Vol.2012-HPC-135 No.5
2012/8/1

分散メモリ環境上におけるタスク並列処理系 MassiveThreads/DMに対する共有メモリ環境上での模擬評価

池上 克明[1,a)]　田浦 健次朗[1,b)]

概要：近年の並列計算環境は大規模化かつ分散化しており，このような環境で広く用いられてきた処理系が MPI であるが，MPI でのプログラミングはプログラムの実行中並列度が固定される点やデータ共有をメッセージで行う点でユーザーにとってプログラムの記述が困難である．そのためプログラムの記述性と性能を両立するには新しいプログラミングモデルに基づいた並列言語や処理系が必要になる．並列性のモデルである細粒度タスク並列モデルと分散環境でのデータ共有モデルである PGAS はこのような性能と記述性を両立しうるものである．本発表では従来のこれらのモデルに基づく並列言語処理系の問題点を踏まえ，より適切なキャッシュ機構を備えた PGAS 分散タスク並列処理系を提案し，この処理系の性能を予備的に共有メモリ環境で推定した．今回のシミュレートでは十分な性能を得られる見積もりがたったので，今後はタスクスチールなどタスク並列処理系全体でのオーバーヘッドや通信の集約の効果などを含めて推定し確認する予定である．

Ikegami Katsuaki[1,a)]　Taura Kenjiro[1,b)]

Abstract: Latest parallel computers have large-scale parallelism and hierarchical architecture, thus it becomes much harder to write programs with traditional parallel programming runtime MPI. MPI have two problems; one is static parallelism and the other is message passing model over processes. To overcome the issue, there must be a more sophisticated parallel and distributed language with newly programming model such as partitioned global address space (PGAS) and fine-grained task parallelism. In this presentation, we propose new PGAS cache semantics that suits to a task parallel runtime. We also evaluated this semantics in not distributed but shared memory environment, and guessed the performance of PGAS preliminarily.

1. 序論

1.1 背景

並列計算は通常の逐次計算では実現できない高性能を得るために行われ，その利用範囲は物理現象のシミュレーションや流体解析のような専門性の高いものから近年ではコンシューマ用まで広い範囲に亘っている．このように並列計算が広く利用されるに至った理由の一つとして，計算機の単体での性能向上が限界を迎え，計算環境の並列度が向上していることが挙げられる．実際現代のスーパーコンピュータは数千個のプロセッサからなることも多く，一方で一般向けのプロセッサにおいてもマルチコア化が進んでいる．

並列計算を記述する必然性は現代の計算機を対象とする以上高まっていると考えられるが，特に高い性能を得られるスーパーコンピュータやクラスタ環境では通常の計算機で利用されているハードウェアによる共有メモリを利用することができず，プロセッサ間の通信は Ethernet や Infiniband などのネットワークを経由して行うことになる．このような分散環境で以前から広く用いられている処理系として MPI[9] が挙げられる．MPI はデータの共有をネットワークを介するメッセージパッシングモデルに基づいて行なっていて，送信側と受信側で共通して大域的にデータを指し示すことができず，データにアクセスするには常にデータの存在するプロセスとデータにアクセスするプロセスの間でメッセージのやり取りを記述する必要がある．また通常のマルチスレッドモデルのプログラミングモデルと

[1] 東京大学
The University of Tokyo
[a)] liquid@eidos.ic.i.u-tokyo.ac.jp
[b)] tau@logos.t.u-tokyo.ac.jp

対照的に，実行中のプロセス数は一定で，ユーザーが自由に並列度を制御することはできない．このような特徴から通常の共有メモリ上での並列プログラミングモデルと異なる点が大きく，記述性が高いとは言えない．

このように従来より大規模計算環境において用いられてきたプログラミングモデルは高い性能を得ることはできるが，プログラムの記述は容易ではない．これに対して並列性の記述の観点とデータアクセスの観点からより高い記述性を持つプログラミングモデルの一つとしてそれぞれ細粒度タスク並列モデルと partitioned global address space (PGAS) モデル [18] が挙げられる．

