計算機アーキテクチャ　144−17
（２００１．７．２６）

ループパーティショニングを用いたショートベクトル化技法

佐渡昭彦[†]　古関聰[††]
小松秀昭[††]　深澤良彰[†]

本稿では、ループパーティショニングを用いたショートベクトル化技法を提案する。ショートベクトルプロセッサは、ベクトル長が短い為、依存の制限が緩く、ベクトル実行しやすいことに着目した。また、通常ではベクトル実行できない命令に対し、ループ交換やループスキューイングなどのユニモジュラ変換を施すことにより、ベクトル化率を高める。ユニモジュラ変換を施すとベクトル実行に必要なメモリ上でのアドレス連続性が失われるが、本手法では、プリフェッチ命令を用いてデータをアドレス連続な領域にコピーして、ベクトル実行を行う。

A Short Vector Extracting Technique Using Loop Partitioning

Akihiko Sado [†], Akira Koseki [††], Hideaki Komatsu [††]
and Yoshiaki Fukazawa [†]

This paper addresses a short vectorization technique based on the loop partitioning. The restriction of dependence in short vector processors is relatively weak because the vector length is short, then vectorized execution can be easily realized. Our technique also includes unimodular loop transformations such as loop interchange and loop skewing to vectorize more instructions that cannot be vectorized in a naive manner. Unimodular loop transformations may breaches the memory contiguity of operations that is required for the vectorized execution. However we prefetch the data onto a contiguous the memory region so that we can proceed on vectorization.

1. はじめに

ベクトル演算装置は、SIMD(Single Instruction Multiple Data) を実現するハードウェア機構である。SIMD とは、１つの命令で複数のデータを同時に処理することであるが、パーソナルコンピュータでは必要なかった為、数年前まではスーパーコンピュータにしか搭載されていなかった。ところが、近年の目覚しいプロセッサ技術の発展と、大量のデータ処理を必要とするマルチメディアソフトウェアの普及により、パーソナルコンピュータでもベクトル長の短いショートベクトル演算装置の組み込まれたプロセッサが使用されるようになった。

このベクトル演算装置を使い、ループの各イタレーションで順番に行われる配列の演算を、配列の要素をベクトルの要素とみなして、ベクトル処理すると、ループを高速に実行できる[1)]。しかし、ベクトル実行では数個の繰り返しが一度に処理されるため、命令の実行順序に変化が起きる場合があり、対象プログラム中に存在する依存の性質によってはベクトル化できない場合もある。

ループの並列化を行う手法としてループパーティショニング[2)]がある。ループパーティショニングとは、依存ベクトルの情報をもとに、イタレーション集合であるパーティション間で依存関係によるサイクルを形成しないようにパーティションを作成し、命令の実行順序を入れかえて、依存の制限を緩める為のプリフェッチを用いた、ループを高速に実行する手法である。また、ユニモジュラ変換であるループスキューイングやループ交換等のループ並列化手法も研究されてきた[3)]。ユニモジュラ変換は、プログラムにおける配列間の依存関係を依存ベクトルという依存距離を成分とするベクトル形式で表し、ループ内の依存ベクトル群に対して一次変換を施すことで、依存の性質を異なったものに変換する手法である。

本研究では、プリフェッチ命令やストリーミング・ライト命令を持ったマルチメディアプロセッサを対象とし

† 早稲田大学理工学部
School of Science and Engineering, Waseda Univ.
†† 日本 IBM(株) 東京基礎研究所
Tokyo Research Laboratory, IBM Japan, Ltd.

て、ユニモジュラ変換を用いたループパーティショニングを行うアルゴリズムを提案し、評価する。

2. 関連研究

ベクトル演算装置に関する研究として、文献 4) がある。この手法は、ベクトル実行を阻害する依存を解析し、ループのイタレーションをマルチプロセッサ上でベクトル実行して高速に実行している。しかし、ベクトル長が長く、ベクトル化の条件が厳しくなるため、ベクトル実行が不可能であることが多かった。それをある程度克服したのが、文献 5) である。この手法は、MMX というベクトル長が 1 〜 8 と短いベクトル演算装置を用いて、特にマルチメディアソフトフェアや DSP に対して優れた性能を示している。しかし、MMX を用いてもベクトル実行できないプログラムは多く存在しており、それらは逐次実行を余儀なくされている。

