ハイパフォーマンス
コンピューティング 65－9
（1997. 3. 6）

非構造格子を対象としたボリュームレンダリング法の並列化

西松 研　瀬戸 麻由子　河野 洋一　福盛 秀雄　村岡 洋一

早稲田大学

非構造格子を対象としたボリュームレンダリング法の高速化を達成するため、最終映像に無関係なデータの計算を省くことで、レンダリング処理の効率化をはかる方法を検証し、さらにそのレンダリング処理の並列化を行なった。富士通社分散メモリ型並列計算機AP1000上に実装し、ボロノイ分割データ6000格子点(39958四面体)に対し、25倍前後の高速化が得られた。

Parallel Volume Rendering for Unstructured Grid

Ken NISHIMATSU　Mayuko SETO　Yoichi KONO
Hideo FUKUMORI　Yoichi MURAOKA

School of Science and Engineering , WASWDA University
3-4-1 Ohkubo Shinjuku-ku , Tokyo 169 , JAPAN

We aim at speed-up of volume rendering for unstructured grid. To get speed-up, this paper presents two method. One method is to omit waste rendering calculation that is no relation with the final image. The other method is to parallelize that rendering algorithm. Paralel method is implemented in Fujitsu AP1000. Using 64 PEs, we achieved about 25 times speed-up for 6000 grid points(39958 tetrahedra).

1　はじめに

数値シミュレーション結果の可視化には等値面表示、切断面表示、ボリュームレンダリング法がある。ボリュームレンダリング法は等値面表示のように、データの一部しか表現できない方法と異なり、透明度を変化させることで、物体の内部状態も表現できる手法である。そのため、データ全体を表現することが可能な手法であるが、計算時間がかかる点が問題とされている。

従来、ボリュームレンダリング法は医療分野で用いられるCT scanやMRIによって生成された構造格子を対象に用いられてきた。しかし、有限要素法等の数値シミュレーション分野では、非構造格子を扱う。構造格子では128^3～256^3のデータを対象に視点を変える等、ユーザとの対話的な処理が可能である（ここでの対話性とは、毎秒数フレームから数十フレーム程度のレートを指す）[3]。そこで、非構造格子を対象に、同じ程度の格子点数で、同様の処理を行なうのに十分なレンダリング速度を達成することを目的とする。

しかし、非構造格子を対象としたボリュームレンダリング法は、その形状から構造格子に比べ、レンダリング処理も複雑となり、可視化時間も多くかかる。そこで、本研究では、非構造格子を対象としたボリュームレンダリング法の高速化を達成するために、レンダリング処理において、最終映像に無関係なデータの計算を省くことによる効率化手法と、そのレンダリング処理の並列化手法を行なった。無駄な計算とは、不透明ピクセルへの混合計算と透明なボクセルに対する処理を指す。不透明ピクセルに対しては、スクリーンスキャンライン単位に不透明ピクセルの連続数を把握し、その先頭にアクセスすると、まとめて飛ばす。一方、透明ボクセルに対しては、データ空間を切断したスライスによって生成された三角形に対して、透明な三角形を外すことで、透明でない領域のみを対象に計算できる。

2　最終映像に無関係な計算を省く方法の検証

最終映像に無関係な計算として、透明なボクセルに対する処理と不透明ピクセルへの混合処理が

ある。この二つの処理に対して効率化をはかった方法に、Lacroute 氏の提案した手法がある [1] [2]。彼らの手法概念は、立方体格子の形状から、データ空間をスライスの集まりとみなす。視点側のスライスから、スキャンライン単位に処理を行なうが、スキャンライン単位に事前に透明なデータが連続している場合、そのまとまりを調べておく。ピクセル側もスキャンライン単位で、不透明ピクセルの連続数を保持しておく。図1において、skip の部分は無駄な計算となるため、その領域にアクセスしたら、計算位置を飛ばし、work の部分のみピクセルへの混合を行なうことで、計算の効率化を図る。なお、サンプリング計算はスライス上で行なうため、2次元の内挿問題となる。