タスク並列はある手続きの一部分を並列に実行できるタスクとして記述するプログラミングモデルで，通常の POSIX threads などもこのモデルに属する．しかし，通常のタスク並列処理系は CPU のコア数を大幅に上回る並列度で計算を行うとタスクの切り替えの際のオーバーヘッドが大きくなるため，プログラマが任意のタイミングでタスクを生成するのは現実的ではない．このオーバーヘッドを削減して細粒度なタスクの生成を可能にしたものとして Cilk [2] を始めとする細粒度タスク並列処理系が知られている．これらの細粒度タスク並列処理系ではユーザーレベルでタスク（スレッド）の切り替えを行うことで通常のコンテキストスイッチより小さなオーバーヘッドを実現している．共有メモリ環境上で動作する細粒度タスク並列処理系は Cilk を始め Nanothreads [11], Qthreads [17], MassiveThreads [20] など多くが研究されている．一方で分散環境でも動作する細粒度タスク並列処理系は多くない．

また PGAS モデルは分散環境上で大域アドレス空間をユーザーに提供することでプログラムを共有メモリインターフェースで記述することができるプログラミングモデルである．以前から分散環境上で共有メモリインターフェースを提供する試みは distributed shared memory (DSM) として行われてきたが，PGAS は DSM と比べてデータが実際にどのノードに置かれるかをプログラマが明示的に指定することができるため性能が低下しづらくなっている．

しかし，現状の PGAS 処理系では一般に性能と記述性のトレードオフが生じている．PGAS 自体がプログラムの記述を容易にするための機構である以上，PGAS による記述は MPI などの従来手法に比べて高い生産性を得る必要がある．この点でおいて Chapel [3] や X10 [5] などといった言語は一定の成功を収めていると言える．しかし，これらの言語では記述の抽象度が高いことから処理系の実装が最適化されていない．そのため，リモートメモリにアクセスする際にも細粒度なアクセスを行なって性能を損なう結果になりやすい．一方で，UPC [4] のような低レベルな言語では透過的な大域アドレスアクセスを行うと性能が低下するのでこれを避けるためにはプログラマ自身がある程度大きな領域に対して PGAS とローカルメモリ間でコピーを行う必要がある．しかし，このような記述を行うと離散的な領域にアクセスする際にはアドレスの変換コストがかかり，なおかつプログラマも記述するのが難しい．

1.2 目的

高生産な並列言語処理系として PGAS を提供する分散細粒度タスク並列処理系を設計する．ここで，大域アドレス空間に対してローカルなアドレス空間をマップしてキャッシュとして用いるという新しいインターフェースを導入することで，通常の PGAS モデルにおいて問題となる記述性と性能の両立を狙う．今回は実際の分散環境で動作する実装ではなく共有メモリ上で実際の動作を模擬的に表現する物を制作し，分散環境で実装した場合どのような性能低下が起こりうるかを確かめる．

2. 関連手法

2.1 分散環境上での細粒度タスク並列処理系

2.1.1 Silkroad

L. Peng らが開発した Silkroad [13] は Cilk が共有メモリ上で動作するのに対して DSM を提供したものである．Cilk には分散環境でも動作する処理系は存在するが，データの共有がプロセスの祖先・子孫間でしか行えないなど強い制約がある．一方で Silkroad では DSM を提供しているので通常の共有メモリインターフェースでプログラムを記述することができる．しかし，DSM ではデータの位置をユーザーが制御することが難しく，PGAS よりもより性能に与える影響は大きいと考えられる．

