2005－ARC－161　(8)
2005／1／21

複数の S-DSM を対象とする開発支援ツール S-CAT の設計と実装

多　　忠行††　吉瀬　謙二†　片桐　孝洋†　弓場　敏嗣†

†電気通信大学大学院情報システム学研究科〒182-8285東京都調布市調布ヶ丘1-5-1

S-DSM はメモリ一貫性を保証するため様々な制御通信を行わなければならず，機構が複雑になる傾向がある．そのため開発中デバッグや動作検証に時間がかかってしまう．また，S-DSM 上で実行するアプリケーションの性能測定の際，詳細な実行情報を取得しにくい．そこで我々は，S-DSM を対象とする開発支援ツール S-CAT を作成した．S-CAT の特徴は以下のとおりである．(1)複数の S-DSM を対象とする．(2)多くの計算機環境で動作する．(3)通信状況や通信メッセージ等の局所的な時系列情報を提供する．(4)通信密度や統計情報等の大域的な情報を提供する．

The design and implementation of development support tool S-CAT for multiple S-DSMs

Tadayuki Ohno‡†　Kenji Kise†　Takahiro Katagiri†　Tositugu Yuba†

†Graduate School of Information Systems, The University of Electro-Communications.
1-5-1, Chofugaoka, Chofu-shi, Tokyo, 182-8285
‡{kis,katagiri,yuba}@is.uec.ac.jp,　‡ohno@yuba.is.uec.ac.jp

Since S-DSM ensures memory consistency, various communications must be performed, and there is a tendency for a mechanism to become complicated. Therefore, there is a problem that complicated debugging and a processing verification will take time in development. Moreover, in performance measurement of application, there is a problem of being hard to acquire execution information in detail. Then, we created development support tool S-CAT for S-DSM. The feature of S-CAT is as follows. (1) It is targeted at multiple kinds of S-DSM. (2) Operate on many platforms. (3) Provide local information, such as a communication situation and a communication message. (4) Provide global information, such as communication density and statistical information.

1.　はじめに

近年 PC クラスタのような大規模クラスタシステムが普及している．クラスタシステム上でアプリケーションを実行する際，高性能で，かつ効率よく利用できる並列プログラミング環境が重要となる．

ソフトウェアで仮想的な共有メモリを実現する Software Distributed Shared Memory (S-DSM)は，分散メモリ環境において共有メモリモデルの並列プログラミングが可能であるという利点を備えている．現在までにいくつかの S-DSM[1][2]が開発されているが，機構が複雑なため開発に時間がかかるという問題がある．

S-DSM では仮想共有メモリを構築する時，分散したメモリ間の一貫性をとるために通信を行う．通信は非同期に行われ，実行状況の把握は困難である．S-DSM 開発時のデバッグ作業においては，printf を用いた断片的な実行状況の取得が一般に行われる．開発を容易にするためには実行状況をわかりやすい形で提示する開発支援ツールが求められる［3］．

新しい S-DSM を開発する，あるいは従来の S-DSM に新しい機構を実装するとき，そのたび毎に新たにツールを作成することは避けるべきである．すなわち，対象とする S-DSM を限定しないツールが求められる．しかし，現在のところ複数の S-DSM を対象とするツールはない．そこで，我々は複数の S-DSM を対象とした開発支援ツール S-CAT を作成した．

本稿の構成を示す．2 章では S-DSM を対象とするツールの設計方針を議論し，3 章で S-CAT のツールとしての所持機能を示す．4 章で実際に使用した事例を示す．5 章で関連研究について述べる．６章で本稿をまとめ，今後の課題について述べる．

