MPC++ Multi-Thread Template Library の様々な通信レイヤ上での実装と性能評価

野田裕介[†] 栄純明[†]
松岡聡[†] 小川宏高[†]

MPC++を含めた並列プログラミング言語は，ユーザレベルスレッド，リモートメソッド起動，リモートメモリアクセスなど細粒度並列処理に必要な様々な機能を言語機能として有する．これらの言語は，その性質ゆえ，特定のハードウェアに依存して実装される場合が多く，コモディティハードウェアによる PC クラスタのように潜在的に多種多様な環境 (プロセッサ，OS，ネットワーク) へのポータブルかつ高効率な実装方法については十分に検討されているとは言い難い．そこで我々は，MPC++の環境依存部分である通信レイヤおよびユーザレベルスレッド機能を他の部分から分離するとともに，通信レイヤを MPI に代表される汎用かつ高速なメッセージ通信ライブラリを利用して実現することで，並列プログラミング言語の実装におけるポータビリティと高効率の両立を試みている．本稿では，通信レイヤとして軽量かつ汎用な通信機構である VIA を用いて MPC++ を実装し，他の様々な実装とともに，基本性能および NAS Parallel Benchmark の CG カーネルの実行性能を評価した．現状の VIA による実装は不十分であるため，3 ノード以上での評価は行えなかったが，既存の MPI を用いた実装に比べて，特に小量のデータ通信において顕著な改善が見られた．また，スループットも 32bytes の場合に 190%改善した．

Implementing and Evaluating of MPC++ Multi-Thread Template Library on Multiple Communication Layers.

Yusuke Noda,[†] Yoshiaki Sakae,[†] Satoshi Matsuoka[†] and Hirotaka Ogawa[†]

Parallel Programming Languages such as MPC++ embody various features required for finer-grained parallel processing such as user-level threads, remote thread invocation, and remote memory access. Such languages are often implemented assuming a fixed, high-performance hardware to eliminate software overhead as much as possible, and portable, high-performance implementations on top of commodity clusters which could involve a variety of execution environment (different CPUs, OS, networks) have not been well investigated. In order to clarify the commodity-level viability of such languages, we have been experimenting with a variety of combinations of the underlying execution environments. In particular, we are testing the use of VIA as an underlying messaging layer for MPC++. Although a general low-level messaging layer, nontheless the semantics of MPC++ makes it non-trivial to perform a straightforward port. Although the current problems in the implementation prohibits us from large-scale benchmarks, the initial experiments with NAS Parallel CG show that implementation of VIA on 100Base-T allows notable speedup compared to MPI implementations due to low-latency messaging, and throughput increasing by 190% for small, 32-byte messages, which is often used for such languages.

1. はじめに

汎用的な OS が動作する・使い慣れた開発環境が使用できる・導入コストが安く済む・技術の進歩の恩恵を受けやすい，などの理由から超並列計算機に代わり，パーソナルコンピュータやワークステーションなどのコモディティハードウェアで構築する，クラスタ型並列計算機が注目されている．しかしクラスタ型並列計算機では汎用ネットワークが要素計算機と比較して必ずしも高速でなかったため，定型的な並列アプリケーションにしか利用できないと考えられてきた．しかし近年，Fibre Channel[2] や Gigabit Ethernet，Myricom 社[3] の Myrinet などのギガビット級の高速ネットワークが比較的安価に入手できるようになってきたため，より幅広い並列アプリケーションにクラスタ型並列計算機を適用可能になってきている．

クラスタ型並列計算機では，開発言語として C や

† 東京工業大学 情報理工学研究科 数理・計算科学専攻
Tokyo Institute of Technology

C++，Fortran に，MPI や PVM といったメッセージ通信ライブラリを用いることが多い．しかしこれらの通信ライブラリではメッセージの受け渡しをプログラマが直接記述しなければならず，大規模なプログラムを書くには労を要する．そこでプログラマがメッセージの受け渡しを直接記述しなくても済むように，より高いレベルで抽象化されたプログラムを記述できる並列言語が開発されてきた．

