ハイパフォーマンス
コンピューティング 52－3
（1994．7．22）

ウェーブフロント型ループの超並列計算機向けコンパイル技法

太田 寛　斎藤 靖彦　海永 正博　小野 裕幸
(株)日立製作所　システム開発研究所

偏微分方程式の求解法などに現れるウェーブフロント型ループの超並列計算機向けコンパイル技法として，タイリングが知られている。これは，通信オーバーヘッドを低減するため、複数のイタレーションをまとめたタイルを単位として通信を行うものである。このとき，タイルサイズを大きくすれば通信回数は減るが、2番目以降のプロセッサの起動が遅れる。本報告ではこのトレードオフを考慮して、一般的ウェーブフロント型ループに対するタイリング方法，特に最適タイルサイズ決定方法を提案する。本方式に従って試作したコンパイラプロトタイプによってSORプログラムを並列化し，実機評価した結果，タイリングしない場合と比べ10倍以上の性能が得られた。

Compiling Wavefront Loops for Massively-Parallel Processors

Hiroshi OHTA, Yasuhiko SAITO, Masahiro KAINAGA, Hiroyuki ONO
Systems Developement Laboratory, Hitachi Ltd.

Tiling is known as a technique to compile wavefront loops for massively-parallel processors. It reduces communication overhead by grouping multiple iterations into a *tile*. While taking larger tiles reduces the number of communications, it causes a delay in starting the 2nd and the following processors. We examine this tradeoff theoretically and show a tiling method for general wavefront loops, especially how to optimize the tile size. We have implemented this method in our compiler prototype, with which we have parallelized an SOR program. The tiled program executes ten times as fast as the one without tiling.

1 はじめに

ウェーブフロント型ループは，偏微分方程式の緩和法による求解などの現実的な問題に現われるループであり，ループ運搬依存のある多重ループとして特徴付けられる。このループを超並列計算機向けにコンパイルする場合，通常のDOACROSS型の並列化を行うと，ループのイタレーションごとにプロセッサ間通信が発生するため，良い性能が得られない。

この問題の解決のために，複数のイタレーションをまとめた「タイル」と呼ばれる単位ごとに通信を行う，「タイリング」という技法が提案されている[1][3][5]。タイリングには，タイルサイズを大きくすれば通信回数は減るが，2番目以降のプロセッサの処理の開始が遅れるというトレードオフがある。Ramanujamら[5]は，ウェーブフロント型ループの特性を数学的に取り扱い，タイルの形状やタイルサイズについて論じた。Hiranandaniら[6]はRice大のFortran Dコンパイラに関する論文の中で，単純なウェーブフロント型ループに対して，最適タイルサイズを示している。しかし，これらの研究はいずれも，コンパイラで採用できるような一般的なタイリング方法を示してはいなかった。

本報告では，イタレーション空間の次元数や，依存ベクタの方向などが任意であるような一般的場合を理論的に取り扱い，タイリング方法，特に最適タイルサイズの決定方法を示す。

2章では例を用いてウェーブフロント法とタイリングの概念を説明する。3章ではイタレーション空間が2次元の場合の最適タイルサイズを示す。4章では3章の結果を3次元以上のイタレーション空間に拡張する。5章では実機評価によってタイリングの効果を示す。

2 ウェーブフロント法とタイリング

2.1 ウェーブフロント法

ウェーブフロント型ループの例として，次の2重ループを考える。

```
for(i1=1;i1<N1;i1++)
  for(i2=1;i2<N2;i2++)
    a[i1][i2] = a[i1-1][i2]+a[i1][i2-1];
```

このループのイタレーションの集合は，図1に示すような2次元のイタレーション空間を構成する。個々のイタレーションは黒い点で示されている。配列参照に伴うデータの流れにより，イタレーション間には矢印で示されるような依存関係(実行順序の制約)がある。この依存関係は，2種類の依存距離ベクタ(1,0)，および(0,1)で表される。i1ループ，i2ループのいずれのループについても，ループ運搬依存があるのでDOALL型の並列実行はできない。