ループパーティショニングに関する研究としては、文献 2) がある。これは、プリフェッチ命令を用いることにより演算装置とメモリとのスケジューリングを組み合わせて高速実行することを実現している。しかし、全体を逐次実行しているため、本稿では、ベクトル演算装置を用いて、パーティションの内部をベクトル実行することによって高速実行することを考える。

3. 本手法の特徴

3.1 ショートベクトルプロセッサへの特化

本研究の対象はベクトルプロセッサの中でも一度に計算できる要素数が少ない、ショートベクトルプロセッサである。

ループパーティショニングは、キャッシュにデータが収まる程度の小規模なパーティションを作成して実行するので、ベクトル長の短いショートベクトルプロセッサに適応しやすい。本稿では、各パーティション内をベクトル実行することによってパーティショニングされたループを高速に実行することを試みる。

3.2 プリフェッチ命令によるベクトル化支援

ベクトル実行を行うためには演算に用いるデータがメモリ内でアドレス連続である必要がある。本研究では、イタレーション空間でパーティションの形を決定するのに依存ベクトルを用いる。また、パーティションの形を決定する 2 つの独立なベクトルをパーティション基本ベクトルといい、データがアドレス連続になるようにその 1 つをイタレーション空間で逐次実行が行われる向きに基本ベクトルをとる。しかし、それでは依存ベクトルに阻害されてベクトル実行できない場合があるので、その場合には、ユニモジュラ変換を用いて依存ベクトルを変換することによってベクトル実行を可能にする。

例えば、図 1のようなプログラムを考える。このプログラムをイタレーション空間 (j,i) で考えると、真依存である依存ベクトル (1,0)(2,1) が存在する。図 2(a) を見ると、メモリ上でアドレス連続な方向は内周ループの j の方向となり、依存ベクトル d1 がベクトル実行を阻害している。ただし、図中の IN とは内周ループのことであり、 OUT とは外周ループの事である。

```
For i=1 to 100
  For j=1 to 100
    A[i][j] = A[i][j-1] + A[i-1][j-2]
  End For
End For
```

図 1　ベクトル実行不可能な場合

このような場合、本研究ではユニモジュラ変換を用いて依存の形を変形させることによってベクトル実行を可能にする。図 1のプログラムに対してループ交換を行うと、図 3のプログラムのようになり、変換後の内周ループの方向にベクトル実行可能であることが分かる。

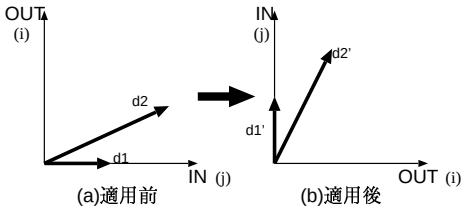

図 2　ループ交換による依存ベクトルの変形

```
For j=1 to 100
  For i=1 to 100
    A[i][j] = A[i][j-1] + A[i-1][j-2]
  End For
End For
```

図 3　ベクトル可能な場合

しかし、メモリ上でアドレス連続な方向は、図の IN 軸方向になり、実行順序とは異なってしまう。そこで、仮想メモリにアドレス連続になるようにコピーする必要がある。その際、 1 つずつコピーをしていたのでは時間がかかってしまうので、必要なデータを前もってキャッシュに収めるプリフェッチ命令やデータのメインメモリへの読み書きを他の処理とオーバーラップさせることができるストリーミング・ライト命令を用いることにより、コピーの時間を最小限に抑える。

3.3 パーティションの大きさ

パーティション内をベクトル実行する場合、なるべく多くのイタレーションがパーティション内に含まれるよう考慮することで実行効率を上げることができる。

パーティションは、形を決定する 2 つの 1 次独立なベクトル長 1 のパーティション基本ベクトル P_{X_0}、 P_{Y_0} によって決定する。パーティション基本ベクトルを数倍したものがパーティションベクトル P_X、 P_Y となり、パーティションの形と大きさを決定する。まず、パーティションベクトル P_Y の大きさを任意に決定してしまうと、図 4のように OUT 軸方向の上方部数行のイタレーションがパーティションに入らなくなってしまい、逐次実行するところが増えてしまう。そこで、本研究ではパーティションベクトル P_Y の OUT 軸方向の長さを外周ループの周回数の約数とする。こうすることにより、上方部をパーティションに的確に割り当てることができる。もし、外周ループの周回数が素数であったり、あまりにも長さが不適当であったりした場合には、外周ループの周回数から 1 を引いた約数で考えて、一番最後に残る 1 段は逐次実行するようにする。