2.1.2 Scioto

Scioto [8] はより低レベルな分散タスク並列モデルに基づくフレームワークである．Silkroad が Cilk 同様にプログラミング言語としてユーザーに機能を提供しているのに対して，Scioto は他の分散並列プログラミング言語上でタスク並列処理系を構築する．そのため，MPI のみならず UPC や CAF [14] など多くの PGAS 言語上でも動作することができる．その一方で，当然 PGAS の性能は下のレイヤとして用いる処理系に当然依存することになる．

2.2 連続しない領域へのアクセスを提供する GAS 処理系

通常の PGAS 言語においては，連続しない領域へアクセスするような API を持たない．つまり，ユーザーは離散的な領域へアクセスするためにはその連続したブロック数だけ API を呼び出し自分で適切に場所を管理する必要がある．その点で連続した領域に対するアクセスを利用できる API があると通信の最適化を処理系に任せることができる．Global Arrays [12] や DMI [19] はこのような API を提供している．一方で，実際にデータを読み書きした後のデータの配置はアドレスの変換が必要である．

2.3 PGAS の通信の集約による性能最適化

PGAS モデルでは記述性に関してはユーザーがプログラムを共有メモリのインターフェースで記述することを目的にしている．しかし通常の共有メモリのようにプログラムを記述すると，大域アドレスに対するアクセスは粒度が細かすぎて実際の通信を行うと性能を損ねることが多い．そこでユーザーには細粒度な PGAS への記述を許しつつ，実際には処理系が粗粒度なネットワークアクセスに集約することでオーバーヘッドを削減する言語処理系が存在する．具体的には UPC においてはコンパイラの静的解析結果に基づいた通信の集約 [7] や実行時に時間的に近接したアクセスを自動的に集約するような手法 [6] が研究されている．その他にも Titanium におけて実行時にプログラムを実際に走行させた結果を用いて動的に通信を集約する手法 [15], [16] や high performance fortran (HPF) [10] における HALO というアクセス範囲をユーザーに明示させる仕組みを用いた通信の集約 [1] も知られている．しかし，これらの通信の集約は SPMD 型の並列プログラミングモデル上で行われており，分散タスク並列モデル上では行われていない．

3. 提案手法

3.1 概要

本発表ではまず我々の研究グループで開発している新しい PGAS インターフェースに基づくタスク並列処理系と通信の最適化を説明する．次にこれを共有メモリ上でシミュレーションする手法を説明する．

まずタスク並列処理系の部分について簡単に触れる．我々の研究グループが開発している MassiveThreads/DM は共有メモリ上で動作するタスク並列処理系 MassiveThreads を分散環境で動作するようにしたものである．細粒度タスク並列処理系であり，Cilk などと同様に LIFO スケジューリングといって生成されたタスクが LIFO，つまりは通常の逐次プログラムにおけるコンテキスト・スタック同様に実行されていく．ワーカーはこのスタックとして使われているタスクキューの中のタスクがなくなるまで実行を続ける．一方で，実行するタスクがないワーカーは周囲のワーカーからタスクを盗んでくる．ワーカーがタスクスチールする際は，相手のタスクキューのうち，通常ワーカーがタスクを取る側と反対側からタスクを取ってくる．これにより，タスクスチールする際にはなるべく呼び出し元に近いタスクをスチールすることになり，これは分割統治法でプログラムが記述されている場合は「まだ十分多くのタスクを生成しうるタスク」である．これによりタスクスチールをあまり頻繁に行うことなく負荷分散を実現できる．

これを満たすタスク並列処理系を分散環境上で構築することは自明ではないが，本発表ではこのようなスレッドモデルを提供することだけを示しメモリモデルの説明に移りたい．以降ではこのタスク並列処理系に必要な GAS を考えることにする．

3.2 新しい PGAS インターフェース

3.2.1 PGAS の基本的な機能

GAS は仮想的な共有メモリインターフェースのアドレス空間であるが，実際にはそのデータは計算に参加する各々のプロセス上のメモリとして確保される．この時，このデータの最小単位のことをページと呼び，実際にページが位置するプロセスをオーナーと呼ぶ．

