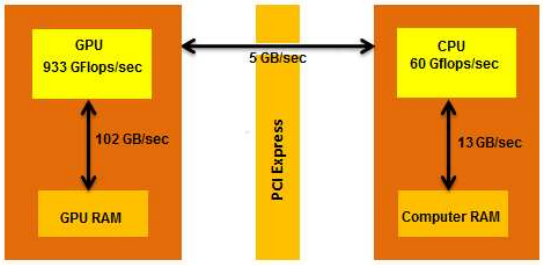
Şekilden görüldüğü gibi algoritma wallclock süresi üzerinde MPI I/O ve hesaplama süreleri aynı ağırlıkta etki etmektedir. Bu etki düğüm başına düşen veri hacmi 0.9GB'a kadar düşene de değişmemektedir. Bu noktadan sonra düğüm satısının artması fiziksel olarak girdi dosyasının daha hızlı aktarımını sağlamadığından MPI I/O süresi sabit kalmış, buna karşın düğüm sayısı arttıkça hesaplama süresi beklendiği gibi azalmaya devam etmiştir. Fakat MPI I/O süresinden dolayı algortimanın başarımı bu noktadan sonra azalmaya başlamıştır. Buradan çıkarılacak sonuç bu sistemde en uygun düğüm sayısı, her bir düğüme 0.9GB'dan daha az veri hacmi düşmeyecek şekilde bir seçim yapılmasıdır.

6. YBMG Yönteminin CUDA ile Koşutlaştırılması

Son birkaç yılda "watt başına başarım" gücü yüksek olan GPU işlemcileri kullanan CUDA kitaplığı oldukça popüler bir hale gelmiştir. Bilimsel yazında diğer koşutlaştırma kitaplıklarına göre oldukça yeni olan CUDA'nın bu kadar başarımlı çalışmasının arkasında yatan etmenler donanımdan yazılıma kadar bu bölümde açıklanacaktır. Daha sonra CUDA kitaplığını kullanarak YBMG yönteminin koşutlaştırma adımları gösterilecektir.

6.0.1 GPU Donanımının Temel Özellikleri:

CUDA kitaplığına değinmeden önce bu kitaplığın kullandığı GPU işlemcisi ve bağlı olduğu donanım özelliklerine değinilecektir. GPU işlemciler ekran kartı üzerinde bulunur. Bu nedenden dolayı ekran kartı ile yönetici makinanın (ing:*host machine*) ile ilişkisi irdelenmelidir. Şekil 6.1'de sol yanda bir ekran kartını temsili, sağ yanda ise ekran kartının bağlı olduğu ve CPU işlemci kullanan yönetici makinanın temsili görülmektedir.



Şekil 6.1: CPU ve GPU sistemlerin veriyolu başarım yapılmalıdırlıkları

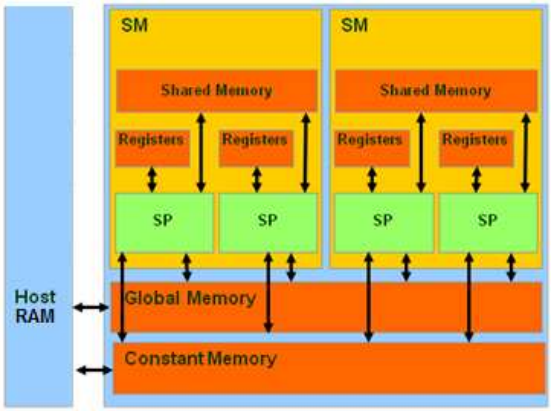
Ekran kartı ile yönetici makina arasındaki veri transferi PCI express veri yolu ile sağlanır. Bu veriyolu saniyede 5GB aktarım hızına sahiptir. Şekil 6.1'de her iki sistemi kabaca karşılaştıracak olursak günümüz CPU'ların 60 Gflops/sn. başarıma ulaştığı görülmektedir. Buna karşın test için kullanılan Nvidia Tesla C1060 kartı ile 933 Gflops/sn. başarım elde edilebilmektedir. Hem GPU hem de CPU'nun verileri işleyebilmesi için makinanın belleğinden verileri çekmeleri gerekmektedir. Bu veri akışı CPU'da 13GB/sn. hıza sahipken GPU'da bu hız 102GB/sn. olmaktadır. Temsili

şekillerden başarım farkının nasıl oluştuğu temel olarak görülmektedir. GPU'nun CPU'ya göre hesaplama başarımının bu kadar yüksek olmasının farkını görmek için yonga (ing:*chipset*) mimarisi arasındaki farklara bakmak gerekmektedir. Bu farklar Şekil 6.2'de görülmektedir.

Şekil 6.2: CPU ve GPU'nun yonga mimarileri farkları

CPU'nun en önemli görevi, işletim sistemi ile donanım arasında köprü oluşturan programları çalıştırmaktır. Doğal olarak bu iş yüküne cevap vermek için sınırlı ALU (Arithmetic Logic Unit) sayısına karşın geniş ve oldukça hızlı bir yonga üzeri hafıza (ing:*onchip memory*) kullanır. Gereksinim duyduğunda bellek adı verilen ve büyük olmasına karşın başarımı düşük sistem belleğini kullanır.

GPU ise kullanım amacına uygun bir biçimde, CPU'nun görevleri dışında kalan görüntüleme işlerinin gereksinim duyduğu yüksek hesaplama yükünü karşılamak için üretilmiştir. Dolayısıyla bu yüksek hesaplama maliyetini karşılamak için çok sayıda ALU biriminden yararlanır. GPU'daki her bir ALU birimine SP (Streaming Processor) denilmektedir. GPU çok sayıda ALU ve daha sınırlı büyüklükte GPU yonga üzeri hafıza sahiptir. Gereksinim duyduğunda ekran kartın üzerinde bulunan bellekleri de kullanır. Bu bellekler aynı zamanda host belleği ile veri alışverişi yapabildiği kısmıdır. GPU bileşenlerinin ekran kartı üzerinde bulunan hafıza türlerini kullanım biçimi Şekil 6.3'de incelenebilir.



Şekil 6.3: GPU'nun hafıza kullanım Şeması

GPU üzerinde bulunan her bir SP genel hafızaya (ing: *Global Memory*) erişip yazma yetkisine sahiptir. Bu hafıza türü, ekran kartı üzerindeki en kapasiteli hafızadır. Fakat kaşelenemediği (ing:*caching*) için erişim hızı diğer hafızalara göre oldukça düşüktür. Sabit hafıza (ing: *constant Memory*) kaşelenebilir, fakat SP'ler sabit hafıza içindeki veriyi sadece okuyabilir, yazamaz. Ayrıca kapasitesi de çok daha küçüktür. SP'ler belirli sayıda gruplar halinde SM (Streaming Multiprocessor) içinde bulunurlar. Her bir SM içindeki SP'ler kendi bulunduklari SM üzerinde bulunan ve paylaşılan hafıza (ing: *Shared Memory*) adı verilen ve GPU üzerinde bulunan hafıza türünü kullanırlar. Bu hafıza türü kaşelenebilmektedir. Aynı SM içinde bulunan SP'ler bu hafıza türü aracılığı ile veri paylaşımı yapabilirler. Ayrıca SP'ler bu hafıza üzerinde veri okuma ve yazma işlemleri yapabilirler. Fakat bu hafızanın kapasitesi genel bellek yanında KB'lar seviyesindedir. En küçük kapasite ve en hızlı işleme sahip hafıza türü ise her bir SP'ye özel olarak atanmış yazmaçtır (ing:*register*). Bu hafıza türüne yazılabilecek değisken sayısı sınırlıdır ve her bir yazmaç kendi SP'si tarafından kullanılabilir.

6.1 CUDA Programlama Yapısı

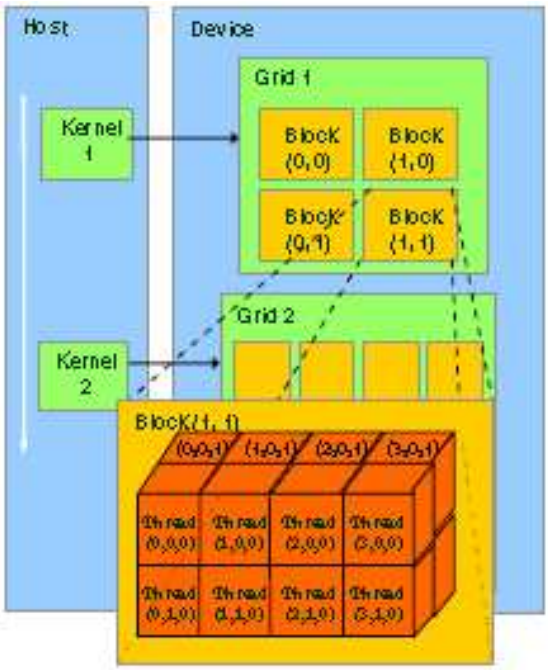
GPU temel olarak SIMD (Single Instruction Multiple Data) yapısına sahip ve işyükünü iş parçacıklarına (ing:*thread*) bölerek koşutlaştırmaktadır. İşyükünün başarımlı bir biçimde dağıtılabilmesi için kullanılan programlama yapısının bilinmesi zorunludur.