一方ギガビット級の通信速度を持つ高速ネットワークでは，従来からの TCP/IP などの通信プロトコル上でノード間通信を行っていたのでは，システムコールやデータコピーなどのオーバーヘッドが大きく，ネットワークの物理的性能を生かしきれない．そこで UCB の AM[4]，イリノイ大学の FM[5]，コーネル大学の U-NET[6] などの低レイテンシ，高スループットを実現するユーザレベルの通信ライブラリが開発された．またハイパフォーマンスな並列言語を実現するにはこれらの通信ライブラリに密接していることが必要と考えられ，通信ライブラリごとに並列言語が開発されてきた．

そのような中で，新情報処理開発機構 (RWCP)[7] では Myrinet 上に独自に PM 通信ライブラリ [8] を開発し，Myrinet を用いてノード間を接続したクラスタ型並列計算機用の並列プログラミング環境である SCore の研究を行ってきた．また SCore 上での開発言語として MPC++ Version 2.0 を開発した．

MPC++の核部分である level 0 は C++の継承，テンプレート機能のみを用いて実装されており，メッセージ通信ライブラリ PM および setjmp，longjmp 相当の機能さえ用意できれば，どんなクラスタ型並列計算機上でも動作させることが出来る．しかし，当初の PM は Myrinet 上にのみ実装されていたために，他の多種多様なネットワーク機器には対応できず，これが MPC++のポータビリティの制約となっていた．

この制約を取り除く 2 通りの方法が考えられる．一つは並列言語が用いる通信ライブラリをより多くのネットワーク機器上に実装することであり，RWCP では Myrinet 以外の Ethernet や UDP[9]，などへの PM の移植作業が行われている．もう一つの方法は，PM のような特殊な低レベルのメッセージ通信ライブラリの代わりに，より汎用性のある通信レイヤを用いることである．栄ら [10] は MPI を通信レイヤとして用いることで MPC++の可搬性を実現している．しかしこの二つの方法にはそれぞれ欠点がある．前者の方法は，それぞれのハードウェアに特化した高効率な PM を実現できる反面，個々の実装に必要な労力が少なくない．後者は，多くの環境に適用するのが容易である反面，PM には本来不必要な処理を含んでいるため効率が必ずしも良くはない．そこで我々は，栄らの MTTL-MPI の実装を元に，汎用的で，より低レベルな通信レイヤである VIA[11] を用いた MTTL-VIA を実装した．MTTL-VIA は，MTTL-MPI の効率の悪さを補うと共に，MPC++の可搬性も実現できることが期待される．

本稿では，MTTL-VIA の実装の詳細を述べ，まだ完成度が十分ではないため，そのベースラインの性能と，Nas Parallel Benchmarks の CG カーネルの（不完全な）結果を示す．

2. MPC++ - MTTL の概要

MPC++ は RWCP で開発されているクラスタ型並列計算機向けの言語であり，C++ のオブジェクト機能，細粒度ユーザーレベルマルチスレッド，リモートメソッド起動，リモートメモリリード/ライト，同期構造体，グローバルポインタなどの機能を提供する．MPC++ Version 2.0 は level 0 と level 1 からなり，level 0 では C++ 言語仕様を拡張，変更すること無く，C++ のテンプレート機能を用いて先の並列言語としての機能を定義・実装している．Level 1 ではアプリケーションに特化した拡張を提供するメタレベルアーキテクチャーを規定する．

本稿では MPC++ Multi-Thread Template Library (MTTL) と呼ばれる level 0 のみを扱う．

2.1 MPC++のプログラミングモデルと実行モデル

MPC++ は分散メモリ環境での SPMD プログラミングモデルをサポートする．各プロセスには複数のスレッドが含まれ，これらはプリエンプティブではない．すなわち，スレッドの実行は同期を行うか，プログラマが明示的に yield するか，その実行が終了するまで止らない．

すべての変数は基本的にプロセスローカルである．ファイルスコープで定義された変数はスタートアップ時に各プロセスに割り付けられる．したがってプログラム実行時の変数参照は，そのスレッドの実行されているプロセス内のアドレススペースでのアクセスであり，他のプロセスの変数にアクセスするためには後述の *global pointer* の機能を用いる．

MPC++ プログラムのメインルーチンである `mpc_main` は基本的にはファイルスコープ変数の初期化後，プロセス 0 だけで実行される．他のプロセッサ上のプロセスは，ファイルスコープ変数の初期化後メッセージハンドラの実行に入り，プロセス 0 からのメッセージを待つ．