2.　設計方針

S-DSM 開発を支援するツール S-CAT を作成するにあたり，以下の設計方針をおく．

i)　様々な S-DSM を対象とする
　現在多様なメモリ一貫性モデルに基づく様々な S-DSM があり提案されてい

る，またさらに新しい S-DSM が開発される可能性がある．これらの S-DSM のために専用の開発支援ツールを作成することは無駄が多い．そのため，ツールの支援対象を特定の S-DSM に限定するべきではない．S-DSM への基本的な支援機能を有するツールを拡張して，専用化を図ることにより開発効率がよくなる．

ii) S-DSM 処理を把握可能とする

S-DSM はメモリ一貫性を保証するために通信を行っている．しかし S-DSM ユーザは，実行時にはメモリ一貫性保証の機構が見えないため，最適化やデバッグが困難になる．そのため処理の時間推移を把握するための機能を備える．

iii) ツールが S-DSM へ与える負担を軽減する

アプリケーション実行時に伴うツールのための処理が S-DSM に与える影響を最小限にするべきである．そのため実行時に行うツールのための処理は「情報の保存」のみにし，不要な処理の遅延を最小限にする．採取した情報ログは実行後にツールにより表示される．

iv) ツールのソフトウェア構成が単純でありかつ拡張性がある

開発支援ツールユーザが求める機能は多様であるため，基本機能だけを備えたツールを拡張することで要求を満たす．S-DSM 開発支援ツールとして拡張性のあるプロトタイプを作成し，これに機能を追加していくことにより柔軟に開発支援ができる．

3. 設計

ここでは 2 章の設計方針を元に、本ツールの設計の詳細を示す．

3.1. S-CAT の使用

本ツールの使用は図 1 の様な手順をとる．

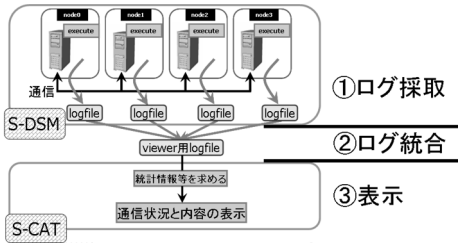

図 1:使用手順

① ログ採取

S-DSM 上でログを採取しながらアプリケーションを動かす．この際，S-DSM にログを保存する機能を付け加えてあり，S-CAT 入力用のフォーマットにしたがっていることが必要である．また、分散環境では個々のノードの時計が異なっているため,時計合わせを行っている必要がある．

② ログ統合

ログファイルはすべてのノードのローカル領域にファイルとして格納される．それぞれのログファイルには送信時刻と通信時間とメッセージの内容が書かれている．それらのファイルを集め，1 つのログファイルに統合する． 統合には unix の sort コマンドを使用し,時刻でソートを行う．

③ 処理状況の表示

生成されたログファイルを元に S-CAT により視覚化を行う．これによりツールユーザは開発中の S-DSM の性能ボトルネックの所在を知ることができる．

3.2. 対象

S-DSM としてホームベース S-DSM である JIAJIA[1]を用いる．JIAJIA はメモリ一貫性保証のために様々なオペレーションを含む通信を行うが,すべてのオペレーションについて同一の構造体で扱うため，支援ツールの単純な対象として適していると考えた．しかし，設定ファイルの記述により，複数の S-DSM を考慮した設計となっている．

3.3. 機能

3.3.1. 通信視覚化

メモリ一貫性処理の中でユーザから見えない通信処理に焦点をあて，これを時系列順に表示することにより処理の流れを視覚的に把握することができる．

図 2 は本ツールの画面写真の一部である．図の横方向を時間軸，縦方向をノード番号とし，矢印がノード間通信を表す．全ての通信をそれぞれ矢印で表すことで，時系列順の処理の流れを把握することができる．

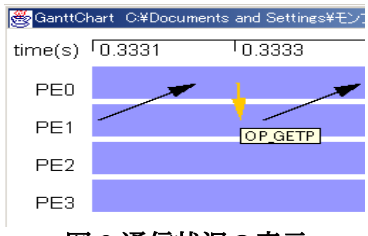

図 2:通信状況の表示

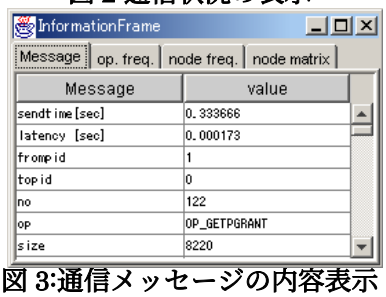

図 3:通信メッセージの内容表示

3.3.2. 通信メッセージの内容表示

一つの通信に含まれるメッセージ内容を表示する．図 2 の矢印をマウスでクリックすること

で一つの通信を選択することができる．選択された矢印の通信内容は，図 3 のように表示される．図 3 では以下の要素が詳細情報となっている．

① 送信時刻(sendtime)[秒]
② 通信時間(latency)[秒]
③ 送信元 ID(frompid)
④ 送信先 ID(topid)
⑤ 通信全体における順次番号(no)
⑥ オペレーションの種類(op)
⑦ 送信したバイト数(size)