NVIDIA C Derleyici (ing: NVIDIA C Compiler) kısaca NVCC ikiye ayrılabilir. Sistem üzerinde çalışan C/Fortran kodudur ve sistemin C/Fortran derleyicisi tarafından derlenir. Ekran kartında GPU üzerinde çalışan kısım ise çekirdek (ing:*kernel*) adı verilen koşut fonksiyonlarından oluşur. Çekirdekler verinin koşutlaştırılabilmesi için iş parçacıkları adı verilen ve belli kurallarla oluşturulabilinen yapılar üretir. Bu aşamada çekirdekten başlayarak iş parçacıklarına doğru giden yapı incelenmelidir.

6.1.1 CUDA Programlamada İş Parçacıkları Yapısı

Şekil 6.4'de CUDA iş parçacıkları düzenleme yapısı gösterilmektedir.

Çalıştırılan bir CUDA programı içinde birden çok çekirdek olabilir, fakat sırayla çalışabilir, aynı anda birden çok çekirdek çalıştırılamaz. Çekirdekler de kendi içinde işyükünün iş parçacıklarına bölünebilmesi için ızgara (ing:*grid*) adı verilen bir yapı



Şekil 6.4: CUDA İş parçacığı düzenleme yapısı

kullanırlar. Benzer bir biçimde her bir çekirdek yalnızca bir adet ızgara yapısı içerebilirler. Izgara, 2 boyutlu bir yapıya sahiptir ve ızgaranın her bir boyutu en fazla 65,535 adet blok (ing:*block*) adı verilen yapıları barındırabilirler. Şekil 6.4'de Izgara1 4 adet bloğa sahiptir ve her bir boyutta 2 adet iş parçacığı vardır. Bloklar ise 3 boyutlu bir yapıya sahiptir. G80 GPU işlemci için ibr blok toplam olarak en fazla 512 adet iş parçacığı içerebilir. Aynı işlemci için her bir blok iki boyutun her birinde maksimum 512 adet iş parçacığı içerebilir, üçüncü boyutta ise bu sayı 64 ile sınırlıdır. Bir ızgaradaki her bloğun boyut uzunlukları aynı olmalıdır. Şekil 6.4'de ele alınan Block(1,1) içinde 16 adet iş parçacığı vardır ve bloklar (4,2,2) biçiminde düzenlenmiştir.

Örnek olarak 1000x1000 boyutunda iki matrisin çarpımı ele alınsın. Burada hesaplanacak matrisin eleman sayısı 1.000.000 olacaktır. Hesaplanacak her bir eleman için 1 adet iş parçacığı yaratılacağından bu problem için toplam 1.000.000 adet iş parçacığı yaratılacaktır. Daha önceden değinildiği gibi bu sayıda iş parçacığı oluşturmak, yapısı itibariyle GPU için oldukça düşük bir maliyettir. Bu problem için blok yapısı (16,16,1) olacak şekilde oluşturulabilir. (32,32,1) blok yapısı geçersiz bir yapıdır, çünkü bloktaki eleman sayısı toplamı 512 sınırını aşarak 1024 olmaktadır. Bloğun ilk iki boyut uzunluğu 16 olduğundan 1000/16 = 62,5 olur. Bu nedenle ızgaradaki blok yapısının (63,63) şeklinde düzenlenmesi gerekmektedir.

6.1.2 İş Parçacığı Yapısının Donanımsal Kısıtları

G80 işlemcisine ait her bir SM'de 768 iş parçacığı aynı anda çalışabilmektedir. Bu da G80'nin 12.000'den çok iş parçacığını aynı anda çalıştırabildiğini gösterir. Intel tabanlı CPU'larda bu rakam her bir çekirdek için en yüksek sayı 8'dir. Üstelik iş parçacığı yaratma işleminin GPU açısından maliyeti ihmal edilecek kadar azdır. Tüm bu farklılıklar bir araya geldiğinde GPU'ların başarımı CPU'ya göre bazı uygulamalarda 200 kat çok olabilmektedir.

Şekil 6.3'deki bellek yapısı gözönüne alınarak donanım ile iş parçacığı arasındaki özellikler şu şekilde gösterilebilir:

Her iş parçacığı, çalıştığı SM içindeki yazmaç belleğe yazıp erişebilir, diğer SM'lerinkine erişemez.

Her blok, çalıştığı SM içindeki paylaşılan belleğe yazıp erişebilir, diğer SM'lerinkine erişemez.

Her ızgara, genel belleğe yazıp erişebilir, sabit belleğe sadece erişebilir.

Koşullardan da anlaşılacağı gibi her bir blok tek bir SM içinde çalıştırılabilir. G80 için aynı anda her bir SM'de sadece 8 blok çalışır. Toplamda 128 blok aynı anda G80 işlemcisinde çalışabilir.

6.1.3 İş Parçacığı Çalıştırım Sırası

Bir SM, çalıştırması gereken iş parçacıklarını 32'lik birimlere ayırır. Her bir birime WARP adı verilir. Aslında bu tip bir ayırma CUDA programlamasında yoktur, bu GPU işlemcisinin kendi mimarisinden kaynaklanmaktadır. Fakat programlama açısından önemlidir. Eğer bir SM içerisindeki herhangi bir WARP bir sonraki WARP verisine gereksinim duyarsa, o ana kadar çalışmış WARP bekler ve diğer WARP gereksinim duyulan veriye kadar çalışır. Bu da gecikmeye ve başarım yitimine neden olur.

İş parçacığı düzenlemesinde diğer önemli konu da işleme alınacak iş parçacıklarının sırasıdır. Örnek olarak 8x8'lik bir blok gözönüne alınsın. Toplamda bu blok 2 WARP'tan oluşacaktır. İlk WARP $T_{0,0}$'dan başlar $T_{3,7}$' a kadar devam eder, son WARP ise $T_{4,0}$'dan başlar $T_{7,7}$' ye kadar devam eder. 3 boyutlu 4x8x2 blokta ise ilk WARP $T_{0,0,0}$'dan başlar $T_{3,7,0}$' a kadar devam eder, son WARP da $T_{0,0,1}$'dan başlar $T_{3,7,1}$' a kadar devam eder.

İş parçacıklarının işlem sırası özellikle verinin genel bellekten paylaşılan ya da sabit belleğe alınması sürecinde etkilidir.

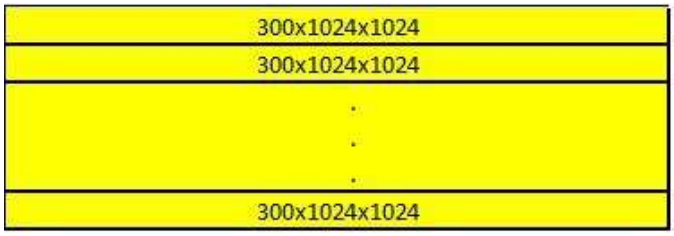
6.1.4 YBMG Terimleri Hesabının CUDA ile Koşutlaştırılması

Daha önceden değinildiği gibi YBMG, hesaplama maliyeti verinin boyut ve bileşen sayısı ile orantılı olarak artan bir yöntemdir. Gerçek bir hesaplama maliyeti yüksek hacimli verilerde ortaya çıkmaktadır. Buna karşın günümüzde GPU işlemcilerin kullanabildiği genel hafıza en fazla 6GB'dır. Bu nedenle yüksek hacimli problemleri GPU ile çözmek, verinin ekran kartının genel hafızaya parça parça gönderilmesi ile mümkündür. Bu parçalama öyle bir şekilde yapılmalıdır ki CUDA programlamaya uygun olan SIMD koşutlaştırma türünde olsun. Özetlemek gerekirse, veriler parça parça ekran kartının genel hafızasına uygun parçalar halinde gönderilmeli, öyle ki her bir SP aynı işlemi farklı verileri kullanarak hesaplama yapabilsin.