このようなループの並列実行方法として，以下に述べるウェーブフロント法が知られている。図1に示すように，イタレーション空間をi1次元で分割して1列ずつ各

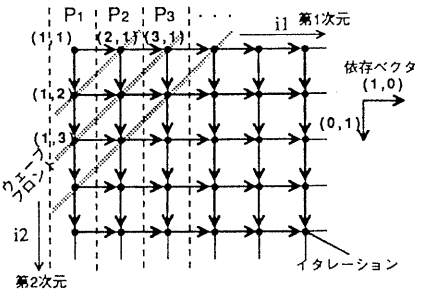

図1: イタレーション空間とウェーブフロント法

プロセッサP_1,P_2,... に割り付ける。最初のステージで，プロセッサP_1がイタレーション(1,1)を実行し，計算結果a[1][1]をプロセッサP_2に転送する。次のステージで，イタレーション(1,2)と(2,1)をプロセッサP_1とP_2が並列に実行し，計算結果をそれぞれ隣のプロセッサに転送する。以下同様に図に示したウェーブフロント(波頭)上のイタレーションを各ステージで並列に実行する。

ウェーブフロント法は，見方を変えれば，i1次元に関するDOACROSS型並列実行に他ならない。したがって，i2次元についてのループ運搬依存(0,1)がなく，i1次元についてのループ運搬依存(1,0)のみがある場合も，全く同様のウェーブフロント法が可能なことに注意してほしい。そのような場合も含めて，本報告ではウェーブフロント型ループを次のように定義する。

ウェーブフロント型ループとは，次の条件を充たすループネストである。

(a) 並列実行したい次元についてループ運搬依存がある。
(b) 並列実行したい次元の内側に別の次元を含む。

どの次元で並列実行するかは，ユーザの指示文などに従って決まっているものとする。

2.2 タイリング

ウェーブフロント法を超並列計算機で実行する場合には，プロセッサ間通信の起動時間がプロセッサ内演算時間に比べて非常に大きいため，上記の方法はそのままでは実用にならない。実際，多くの超並列計算機で，mバイトのデータの転送時間$T_{tr}(m)$は，近似的に線形式

$$T_{tr}(m) = \alpha + \beta m \qquad (1)$$

で表され，α = 数百CPUステップ，β = 数CPUステップ／バイト程度である。したがって，上記の単純なウェーブフロント法のように個々のイタレーションごとにデータ転送を行うと，性能が極端に低下する。

この問題の解決のため，タイリングという方法が提案されている[1][3][5]。タイリングされたウェーブフロント法では，イタレーション空間内の矩形で表される複数のイタレーションをまとめて，タイルを構成する。例え

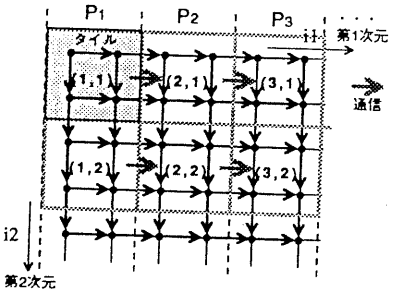

図 2: タイリングされたウェーブフロント法

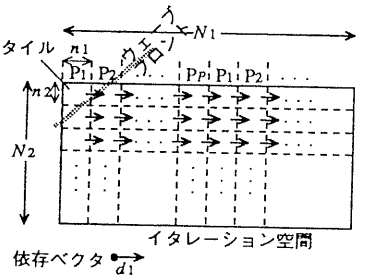

図 3: $d_2 = 0$ の場合のタイリング

ば図2は，サイズが 2×2 のタイルによるタイリングを示している。このタイルを単位として，ウェーブフロント型の並列実行を行う。

処理の進行は以下の通りである。図2に示すように，タイルを単位としてイタレーションをプロセッサに割り付ける。最初のステージで P_1 がタイル (1,1) の処理を実行し，計算結果の a[1][2], a[2][2] を P_2 に転送する。次のステージで P_1 がタイル (1,2) を，P_2 がタイル (2,1) を並列に実行し，計算された配列要素の値を隣接プロセッサに転送する。以下のステージも同様である。

この方法では，タイリングしない場合に比べて通信回数が少なく，通信起動オーバーヘッドを低減できる。図の例では通信回数は 1/4 になっている。タイルサイズが大きいほど，通信回数は減る。しかし一方で，タイルサイズをあまり大きくすると，2 番目以降のプロセッサが処理を開始するまでの時間が長くなってしまい，並列性が十分に生かせないという問題が起きる。なぜなら，プロセッサ P_1 でタイル (1,1) の処理が終わるまで，プロセッサ P_2 はタイル (2,1) の処理を開始できず，以下のプロセッサについても同様だからである。したがって，タイリングでは，このトレードオフを考慮して最適なタイルサイズを決定する必要がある。