ここで，一般にタスク並列で更に分割統治法を利用してプログラムを記述する場合を考えると，タスクがどのワーカーに割り当てられるかはタスクスチールに依存しているので計算の始めからどのプロセスにどのデータを割り振るかを決めるのは難しいことである．そこでページのオーナーは動的に変化させることが可能になっている．

3.2.2 従来手法の問題点

PGAS モデルは仮想的なアドレス空間を提供するが，一般に DSM とは異なり通常のポインタと PGAS 上のポインタは区別される．UPC や Chapel など PGAS 上のデータに通常のメモリ領域同様透過的にアクセスを提供する言語もあるが，多くの PGAS 言語処理系は PGAS 上のデータとローカルなメモリ領域のデータをやり取りする API を持つ．ここで，従来の PGAS 言語処理系で利用されてきたこのインターフェースの基づくセマンティクスを put/get セマンティクスと呼ぶことにする．put/get はそれぞれ PGAS への書き込み・PGAS からの読み込みを行うインターフェースである．この put/get による PGAS に対するアクセスと，何らかの同期機構があれば分散環境でも共有メモリのスレッドプログラミングに近い記述性を得ることができる．

しかし，put/get インターフェースには問題がある．このインターフェースを細粒度に用いると PGAS 上のデータ領域に頻繁にアクセスすることになる他，大域アドレスと対応するローカルアドレスを変換するような処理もコストが大きく高い性能を得ることは難しくなる．一方で，これをなるべく粗粒度に使おうとするとユーザーは put/get で利用するローカルなメモリ領域を自分の責任で適切に使いまわす必要が出る．これはタスクマイグレーションによってローカルなメモリ領域にアクセスできなくなる可能性がある分散タスク並列処理系にとっては使いづらい．

また，連続した領域の場合であれば一度 put/get しさえすれば同じように連続したローカルなメモリ領域として扱うことができるが，これがストライドや非定型なアクセスに対して行われる場合はそれも不可能である．例えば図 1 のようにあるメモリ領域に対して一部の必要な要素だけにアクセスする場合を考える．ここで必要な要素だけにアクセスするが，1 要素にアクセスする度に 1 回通信していて

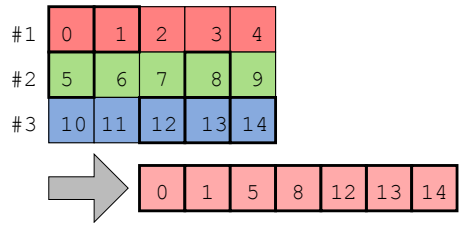

図 1 put/get セマンティクスにおける非定型データへのアクセス

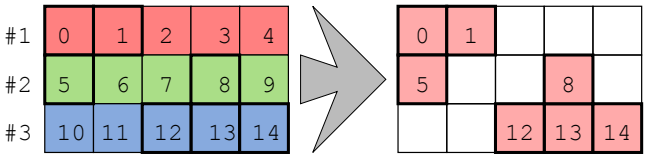

図 2 新規 PGAS インターフェースによる非定型データへのアクセス

は性能を引き出すことはできないので，データを集約して読み書きする必要がある．そのため，単純に必要な要素を全てまとめて読んでくると図のように受け取ったデータが密に詰まった状態になる．この状態で問題になるのは，大域アドレスからローカルアドレスを計算するのに無視できないコストがかかる事である．

3.2.3 新規 PGAS インターフェースの仕様

これに対して新規 PGAS インターフェースは異なったセマンティクスを持つ．まず，新規 PGAS インターフェースは次の 3 つの API からなる．

- `localize`
- `unlocalize`
- `commit`

`localize` は既存のモデルでいう get, `commit` は同様に put に近い作用をするが, put/get と異なる点がある．`localize` は大域アドレスとそのどの領域にアクセスするかを引数として受け取り，この大域アドレス空間上の領域に対応付けするのに必要なだけのローカルなメモリバッファを確保し，これを大域アドレスと紐付けして先頭アドレスを返す．実際の計算はこのローカルなメモリアドレスに対して行い，結果を他のプロセスに反映させる場合は `commit` を反映させたい部分を指定して呼び出す．最後にローカルな領域が不要になった時点で `unlocalize` の呼び出しを持ってこの対応付けは破棄される．