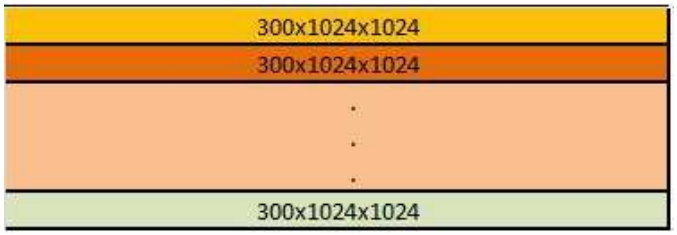
Bu durumu daha iyi açıklamak için yeni bir örnek noktalar kümesi alınacaktır. Bu kümede 1. boyuta ait bileşen sayısı 300, 2. ve 3. boyutlara ait bileşen sayısı 1024, 4. boyuta ait bileşen sayısı 300 alınmıştır. Boyutlardaki bileşen sayıları rastlantısal alınmamıştır. MPI algoritmasında kullanılan veri bölüntüleme stratejisi burada da kullanılacaktır. Bölüntüleme sonrasında çıkan her bir satır GPU genel hafızasına aktarılabilecek sığadadır. 1024 adet bileşen, işyüklerini 32'lik parçalar halinde çalıştıran CUDA programlama yapısı için uygun bir sayıdır. 300 adet bileşen ise CUDA işyükü paylaşımı açısından çok uygun bir sayı değildir. Eğer tek bir veri satırı için YBMG bileşenleri hesabı CUDA ile yapilabilirse, diger satırlar için de aynı algoritma çalıştırılıp nihai sonuçlar elde edilir. Bu nedenle tek bir veri satırı için yapılan YBMG terimleri hesabı üzerinden algoritma oluşturulacaktır. Algoritma oluşturulurken örnek kümenin 2. boyutuna ait bileşenlerin YBMG terimleri hesabından başlanacaktır. İkinci boyuta ait YBMG terimlerinin hesabında kullanılan tüm toplamlar aslında hem YBMG değişmez hem de YBMG birli terimleri hesabında kullanılan toplamlardır. Dolayısıyla bu terimlerin hesabı pratik bir şekilde yapılabilir.

6.1.4.1 2. Boyuta Ait YBMG Terimleri Hesabının CUDA ile Koşutlaştırılması

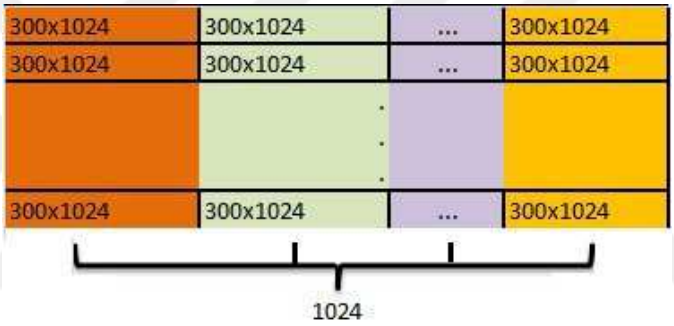
Programlamanın nasıl yapıldığına geçmeden önce YBMG değişmez, 1. ve 2. boyutlardaki YBMG terimleri hesabında kullanılan veri çizemleri gösterilmelidir.



Şekil 6.5: YBMG değişmez terimi f_0'ın veri kullanım çizemi



Şekil 6.6: 1. boyuta ait düğüm noktalarının YBMG birli terimlerinin veri kullanım çizemi



Şekil 6.7: 2. boyuta ait düğüm noktalarının YBMG birli terimlerinin veri kullanım çizemi

Önceden değinildiği gibi, Şekil 6.7'deki 2. boyuta ait bileşenlerin YBMG terimlerini hesaplayan toplamlar kullanılarak Şekil 6.5'de gösterilen YBMG Değişmez Terimi ve Şekil 6.6'da gösterilen 1. boyuta ait YBMG birli terimleri hesaplanabilir. 2. boyuttaki bileşen sayısı 1024 tanedir. Dolayısıyla bu boyutta hesaplanacak 1024 adet YBMG terimi vardır. İlk satırdakı verilerin GPU genel hafızaya aktarıldığında ilk problem ortaya çıkar. İlk satırda 300x1024x1024 = 314.572.800 eleman vardır. Önceki bilgilerimizden tek bir boyutta GPU'nun en fazla 65.535 adet bloğa izin verdiğini bilmekteyiz. Her bir blokta 512 iş parçacığı kullanılsa bile 65.535x512 = 33.553.920 işparçacığı tek bir boyutta kullanılabilmektedir. Bu sorunu ortadan kaldırmak için veri iki boyutlu hale getirilebilir. Fakat bu durumda da hesaplanması gereken ikinci bir indeks ve hesaplama karmaşası ortaya çıkmaktadır. Bu tür bir

problemi çözmek için *CUDA Reduction* [18] algoritmasında kullanılan yöntem önem taşımaktadır. Algoritmanın CUDA çekirdek kodu aşağıdaki gibidir.

```
1  __global__ void reduce6(int *g_idata, int *g_odata, unsigned int n)
2  {
3     extern __shared__ int sdata[];
4     unsigned int tid = threadIdx.x;
5     unsigned int i = blockIdx.x*(blockSize*2) + tid;
6     unsigned int gridSize = blockSize*2*gridDim.x;
7     sdata[tid] = 0;
8     while (i < n)
9     {
10       sdata[tid] += g_idata[i] + g_idata[i+blockSize];
11       i += gridSize;
12    }
13     __syncthreads();
14
15   // do reduction in shared mem
16   if (blockSize >= 512) { if (tid < 256) { sdata[tid] += sdata[tid + 256]; } __syncthreads(); }
17   if (blockSize >= 256) { if (tid < 128) { sdata[tid] += sdata[tid + 128]; } __syncthreads(); }
18   if (blockSize >= 128) { if (tid <  64) { sdata[tid] += sdata[tid +  64]; } __syncthreads(); }
19
20   #ifndef __DEVICE_EMULATION__
21   if (tid < 32)
22   #endif
23   {
24      if (blockSize >=  64) { sdata[tid] += sdata[tid + 32]; EMUSYNC; }
25      if (blockSize >=  32) { sdata[tid] += sdata[tid + 16]; EMUSYNC; }
26      if (blockSize >=  16) { sdata[tid] += sdata[tid +  8]; EMUSYNC; }
27      if (blockSize >=   8) { sdata[tid] += sdata[tid +  4]; EMUSYNC; }
28      if (blockSize >=   4) { sdata[tid] += sdata[tid +  2]; EMUSYNC; }
29      if (blockSize >=   2) { sdata[tid] += sdata[tid +  1]; EMUSYNC; }
30   }
31
32   // write result for this block to global mem
33   if (tid == 0) g_odata[blockIdx.x] = sdata[0];
34 }
```

Reduction algoritması bir veri dizisi içindeki elemanların toplamını etkin bir şekilde hesaplar. Algoritmadaki ilk satırda kullanılacak parametreler yer almaktadır. İlk parametre ekran kartının genel belleğinde bulunan ve veri dizisini ifade eder. İkinci parametre ise sonuçları içeren çıktı dizisini ifade eder. Son parametre ise veri dizisinin uzunluğunu ifade eder. 3. satırda her bir bloğun kullanacağı paylaşılan bellek alanı açılır. Bir sonraki satırda iş parçacıklarının indeksini tutan değişken yerine kullanılan ve karmaşayı önlemek için oluşturulan *tid* (ing: thread id) değişkeni yer almaktadır. 4. satırda genel bellekteki dizinin indeksini kontrol etmek için oluşturulan *i* değişkeni vardır. Bu değişken her bir bloğun indeksine ve bloklarının uzunluğuna bağlıdır. Dikkat edilirse her bloğun başlangıç noktası blok indeksi ile blok uzunluğunun çarpımının 2 katı olan değerden başlar. Böylece her bir blok başlaması gereken noktanın iki katı ötesinden başlar. *tid* değişkeni de bloklar içindeki iş parçacıklarının sorumlu olduğu indeksi belirlemede kullanılır. *gridSize* değişkeni ise ızgara uzunluğunun iki katını ifade eden değere eşittir. 7. satırda bloklar içindeki paylaşılan bellekte ayrılan dizi elemanları sıfır atanır. while dizisi global bellekteki kontrolü sağlayan indeks veri