2.2 invoke, ainvoke 関数テンプレート

invoke 関数テンプレートは，同期的にローカルもしくはリモート関数呼び出し機能を提供する．つまり invoke を呼び出したスレッドは invoke した関数から戻るまでブロックされる．ainvoke 関数テンプレートは，非同期的にローカルもしくはリモート関数呼び出し機能を提供する．invoke，ainvoke の呼び出しには新しいスレッドの生成と，スレッドのコンテキストスイッチを伴う．

2.3 同期構造体 Sync クラステンプレート

multiple readers/writers 通信モデルを実現するために， FIFO 型の通信バッファとして振る舞う同期構造体 Sync クラスを提供する．writer がデータを書き込むと Sync オブジェクトの queue に入れられ，reader がデータを読むと queue の先頭から削除される．reader が読み出すときに queue にデータがないときには reader スレッドはブロックする．またメンバ関数 peek() によって queue からデータを削除せずに読み出すこともできる．そのほか queue の長さを得る queueLength() なども提供する．

2.4 グローバルポインタ GlobalPtr クラステンプレート

他のプロセッサ上のメモリを参照するためのグローバルポインタを実現するために，GlobalPtr クラステンプレートを提供する．GlobalPtr クラスでは配列へのグローバルポインタ，グローバルポインタへのグローバルポインタ，グローバルポインタの指すリモートオブジェクトのメンバ関数の起動，リモートメモリの read/write などの機能を提供する．

3. VIA の概要

VIA[11] とは，Intel[14]，Microsoft[15]，Compaq[16] の三社によって提案された高速インタコネクトの標準 API である．VIA ではネットワークを Virtual Interface (VI) で隠蔽することによって，ユーザが実際のネットワークを意識する必要をなくしている．また VI はユーザレベルで動作するので，ユーザ空間からカーネル空間へのバッファのメモリコピーのオーバーヘッドを受けない．以下では VIA の構成および VIA の持つ機能について概要を説明する．

3.1 VIA の構成

VIA は，VI，Completion Queue，VI Provider，VI Consumer の 4 つの要素から構成される．以下では VIA の各要素について述べる．

3.1.1 Virtual Interface (VI)

VI は VI Consumer がデータ転送操作を行うために直接 VI Provider にアクセスすることを可能にするメカニズムである．

VI は二組のワーク・キュー，Send Queue と Receive Queue から成り立つ．VI Consumer はワーク・キューに対して，デスクリプタの形でデータ送信または受信リクエストを発行する．VI Provider は発行されたデスクリプタを非同期に処理し，完了するとそれらにマークをつける．VI Consumer は完了したデスクリプタをワーク・キューから取り除き，次のリクエストにそのデスクリプタを再利用する．各ワーク・キューは，新しいデスクリプタがワーク・キューに発行されたことを VI ネットワークアダプタに知らせるために使われるドアベルを持っている．ドアベルはアダプタによって直接実装され，実行に OS は介在しない．

Completion Queue を用いることで，VI Consumer は複数の VI のワーク・キューの完了の通知を一箇所で受けることが出来る．Completion Queue については後述する．

3.1.2 VI Provider

VI Provider とは VI を提供するハードウェアとソフトウェアコンポーネントのセットである．VI Provider はネットワークインタフェースコントローラ (NIC) とカーネルエージェントからなる．

VI NIC は VI と Completion Queue を実装し，データ転送機能を直接実行する．

カーネルエージェントは，オペレーティングシステムの特権領域，通常は VI NIC ベンダによって供給されるドライバである．それは VI Consumer と VI NIC の間に VI を保持するのに必要なセットアップとリソースマネジメント機能を持つ．これらの機能は VI の生成 / 破棄，VI 接続のセットアップ/切断，割り込みの管理や処理，VI NIC が使用するシステムメモリの管理，エラーハンドリングなどを含む．

3.1.3 VI Consumer

VI Consumer は VI のユーザの代理を務める．VI Consumer はソケットや MPI のような標準 OS プログラミングインタフェースを通してコミュニケーションサービスにアクセスする．