図 3 は S-DSM として JIAJIA を使用している際の表示例であるが, 3.3.5 で示すように設定ファイルの記述により表示方法を変えることができる．

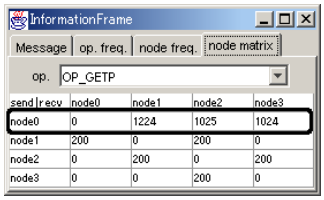

図 4: ノード間通信回数の表示

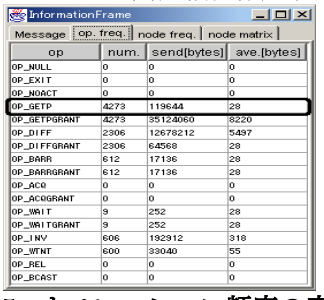

図 5: オペレーション頻度の表示

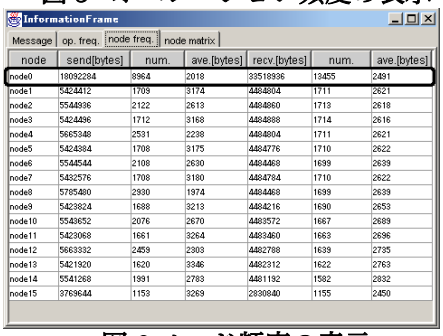

図 6:ノード頻度の表示

3.3.3. 統計情報の表示

ノード毎の通信回数やバイト数等の統計データの表示を行う．図 4,5,6 は本ツールの統計情報を表示する部分である．

図 4 はオペレーション毎に，あるノードからあるノードへの送信回数の表を示している．縦方向が送信ノード, 横方向が受信ノードであり，例えばページ要求通知である OP_GETP というオペレーションの通信についてノード 0 が他のノードに送信している回数は,ノード 1 に 1224 回,ノード 2 に 1025 回,ノード 3 に 1024 回行っていることを示す．

図 5 はオペレーション毎の送信回数，バイト数，平均バイト数を表示している．例えば，OP_GETP というオペレーションを含むメッセージ通信が 4,273 回行われ，合計で 119,644bytes のデータが送られ, 平均で 28bytes であることを示す．

図 6 はノード毎の送信回数，受信回数，総バイト数，平均バイト数を表示している．例えば，ノード 0 は 18,092,284bytes，8,964 回送信し，その平均送信バイト数が 2,018bytes であり，33,518,936bytes，13,455 回受信し，その平均受信バイト数が 2,491bytes であることを示す．

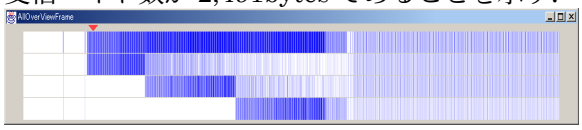

図 7:通信密度の表示

3.3.4. 通信密度の表示

通信密度を視覚的にわかりやすく表示する．図 7 は本ツールの通信密度表示部分である．縦方向がノード番号，横方向が時間軸である．図の左端と右端がそれぞれアプリケーションの実行開始から終了までに対応しており，図の色の濃い部分は通信が密な箇所，薄い部分は通信が疎な箇所である．これにより全体の通信状況を把握することができる．

3.3.5. 通信メッセージの定義ファイル

使用している S-DSM の通信メッセージを定義することができる．図 8 は定義ファイルの 1 部分である. XML を使用して S-DSM の通信メッセージの構造体を定義することができる．これにより S-CAT は様々な S-DSM に適応することができる．また，S-CAT で表示のためのシノニムを定義することができる．