以下，本報告では，問題を一般的に定式化し，タイリング方法，特に最適タイルサイズ決定方法を示す。

3 2 次元空間のタイリング

本節では 2 重ループ，すなわちイタレーション空間が 2 次元の場合のタイリング方法を示す。なお，本報告では以下，ループは完全ネストとする。

イタレーション空間の各次元のサイズ (イタレーション数) を N_1, N_2 とする。第 1 次元で分割するものとし，プロセッサ数を P とする。**依存距離ベクタ** (以下，単に**依存ベクタ**と呼ぶ) を $\boldsymbol{d} = (d_1, d_2)$ とする。第 1 次元に関してループ運搬依存がある，すなわち，依存ベクタの第 1 次元成分 d_1 は 0 でないものとする。なお，依存ベクタの一般的性質として，第 1 次元成分は 0 でなければ正である。

依存ベクタが複数ある場合は，それらの中で負方向の傾斜が最も急なもの，すなわち，$-d_2/d_1$ が最大であるものを $\boldsymbol{d}$ とする。明らかに，このベクタで表される依存関係を守れば，他の依存関係も守られる。

d_2 の符号によって，3 通りの場合に分かれる。

3.1 $d_2 = 0$ の場合

初めに，基本的な場合として，$d_2 = 0$ の場合を考える。2 章の例はこの場合に該当する。図3にタイリングされたイタレーション空間全体の様子を示す。各次元のタイルサイズを n_1, n_2 とする。タイルは図に示すように，P_1, P_2, ... の各プロセッサに割り当てる。$n_1 P < N_1$ のときは，割り当ては巡回的になる。最適タイルサイズを決定するために，以下，実行時間をタイルサイズの関数として表してみる。

1 タイル当たりの実行時間 T_{tile} は，

$$T_{tile} = n_1 n_2 t + a + b n_2 \tag{2}$$

となる。第 1 項は，1 タイル当たりのプロセッサ内演算時間である。ここで，t は 1 イタレーション当たりのプロセッサ内演算時間を表す。また，第 2,3 項はタイル当たりの通信時間である。第 2 項の a は，式 (1) の α に等しい。また，第 3 項は転送バイト数が n_2 に比例することに起因し，$b = \beta l d_1$ と表される。ここで l は配列要素当りのバイト数である。

最終プロセッサ起動までに実行されるタイル数 M_s は，

$$M_s = P \tag{3}$$

となる (厳密には $P-1$ であるが，P で近似する)。最終プロセッサが実行するタイル数 M_p は，

$$M_p = \frac{N_1 N_2}{P n_1 n_2} \tag{4}$$

である。結局，全実行時間 T は，

$$\begin{aligned} T &= T_{tile}(M_s + M_p) \\ &= (n_1 n_2 t + a + b n_2)\left(P + \frac{N_1 N_2}{P n_1 n_2}\right) \end{aligned} \tag{5}$$

となる。

次に n_1, n_2 を変化させたときの T の最小値を求める。式(5)で $n_1 n_2 = s$ と置いて，T を s と n_1 の関数として表すと，

$$T = \left(st + a + \frac{bs}{n_1}\right)\left(P + \frac{N_1 N_2}{Ps}\right) \qquad (6)$$

となる。s(すなわち，タイルの面積)を固定すれば T は n_1 の単調減少関数である。本モデルで n_1 の取りうる最大値は N_1/P であるから(これ以上になると最終プロセッサは何もしないことになってしまう)，結局，T は

$$n_1 = \frac{N_1}{P} \qquad (7)$$

(すなわち，非巡回ブロック分割)のときに最小になると結論できる。これを，式(5)に代入すると，

$$T = \left(\left(\frac{N_1 t}{P} + b\right) n_2 + a\right)\left(P + \frac{N_2}{n_2}\right) \qquad (8)$$

となる。$dT/dn_2 = 0$ を解くことにより，T を最小にする n_2 の値 n_{2opt} とそのときの T の値 T_{opt} は，

$$n_{2opt} = \sqrt{\frac{N_2 a}{\left(\frac{N_1 t}{P} + b\right) P}} \qquad (9)$$

$$T_{opt} = \left(\sqrt{N_2\left(\frac{N_1 t}{P} + b\right)} + \sqrt{aP}\right)^2 \qquad (10)$$