このインターフェースは**図 2** に示すように離散的な領域に対してアクセスする際にはそのデータ配置をそのままに保つことでプログラマが元の PGAS 上のデータの位置から対応するマップ先のローカルなメモリのアドレスを知ることが容易になっている．

このインターフェースの基づくセマンティクスのうち重要なものとして，複数回の `localize` で返されるローカルなメモリバッファは再利用されうるという事である．つまり，ある領域を `localize` した後にその領域の部分集合となるような領域を `localize` すると，前回の `localize` した際のバッファの対応する位置のアドレスが返ってきうるという事である．この挙動からはメモリバッファを一貫したコヒーレンスを持たないキャッシュと考えることができる．このようなセマンティクスを持つことで，処理系は一度 `localize` した位置に対して繰り返しデータの移動を行う必要がなく，結果として通信の集約を行うことが可能になる．ここで複数のタスクがローカリティの高い近接した領域について計算を行う場合はこのような通信の集約の恩恵を強く受けることになる．

```
1  void merge_sort(gas_ptr<int> gas_begin,
2                  gas_ptr<int> gas_end)
3  {
4    // simply localize [gas_begin..gas_end)
5    int * begin = localize(gas_begin, gas_end -
         gas_begin);
6    if(end - begin < threshold){ /* seq sort */ }
7
8    int * middle = begin + (end - begin) / 2;
9    spawn merge_sort(begin, middle);
10   spawn merge_sort(middle, end);
11   sync;
12   // rest of codes
13   commit(gas_begin, gas_end - gas_begin);
14   unlocalize(gas_begin, begin);
15   return;
16 }
```

図 3 マージソートに PGAS の API を追加した例

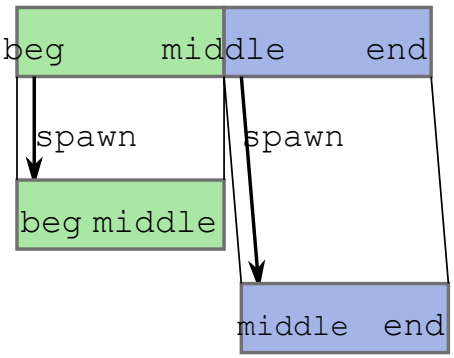

図 4 マージソートにおける localize を利用した通信の集約

図 3 にマージソートにこのインターフェースによる記述を追加したコード例を示す．

このコードでは冒頭で必ず `gas_begin` から `gas_end` で示される，与えられた範囲全体を `localize` によってローカルなアドレスに対応づけしている．子のタスクでは親のタスクで既にアクセスする範囲は全て読み込まれていて，実際には図 4 のように以前 localize した領域中の位置的に対応するアドレスを返す動作が期待できる．この期待は子がワークスチールされない限り成り立つ．つまりこのように `localize` を祖先の段階でも子孫の段階でも記述するこ

とによって通信の集約を行うことができる．

3.3 共有メモリ環境におけるシミュレーション

このように提案する処理系はこれまでの PGAS よりもより高い性能を発揮すると考えられるインターフェースを持つが，実際に高い性能になるかどうかは実際に処理系を実装してみなければわからない．しかし，実装の完成より先に模擬的にでも性能を推し量ることが出来れば有意義である．そこでより実装が容易な共有メモリ上でいくつかのベンチマークを走行させ，処理系の性能を推定することにした．