図 8 の<message>タグで囲まれた部分が S-DSM 通信メッセージ構造体定義部である．上から送信ノード ID，受信ノード ID，シーケンシャルナンバ，オペレーション，サイズというデータを含む通信メッセージ構造体を使用することを定義している．さらに，<synonym>タグで囲まれた部分がシノニム定義部である．シノニム定義では, メッセージの op の値が 110 の時は”OP_GETP”と表示し，111 の時は“OP_GETPGRANT”と表示することを定義している．

```
<structure>
 <message>
  <variable type="short"         name="frompid"/>
  <variable type="short"         name="topid"/>
  <variable type="unsigned int" name="no"/>
  <variable type="short"         name="op"/>
  <variable type="short"         name="size"/>
 </message>
 <display>
  <synonym_list>
   <target name="op"/>
   <synonym value="110" string="OP_GETP"/>
   <synonym value="111" string="OP_GETPGRANT"/>
  </synonym_list>
 </display>
</structure>
```

図 8:通信メッセージ定義ファイル

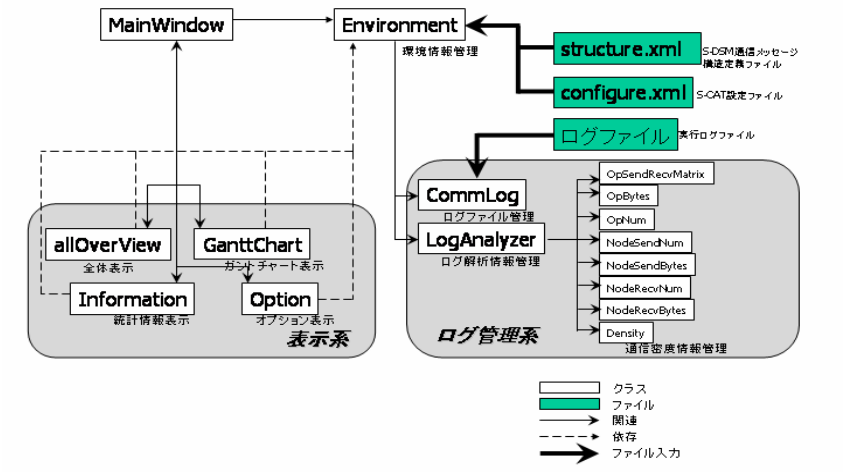

図 9:ソフトウェアアーキテクチャ

3.4.　実装

図 9 に本ツールのソフトウェアアーキテクチャを示す．本ツールは Java を使用して記述されており，様々な計算機環境で動作する．本ツールは複数のクラスで構成されている．

MainWindow クラスは始めに実行されるクラスで，メインウィンドウとその他のウィンドウを管理する．Environment クラスは本ツール全体の管理をする．Information クラスは通信メッセージ情報や，統計情報を表示する．GanttChart クラスは時系列順に並べた通信を表示し，局所的な情報を提示する．AllOverView クラスは通信密度を表示し，大域的な情報を表示する．Option クラスは表示における表示倍率の変更，表示するメッセージの種類の選択を行う．CommLog クラスは通信ログファイルを読み込み，管理する．LogAnalyzer クラスはログファイルの解析情報を管理する．

4.　S-CAT の使用事例

一般に，MPI 等を使用した並列アプリケーションに比べて,S-DSM を使用した並列アプリケーションは性能が低い．これには様々な原因があるが，性能のボトルネックとなっている原因を特定することはツールなしには困難である．そこで，S-DSM の性能低下につながるオーバヘッドの特定に S-CAT を使用した事例を示す．

アプリケーション SOR(Successive Over-relaxation)について 2 種類のデータ配置方式を使用して実行し比較した際，方式の違いにより実行時間が大幅に異なった.原因を解明するために実際に S-CAT を使用した．

4.1.　アプリケーション SOR

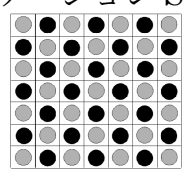

図 10:SOR のデータ配置

SOR は図 10 のように格子状に並べられた赤と黒の値を順次大域更新していくアプリケーションである.赤一つを更新するためには，その上下左右の黒の値を必要とし，黒一つを更新するためには，その上下左右の赤の値を必要とする．赤全部、黒全部の更新を 1 iteration で行い，これを指定数だけ繰り返し計算する.使用した SOR の擬似プログラムは図 11 の通りである．