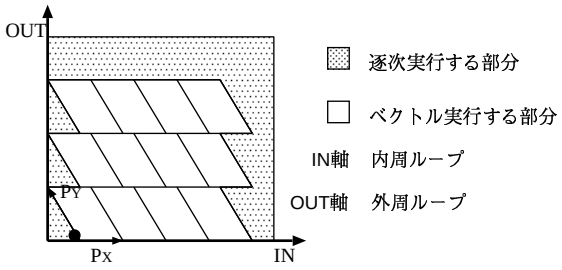

図 4　基本ベクトル P_X, P_Y のサイズの決定

同様に、パーティションベクトル P_X の大きさも逐次実行する部分をなるべく少なくすることを考える。図 4のような場合、 IN 軸方向の右端のイタレーションがパーティションに入らなくなってしまい、逐次実行する事になってしまう。よって、この逐次実行する部分を減らすようにする一方で、本手法では、パーティション内を無駄なくベクトル実行することを考えるため、パーティション内に逐次実行するイタレーションがないという条件のもとに大きさを決定する。

4. 本手法の流れ

本手法では、プリフェッチ命令やストリーミング・ライト命令を持ったベクトル長 4 であるマルチメディアプロセッサを対象とした 2 次元のループパーティショニングを行う。具体的には、図 5 で示すような流れに従ってパーティションを作成し、パーティション内部をベクトル実行する。

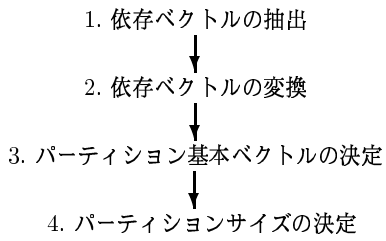

図 5　本システムの流れ

4.1 依存ベクトルの抽出

入力されたプログラムのうち、配列の添え字式同士を比較して得られる差分を求め、その値を成分とする依存ベクトルを抽出する。逆依存の場合は、ベクトル実行を阻害しないので、真依存を表す依存ベクトルの情報をもつ場合にループパーティショニングを行う。

4.2 依存ベクトルの変換

メモリ上でアドレス連続なところに、ベクトル長 4 よりも短い依存ベクトルが存在するとベクトル実行できない。そこで、ベクトル実行できない場合にはループ交換やループスキューニング等のユニモジュラ変換を用いて依存ベクトルを変形させる。

依存ベクトルがイタレーション空間の第一象限か IN 軸正の方向にのみ存在した場合、ループ交換を用いることによってベクトル実行を可能にする。

また、第一象限に IN 軸の正の方向か OUT 軸の正の方向にのみ依存ベクトルが存在した場合は、ループスキューイングを用いることによってベクトル実行を可能にする。

両方の条件を満たした場合には、ループ交換を用いる。なぜなら、ループスキューイングはイタレーション空間の形を変形し、逐次実行をする部分が存在してしまうが、ループ交換では逐次実行をなくし、ベクトル実行のみで実行することが可能な場合がある為、高速に実行できるからである。

例えば、図 6(a) の場合、 IN 軸方向に依存ベクトル d1 が存在してしまい、 d1 のベクトル長がショートベクトルプロセッサのブロック数より短いとベクトル実行できない。そこで、ループスキューイングを用いると図 6(b) のように変換される。すると、 IN 軸方向には依存が存在しなく、ベクトル実行できることが分かる。

しかし、ループ交換やループスキューイングを用いた

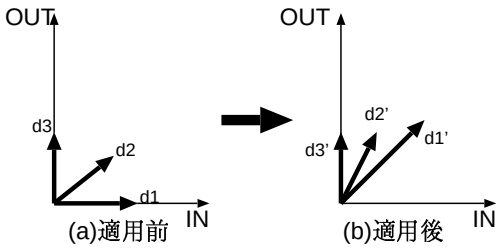

図 6　ループ交換による依存ベクトルの変形

ことにより、依存ベクトルを変形し、ベクトル実行可能なイタレーションを得ることができるが、ベクトル演算装置の条件となるメモリ上でアドレス連続なイタレーションとはならない。例えば、図 6(b) の場合、アドレス連続な方向は d1' 方向となっている。本研究では、メモリ上でアドレス連続でない部分をアドレス連続な仮想メモリにあらかじめプリフェッチしておくことによってコピーする時間を最小限に抑えて、ベクトル実行していく。