まず，動作を模擬するにあたってまずは `localize` の特徴である大域アドレスにひも付けされたローカルなメモリ領域を確保・管理する必要がある．この `localize` のシミュレーションには二つの方法を利用することにした．一つは仮想的に PGAS として扱うメモリ領域のアドレスをそのままひも付けされたローカルなメモリ領域として返す方法である．この方法では大域アドレスとローカルなメモリアドレスが常に等しいので `localize` や `commit` はデータに関して何もすることはない．

もう一つの方法は処理系が別のメモリ領域を確保してこれをひも付けされたキャッシュバッファ領域として用いる方法である．後者の手法は前者に比べてキャッシュと実際の領域が異なっている点で実際の分散環境に近く，そのためユーザープログラムの記述が正しいかどうかの検証もより正確に行える．これに比べて前者は仮想的な PGAS の領域がそのままキャッシュと一致するので `localize` や `commit` を適切に行わなくてもプログラムが動作してしまい，ユーザープログラムの検証としてはもう一方の方法より弱いといえる．ただし，今回は前者しか実装できなかった．

また，PGAS としての性能を推定するには当然ノード間でのページの転送コストを推定する必要がある．これに関しては単純にオーナーを変更する操作の際にオーナーを更新し，実際に `localize` や `commit` で読み書きする際にオーナーに合わせたコストを加算する．ここではリモートアクセスの際にかかるコストの単純なモデルとして遅延 d をレイテンシ l とバンド幅 B，転送するページの大きさ s を用いて

$$d = h + \frac{s}{B}$$

と仮定して人工的に遅延を加えることにした．

4. 評価

4.1 実験

今回は簡単なベンチマークアプリケーションとして Cilk の example として付属する物を移植して評価を行った．今回は 10G Ethernet を想定して $l = 1.8$msec., $B = 9$Gbps というパラメータを設定した．cilksort というアプリケーションはマージソートを並列実行に適した形に修正したものであり，matmul は密行列積である．これらの実行結果のスケーラビリティを**図 5**，**図 6** に示す．また 32 コアで実行した際の実行時間を**図 7**，**図 8** に示す．なおそれぞれのアプリケーションにおいて PGAS のページサイズは cilksort で 2^{18} バイト，matmul で 2^{14} バイトである．また行列のサイズは 1024 である．