dizisi uzunluğuna kadar devam eder. Döngünün ilk satırında bloklardaki paylaşılan belleklere açılan diziler içine genel bellekteki veriler toplanarak aktarılır. *i* değişkeni oluştururken blokların başlangıç noktalarının olması gerekenden 2 kat ötelenmiş şekilde açıldığı belirtilmişti. 10. satırda da paylaşılan bellek içindeki diziye hem bloktaki indeks değer hem de blok uzunluğu kadar ötelenmiş değer aynı anda toplanır. Böylece tek seferde bir iş parçacağı iki değeri toplayarak paylaşılan belleğe yazar. Bloklar içindeki hesaplamalar bitince algoritma 11. satıra kadar gelir. Bu satırda indeks değeri ızgara uzunluğunun iki katı kadar ötelenir. Böylece ötelendiği yerden itibaren tekrar bloklar içinde işlemler yapılır ve veri dizisi sonuna kadar bu döngü tekrarlanır. Bu döngü sayesinde ızgarada açılan blok sayısı kısıtı olmaksızın genel bellekte bulunan her uzunluktaki veri dizisi elemanlar bloklar içindeki paylaşılan belleklerde toplanır. Bu aşamada toplamlar bloklar içindedir, paylaşılan bellekteki dizi elemanları henüz toplanmamıştır. Algoritmada bu toplamları hesaplamaları 16. satır ile 29. satır arasında yapılır. Burada CUDA'nın kısıtları ve özellikleri en iyi şekilde kullanılarak eniyileme yapılmıştır. Önceden değinildiği gibi CUDA'da blok uzunluğu 512 ile sınırlandırılmıştır. Bu nedenle kodda ard arda kontrol döngüleri yazmak yerine mümkün olan tüm olasılıklar gözönüne alınarak if koşulları koyulmuştur. Satır 16'da başlayan bu if koşulları öyle çalışmaktadır ki seçilen blok uzunluğu uygun olan if koşuluna girdikten sonra devam eden diğer koşullar bir önceki koşulun hesapladığı işi devralır. Örnek olarak boyut uzunluğu 512 olan bir blok Satır 16'yı sağlar ve paylaşılan bellekteki ilk yarı olan 256 eleman ile bir sonraki 256 eleman toplanır. Dolayısıyla sonuçlar paylaşılan bellekteki ilk 256'lık kısımda toplanır. Bunun işlem sonrasında blok uzunluğu Satır 17'deki koşulu da sağlar. Bu koşul ile paylaşılan bellekteki ilk 128 eleman bir sonraki 128 eleman ile toplanır ve bu koşullar Satır 21'e kadar devam eder. Burada da CUDA'daki WARP yapısı düşünülerek 32 ve sonraki döngüler tek bir if koşulu içine alınmıştır. Değinildiği gibi CUDA işlemleri 32'lik sıralar ile yapar. Dolayısıyla 32 ve daha aşağı işlemlerin tek WARP içinde yapılacağı kesin olduğundan ayrıca ayrı ayrı if koşulları açmaya gerek yoktur. Bu sayede if koşullarının yaratacağı gereksiz zaman kaybı da önlenir.

Algoritma etkin olmasına karşın YBMG yöntemi için bu haliyle uygun değildir. *Reduction* algoritması tüm veri dizisini toplamaktadır. Oysa yöntem için veri parçalarının ayrı ayrı toplanması gerekmektedir. Burada *Reduction* algoritması şu şekilde iyileştirilmiştir: Hesaplanacak YBMG terimleri kadar blok sayısı içeren bir

ızgara oluşturulur. Her bir blok tek bir YBMG teriminin hesaplanmasından sorumlu olacağından 1024 adet blok yaratılmalıdır. Buradaki kritik nokta, blokların sadece sorumlu oldukları veri parçaları içerisindeki elemanların toplamasını sağlamasıdır. *Reduction* algoritmasından farklı olarak bu sefer ızgara değil ızgara içindeki bloklar ötelenmektedir. Yani ızgara tüm veriyi kapsar, bloklar kendilerine düşen veri dizisi bitimine kadar ötelenir ve bu sırada elemanlar bloğa ait paylaşılan bellekte toplanır. Bu amaçla tanımlı *gridDim*, *blockIdx* ve *threadIdx* değişkenleri kullanarak ilgili hesaplamaları yapan kod *SumArray* çekirdeği olarak aşağıda tanımlanmıştır.

```
 1  __global__ void SumArray(float *data, float *odata, unsigned long n)
 2  {
 3     unsigned long VirtualBlockSize = n/gridDim.x;
 4     unsigned int tid = threadIdx.x;
 5     unsigned long i = blockIdx.x * VirtualBlockSize + tid;
 6     unsigned long finish = (blockIdx.x + 1) * VirtualBlockSize;
 7     unsigned int step = blockDim.x;
 8     unsigned int blockSize = blockDim.x;
 9
10      extern __shared__ float sdata[];
11
12      sdata[tid] = 0;
13
14      while(i<finish)
15      {
16            sdata[tid] += data[i];
17            i += step;
18      }
19       __syncthreads();
20
21
22      // do reduction in shared mem
23      if (blockSize >= 512) { if (tid < 256) { sdata[tid] += sdata[tid + 256]; } __syncthreads(); }
24      if (blockSize >= 256) { if (tid < 128) { sdata[tid] += sdata[tid + 128]; } __syncthreads(); }
25      if (blockSize >= 128) { if (tid <  64) { sdata[tid] += sdata[tid +  64]; } __syncthreads(); }
26
27      #ifndef __DEVICE_EMULATION__
28      if (tid < 32)
29      #endif
30      {
31          if (blockSize >=  64) { sdata[tid] += sdata[tid + 32]; EMUSYNC; }
32          if (blockSize >=  32) { sdata[tid] += sdata[tid + 16]; EMUSYNC; }
33          if (blockSize >=  16) { sdata[tid] += sdata[tid +  8]; EMUSYNC; }
34          if (blockSize >=   8) { sdata[tid] += sdata[tid +  4]; EMUSYNC; }
35          if (blockSize >=   4) { sdata[tid] += sdata[tid +  2]; EMUSYNC; }
36          if (blockSize >=   2) { sdata[tid] += sdata[tid +  1]; EMUSYNC; }
37    }
38
39    // write result for this block to global mem
40    if (tid == 0) odata[blockIdx.x] = sdata[0];
41
42 }
```

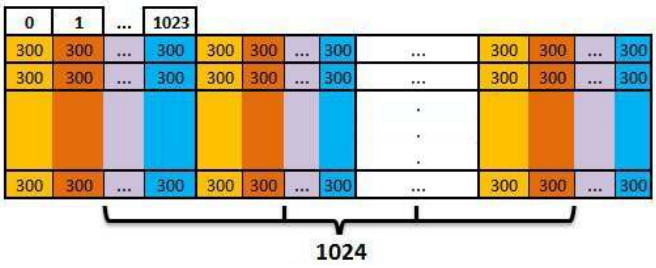
İlk olarak veri satırındaki her bir YBMG terimi hesaplanmasında kullanılacak veri parçası uzunluğunun saptanması gereklidir. Bu hesaplama 3. satırdaki $VirtualBlockSize = n/gridDim.x$ ile belirlenir. Burada n, veri satırının uzunluğunu temsil etmektedir. Örneğimizde $VirtualBlockSize = n/gridDim.x = 300x1024x1024 /1024 = 300x1024$ olarak hesaplanır. Bu da her bir YBMG terimini hesaplamada kullanılacak eleman sayısına eşittir. $tid = threadIdx.x$ atamasını kodda karmaşayı

önlemek için tanımlanmıştır. Genel bellekteki verinin paylaşılan belleğe aktarımını kontrol için 5. satırdaki *i* değişkeni kullanılmaktadır. Dikkat edilecek olursa her bir blok *i* değişkeni yardımıyla hesaplamaya başlayacakları elemanı tam olarak bulurlar ve her bir blok içindeki iş parçacıkları ise *tid* yardımıyla sorumlu oldukları elemanı paylaşılan bellekteki ilgili eleman ile toplanmasını sağlarlar. İndeks ötelemelerinin 300x1024 eleman içinde sınırlı kalması için 6. satırdaki *finish* değişkeni tanımlanmıştır. Bu değişken bir sonraki bloğun başlangıç elemanının indeksini hesaplar. Böylece her bir blok bu indekse geldiğinde hesaplamayı tamamlar. Blokların içindeki iş parcacıkları sorumlu oldukları işleri tamamladıklarında bloğun kendi uzunluğu kadar sağa ötelenmesi gereklidir ki bloğun sorumlu olduğu dizi parçasındaki tüm hesaplamaları tamamlayabilsin. Bunun için 7. satırdaki *step* değişkeni tanımlanmıştır. Bu değişken sayesinde bloklar içindeki hesaplamalar *finish* indeksine kadar devam eder. Kodun 10. satırında her bir blokta paylaşılan bellek açılır ve başlangıç için paylaşılan bellekteki tüm elemanlar sıfırlanır. 14. satırdaki while döngüsü ile hesaplamaların sınırı belirlenir. 16. satırda bloklar içindeki her bir iş parçacığı paylaşılan bellekte sorumlu olduğu eleman ile genel bellekteki ilgili elemanı toplar. Böylece blok içindeki her bir iş parçacığı aynı anda bu işlemi tamamlamış olur. Bu işlem tamamlandıktan sonra genel bellekteki indeksi kontrol eden *i* değişkeni 17. satırda görüldüğü gibi blok uzunluğu kadar ötelenir. Bu öteleme ve paylaşılan bellek üzerindeki toplama işlemi *i* indeksi *finish*'e ulaşana kadar devam eder ve while döngüsü sonlanır. Bundan sonraki kısım daha önce *Reduction* algoritmasında anlatıldığı gibi bloklardaki paylaşılan bellekteki elemanların tek bir eleman üzerinde toplanmasıyla sonlanır. Burada kritik nokta blok sayısının boyuttaki bileşen sayısına eşit olacak şekilde atanmasıdır. Elde edilen sonuçlar her bir veri satırı için toplanıp ağırlıkları ile çarpıldığında 2. boyuta ait YBMG terimleri elde edilmiş olur.