今回パラメータを以下の通り設定し実行した．

・iteration 数=100
・M=2046
・N=1023

```
main(){
  init();                          } S-DSM 初期化
  赤,黒の共有メモリ確保              SOR 初期化
  ノード 0 が赤、黒を初期化
  for(100 回){                     }
    自分の範囲の赤を計算             } 計算コア
    barrier();
    自分の範囲の黒を計算
    barrier();
  }
}
```

図 11:SOR 擬似プログラム

4.2.　使用した S-DSM 環境

使用している S-DSM は JIAJIA[1]である．JIAJIA では，最新のデータを保持するノードを HomeNode と呼ぶ．HomeNode がもっているデータを他のノードが使用するときは，HomeNode からそのノードへデータを受け渡すことで仮想共有メモリを実現している．そのため，データに対する HomeNode の割り当て方により，通信回数が変化することがある．

JIAJIA ではメモリ管理をページ単位で行っている．我々が使用している JIAJIA は 1 ページが 8KBytes である．

HomeNode からページを受け取ったノードがページに変更を加えた際には，変更後のページと変更前のページの diff(差分)を barrier 処理

時に HomeNode に返す.これにより常に HomeNode が最新データを保持していることを保証する.

4.3. 集中型データ配置

この方式を全 9 ノードで実行する. ノード 0 は図 12 の様に赤黒の 2 次元配列全部のデータを保持するが計算は行わず, ノード 1～8 はデータを保持しないが計算を行う.ノード 0 が保持するデータ量は, 赤黒それぞれ(M+1)×((N+1)×sizeof(float)) bytes の大きさの配列データである.

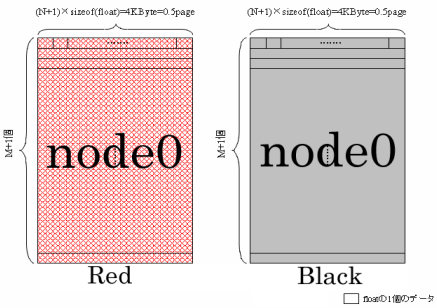

図 12:集中型のデータ配置

4.4. ブロック分割型データ配置

この方式を全 8 ノードで実行する.図 13 の様にデータをブロックで分割して保持し, 全ノードで計算を行う. ノード毎に赤黒それぞれ((M+1)/8)×((N+1)×sizeof(float)) bytes の大きさの配列データである.

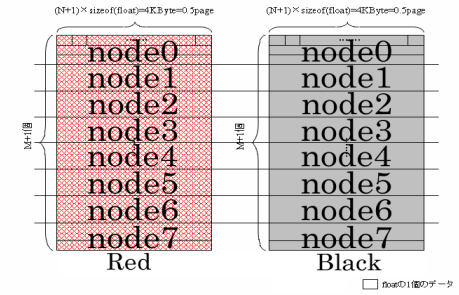

図 13:ブロック分割型のデータ配置

4.5. アプリケーション実行結果

それぞれの方式での実行結果は, 計算コア部分での実行時間測定で,

・集中型: 24.4sec

・ブロック分割型:0.7sec

となった.ここで, 方式の違いによる実行時間の差がどのような原因から発生したのかを解明するために S-CAT を使用した.

4.6. 原因究明

2 方式の実行それぞれについてログファイルを生成した.このログ採取機能は JIAJIA を改良して付け加えた.

ログファイルを S-CAT で表示した画面写真は図 14,15 である. さらに方式 1,2 それぞれの密度表示部分を図 16,17, 統計情報を図 18,19 として示す.