4.3　パーティション基本ベクトルの決定

ここで、パーティションの形を決定するパーティション基本ベクトル P_{X_0}、 P_{Y_0} の決定の際に重要な情報となるのような、図 7のような CW(=clockwise) ベクトルと CCW(=counterclockwise) ベクトルを定義する。CW とは、依存ベクトルのどれかと平行であり、 CW ではない他の依存ベクトルとのなす角は、全て CW から見て反時計回りに 180 度以内となるベクトルである。 CCW とは、依存ベクトルのどれかと平行であり、CCW ではない他の依存ベクトルとのなす角は、全て CCW から見て時計回りに 180 度以内となるベクトルである。。

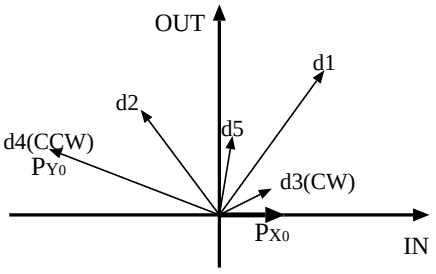

図 7　CW, CCW と依存ベクトル

例えば、図 7のようにプログラム中に 5 本の依存ベクトルが存在したと仮定する。この 5 本のベクトルの中から上記の CW と CCW の定義に値するものは CW が d3 であり、 CCW が d4 となる。

パーティション基本ベクトルは、 CW と CCW の間にとることはできない。もし、パーティション基本ベクトルが CW と CCW の間に存在したとすると、パーティション毎に実行することができなくなるからである。例えば、図 8のようなイタレーション空間に、 P_X,P_Y のようなパーティションベクトルを決めたとする。ここで、イタレーション内に存在しているベクトルは、点線で表されている d1,d2 の 2 種類である。ただし、 2 つのパーティションの境界線を中心に依存ベクトルを表記しているので、図中以外へのイタレーションに向かったり、図中以外のイタレーションから受けたりしている依存ベクトルは表記していない。依存ベクトル d1 がパーティション 1 からパーティション 2 へ依存を持ち、依存ベクトル d2 がパーティション 2 からパーティション 1 へ依存を持っているので、パーティション 1 を実行してからパーティション 2 を実行することも、パーティション 2 を実行してからパーティション 1 を実行することもできない。つまり、 2 つのパーティションが双方向の依存ベクトルを持っていると、実行することはできないため、このようなパーティションベクトルの取り方はできない。

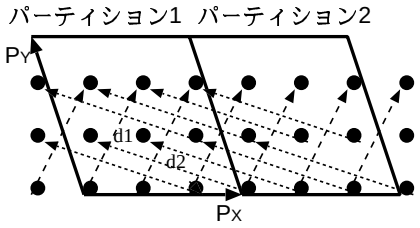

図 8　基本ベクトルの悪い取り方の例

パーティション基本ベクトルのうちの 1 つ P_{X_0} ベクトルを図 7の IN 軸と平行な $P_{X_0} = (1, 0)$ とする。 P_{X_0} を IN 軸と平行な方向にするのは、イタレーションの実行順序が IN 軸に平行なために、ベクトル実行するときにアドレス連続なイタレーションをとるのに有効であるからである。また、もう 1 つのパーティション基本ベクトル P_{Y_0} を CCW とする。ただし、 CCW が第一象限に存在した場合、 OUT 軸と平行な $P_{Y_0} = (0, 1)$ とする。これは、できるだけ多くのイタレーションをパーティションにおさめる為である。

4.4　パーティションサイズの決定

パーティションサイズを決定する際、以下の 4 つの条件を満たす必要がある。

パーティション内に含まれるイタレーションは、パーティションの境界線の左側と下側の線上とその内部であり、上と右側の線上の部分は含まない。

4.4.1 パーティションサイズの下限 (内周ループ)