6.1.4.2 3. Boyuta Ait YBMG Terimleri Hesabının CUDA ile Koşutlaştırılması

Üçüncü boyuta ait düğüm noktalarının YBMG terimleri hesabı için kullanılan veri kullanımı Şekil 6.8'de görülmektedir.

Üçüncü boyuta ait bileşenlerin CUDA programlaması, son boyuta kadar tüm boyutların bileşenlerinin CUDA programlaması ile aynıdır. Aralarındaki farklar dizi parçalarının uzunluğu, dizi parçaları arasındaki uzaklık ve dizi parçalarının sayısıdır. Dolayısıyla



Şekil 6.8: 3. boyuta ait düğüm noktalarının YBMG birli terimlerin veri kullanım çizemi

burada açıklanacak yöntem, ikinci ve sonuncu boyutlar arasındaki tüm boyutların YBMG terimleri hesabında kullanılabilir.

3. boyutta da 1024 adet bileşen olduğundan 1024 adet YBMG terim değeri hesaplanmalıdır. Şekil 6.8'de görüleceği gibi ilk 300x1024 veri için kullanılan çizim, veri dizisi sonuna kadar 1024 kez tekrarlanmaktadır. Dolayısıyla burada YBMG terimlerini hesaplamak için uygulanacak yöntem, 2. boyuttaki YBMG terimlerini bulurken kullanılan yöntemin 1024 kez tekrarlanmasıdır. 2. boyutta YBMG terimleri ızgara içindeki blokların ötelenmesi ile bulunmaktaydı. Bu kez hem ızgara hem de blok ötelenmesi işlemi uygulanacaktır. Şimdiki sorun blok içindeki 300 adet toplamın iş parçacıklarına nasıl paylaştırılacağıdır. 300 eleman tabi ki CUDA blok yapısının izin verdigi sınırlar içerisindedir, fakat burada genel bir algoritma oluşturulmaktadır. Bu nedenle ardışık olarak toplanacak 300 elemanın blok sınırları içerisine sığmadığı varsayılır. Burada oluşturulan kod, boyuttaki bileşen sayısı ne olursa olsun doğru sonuç verecek şekilde çalışmalıdır.

Bir diğer problem, ele alınan bloktaki iş parçacığı sayısı genel GPU *Reduction* algoritması için uygun olmamasıdır. Uygun algoritma oluşturulabilir, fakat amaç, genel kod olduğundan aynı *Reduction* algoritmasının kullanılması tercih sebebidir. Yukarıda anlatılan ihtiyaçları karşılayacak şekilde oluşturulmuş *SumArray*2 çekirdeği aşağıdaki gibidir.

```
1 __global__ void SumArray2(float *data, float *odata, unsigned long n, unsigned long BBS,
unsigned int BS, unsigned int PS, unsigned int blockSize)
 2 {
 3    // BBS: Boyut Bileşen Sayısı
 4    //  BS: Blok Sayısı
 5    //  PS: Parça Sayısı, her bir dizi parçasını hesaplamada kaç tane parça kullanıldığını ifade eder
 6    // SPU: Son parçanın uzunluğu
 7    // blockSize: Her bir parçanın (bloğun) uzunluğu
 8
 9    unsigned int tid = threadIdx.x;
10    unsigned long start = blockIdx.x * BBS + tid;
11    unsigned long SPU;
12    unsigned int j;
13    unsigned long gridStart=0;
14    unsigned long gridSize = BBS * BS;
```

```
15
16    extern __shared__ float sdata[];
17
18    sdata[tid] = 0;
19
20    while(gridStart<n)
21    {
22          for(j = 0; j < (PS-1); j++)
23             sdata[tid] += data[gridStart + start + j * blockSize];
24          __syncthreads();
25
26          SPU = BBS - (PS-1) * blockSize;
27
28          if(tid < SPU)
29             sdata[tid] += data[gridStart + start + (PS-1) * blockSize];
30           __syncthreads();
31
32           gridStart+=gridSize;
33    }
34    __syncthreads();
35    // do reduction in shared mem
36    if (blockSize >= 512) { if (tid < 256) { sdata[tid] += sdata[tid + 256]; } __syncthreads(); }
37    if (blockSize >= 256) { if (tid < 128) { sdata[tid] += sdata[tid + 128]; } __syncthreads(); }
38    if (blockSize >= 128) { if (tid <  64) { sdata[tid] += sdata[tid +  64]; } __syncthreads(); }
39
40    #ifndef __DEVICE_EMULATION__
41    if (tid < 32)
42    #endif
43    {
44          if (blockSize >=  64) { sdata[tid] += sdata[tid + 32]; EMUSYNC; }
45          if (blockSize >=  32) { sdata[tid] += sdata[tid + 16]; EMUSYNC; }
46          if (blockSize >=  16) { sdata[tid] += sdata[tid +  8]; EMUSYNC; }
47          if (blockSize >=   8) { sdata[tid] += sdata[tid +  4]; EMUSYNC; }
48          if (blockSize >=   4) { sdata[tid] += sdata[tid +  2]; EMUSYNC; }
49          if (blockSize >=   2) { sdata[tid] += sdata[tid +  1]; EMUSYNC; }
50    }
51
52    // write result for this block to global mem
53    if (tid == 0) odata[blockIdx.x] = sdata[0];
54 }
```

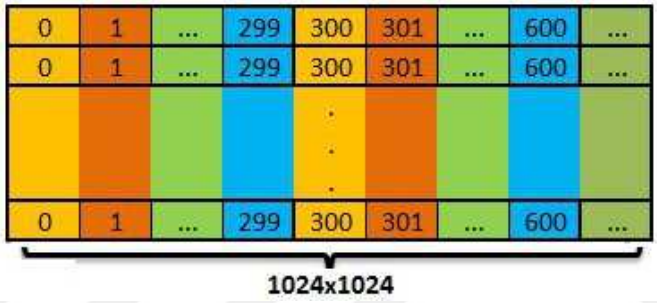
Burada ek olarak çekirdekte kullanılan bazı parametreler eklenmiştir. Bunlardan ilki boyuttaki bileşen sayısı için kullanılan *BBS* (Boyut Bileşen Sayısı)'dır. *BS* ızgarada kullanılan blok sayısını gösterir. *PS* ise hesaplanması gereken dizi parçaları uzunluğunu kaç blok uzunluğu ile kapsanacağını gösteren bir parametredir. Örnek olarak dizi uzunluğu 300 blok uzunluğu 128 olduğunda 3 adet blok dizi parçasını kapsar. *SPU* parametresi ise son parçanın uzunluğunu ifade eder. Örnekte $BBS-(PS-1)*PU = 300 -2*128 = 44$ son parça sayısını ifade eder. *start* değişkeni bir önceki kodda olduğu gibi blokların hesaplamaya başlayacağı elemanın indeksini hesaplar. Burada hem bloklar hem de ızgara hesaplamalar tamamlandıkça öteleneceği için ek olarak gridSize değişkeni vardır. Bu değişken ızgaranın ne kadar öteleneceği hesabında kullanılır.

20. satırdaki while döngüsü ile başlanır. Bu döngü ızgaranın hesaplamalar sırasında ötelenmesi için kullanılır. daha sonra 2. boyuttaki YBMG terimleri hesaplamakta kullanılana benzer bir kod kullanılır. Burada farklı olarak başka bir döngü içinde olduğundan en optimize olacak şekilde yazılmıştır. 22. satırdaki for döngüsü en son

blok ötelemesi hariç tüm işlemleri tamamlar. Bu döngü sonrasında 26. satırda son blok ötelemesinde ne kadar elemanın hesaplamaya dahil edileceği hesaplanır. 28. satırda da bu hesaplama kullanılarak hesaba katılmaması gereken fakat blok sınırları içerisinde kalan elemanlar dışlanır. Bu işlemden sonra ızgara içindeki tüm blokların sorumlu oldukları hesaplamaları tamamlanır. 32. satırda ızgaranın ötelenmesi için başlangıç noktası ızgara uzunluğu kadar arttırılır. Bu süreç veri dizisi sonuna kadar devam eder. Böylece boyuttaki YBMG bileşenleri hesabında kullanılacak toplamlar elde edilmiş olur.

6.1.4.3 Son Boyuta Ait YBMG Terimleri Hesabının CUDA ile Koşutlaştırılması

Son boyuta ait düğüm noktalarına ait YBMG terimleri hesabı için kullanılan veri çizemi Şekil 6.9'de görülmektedir.



Şekil 6.9: Son boyuta ait düğüm noktalarının YBMG birli terimlerin veri kullanım çizemi

Özyineli ilişkilerden bilinmektedir ki son boyutun birinci düğüm noktasının YBMG terimi hesabında veri satırının ilk elemanı alınır, son boyuttaki bileşen sayısı kadar ötelenir ve ötelendiği yerdeki veri ile toplanır. Bu süreç veri satırı sonuna kadar devam eder. Benzer süreç diğer düğüm noktalarının YBMG terimleri hesabında da kullanılır.