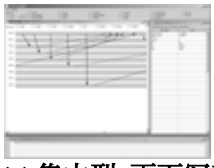

図 14:集中型 画面写真

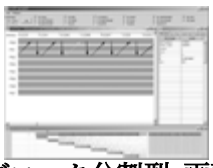

図 15:ブロック分割型 画面写真

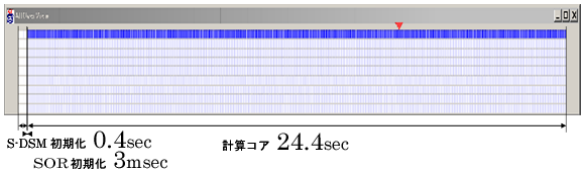

図 16:集中型 S-CAT 密度表示

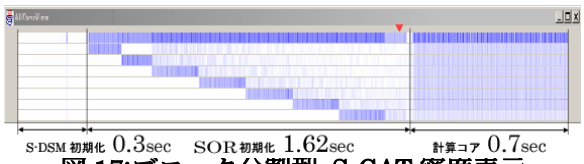

図 17:ブロック分割型 S-CAT 密度表示

	num	byte	average
OP_NULL	0	0	0
OP_EXIT	0	0	0
OP_NOACT	0	0	0
OP_GETP	6896	193088	28
OP_GETPGRANT	6896	56655120	8220
OP_DIFF	1024	1681916084	16424
OP_DIFFGRANT	102400	2867200	28
OP_BARR	1656	46368	28
OP_BARRGRANT	828	23184	28
OP_ACQ	0	0	0
OP_ACQGRANT	0	0	0
OP_WAIT	24	672	28
OP_WAITGRANT	12	336	28
OP_INV	800	13116800	16396
OP_WTNT	1600	1681600	1051
OP_REL	0	0	0
OP_BCAST	1640	13140320	8012

	send byte	num	average	recv. Byte	num	average
node0	72692640	110936	655	1683837812	112576	14957
node1	209743716	13922	15065	9496120	13922	682
node2	214010096	14532	14726	11140120	14122	788
node3	210725016	14122	14921	11140120	14122	788
node4	217295176	14942	14542	11140120	14122	788
node5	210725016	14122	14921	11140120	14122	788
node6	214010096	14532	14726	11140120	14122	788
node7	210725016	14122	14921	11140120	14122	788
node8	209744000	13922	15065	9496120	13922	682

図 18:集中型 統計情報

	num	byte	average
OP_NULL	0	0	0
OP_EXIT	0	0	0
OP_NOACT	0	0	0
OP_GETP	6385	178780	28
OP_GETPGRANT	6385	52494700	8220
OP_DIFF	2820	14794876	5246
OP_DIFFGRANT	2820	78960	28
OP_BARR	1442	40376	28
OP_BARRGRANT	1442	40376	28
OP_ACQ	0	0	0
OP_ACQGRANT	0	0	0
OP_WAIT	21	588	28
OP_WAITGRANT	21	588	28
OP_INV	1414	596176	421
OP_WTNT	1400	63824	45
OP_REL	0	0	0
OP_BCAST	0	0	0

	send byte	num	average	recv. Byte	num	average
node0	17181996	9682	1774	31302048	9668	3237
node1	7536364	2201	3424	5520268	2203	2505
node2	7536504	2206	3416	5520408	2208	2500
node3	7535832	2182	3453	5519736	2184	2527
node4	7543996	2181	3458	5519716	2183	2528
node5	7534040	2118	3557	5517844	2120	2602
node6	7531940	2043	3686	5515844	2045	2697
node7	5878572	1537	3824	3863280	1539	2510

図 19:ブロック分割型 統計情報

通信密度表示画面において双方で大きく異なる部分は, SOR 初期化にかかっている時間である.集中型では全てのデータを HomeNode(ノード 0)が保持しており, そのデータをノード 0 が

初期化するため全く通信が発生しない.そのため 3ms で初期化作業が完了している.これに対してブロック分割型ではデータをそれぞれのノードが均等に保持しているが，初期化作業を行うのがノード 0 であるため，図の様にノード 0 と他のノード間での通信が行われていることがわかる.そのため初期化作業の時間が大幅に大きくなっている.