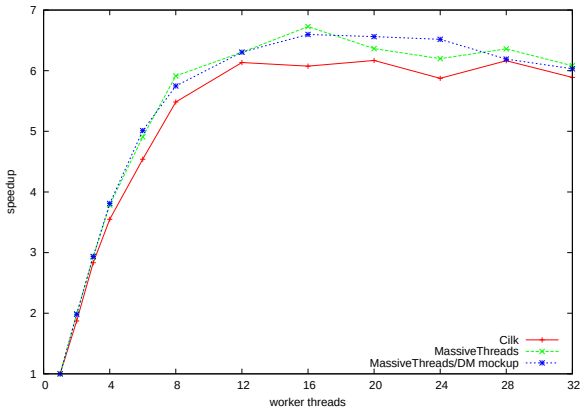

図 5 Speed-up of benchmark cilksort

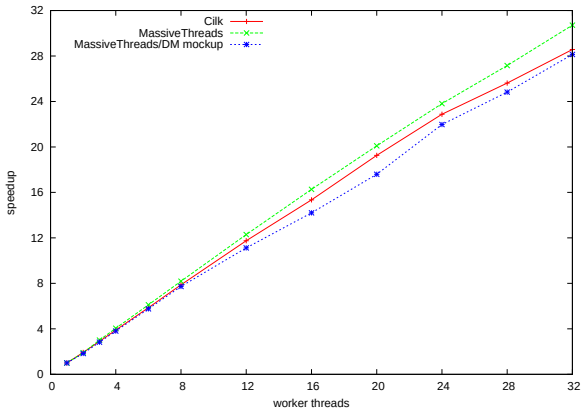

図 6 Speed-up of benchmark matmul

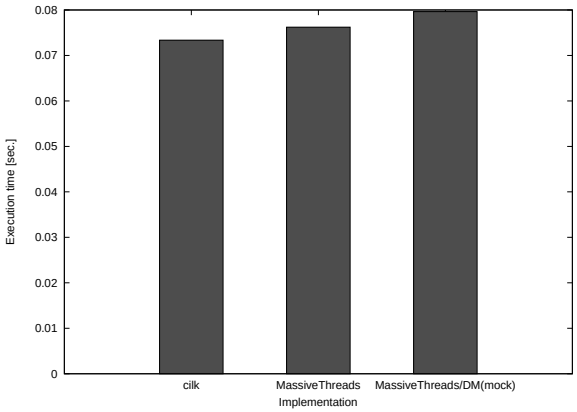

図 7 Execution time of cilksort with 32 cores

4.2 考察

Cilk と MassiveThreads は 32 コアの時点でほぼ互角の性

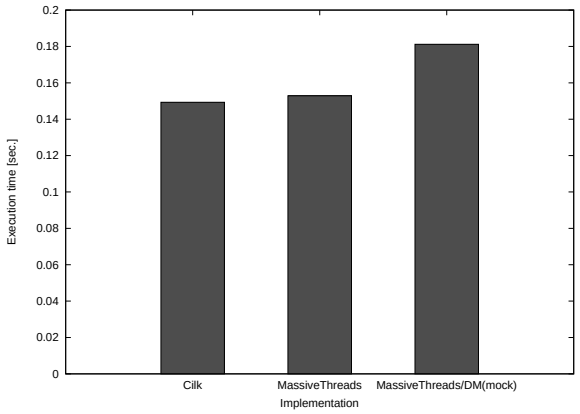

図 8 Execution time of matmul with 32 cores

能となっているが, Cilk の方が高い性能を持っていた．一方 MassiveThreads/DM における推定結果は MassiveThreads に対して 1 割から 2 割ほどのオーバーヘッドが大きくなっている．これは当然ページの移転に対してかけた遅延が性能に悪影響を与えていると考えられる．

5. 結論

5.1 まとめ

分散タスク並列処理系 MassiveThreads/DM とそれに用いるための GAS のセマンティクスの予備的評価として共有メモリ上でその動作を確認した．特に今回はネットワークによる遅延に着目し，これを模擬的に再現することで分散環境での計算を推定した．現在のシミュレーションの結果では十分な性能が得られることがわかった．

5.2 今後の課題

上ではユーザーが利用するデータ領域のアクセスの際のネットワークの遅延については模擬的に再現したが，実際に分散環境で計算を行う際にはタスクスチールの際のタスクの問い合わせやスタックの転送でのコストもネットワークの遅延によって大きくなり，結果的に性能が低下するといったことも当然考えられる．

これらに関してもユーザーの利用するデータ同様にシミュレーションを行うことでより正確な性能の推定を行う必要がある．

参考文献

[1] Siegfried Benkner. Optimizing irregular hpf applications using halos. In *Proceedings of the 11 IPPS/SPDP'99 Workshops Held in Conjunction with the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing*, Vol. 1586, pp. 1015–1024, 1999.

[2] Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall, and Yuli Zhou. Cilk: An efficient multithreaded runtime system. In *PPoPP '95 Proceedings of the fifth ACM SIGPLAN symposium on Principles and practice of parallel programming*. ACM, October 1996. ISBN:0-89791-700-6.

[3] David Callahan, Bradford L. Chamberlain, and Hans P. Zima. The cascade high producivity language. In *9th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS 2004)*, pp. 52–60. IEEE Computer Society, April 2004.

[4] William W. Carlson, Jesse M. Draper, David E. Culler, Kathy Yelick, Eugene Brooks, and Karen Warren. Introduction to UPC and language specification. *IDA Center for Computing Schences*, May 1999.