Diğer algoritmalarda hatırlanacak olursa hesaplanması gereken YBMG bileşen kadar ızgarada blok açılıyor ve bloklar sorumlu oldukları dizi parçaları üzerinde işlemlerini tamamlıyorlardı. Şekil 6.9'de görüldüğü gibi ardışık elemanlar arasında herhangi bir işlem bulunmamaktadır. Dolayısıyla önceki yöntemlerden farklı olarak burada her bir blok tek bir YBMG toplamından sorumlu olmak yerine birçok YBMG teriminin toplamından sorumlu olacaktır. Fakat bu aşamada farklı bir sorun ortaya çıkar. Bilindiği gibi CUDA'da blok içinde iş parçacığı açma sayısı sınırlıdır, hesaplanması gereken YBMG terimlerinin sayısı bu limitleri aşabilir ki diğer algoritmalarda da bu durum düşünülerek kodlar oluşturulmuştu. Diğer algoritmalara benzer şekilde bloklar

ötelenebilir, fakat tek bir blok içindeki iş parçacıklarının toplamı terim sayısından az ise yanlış elemanları birbirleri ile toplayacaktır. Tüm bu durumlar göz önüne alınarak aşağıdaki *SumArray*3 çekirdeği oluşturulmuştur.

```
 1 __global__ void SumArray3(float *data, float *odata, unsigned long n, unsigned int
BBS, unsigned int blockSize, unsigned int SPU, unsigned int BS, unsigned int PS)
 2 {
 3
 4   // BBS: Boyut Bilesen Sayisi
 5   // blockSize: Blok uzunlugu
 6   // SPU: Son parca uzunlugu, dizi parcasi uzunlugunun blockSize a bolumunden kalan parca
 7   // BS: Blok Sayisi
 8   // PS: Parca sayisi, dizi parcasini kac adet blogun hesapladigini gosterir
 9
10   unsigned int tid = threadIdx.x;
11   unsigned int start = ( (blockIdx.x/PS) * BBS ) + ( (blockIdx.x% PS) * blockSize );
12   unsigned int i = start + tid;
13   unsigned int gridSize = ( BS/PS ) * BBS;
14   unsigned int globalIndex = (blockIdx.x % PS) * blockSize + tid;
15
16   extern __shared__ float sdata[];
17
18   sdata[tid] = 0;
19
20   while(i<n)
21   {
22
23      if( blockIdx.x%PS <(PS-1) )
24      {
25         sdata[tid] += data[i];
26      }
27      else if (tid<SPU)
28      {
29         sdata[tid] += data[i];
30      }
31      __syncthreads();
32
33      i += gridSize;
34   }
35   __syncthreads();
36
37   atomicFloatAdd(&odata[globalIndex], sdata[tid]);
38 }
```

Kodda *start*, *gridSize* ve *globalIndex* değişkenleri hesabı en kritik hesaplamalardır. Bu hesaplamaları daha iyi anlatmak için örnek uzay üzerinden anlatım yapılacaktır. 6.9'de görüldüğü gibi her 300 elemanda bir yapı tekrarlanmaktadır. Izgaradaki blok uzunluğu 128 olarak seçilsin. Dolayısıyla tekrarlanan yapıyı 3 adet blok kapsamaktadır. Parça sayısı 3 olarak belirlendikten sonra değişkenlerin hesabına başlanabilir. 11. satırdaki *start* değişkeninin hesabında iki adet toplam bileşeni yer almaktadır. Hesaplamanın ilk parçası olan $((blockIdx.x/PS) * BBS)$ ile hangi bloğun hangi 300'lük yapının hesabında kullanılacağını belirler. $((blockIdx.x\%PS) * blockSize)$ ise bu yapı içinde bloğun sorumlu olduğu parçanın başlangıç elemanının yerini hesaplar. *gridSize*, ızgaranın ne kadar öteleneceğini belirler. *globalIndex* değişkeni ise kodun ilgili kısmına gelindiğinde açıklanacaktır.

Bir önceki koddaki gibi en dıştaki while döngüsü ızgara ötelemesini kontrol eder. Daha sonra gelen if döngüsü ile 3 adet blok genel bellekteki 300'lük yapıların içindeki veriyi paylaşılan belleğe toplayarak aktarır. Son parçadan sorumlu blok ise bir sonraki 300'lük yapının başlangıcına kadar olan elemanların işlemlerinden sorumlu olacak şekilde ayarlanmıştır. Bu şekilde ardışık her bir 3'lü bloktaki paylaşılan bellekler 300'lük yapıdaki veri parçaları içindeki elemanların düzgün sırada toplanmasını sağlar. Bu aşamada önemli nokta doğru sıradaki bloğun doğru sıradaki diğer blok içindeki veri ile toplanmasıdır. Burada CUDA'nın yapısından kaynaklanan bir kısıt vardır. Blok içindeki iş parçacıklarının eşleşmeleri (ing: sychronization) *syncthread*() fonksiyonu ile kontrol edilir. Fakat CUDA'da bloklar arasında eşleştirme yoktur. Bloklar arasında yapılacak işlemler için *atomic* fonksiyonlar kullanılmalıdır. Bu fonksiyonlar bloklar arasındaki bu tür işlemlerin yapılmasına olanak verir, fakat bun işlemleri seri olarak yapar. Bloklar paylaşılan belleklerindeki veriyi paylaşılan belleklerinden sırayla aktarırlar. Bu nedenle başarımlı fonksiyonlar değildirler. Bu özelliklerinden dolayı yapılan testlerde blok sayısı ne kadar fazla ise başarımın düşük olduğu gözlenmiştir. Testler sonucunda en optimum blok sayısının 64 olduğu tespit edilmiştir. Bu aşamada kod açısından bir kısıtın belirtilmesi gerekmektedir. Blok sayısının parça sayısının katları olması gerekmektedir. Bu kısıt sağlanmaz ise ızgara ötelenmesi sonucu yanlış noktadan başlanır ve hesaplama yanlış sonuç verir. Fakat bu önemli bir kısıt değildir, 64 sayısından başlanarak basit bir for döngüsü ile hangi sayı parça sayısına tam bölünüyor ise o sayı blok sayısı olarak atanır. 37 satırdaki *atomicAdd*() fonksiyonu ile her bir blok paylaşılan belleklerindeki verileri genel bellekteki *odata* dizisine sırayla aktarır. Burada kritik nokta *globalIndex* hesaplamasıdır. Bu hesaplama ile her bir üçlü blok paylaşılan bellekteki verileri genel bellekteki doğru adres üzerinde toplayarak hesaplamayı tamamlarlar.

6.1.4.4 Algoritmanın Başarım Değerlendirmesi

Başarım değerlendirmesine geçmeden önce CPU ve GPU sistem özellikleri, karşılaştırma için verilmelidir.

CPU System:

- 2 x Intel(R) Xeon QuadCore X5550 2.67GHz

- 32GB 1333Mhz DDR3 667Mhz

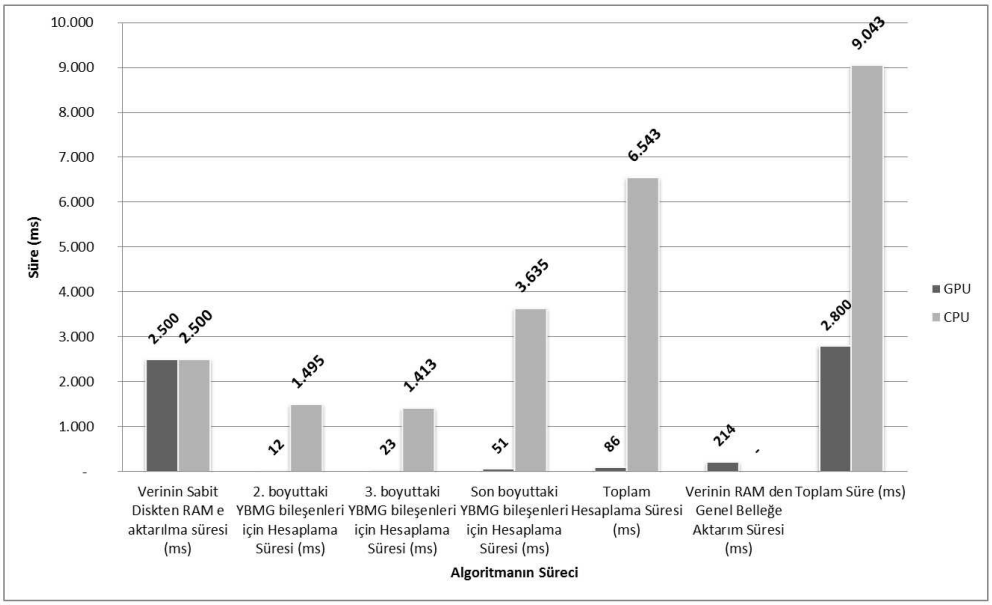
GPU System:

- Nvidia(R) Tesla C1060 GPU

- 240 x 1.296Ghz Stream Processors

- 4GB DDR2 667Mhz Genel Hafıza

CUDA başarım analizi için *NVIDIA Visual Profiler* kullanılmıştır. Alınan veri setinin ilk veri satırı için başarım değerlendirmesi Şekil 6.10'da görülmektedir.