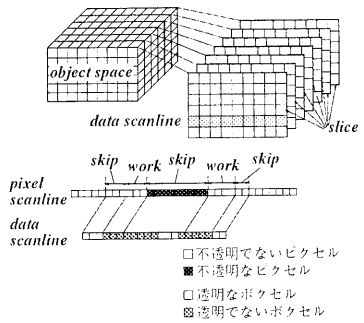

図 1: Lacroute 氏の手法による計算の効率化

レンダリングでは、視点方向に対し、最も角度の少ない基本軸 (x,y,z 軸のいずれか) を選び、その軸に対して垂直なスライスを用いる。光線はスライスに対して垂直にあたるという条件で見ると、スライスと光線との位置関係は、図2のようになる (図2 は z 軸に垂直なスライスが使われる場合)。

全スライスに対する作業終了後、得られた画像に2次元変換を行ない、最終画像を得ている。

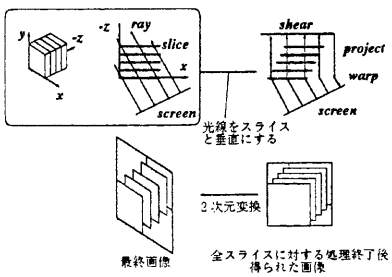

図 2: スライスと光線の関係図

四面体格子への適用

立方体格子のような構造格子では、3 次元データの場合、3 次元配列に物理量 $P(i,j,k)$ を入れることで、配列の添字がそのまま、その点の位置を示している。また、隣接している格子は添字から簡単にわかるため、隣接情報もすでに合わせ持っている。一方、非構造格子では、通常、格子点に番号をつけ、その点の頂点座標を格納する配列と、格子がどの点で構成されているかを格納する、格子構成情報の配列からなるため、格子と格子の隣接関係は分からない。

そこで、Lacroute 氏の提案した手法の概念を四面体格子にも適用するため、スライス作成及び、スライス上をスキャンライン単位でアクセスできるようなテーブルを作成した。

1. 非構造格子データに対し、基本となる x,y,z 軸方向に垂直なスライスをそれぞれ作り、その面と四面体が交差する面を求める。交差によってできた面をそのスライス上の情報として持たせる。交差によってできた面が四角形のときは2つの三角形に分割することで、全て三角形のデータとする。
2. スライス上の三角形に対し、その三角形の辺の隣接情報 (辺の共有情報) を作成する。
3. 三角形の辺で、隣接情報をもたない辺は、外枠の辺であると定義する。外枠の辺が、最初にスキャンラインと交差する辺の候補であるため、この辺を外枠の辺のリストに登録する。

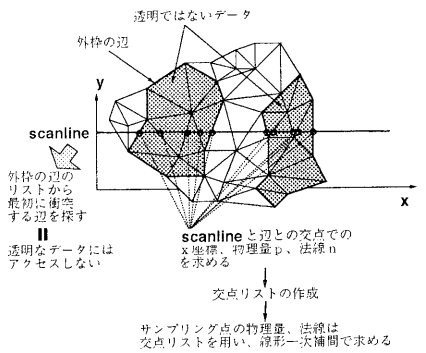

図 3: レンダリング処理

2 の段階で、三角形の頂点の物理量と透明かそうでないかの閾値を比較し、三点が全て、閾値以下の場合、その三角形は透明であると判断し、隣接関係を調べるものから外す。そうすることで、レンダリング処理のときには、そのデータにアクセ

スしない。また、閾値を変更した場合、スライス情報を保持していれば、2の操作からやり直すことで対処できる。また、不透明ピクセルへのアクセス削減はLacroute氏の手法をそのまま用いる。

3 レンダリング処理の並列化

レンダリング処理を並列化する方法として、前章のアルゴリズムの計算を単純に各PEで分散させる方法が望ましい。並列化の方法として、オブジェクト空間を切り分け、仕事として与える方法と画像空間を切り分け、仕事として与える方法がある。視点から近いデータ順(Visibility Order)に処理を行なうため、画像空間を切り分け、必要に応じて、他PEからデータを取りよせる方法をとる。