であることが分かる。通常の場合，

$$\frac{N_1 t}{P} \gg b \qquad (11)$$

と近似できるので，式(9), (10)は

$$n_{2opt} = \sqrt{\frac{N_2 a}{N_1 t}} \qquad (12)$$

$$T_{opt} = \left(\sqrt{\frac{N_1 N_2 t}{P}} + \sqrt{aP}\right)^2 \qquad (13)$$

のように簡単になる。

なお，式(7)よれば，第1次元方向のイタレーションの分割は**非巡回ブロック分割が最適**であることが分かる。

3.2 $d_2 < 0$ の場合

次に，依存ベクタの第2成分が負の場合を考える。図4に示すように，イタレーション空間をこのまま矩形にタイリングすると，依存ベクタがタイル境界を下から上へ横切るので，正しく実行できない。すなわち，タイルを上から順に実行すると依存関係を破ることになってしまう。なお，図には示していないが，下向きの依存ベクタもあって，タイルを下から順に実行することもできないものとする。

このような場合に正当なタイリングを行うための手法として，スキュー変換が知られている[2][4]。スキュー変

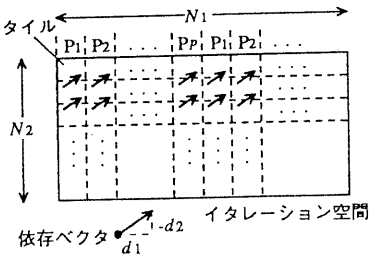

図 4: $d_2 < 0$ の場合の誤ったタイリング

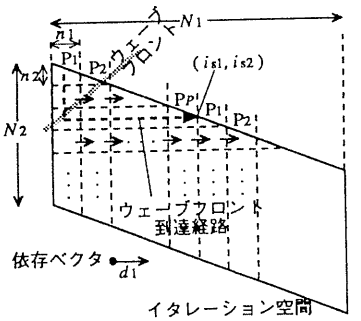

図 5: スキュー変換後のタイリング

換とは，イタレーション空間の座標系の変換である。$i_1' = i_1, i_2' = i_2 + f i_1$ という座標系を導入する。f はスキュー係数と呼ばれる定数である。この変換によって，依存ベクタ (d_1, d_2) は，$(d_1', d_2') = (d_1, d_2 + f d_1)$ に変換される。

変換後の座標系において，矩形のタイリングを可能とするためには，依存ベクタの第2成分が0になるようにスキュー係数 f を決定すれば良い。すなわち，

$$f = -d_2/d_1 \qquad (14)$$

とすれば良い。この変換は，行列形式で，

$$\begin{bmatrix} i_1' \\ i_2' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ f & 1 \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} \qquad (15)$$

と表される。変換後のイタレーション空間と依存ベクタを図5に示す。

図5で，全実行時間をタイルサイズの関数として表してみる。1タイル当たりの実行時間 T_{tile}，および，最終プロセッサが実行するタイル数 M_p は，前と同様に，それぞれ式(2)，(4)で表される。

最終プロセッサ起動までに実行されるタイル数 M_s を求めるには，最終プロセッサが担当する点 ${}^t(n_1 P, 0)$ の

スキュー変換後の座標

$$\begin{bmatrix} i_{s1} \\ i_{s2} \end{bmatrix} = \begin{bmatrix} 1 \\ -d_2/d_1 \end{bmatrix} n_1 P \qquad (16)$$

にウェーブフロント面が到達するまでのタイル数を計算すれば良い。その数は図の太破線で示した経路のタイル数であるから

$$M_s = \frac{i_{s1}}{n_1} + \frac{i_{s2}}{n_2} = \left(1 - \frac{d_2 n_1}{d_1 n_2}\right) P \qquad (17)$$

である。

以上より，全実行時間Tは，次のようになる。

$$\begin{aligned} T &= T_{tile}(M_s + M_p) \\ &= (n_1 n_2 t + a + b n_2) \cdot \\ &\quad \left(\left(1 - \frac{d_2 n_1}{d_1 n_2}\right) P + \frac{N_1 N_2}{P n_1 n_2}\right) \end{aligned} \qquad (18)$$

最適タイルサイズを求めるために，式(18)を解析する。まず，n_1固定の条件下で，n_2で微分することによってTの最小値$T_{opt}(n_1)$を求めると，

$$n_2 = \sqrt{\frac{a(-d_2 n_1/d_1 + N_1 N_2/P^2 n_1)}{n_1 t + b}} \qquad (19)$$

のときに，

$$T_{opt}(n_1) = \left(\sqrt{(n_1 t + b)\left(-\frac{P d_2 n_1}{d_1} + \frac{N_1 N_2}{P n_1}\right)} + \sqrt{aP}\right)^2 \qquad (20)$$