Şekil 6.10: CPU ve GPU sistemlerinin YBMG yöntemi için başarım karşılaştırması

Şekil 6.10'da yatay eksende veri setinin ilk satırı için tüm süreç sıralanmıştır. İlk olarak veri sabit diskten RAM'e aktarılmıştır. Daha sonra GPU hesaplamaları için veri ekran kartının genel belleğine aktarılmıştır. Bu sırada CPU hesaplama sürecine başlamıştır. İkinci, üçüncü ve son boyutlardaki toplamlar hem CPU hem de GPU için hesaplanmıştır. İkinci boyuttaki hatırlanırsa 1024 adet blok ve her bir blokta 256 iş parçacığı vardı. Bu seçim GPU için en performanslı seçimlerden biridir. Seçilen 256 adetlik iş parçacığı sayısı 32'lik WARP yapısına uygundur. SM'lerin tam başarımlı çalışıp çalışmadığını sorgulayan *Occupancy* 1 (%100) olarak ölçülmüştür. Yazmaç kapasitesi sınırda kullanılmuştır (Yazmaç kullanım oranı: 1 (16384 / 16384) [İş parçacığı başına 15 yazmaç]), fakat sınır aşılmadığı için herhangi bir başarım kaybı sözkonusu

değildir. En önemli başarım ölçütlerinden biri de çekirdeğin genel bellekle olan veri aktarım bant genişliğidir. Bu değer teorik sınıra ne kadar yakın ise çekirdek o denli başarım sağlamış kabul edilir. Genel olarak teorik en üst limitin %70 ve üzeri sonuçları başarılı olarak kabul edilir. Kullanılan ekran kartının genel bellek veri aktarım bant genişliği 102,4GB/sn'dir. Başarımı analiz edilen *SumArray* çekirdeğinin ulaştığı bant genişliği 85,84GB/sn'dir. Bu bakımdan diğer analizlerle de birleştirildiğinde başarımlı bir CUDA çekirdeği olduğu söylenir. Hatta en iyi rakibi olan *CUDA Reduction* çekirdeği bile aynı bant genişliğinde bir başarıma sahip olmasına karşın hesaplama süresi olarak az bir farkla da olsa daha yavaştır (*SumArray* hesaplama süresi 11,87 ms iken *CUDA Reduction*'da aynı süre 14.85ms'dir). Üstelik *CUDA Reduction* sadece tüm veri setindeki elemanları toplamaktadır. Oysa *SumArray* bir adım ileri giderek veri dizisi içinde istenilen veri parçaların grupları içindeki elemanları da kendi aralarında toplayabilmektedir.

Üçüncü boyut bileşenlerinin YBMG terimleri hesaplama başarımına bakılacak olursa, yine 1024 adet blok ve her bir blokta 256 iş parçacığı kullanılmıştır. Benzer şekilde bloklarda 256 iş parçacığı kullanılması 32'lik WARP yapısına uygundur. Dolayısıyla genel başarım ölçütleri bir öndeki çekirdek ile aynıdır. Fakat burada dikkat edilmesi gereken önemli bir nokta vardır. *SumArray* çekirdeğinde her bir bloğun hesaplaması gereken dizi parçası gözönüne alındığında 32'lik WARP yapısına tamamen uymaktadır. Her blok 300x1024 adet bileşenden sorumluydu ve bloklarda hiçbir iş parçacığı işlevsiz kalmamıştır. Bloklardaki tüm iş parçacıkları istisnasız verideki bir elemandan sorumlu olmuşlardır. Üçüncü boyutta ise durum farklıdır. Hatırlanırsa her bir veri parçasının uzunluğu 300 idi ve bu sayı 32 ile tam olarak bölünmez. Dolayısıyla WARP yapısına uygun değildir. CUDA programlamada işler 32'lik WARP'lar olarak ayrılıp yapılmaktadır ve *SumArray*2 çekirdeğinde veri parçasının son kısmı hesaplanırken bazı iş parçacıkları işlevsiz kalmaktadır. Her ızgara ötelendiğinde de bu durum tekrarlanmaktadır. Dolayısıyla bu durum *SumArray*2 çekirdeğinin genel bellek veri aktarım bant genişliğinin 69,94GB/sn olmasında etkilidir. Fakat bu durumda bile başarım literatürdeki yüksek başarımlı sonuçlar içinde kalır. Ayrıca CPU'ya göre hesaplamayı 50 kattan hızlı bir şekilde yapmıştır.

Son boyutta CPU performansının oldukça kötü olduğu gözlemlenir. Bunun sebebi de açıktır, hesaplamada yer alan iç içe for döngülerinde toplama işlemi ardışık değildir. Bu nedenle her bir hesaplama oldukça fazla ötelemeli ve her ötelemede sadece 1 toplama

işlemi yapılarak tamamlanır. GPU programlamada, kendine özgü yapısı sayesinde bu durumda bile çok büyük bir başarım kaybı yaşanmamaktadır. Buradaki başarım kayıpları bir önceki çekirdektekiler dışında en önemli neden *atomic* fonksiyonların kullanılmak zorunda kalınmasıdır. Bu zorunluluk yüzünden daha fazla başarım kaybı yaşamamak için blok sayısı 64 ile kısıtlanmıştır. Bu durumda çekirdekte çok daha fazla ızgara ötelemeleri meydana gelmiştir. Aynı zamanda *atomic* kullanılması genel bellek veri aktarım bant genişliğini önemli oranda düşürmüştür. Tüm bu etmenler gözönüne alındığında veri bant genişliği 61,48GB/sn olarak görülmektedir. *Occupancy* etkinliği de bunlara bağlı olarak 0.5 (%50) ile 0.5 (%75) arasında olarak gözlenmiştir.

Buraya kadar incelenen süreler hesaplama süreleridir ve en kötü durumda bile CPU'ya göre kat ve kat üstündür. Fakat CUDA programlamaya özgü zaman yitimleri söz konusudur. GPU'nun veriyi işleyebilmesi için öncelikle veriyi bilgisayarın RAM'inden kendi genel belleğine aktarması gerekmektedir. Aktarım için geçen süre 214 ms'dir. Bu süre, GPU üzerinden yapılan toplam hesaplama süresinden çok daha büyüktür. Ek olarak verinin hem CPU hem de GPU üzerinde işlenebilmesi için öncelikle verinin sabit diskten bilgisayarın RAM'ine aktarılması gerekmektedir. Bu süre de ortak olarak 2500 ms'dir. Sabit diskten RAM'e aktarım süresi gözönüne alındığında GPU üzerinden çalıştırılan algoritmada bulunan hesaplamalar ve veri aktarım süreleri ihmal edilebilecek kadar düşüktür. Bunun nedeni *Reduction* türü algoritmaların eleman başına 1 $flop$ işlem maliyetinin olmasıdır. Veri yanında hesaplama maliyeti düşük kalmaktadır. Dolayısıyla bu durum sonuçlara da yansımaktadır. Parallel I/O özelliğine sahip MPI programlama algoritmanın bu yapısından dolayı CUDA programlamaya üstün gelecektir. Bu tür problemler için en optimal çözüm MPI ve CUDA programlamanın birlikte kullanılabildiği donanımlardır. Verilerin düğüm node'larına aktarımı ve dağıtımı MPI Parallel I/O ile gerçekleşir, hesaplamalar ise GPU üzerinde CUDA programlama ile yapılır. Testeki durumda bile tüm süreler göz önüne alındığında GPU, CPU'ya göre 3 katın üzerinde başarım elde etmiştir.