実装方法

実装の全体の流れを図4に示す。

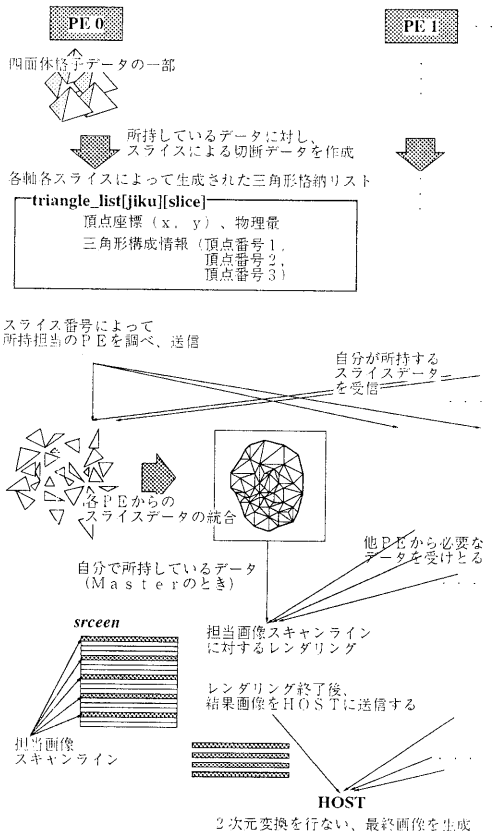

図 4: 並列化処理の全体の流れ

各PEは最初、全体の四面体格子データの一部を所持しているが、スライス単位でデータを保持するよう、データの再配置を行なう。

また、各PEは担当スキャンラインに対し、レンダリング処理する。レンダリング処理の進行はスライス単位で行なう。あるスライスに対して、レンダリング処理を行なうとき、そのスライスを持っているPEをマスターとし、残りのPEをスレーブとする。マスターPEは各データスキャンラインを、必要としているスレーブに送信する。送信されたデータスキャンラインは、スレーブでピクセルへの混合処理に使われた後は、現在の画像生成には再度利用されないため、キャッシュを作る必要はない。

各PEが全スライスに対する作業を終了すると、レンダリング結果のデータをHOST(フロントエンドマシン)に送信する。HOSTはデータを受信し、2次変換を行ない、最終画像を生成する。

画像空間に対して並列化を行なう際、最大の問題点は、データが送信されるまでの待ち時間である。そこで、通信時間を隠蔽するために以下の処理を行なう。各PEは自分が次にマスターPEになるスライス番号が分かっているため、空き時間にスキャンライン情報(スキャンラインと辺との交点座標、交点座標における物理量と法線ベクトル)をあらかじめ計算し、格納する。現在のスライスの次にマスターになるPEは、送信できる準備のできたデータを先送りする。

4 評価

ボロノイ分割によって生成された四面体格子データ[4]を用い、実験を行なった。格子点の座標を$t(x, y, z)$とすると、tは、

$$(0 \leq x \leq 1) \wedge (0 \leq y \leq 1) \wedge (-1 \leq z \leq 0)$$

を満たす空間上に存在する。格子点tは物理量$P(t)$を持っていないため、以下の式で与える。

$$p(t) = 0.86 - [(0.5, 0.5, 0.5) \text{ から } t \text{ までの距離}]$$

ボロノイ分割データに対し、物理量から色と不透明度を算出する関数(transfer function)に$func1$、$func2$、$func3$を用意した(付録参照)。データ全体のうち、物理量が0.20以上、0.60以上、0.80以上の占める割合はそれぞれ99.8%、39.3%、0.892%である。計測時間は、視点方向、並行光源共に$(0, 0, -1)$の方向で与え、並行投影による画像生成を行なう。スキャンライン同士の間隔と同間隔でスキャンライン上をサンプリングする。レンダリング時間には前処理と2次元変換の時間を含めていない。また、テーブル作成時間は、一方向(z軸に垂直なスライス)のみ作成した時間である。

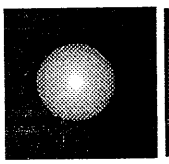
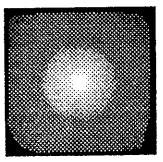
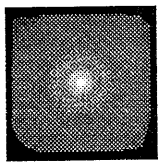

図 5: func1　　図 6: func2　　図 7: func3
(全体で透明なデータの割合、不透明なデータの割合)
func1(60%、40%)func2(0.002%,40%)func3(0%,0.009%)