となる。さらにn_1を変化させたときの最小値を求めるためには，$dT_{opt}(n_1)/dn_1 = 0$を数値的に解く必要があり，ここではこれ以上解析的に調べることはしない。

ここで興味深いのは，$d_2 = 0$の場合と違って，$d_2 < 0$の場合は必ずしも非巡回ブロック分割が最適とは限らないことである。例えば，d_1=1, d_2=1, N_1=N_2=8K, P=512, b=tのときに，最適なn_1を数値的に求めると，n_1=5となる。この結果は，非巡回ブロック分割$n_1 = N_1/P = 16$とは異なる。

現実のプログラムでは，他のループとの関係で非巡回ブロック分割で実行したい場合も多いと思われる。その場合のために，$n_1 = N_1/P$の条件下で，$d_2 = 0$の場合と同様にしてn_{2opt}とT_{opt}を求めると，

$$n_{2opt} = \sqrt{\frac{(N_2 d_1 - N_1 d_2)a}{N_1 d_1 t}} \qquad (21)$$

$$T_{opt} = \left(\sqrt{\frac{(N_2 d_1 - N_1 d_2)N_1 t}{P d_1}} + \sqrt{aP}\right)^2 \qquad (22)$$

となる。

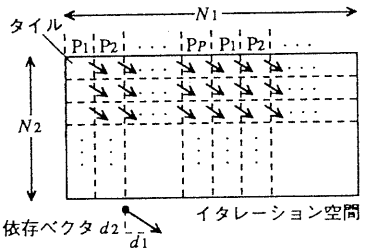

図 6: $d_2 > 0$の場合のスキュー変換前のタイリング

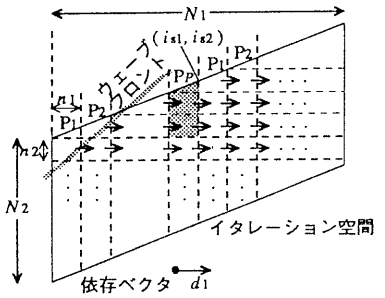

図 7: $d_2 > 0$の場合のスキュー変換後のタイリング

3.3 $d_2 > 0$の場合

次に，依存ベクタの第2成分が正の場合を考える。この状況を図6に示す。この場合は，$d_2 < 0$の場合と違って，イタレーション空間をこのまま矩形にタイリングしても，実行結果が誤りとなる訳ではない。しかし，イタレーション空間をスキュー変換してからタイリングすれば，最終プロセッサがより早く起動するようになり，全実行時間を短縮できる。

前と同じく，依存ベクタの第2成分が0になるようにスキュー変換する。スキュー係数fの形は式(14)と同じだが，今の場合は$d_2 > 0$なのでfは負の値となる。変換後のイタレーション空間と依存ベクタを図7に示す。

図7で，全実行時間をタイルサイズの関数として表す。T_{tile}, M_pは，前と同様に，それぞれ式(2), (4)で表される。M_sは以下のように求める。

最初のプロセッサから発したウェーブフロントが最終プロセッサに伝播する前に，最終プロセッサは網かけで示した部分のタイルを実行して良い。その個数は，$(0 - i_{s2})/n_2 = d_2 n_1 P/d_1 n_2$である。これが$P$より大きいかどうか，すなわち，

$$d_2 n_1 \geq d_1 n_2 \qquad (23)$$

かどうかによって場合分けする。

(A) $d_2 n_1 \geq d_1 n_2$ **の場合**

最終プロセッサは最初のプロセッサと同時に処理を開始できるので，$M_s = 0$ となる。全実行時間は，

$$\begin{aligned} T &= T_{tile} M_p \\ &= (n_1 n_2 t + a + b n_2) \frac{N_1 N_2}{P n_1 n_2} \\ &= \left(1 + \frac{a}{n_1 n_2} + \frac{b}{n_1}\right) \frac{N_1 N_2}{P} \end{aligned} \tag{24}$$

となる。これは，n_1, n_2 の単調減少関数であるから，場合分けの条件を充たす範囲内で n_1, n_2 が最大のとき，すなわち，

$$n_1 = \frac{N_1}{P},\ n_2 = \frac{N_1 d_2}{P d_1} \tag{25}$$

のときに T が最小となる。T の最小値 T_{opt} は，

$$T_{opt} = \frac{N_1 N_2 t}{P} + \frac{a P N_2 d_1}{N_1 d_2} + N_2 b \tag{26}$$

である。

(B) $d_2 n_1 < d_1 n_2$ **の場合**

この場合は，$M_s = (1 - d_2 n_1 / d_1 n_2) P$ となる。全実行時間 T の式は，$d_2 < 0$ の場合の式 (18) と同じ形になる。n_2 で微分することによって前と同様に式 (20) が得られるが，今回は $d_2 > 0$ なので，式 (20) は n_1 の単調減少関数となる。したがって，非巡回ブロック分割 $n_1 = N_1/P$ が最適である。