[5] Philippe Charles, Christian Grothoff, Vijay Saraswat, Christopher Donawa, Allan Kielstra, Kemal Ebcioglu, Christoph von Praun, and Vivek Sarkar. X10: An object-oriented approach to non-uniform cluster computing. In *OOPSLA '05 Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. ACM, October 2005.

[6] Wei-Yu Chen, Dan Bonachea, Costin Iancu, and Katherine Yelick. Automatic nonblocking communication for partitioned global address space programs. In *ICS '07 Proceedings of the 21st annual international conference on Supercomputing*, 2007.

[7] Wei-Yu Chen, Costin Iancu, and Katherine Yelick. Communication optimization for fine-grained UPC applications. In *the International Conference on Parallel Architecture and Compilation Techniques*, 2005.

[8] James Dinan, Sriram Krishnamoorthy, D Brian Larkins, Jarek Nieplocha, and P Sadayappan. Scioto : A Framework for Global-View Task Parallelism. In *Parallel Processing, 2008. ICPP '08. 37th International Conference on*, pp. 586 – 593, 2008.

[9] MPI Forum. Message passing interface forum. `http://www.mpi-forum.org/`.

[10] D.B. Loveman. High performance fortran. *Parallel & Distributed Technology: Systems & Applications, IEEE*, Vol. 1, pp. 25–42, February 1993.

[11] X. Martorell, J. Labarta, N. Navarro, and E. Ayguadé. Nano-threads library design, implementation and evaluation. Dept. d 'Arquitectura de Computadors-Universitat Politècnica de Catalunya. Technical Report: UPC-DAC-1995-33, 1995.

[12] Jaroslaw Nieplocha, Robert J. Harrison, and Richard J. Littlefield. Global arrays: a portable "shared-memory" programming model for distributed memory computers. In *Supercomputing '94 Proceedings of the 1994 ACM/IEEE conference on Supercomputing*, 1994. ISBN:0-8186-6605-6.

[13] L. Peng, W.F. Wong, M.D. Feng, and C.K. Yuen. SilkRoad: A multithreaded runtime system with software distributed shared memory for SMP clusters. In *Cluster Computing, 2000. Proceedings. IEEE International Conference on*, pp. 243–249, 2000.

[14] John Reid. Co-array fortran for parallel programming. *ACM SIGPLAN Fortran Forum*, Vol. 17, No. 2, pp. 1 – 31, Aug. 1998.

[15] Jimmy Su and Katherine Yelick. Array prefetching for irregular array access in Titanium. In *Sixth Annual Workshop on Java for Parallel and Distributed Processing Symposium*, p. 158, April 2004.

[16] Jimmy Su and Katherine Yelick. Automatic support for irregular computations in a high-level language. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, p. 53, April 2005.

[17] K.B. Wheeler, R.C. Murphy, and D. Thain. Qthreads:

An api for programming with millions of lightweight threads. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pp. 1–8. IEEE, 2008.

[18] Katherine Yelick, Dan Bonachea, Wei-Yu Chen, Phillip Colella, Kaushik Datta, Jason Duell, Susan L. Graham, Paul Hargrove, Paul Hilfinger, Parry Husbands, Costin Iancu, Amir Kamil, Rajesh Nishtala, Jimmy Su, Michael Welcome, and Tong Wen. Productivity and performance using partitioned global address space languages. In *PASCO '07 Proceedings of the 2007 international workshop on Parallel symbolic computation*, 2007.

[19] 原健太朗, 田浦健次朗, 近山隆. DMI: 計算資源の動的な参加/脱退をサポートする大規模分散共有メモリインタフェース. 情報処理学会論文誌, Vol. 3, No. 1, pp. 1–40, March 2010.

[20] 中島潤, 田浦健次朗. 高効率な I/O と軽量性を両立させるマルチスレッド処理系. 情報処理学会論文誌プログラミング（PRO）, Vol. 4, No. 1, pp. 13–26, 2011.