4.1 計算の効率化に関する実験と評価

Sun E3000 使用で、ボロノイデータ格子点 12000 点 (四面体数 80388) を使用し、64 スライスを作成、1 スライスあたり 256 スキャンラインを用いて、透明な領域を隣接関係のテーブルに登録した場合としない場合 (A)、不透明なピクセルをまとめて飛ばす処理を行なう場合と行なわい場合 (B) のそれぞれに対し、ピクセルにアクセスした回数 (C) とピクセルへの混合を行なう回数 (D)、レンダリング時間 (E 単位は [sec]) を測定した。サンプリング点が透明なときは混合は行なわない。func3 を用いたときは、全データが表示対象となり、不透明度を持つため、透明なボクセルの削除の作業はない。

表 1: func1 を用いたとき

A	×	×	○	○
B	×	○	×	○
C	3551401	2941997	362209	38957
D	328316	14490	323316	14490
E	32.627	28.191	3.113	2.580

表 2: func2 を用いたとき

A	×	×	○	○
B	×	○	×	○
C	3551401	2913010	3546300	2907909
D	3525767	2881193	3525767	2881193
E	98.507	85.975	97.646	83.468

表 3: func3 を用いたとき

B	×	○
C	3551401	3482617
D	3551401	3480834
E	101.393	102.936

A:透明な領域を削減する (○) しない (×) B:不透明なピクセルをまとめて飛ばす (○) 飛ばさない (×) C:ピクセルアクセス回数 D:ピクセルへの混合回数 E:レンダリング時間 ([sec])

計算の効率化に関する考察

表 1、表 2から、透明なボクセルが存在するような transfer function が与えられた場合、そのボクセルを省く処理を行なうことで、レンダリング時間を短縮できることがわかった。また、不透明ピクセルにはアクセスしないよう、不透明ピクセルをまとめて飛ばす処理は有効である。しかし、表 3のように、混合により、なかなか不透明値にならないような transfer function が与えられた場合、ピクセルへのアクセス回数を減らす効率化と、スキャンラインで自分の位置から何個連続で不透明ピクセルが存在するかを調べる処理が同等かもしくは、重くなってしまうことが推測される。

4.2 並列化に関する実験と評価

本手法を富士通社の分散メモリ型並列計算機 AP1000 に実装した。64PE で、z 軸に垂直な 64 スライスを作成したときの前処理のテーブル作成時間及び、レンダリング処理時間を測定した (図 8)。2 次元変換は、HOST が、PE からレンダリングされた小領域を集めた後、行なっている。スキャンライン数による処理時間を表 4に示す。

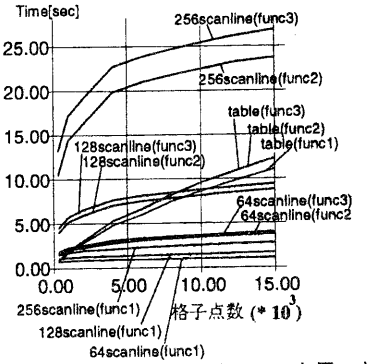

図 8: 64PE 使用で、ボロノイデータを用いたときの処理時間

また、色と不透明度の算出に $func3$ を用いたときの、テーブル作成時間の内訳を示す (図 9)。

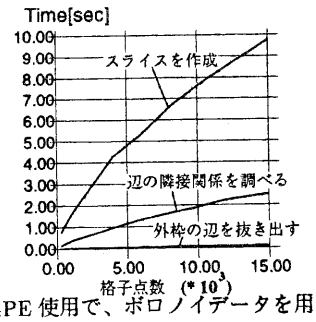

図 9: 64PE 使用で、ボロノイデータを用いたときの前処理時間の内訳

表 4: スキャンライン数と 2 次元変換の処理時間

スキャンライン数	64	128	256
2 次元変換 [sec]	0.4090	1.360	5.020

また、PE 台数によるレンダリング時間、合計時間 (テーブル作成時間とレンダリング時間の合計) とスピードアップを格子点 2000 点 (四面体数 13058)(図 10、図 11)、6000 点 (四面体数 39958)(図 12、図 13)、15000 点 (四面体数 100524)(図 14、図

15、図16、図17) に対して調べた。スライスによるメモリ消費のため、1PE で測定できたのは、6000格子点データまでである。

hipiph データの可視化

格子点数262144(64*64*64) をもつ立方体格子のhipiph データ [5] を四面体に分割し、1250235 四面体のデータを生成した。そのデータに対し、64PE を用い、64 スライス、1 スライス内では128 スキャンラインで画像を生成した。その際、テーブル作成は33.493 秒、レンダリング処理は5.137 秒、2次元変換は1.568 秒 (合計40.198 秒) だった。表示対象となっているデータは全データの5~10 %である。透明なデータへのアクセスを省略させない場合、レンダリング処理は12.486 秒だった。