最適な n_2 やそのときの T も前と同じ形の式 (21), (22) で与えられる。ただし，そのようにして求められた n_1, n_2 が場合分けの条件を充たしていることを確認する必要がある。もし充たしていなければ，場合 (A) の方を適用しなければならない。

なお，これらの結果から分かるように，$d_2 > 0$ の場合は，常に**非巡回ブロック分割が最適**である。

4 L 次元空間のタイリング

本節では，前節までの結果を，3 次元以上のイタレーション空間の場合に拡張する。イタレーション空間の次元数 (すなわちループのネスト数) を L とする。

L 次元の場合を完全に一般的に取り扱おうとすると複雑になり過ぎるので，本報告では，スキュー変換不要の場合のみを扱うことにする。スキュー不要とは，結局，負の成分を持つ依存ベクタがないことと同値であり [4]，この制限の下でも現実の問題の多くに適用可能である。

外側の λ 個の次元で分割するものとし，各次元のプロセッサ数を $P_1, \ldots, P_\lambda$ とする。全プロセッサ数はこれらの積である。分割は次元軸に垂直な面によって行う，すなわち，λ 次元の超矩形で分割する。スキュー不要という条件があるので，このような分割が可能である。例として図 8に 3 次元の空間を 2 次元分割した場合を示す。

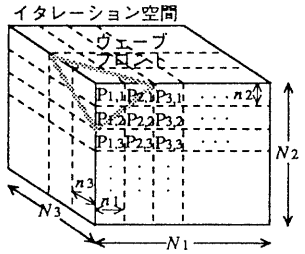

図 8: L 次元空間のタイリング

各次元のタイルサイズを，n_i $(i = 1, \ldots, L)$ とする。$i \leq \lambda$ ならば n_i は分割のブロックサイズを表す。また，タイルの体積 s を次のように定義する。

$$s = \prod_{i=1}^{L} n_i \tag{27}$$

1 タイル当りの実行時間 T_{tile} は，次のようになる。

$$T_{tile} = st + \sum_{i=1}^{\lambda} \left(a + \frac{b_i s}{n_i}\right) \tag{28}$$

第 1 項は，1 タイル当りのプロセッサ内演算時間，第 2 項は 1 タイル当りの，各分割次元方向に関する通信時間の和である。b_i は第 i 次元方向の転送時間のタイルサイズ比例係数である。

最終プロセッサ起動までに実行されるタイル数 M_s，および，最終プロセッサが実行するタイル数 M_p は，

$$M_s = \sum_{i=1}^{\lambda} P_i,\ M_p = \frac{N_1 N_2 \cdots N_L}{s P_1 P_2 \cdots P_\lambda} \tag{29}$$

である。

全実行時間 T は，次のようになる。

$$\begin{aligned} T &= T_{tile}(M_s + M_p) \\ &= \left(st + \sum_{i=1}^{\lambda} \left(a + \frac{b_i s}{n_i}\right)\right) \cdot \\ &\quad \left(\sum_{i=1}^{\lambda} P_i + \frac{N_1 N_2 \cdots N_L}{s P_1 P_2 \cdots P_\lambda}\right) \end{aligned} \tag{30}$$

s を固定すれば T は $n_1, \ldots, n_\lambda$ の単調減少関数であることが分かる。したがって，分割次元についての最適タイルサイズは，

$$n_{i,opt} = \frac{N_i}{P_i} \quad (i = 1, \ldots, \lambda) \tag{31}$$

であると結論できる。すなわち，この場合も**非巡回ブロック分割が最適**ということが分かる。

これを，式 (30) に代入して整理すると，

$$T = (Av + B)\left(C + \frac{D}{v}\right) \tag{32}$$

となる。ここで，各記号の定義は次の通りである。

$$v = \prod_{i=\lambda+1}^{L} n_i \tag{33}$$

$$A = \left(\prod_{i=1}^{\lambda} \frac{N_i}{P_i}\right) \cdot t + \sum_{i=1}^{\lambda} b_i \prod_{\substack{j=1 \\ i \neq j}}^{\lambda} \tag{34}$$

$$B = \lambda a,\ C = \sum_{i=1}^{\lambda} P_i,\ D = \prod_{i=\lambda+1}^{L} N_i \tag{35}$$