本手法では、パーティションサイズの下限をあるパーティションから伸びる依存ベクトルが隣接するパーティションを超えることのないサイズとする。

例えば、図 9のようにあるパーティション 1 から伸びる依存ベクトルが、パーティション 1 に隣接するパーティション 2 及びさらにその隣接であるパーティション 3 に到達していたとすると、各パーティションをパイプライン処理させる場合に、パーティション同士の依存が複雑になってしまい、好ましくない。

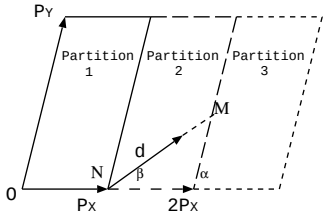

図 9 基本ベクトル P_X の制限

4.4.2 パーティションサイズの下限 (外周ループ)

図 10のように依存がパーティションの P_Y 方向の外に出ないような十分な大きさにもする。あまりにも小さいと、パーティションを作ってもせっかくのキャッシュを効果的に使うことができないからである。

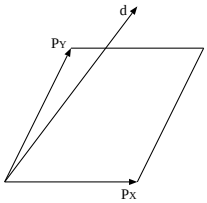

図 10 パーティションサイズの下限（外周ループ）

4.4.3 パーティションサイズの上限 (キャッシュ)

パーティショニングをする利益は、パーティションに区切ると、パーティション内の実行がキャッシュにすべてのデータが収まるので主メモリにアクセスする必要がなくなるということである。

そのため、パーティションサイズを極端に大きくとると、パーティション内の計算に必要なデータがキャッシュの許容量を超過してしまう。こうなることにより、プロセッサと主記憶装置の速度があまりにもかけ離れてしまっている現在の計算機においては、主記憶装置からデータを取得する行為自体が高速な計算の妨げになってしまう。

4.4.4 逐次実行を減らすためのパーティションサイズ

パーティショニングを行った時にパーティション内にイタレーションが少しでも多く入るように、パーティションベクトル P_Y の y 軸方向の長さを外周ループの周回数の約数にする。もし、素数であったり、上限下限を考えた場合に適当でない約数しかなかったりした時には、外周ループの周回数の回転数より 1 を引いた約数で考えて、一番上のイタレーションの行のみは逐次実行する。

P_X 方向も逐次実行を減らすように作成するが、パーティション内のイタレーションを全てベクトル実行することを優先する。つまり、 P_X 方向のベクトル長は、ショートベクトルプロセッサのブロック数の 4 の倍数とする。

4.5 ループパーティショニングの実行

以上のアルゴリズムに従ってパーティションを作成し、実行していく。パーティション外部は、逐次実行し、パーティション内部をベクトル実行する。現在のショートベクトルプロセッサでは、ベクトル命令によって並列化できる配列演算は、メモリ連続な配列に限られるといった制限がある。その為、各パーティション内のイタレーションの実行順序は、まず P_X 方向に実行し、P_X 方向が済んだら、 P_Y 方向に 1 単位移動し、 P_X 方向に実行していくとする。

5. 評　　価

本アルゴリズムを採用し、ループパーティショニングを用いたショートベクトル化技法の評価を行う。本手法と、逐次実行、従来手法やベクトル実行との性能比較を行った。その結果を表 1と 2に表す。プログラムには、 DSP ベンチマークを用いた。表中の WDF, IIR, DPCM, 2D, Floyd, MDFG は、それぞれ Wave Digital filter, Infinite Impulse Response, Differential Pulse-Code Modulation device, Two Dimensional filter, Floyd-Steinverg algorithm, Multi-dimensional Data-flow Graphs を指す。

評価にはプリフェッチ命令とストリーミング・ライト命令を持つショートベクトルプロセッサを搭載した仮想マシンを用いた。仮想マシンはロード命令にコストが 2 かかり、他の命令にはコストが 1 かかるとする。評価値は、逐次実行を 1 とした性能比を表している。プリフェッチ命令の発行後は他の命令を実行できるが、プリ