次に計算コア部分の実行時間が異なる事がわかる.通信密度表示画面からみた大域的な通信パターンに大きな違いが見られないため，統計データを参照した.図 18,19 から OP_DIFF に関する通信が大幅に異なっていることが分かる.ここで, OP_DIFF とは非 HomeNode が変更したデータを HomeNode へ送信するオペレーションである.ブロック分割型では OP_DIFF の総送信バイト量が約 14MBytes であるのに対し，集中型では約 1.6GBytes であり、大きく異なっている.そこで通信状況表示画面で 1 iteration 間の詳細な通信状況を調べたところ，集中型ではブロック分割型とは異なり barrier 処理の際に大量の OP_DIFF 送信を行っていることが分かった.集中型では barrier 間のページ変更部分のサイズ(diff サイズ)が 1MBytes になる.我々が使用している JIAJIA では 1 回の OP_DIFF 処理に送信できる diff サイズは約 16KBytes であるため，1 回の barrier で OP_DIFF 処理を 64 回行っている事が分かる.実行全体では，集中型で上記の OP_DIFF 処理を 1 ノードにつき

OP_DIFF 処理数×barrier 数×iteration 数 =64×2×100=12800 回

行っている.また, 1 iteration 間の OP_DIFF 処理に約 200ms 要している事が分かる.

つまり集中型とブロック分割型の計算コア実行時間が大幅に異なるのは，集中型が OP_DIFF を送信する処理の分だけ時間がかかっているためである.1 iteration 区間内での OP_DIFF 送信処理に約 200ms かかっていることから，処理全体に対して OP_DIFF に関する処理が支配的なことが分かる.

4.7. S-CAT の有用性

以上が S-CAT を使用した実例である.S-CAT は S-DSM における処理の情報を局所的，大域的に視覚化して提示することにより，ユーザに S-DSM の処理状況を分かりやすく把握させ，問題の原因究明を支援する.上記の事例により S-CAT は S-DSM の性能解析に有用であることを示しえたと考える.

5. 関連研究

[5]では， クラスタ上での並列プログラムの通信解析を容易にしている．マシン間の通信量の時間変移を視覚的に提示するが，具体的な通信内容はわからないため，定量的な通信量を把握することしかできない. 本研究では,並列プログラムの通信解析を S-DSM の観点から詳しく提示するという点で異なる．本研究では通信内容を提示するため,具体的な処理状況を把握することができる.

6. まとめと今後の課題

我々は複数の S-DSM を対象とした開発支援ツール S-CAT を作成した．S-CAT はあらかじめ生成された実行情報を読み込み，視覚的にわかりやすくユーザに提示することで, S-DSM の処理を把握させる．今回使用した事例から，本ツールの有用性を確認できた.

今後の課題としては，支援機能の追加を行うことがあげられる．例えば，アプリケーションの計算に費やされている時間とミドルウェアの処理に費やされている時間を分け，ミドルウェア処理時間が実行全体に対してどれだけの割合かを示すことや，階層性のあるマルチクラスタ環境のためのクラスタ化された通信表示方法の導入等が考えられる．また，ツールの有効性検証として他の S-DSM に対しても実際に使用してみることがあげられる.

我々が JIAJIA をベースに作成した S-DSM である Mocha[3]と共に，フリーソフトウェアとして S-CAT を公開する予定である.

謝辞

本研究の一部は，文部科学省科学研究費補助金(課題番号 16300004「スーパークラスタを指向した性能拡張性を持つソフトウェア分散共有記憶方式の研究」)の援助による.

参考文献

[1] Hu, W., Shi, W. and Tang, Z. : JIAJIA:A Software DSM System Based on a New CasheCoherence Protocol, HPCN Europe, pp. 463-472(1999).

[2] Keleher, P., Dwarkadas, S. Cox, A. L. and Zwaenepoel, W. : TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems, Proceedings of the Winter 94 Usenix Conference, pp. 115-131(1994).

[3] 吉瀬謙二:S-DSM システムの受信通知オーバヘッドを削減する方式,電子情報通信学会デザインガイア 2004,pp.71-76(2004/12)

[4] 城田祐介，吉川克哉，本多弘樹，弓場敏嗣: マルチホーム方式を用いたマルチクラスタ向けソフトウェア分散共有メモリ， 2003 年先進的計算基盤システムシンポジウム SACSIS 2003，pp. 315-322(2003/5).

[5] 菊池康祐， 緑川博子， 飯塚肇: 計算機クラスタ用通信解析ツールの開発, 情報処理学会第 64 回全国大会論文集(1)1-69(2002).