並列化における考察

ボロノイデータ6000 格子点に対し、64PE 使用、64 スキャンラインで、レンダリング処理のみでは25.1 倍の高速化 (合計時間は17.2 倍) が得られた。台数分の効果が得られなかった理由として、負荷が完全に均等でないことによる Idle な PE の発生や、並列化により、単体のときにはないデータ伝送作業や作業終了報告等の通信の発生が挙げられる。通信時間を隠蔽することで、レンダリング処理は高速化が得られたが、前処理の重さも無視できない。図8から、格子点数が大きくなるにつれ、テーブル作成の時間の上昇度が大きいが、図9によれば、スライス作成時間と隣接関係を作成する時間の勾配が大きいことがわかる。スライス作成時間については、送信された三角形の頂点情報とすでに持っている頂点情報で重複するものがあるかの判定計算が重い。また、精度を求めるため、多くのスライスを使えば、所持するスライスの枚数分だけ、隣接関係を調べる処理も重くなる。

対話的に十分な速度が得られたとも言えない。しかし、レンダリング処理に便利なデータの再配置を行なうことで、レンダリング処理だけから見れば、並列化効率は出せることは結果から示せた (図11、図13、図15、図17)。視点を変えた画像を見たい場合には、一度スライスを作成すれば、任意の方向からレンダリング時間のみで表示できる。

ボロノイデータ15000 格子点のメモリ消費量は2391KB である。しかし、スライスに変換し保持することで、1スライスだけで最大590KB の大きさになるものもあった。メモリを多く消費するため、現在の可視化できるデータサイズは数値シミュレーション結果を可視化させるためのサイズに比べると小さいという点は課題である。

今回並列化を行なったデータは空間全体に可視化対象データが存在する場合であった。*func1* のような transfer function が与えられた場合、一部の PE に必要とする情報や負荷が集まってしまう。また、透明なデータの配置によっても負荷が偏ることが予想される。現在は、静的な分散しか行なっていないが、透明なボクセルの分布領域や与えられたスクリーンスキャンラインの仕事の重さによって、動的に負荷分散を行なうことで、より高速化を得るのも可能と考える。

5 まとめ

本論文では、非構造格子を対象としたボリュームレンダリング法の高速化のために、無駄な計算を省いたレンダリング手法とその処理の並列化手法を提案した。大きなデータへの適用とさらなる高速化へ発展させていくためには、メモリを考えたデータ構造とテーブル作成部の高速化、動的負荷分散が今後の課題である。

謝辞

本研究で、AP1000 を使わせて頂いたことを (株) 富士通および並列処理センターに感謝致します。

参考文献

[1] Philippe Lacroute, Marc Levoy : Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation , *A CM SIGGRAPH 94*, pp.451-458, July, 1994

[2] Philippe G.Lacroute : FAST VOLUME RENDERING USING A SHEAR-WARP FACTORIZATION OFTHE VIEWING TRANSFORMATION , *Technical Report: CSL-TR-95-678* , September, 1995

[3] Philippe Lacroute : Real-Time Volume Rendering on Shared Memory Multiprocessors Using the Shear-Warp Factorization, *Proceedings 1995 Parallel Rendering Symposium*, ACM Press, Atlanta GA, October, 1995, pp.15-22

[4] 河野洋一, 西松 研, 福盛 秀雄, 村岡 洋一 : 有限要素法要素分割の並列化 -ボロノイ分割の並列化- , ハイパフォーマンスコンピューティング, 60-8, pp.43-48, 1996

[5] ftp://omicron.cs.unc.edu/pub/projects/softlab.v/CHVRTD/

付録：不透明度算出関数

物理量を P、不透明度を α とする。値が与えられていない部分は前後の値から線形補間した値を使う。

[func1]

$P < 0.60 \quad \alpha = 0.0$
$P \geq 0.60 \quad \alpha = 1.0$

[func2]