フェッチ命令によってフェッチするデータをプリフェッチ命令の直後には使用できないとしている。

使用したマシンは 1.5GHz の Pentium4 と 933Hz の Pentium III の 2 台である。

Floyd や MDFG では、依存ベクトルがベクトル実行を阻害し、ユニモジュラ変換による依存ベクトルの変形もできないため、本手法を用いることができなかった。しかし、他の 4 つのベンチマークでは用いることができた。WDF では、ユニモジュラ変換を施す必要もなく、実行できた為に、逐次実行では 2.8 〜 12.3 倍、従来のショートベクトル化技法では、1.097 〜 1.389 倍の性能向上が見込まれた。また、本手法のユニモジュラ変換を施すことによって実行できた IIR、DPCM、2D の 3 つもベクトル実行できなかったものをベクトル化可能とし、従来のループパーティショニング技法と比較して、1.01 〜 1.10 倍前後の性能向上が見られた。

表 1　Pentium4 による評価

	ベクトル実行有		ベクトル実行無	
パーティショニングの有無	有	無	有	無
WDF	2.821	2.724	1.113	1.000
IIR	5.952	不可	5.435	1.000
DPCM	1.089	不可	1.065	1.000
2D	1.206	不可	1.139	1.000
Floyd	不可	不可	1.104	1.000
MDFG	不可	不可	1.143	1.000

表 2　Pentium III による評価

	ベクトル実行有		ベクトル実行無	
パーティショニングの有無	有	無	有	無
WDF	12.29	8.634	2.766	1.000
IIR	3.154	不可	2.929	1.000
DPCM	1.057	不可	1.045	1.000
2D	1.852	不可	1.776	1.000
Floyd	不可	不可	1.107	1.000
MDFG	不可	不可	1.203	1.000

逐次実行に比べても本手法では 1.05 〜 12.3 倍と速度向上が見られるのは、プリフェッチ命令やストリーミング・ライト命令による、ロードやストアにかかるレイテンシの隠蔽と、ベクトル実行することによってパーティション内が高速に実行されていることを示していると思われる。

ユニモジュラ変換を施した場合には、既存のループパーティショニングを用いた場合よりも性能比が平均 5.13 %の向上にとどまっている。これは、プリフェッチ命令を用いてはいるが、メモリ上でアドレス連続でない部分を仮想メモリにコピーすることによっておこるものと思われる。

しかし、既存のループパーティショニングを用いた場合よりも速度が向上し、ベクトル化手法を用いるよりも速度が向上していることから、ベクトル化とショートベクトルプロセッサに搭載された命令の組み合わせによって効果が確実になっていることがわかる。

6. 終わりに

本稿では、ショートベクトルプロセッサの性質に着眼し、ループ内の配列演算をベクトル実行することによる高速化と、ショートベクトルプロセッサに搭載されたプリフェッチ命令やストリーミング・ライト命令の利用による高速化をねらうループパーティショニングを提案した。

本手法を DSP ベンチマークを用いて評価を行った。結果として、逐次実行した場合、既存手法を適用した場合、一般的なベクトル化手法を用いた場合と比較して、速度の向上が見られることから、本手法がショートベクトルプロセッサに対して有効であることがわかった。

今後の課題としては、本手法でも実行できないプログラムに対して、適用できるような依存ベクトルの変換手法を研究し、ベクトル化率をあげていきたい。

参考文献

1) R. Allen and K. Kennedy: 'Vector Register Allocation', IEEE Transactions on Computers, VOL.41, NO.10, pp.1290-1317, OCT. 1992

2) F. Chen and T. W. O'Neil: 'Optimizing Overall Loop Schedules Using Prefetching and Partitioning', IEEE Transactions on Parallel and Distributed Systems, VOL.11, NO.6, pp.604-614, JUNE. 2000

3) M. E. Wolf, and M. S. Lam: 'A Loop Transformation Theory and an Algorithm to Maximize Parallelism', IEEE Transactions on Parallel and Distributed Systems, VOL.2, NO.4, pp.452-471, OCT. 1991

4) H. Zima and B. Chapman: 'Supercompilers for Parallel and Vector Computers', ACM Press Frontier Series, Addison-Wesley, New York, 1990.

5) R.Bhargava and R.Radhakrishnan and B.L.Evans and L.K.John: 'Characterization of MMX-enhanced DSP and Multimedia Applications on a Gernal Purpose Processor', Digest of Workshop on Performance Analysis and Its Impact on Design , pp. 16-23, JUNE. 1998

6) 岸, 古関, 小松, 深澤、ショートベクトルプロセッサ向けループ並列化技法:「ハイパフォーマンスコンピューティングとアーキテクチャの評価」に関する北海道ワークショップ (HOKKE-2000),2000