式(32)は，3.1節の式(8)と同じ形なので，変数v(非分割次元のタイルサイズの積)について微分することにより，Tを最小にするvの値v_{opt}とTの最小値T_{opt}が，

$$v_{opt} = \sqrt{\frac{BD}{AC}} \tag{36}$$

$$T_{opt} = \left(\sqrt{AD} + \sqrt{BC}\right)^2 \tag{37}$$

と求まる。積がv_{opt}になってさえいれば，各非分割次元のタイルサイズ$n_{\lambda+1}, \ldots, n_L$は任意である。

5 実機評価

理論の有効性を確認するために，超並列計算機nCUBE2による実機評価を行った。評価項目は次の3つとした.

- タイルサイズと実行時間の関係について，理論値と実測値を比較し，理論の有効性を確認する。
- プロセッサ数の増大に伴う性能向上を確認する。
- 現実的な応用プログラムでの効果を示す。

5.1 タイルサイズと実行時間の関係

2重ループを対象とし，$d_2 = 0, d_2 < 0, d_2 > 0$の各々の場合につき，第2次元方向のタイルサイズ$n_2$の変化に伴う全実行時間$T$の変化を測定し，理論値と比較した。なお，ループの並列化は人手で行った。

評価に用いたループのボディは次の通りである。

(a) $d_2 = 0$の場合

```
a[i1][i2] = a[i1-1][i2]+a[i1][i2-1];
```

(b) $d_2 < 0$の場合

```
a[i1][i2] = a[i1-1][i2+1]+a[i1][i2-1];
```

(c) $d_2 > 0$の場合

```
a[i1][i2] = a[i1-1][i2-1]+a[i1][i2-1];
```

その他の条件は，以下の通りである。

- $N_1 = N_2 = 1000, P = 8$とした。
- $n_1 = N_1/P$すなわち非巡回ブロック分割とした。

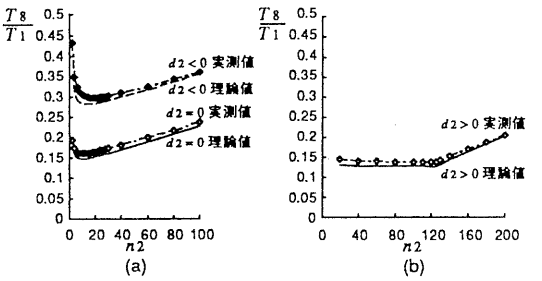

図 9: タイルサイズと実行時間の関係

- 理論値の計算において，1イタレーション当りの演算時間tは，プロセッサ1台での全実行時間をイタレーション数で割った値を用いた。また，通信時間の式に現われるパラメタa, bの値は，本実験とは別に行ったメッセージ通信性能の実測に基づいて，近似的に$a = 100t, b = t$とした。

結果を図9に示す。図9(a)は，$d_2 = 0$および$d_2 < 0$の場合であり，図9(b)は，$d_2 > 0$の場合である。横軸は第2次元のタイルサイズn_2，縦軸は1台時の実行時間によって正規化された実行時間を表す。d_2が0，正，負のいずれの場合についても，実測値と理論値は良く一致している。実測値の方がわずかに大きいのは，タイリング処理のオーバーヘッドによるものと考えられる。

5.2 プロセッサ数と性能の関係

前の実験と同じループを用いて，プロセッサ台数と性能の関係を測定した。実験の条件は以下の通りである。

- プロセッサ数Pは，4,8,16,32とした。
- n_1は，非巡回ブロック分割とした。ただし，$d_2 < 0$の場合は，式(20)を数値的に解いて求めた最適なn_1による巡回ブロック分割の場合も測定した。
- n_2は，理論的に求めた最適タイルサイズを用いた。

なお，具体的なn_1, n_2の値は，表1に示す。

結果を図10に示す。横軸はプロセッサ台数，縦軸は速度向上比(speedup)を示す。ここで，

速度向上比 = P台時性能／1台時性能

である。破線は，速度向上比$=P$という直線を示す。

この結果から，次のことが言える。

- $d_2 = 0, d_2 > 0$の場合，および$d_2 < 0$で非巡回ブロック分割の場合には，プロセッサ数の増加に伴い，ほぼ直線的に性能が向上している。
- $d_2 < 0$のときには 理論の予測通り，非巡回ブロック分割よりも巡回ブロック分割の方が性能が良い。