7. SONUÇLAR

Bu tez çalışmasında özgün olarak geliştirilen ve yukarıda da belirtilen çalışmaları maddeler şeklinde özetlersek:

- Çok boyutlu veri bölüntüleme amacı için geliştirilmiş ve uygulamaları bulunan YBMG yönteminin temel eşitlikleri koşutlaştırma amacıyla iyileştirilmiştir. Yöntemde ele alınan problemin boyut ve bileşen sayılarına göre yapıları ve hesaplama karmaşıklığı değişim gösteren bu eşitlikler iyileştirme süreci sonucunda boyut ve bileşen sayısından bağımsız bir yapıya kavuşmuştur.
- Graph teorisi kullanılarak yöntemde kullanılan eşitliklerin eşitliklerin matematiksel yapısı ortaya çıkarılmıştır.
- İyileştirilmiş yeni yapı sayesinde C/Fortran gibi makina diline daha yakın ve hızlı dillerle yöntemin kodunun yazılması olanaklı hale gelmiştir.
- Yöntemin seri C kodu yazılmış ve başarımlı bir şekilde çalışır hale getirilmiştir. Bu sayede yüksek hacimli problemlere sahip olmayan kullanıcıların da sonuçları daha hızlı bir şekilde elde etmeleri sağlanmıştır.
- Bunun yanısıra tezin amacına uygun olarak MPI kitaplığı kullanılarak yöntem başarımlı bir şekilde koşutlaştırılmıştır. Benzer şekilde CUDA kitaplığı kullanarak da koşutlaştırma başarımı elde edilmiştir. Bu koşutlaştırmalar sayesinde hem supercomputer altyapısına sahip hem de CUDA çalıştırabilecek donanıma sahip kullanıcıların yüksek hacimli problemler üzerinde ele alınan yöntemi kullanma olanağı sağlanmıştır.

ÖZGEÇMİŞ

Ad Soyad: Mehmet Engin Kanal

Doğum Yeri ve Tarihi: İstanbul, 23/08/1977

Lisans Üniversite: D.E.Ü., Matematik Eğitimi Bölümü (1995-1999)

Yüksek Lisans Üniversite: Marmara Ü., Fen Bilimleri Enstitüsü, Uygulamalı Matematik (1999-2002)

Yayın Listesi:

- **Kanal M.E., Baykara N.A., Demiralp M.**, 2011. , Theory And Algorithm of The Inversion Method For Pentadiagonal Matrices, Journal of Mathematical Chemistry, DOI:
- **Kanal M.E., Demiralp M.**, 2011. , A Modified Method of Calculating High Dimensional Model Representation (HDMR) Terms for Parallelization with MPI and CUDA, Journal of Supercomputing, DOI: 10.1007/s11227-011-0695-0
- **Kanal M.E.**, 2010. , Parallel Algorithm on Inversion for Adjacent Pentadiagonal Matrices with MPI, Journal of Supercomputing, DOI: 10.1007/s11227-010-0487-y
- **Kanal M. E.**, Demiralp M., 2008. Data Partitioning via High Dimensional Model Representation by Using Paralel Computing, Proceedings of the 1st WSEAS International Conference on Multivariate Analysis and its Application in Science and Engineering (MAASE 08), İstanbul, Turkey
- Üsküplü S., **Kanal M. E.**, Demiralp M., 2005. Extension of A Finite Regular Data Given To Determine A Univariate Function By Using Forward and Backward Differences, International Conference of Numerical Analysis and Applied Mathematics (ICNAAM 2005), Rhodes, Greece
- **Kanal M. E.**, Üsküplü S., Demiralp M., 2005. Precision Increased Truncated Derivative Formulae in Terms of Forward and Backward Difference Operators, International Conference of Numerical Analysis and Applied Mathematics (ICNAAM 2005), Rhodes, Greece
- **Kanal M. E.**, 2004. Parallel Algorithm on Inversion for Adjacent Pentadiagonal Matrices with MPI, International Conference of Computational Methods in Sciences and Engineering 2004 (ICCMSE 2004), Attica, Greece
- **Kanal M. E.**, Baykara N.A. and Demiralp M., 2003. A Novel Approach To The Numerical Inversion Algorithm For Adjacent Pentadiagonal Matrices, Fourth International Conference (MathTools 2003), St. Petersburg, Russia
- Baykara N.A., **Kanal M. E.**, Yaman İ. and Demiralp M., 2001. Çarpımsallaştırılmış Yüksek Boyutlu Model Gösterilimi Sınama Uygulamaları: Ü ç Köşegenli Matrislerin Determinantlarının Sayısal Hesaplamaları , XII. Ulusal Mekanik Kongresi, Konya

M.Engin Kanal 1977’de İstanbul’da doğdu. Lisans derecesini 1999 yılında Dokuz Eylül Üniversitesi, Buca Eğitim Fakültesi, Matematik Eğitimi Bölümü’nden aldı. Yüksek lisansını Marmara Üniversitesi Fen Bilimleri Enstitüsü Uygulamalı Matematik Programı’ndan 2002 yılında almıştır. 2003 yılında İstanbul Teknik Üniversitesi, Bilişim Enstitüsü, Hesaplamalı Bilim ve Mühendislik Programı’nda doktoraya başlamıştır.

KAYNAKLAR:

[1] Rabitz H. and Alis O.F. and Shorter J. and Shim K., (1999). Efficient Input-Output Model Representations, *Computer Physics Communications*, **117 (1-2)**, 11-20.

[2] Shorter J.A. and Precila C.I. and Rabitz H., (1999). An Efficient Chemical Kinetics Solver Using High Dimensional Model Representation*, J. Phys. Chem. A*, **103**, 7192-7198.

[3] Li G. and J. Hu and S.-W. Wang and Rabitz H., (2002). Practical Approaches To Construct RS-HDMR Component Functions, *Phys. Chem. A*, **106 (37)**, 8721-8733.

[4] Li G. and J. Hu and S.-W. Wang Georgopoulos P.G. and Schoendorf J. and Rabitz H., (2006). Random Sampling-High Dimensional Model Representation (RS-HDMR) and Orthogonality of Its Different Order Component Functions, *Phys. Chem. A*, **110 (7)**, 2474-2485.

[5] Tomlin A.S., (2006). The Use of Global Uncertainty Methods for the Evaluation of Combustion Mechanisms, *Reliability Engineering and System Safety*, **91**, 1213-1231.

[6] Ziehn T. and Tomlin A.S., (2008). A Global Sensitivity Study of Sulphur Chemistry in a Premixed Methane Flame Model Using HDMR, *International Journal of Chemical Kinetics*, **40 (11)**, 742-753.

[7] Tunga M.A. and Demiralp M., (2003). Data Partitioning via Generalized High Dimensional Model Representation (GHDMR) and Multivariate Interpolative Applications, *Mathematical Research,***9**, 447-462.

[8] Ho T.-S. and Rabitz H., (2003). Evaluation of city refuse compost maturity Reproducing Kernel Hilbert Space Interpolation Methods as a Paradigm of High Dimensional Model Representations: Application to Multidimensional Potential Energy Surface Construction, *Journal of Chemical Physics*, **119 (13)**, 6433-6442.

[9] Tunga M.A. and Demiralp M., (2005). A Factorized High Dimensional Model Representation on the Partitioned Random Discrete Data, Applied Numerical Analysis and Computational Mathematics, 1 (1), 231-241.

[10] Tunga M.A. and Demiralp M., (2005). A Factorized High Dimensional Model Representation on The Nodes of a Finite Hyperprismatic Regular Grid. Applied Mathematics and Computation, 164 (3), 865-883.

[11] Sobol I.M., (1993). Sensitivity Estimates for Nonlinear Mathematical Models, Mathematical Modelling and Computational Experiments, 1, 407-414.

[12] Tunga M.A. and Demiralp M., (2006). Hybrid High Dimensional Model Representation (HHDMR) on The Partitioned Data, Journal of Computational and Applied Mathematics, 185 (1), 107-132.

[13] Wolfram Math World Web Site, (Erişim Tarihi: Kasım 2010). Grid Graph, http://mathworld.wolfram.com/ GridGraph.html.

[14] Wolfram Math World Web Site, (Erişim Tarihi: Kasım 2010). Complete k-Partite Graph, http://math\-world.wolfram.com/Completek-PartiteGraph.html.

[15] Wolfram Math World Web Site, (Erişim Tarihi: Kasım 2010). k-Partite Graph, http://mathworld.wolfram.com/k-PartiteGraph.html.

[16] Wolfram Math World Web Site, (Erişim Tarihi: Kasım 2010). Acyclic Digraph, http://mathworld.wolfram.com/AcyclicDigraph.html.

[17] Argonne National Laboratory Web Site, (Erişim Tarihi: Kasım 2010). Acyclic Digraph, http://www.mcs.anl.gov/research/projects/mpi/www/www3/MPI_Reduce.html.

[18] Mark Harris, (Erişim Tarihi: Kasım 2010). Optimizing Parallel Reduction in CUDA, NVIDIA Developer Technology Whitepaper.

[19] Wilkinson B. and Allen M., (2004). Parallel Programming Techniques and Applications Using Networked Workstations and Parallel Computers 2nd ed., Pearson Education Inc.

[20] M. Alper Tunga, (2006). Data Partitioning and Multivariate Interpolation via Various High Dimensional Model Representations, İstanbul Teknik Üniversitesi Doktora Tezi.

[21] Evrim Korkmaz, (2009). Bütünleştirilmiş Küçük Ölçekli Yüksek Boyutlu Model Gösterilimi ve Çok Değişkenli İşlev Yaklaştırımında Kullanımı İstanbul Teknik Üniversitesi Yüksek Lisans Tezi.