$P < 0.2 \quad \alpha = 0.0$
$P = 0.3 \quad \alpha = 0.002$
$P = 0.4 \quad \alpha = 0.004$
$P = 0.5 \quad \alpha = 0.008$
$P = 0.59 \quad \alpha = 0.10$
$P \geq 0.60 \quad \alpha = 1.0$

[func3]

$P = 0.0 \quad \alpha = 0.0$
$P = 0.3 \quad \alpha = 0.002$
$P = 0.4 \quad \alpha = 0.004$
$P = 0.5 \quad \alpha = 0.008$
$P = 0.59 \quad \alpha = 0.10$
$P = 0.60 \quad \alpha = 0.15$
$P = 0.61 \quad \alpha = 0.10$
$P = 0.70 \quad \alpha = 0.002$
$P = 0.75 \quad \alpha = 0.10$
$P \geq 0.80 \quad \alpha = 1.0$

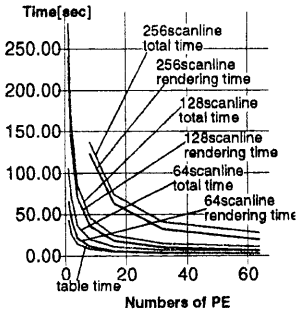

図 10: 2000 点、func3 を用いての PE 台数と処理時間

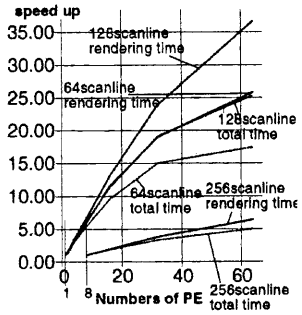

図 11: 2000 点、func3 を用いてのスピードアップ

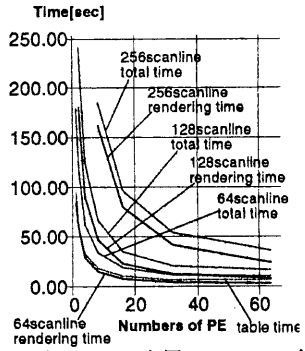

図 12: 6000 点、func3 を用いての PE 台数と処理時間

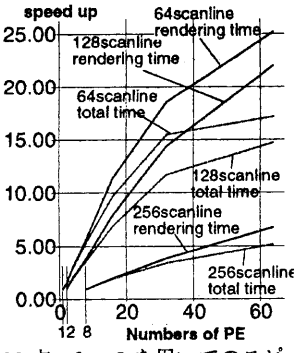

図 13: 6000 点、func3 を用いてのスピードアップ

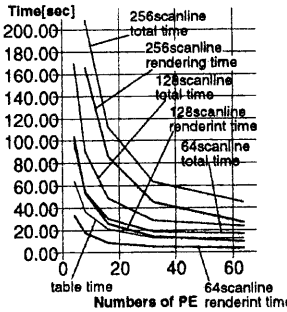

図 14: 15000 点、func3 を用いての PE 台数と処理時間

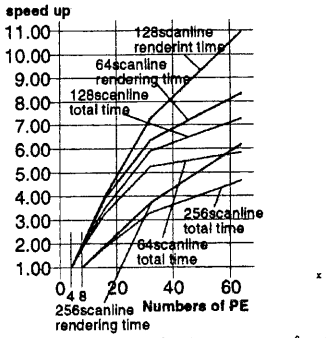

図 15: 15000 点、func3 を用いてのスピードアップ

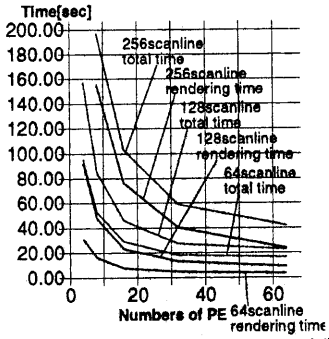

図 16: 15000 点、func2 を用いての PE 台数と処理時間

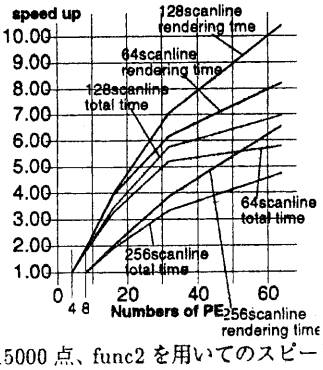

図 17: 15000 点、func2 を用いてのスピードアップ