表 1: 実験に用いたタイルサイズ

	P	n_1	n_2
$d_2 = 0$	4	250	10
	8	125	10
	16	63	10
	32	32	10
$d_2 < 0$ 非巡回ブロック	4	250	15
	8	125	15
	16	63	15
	32	32	15
$d_2 < 0$ 巡回ブロック	4	31	80
	8	20	62
	16	12	51
	32	8	39
$d_2 > 0$	4	250	250
	8	125	125
	16	63	63
	32	32	32

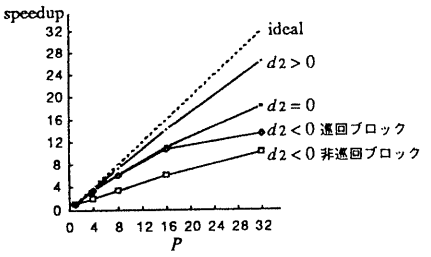

図 10: プロセッサ台数と性能の関係

5.3 応用プログラムでの評価

応用プログラムとして，SOR 法による偏微分方程式の求解プログラムを作成し，性能を測定した。SOR 法の中核ループは，$d_2 = 0$ のウェーブフロント型ループである。$N_1 = N_2 = 128$ とした。

これを次の 3 通りの方法で並列化し，性能を測定した。

(1) **最適タイリング**: 本方式が示す最適タイルサイズによってタイリングしたもの。具体的には，$n_1 = 128/P$, $n_2 = 6$ である。なお，我々は方式の評価実験用に並列化コンパイラプロトタイプを試作中であり，並列化は本プロトタイプによって自動的に行われた。コンパイラはタイルサイズの決定に当り，1 イタレーション当りの演算時間 t を，ループ内の演算数から見積っている。

(2) **第 1 次元のみタイリング**: $n_1 = 128/P$, $n_2 = 1$ としたもの。この並列化は人手で行った。

(3) **タイリング無し**: $n_1 = n_2 = 1$ としたもの。この並列化も人手で行った。

プログラム全体の実行時間の測定結果を図 (11) に示す。横軸はプロセッサ台数，縦軸は速度向上比を示す。なお，タイリング無し，4 台の場合は，メッセージの大量発生のため，測定できなかった。

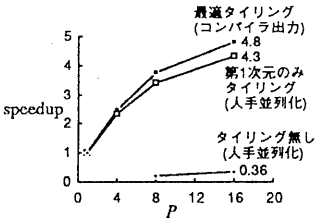

図 11: SOR プログラムの性能

この結果から，次のことが言える。

- タイリング無しの場合は，性能が非常に悪く，1 台時の性能にも遠く及ばない。
- 第 1 次元についてタイリング (ブロック分割) を行うことにより，台数に伴う性能向上が得られるようになる。さらに第 2 次元についてもタイリングを行うと，一層性能が向上する。いずれの場合もタイリング無しと比較して 10 倍以上の性能が得られている。

6 おわりに

ウェーブフロント型ループを超並列計算機向けにコンパイルするための，一般的なタイリング技法を提案し，実機評価により効果を確認した。SOR プログラムにおいて，タイリングしない場合と比べ 10 倍以上の性能が得られた。

今後の課題は，3 次元以上でスキュー変換を必要とする場合の検討などである。

参考文献

[1] M. Wolfe, "More Iteration Space Tiling", *Supercomputing '89*, pp.655–664.

[2] M. Wolfe, *Optimizing Supercompilers for Supercomputing*, Pitman, 1989.

[3] R.Irigoin and, R.Triolet, "Supernode Partioning", *ACM Symp. on Principles of Programming Languages*, pp.319–329, 1988.

[4] M. E. Wolf and M. S. Lam, "A Loop Transformation Theory and an Algorithm to Maximize Parallelism", *IEEE Trans. on Parallel and Distributed Systems*, Vol.2, pp.452-471, 1991.

[5] J. Ramanujam and P. Sadayappan, "Tiling Multidimensional Iteration Spacesfor Multicomputers", *J. of Parallel and Distributed Computing*, Vol.16, pp.108–120, 1992.

[6] S. Hiranandani, K. Kennedy, and C.-W. Tseng, "Evaluating Compiler Optimizations for Fortran D ", *J. of Parallel and Distributed Computing*, Vol.21, pp.27–45, 1994.