3.2 データ転送モデル

VIA には，1)Send/Receive メッセージモデル，2) リモートダイレクトメモリアクセス (RDMA) モデルの二種類のデータ転送モデルがある．

3.2.1 Send/Receive

VI の Send/Receive モデルは 2 点間のデータ転送のごく一般的なモデルに従う．一回の接続で，送信側の send Descriptor と受信側の receive Descriptor の間で一対一の通信が行われる．

3.2.2 Remote Direct Memory Access (RDMA)

RDMA モデルで，データ転送の開始側はソースバッファとデータ転送の目的地バッファの両方を指定する．RDMA オペレーションには RDMA Write と RDMA Read の二種類がある．

3.3 Completion Queue

VI を作成する際に VI を Completion Queue に結びつけることによって，完了したリクエストの通知を Completion Queue に向けることが出来る．ワークキューを Completion Queue に結びつけると，全ての完了の同期を Completion Queue で取らなければならない．

VI のワークキューと同様に，Completion Queue は割り込み無しで VI NIC から完了の通知を受ける．また VI Consumer はカーネルに移行することなく完了の同期を取ることが出来る．

4. MTTL-VIA の実装

MTTL-VIA は MTTL-MPI を元に，通信レイヤを VIA で記述し直すことで実装している．

MTTL-VIA および MTTL-MPI ではポータビリティを高めるために，MTTL を変更し，通信レイヤとスレッド部分を明確に分離している．スレッドはユーザスレッドとして実装しており，この部分のみがプラットフォーム依存であるが，`setjmp()`，`longjmp()` が使える環境であれば特に問題なく移植可能である．

図 1 に MTTL-VIA の構造を示す．

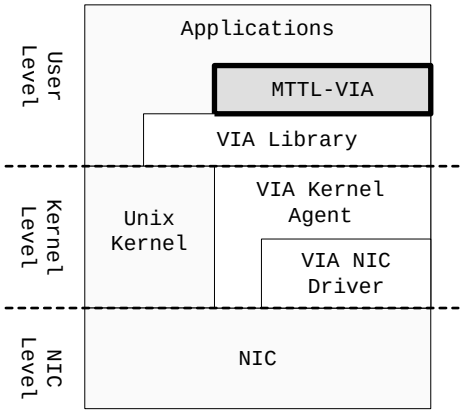

図 1 MTTL-VIA の構造

4.1 通信レイヤ

MTTL-VIA では，それぞれの上位レイヤからは表 18 個の通信プリミティブだけを用いて通信できるように実装されている．そのため他の通信レイヤ上に移植するためには，この 8 個の通信プリミティブのみを書き直せばよい．

しかし，これらの通信プリミティブはもともと MTTL-MPI 用に定義されたものであり，抽象度も MPI と同等なものである．そのためより低レベルで

表 1 MTTL-VIA の通信プリミティブ

_pmttl_cominit	通信の初期化を行う
_pmttl_comfinish	通信の終了処理を行う
_pmttl_prob	受信メッセージのプローブ
_pmttl_asend	非同期送信
_pmttl_nsend	ノンブロッキング送信
_pmttl_senddone	ノンブロッキング送受信の完了確認
_pmttl_brecnode	特定のノードからの受信
_pmttl_brecany	任意のノードからの受信

抽象度の低い VIA でこれらを実装するには注意が必要になる．

以下では特に受信を MTTL を VIA 上へ実装する際に注意する点を挙げる．

MTTL-VIA では任意のノードからのデータの受信を_pmttl_brecany で待つことになるが，VIA での通信は基本的に VI 同士の 1 対 1 通信である．任意のノードからの受信を VIA で実現するにはいくつかの方法が考えられるが、MTTL-VIA では Completion Queue を利用して，任意のノードからの受信を実現している．

以下で MTTL-VIA での_pmttl_brecany の実現方法を述べる．

(1) あらかじめ，受信の完了した (しかし受信確認はまだ行っていない)VI のハンドルとその順番を保存するオブジェクトの linked list を用意しておく．
(2) _pmttl_brecany が呼ばれると，まず linked list の先頭をチェックする．
(3) 先頭にオブジェクトが入っている場合はその VI で受信確認を行い，受信を完了させる．linked list からオブジェクトは取り除く．(完了)
(4) linked list が空の場合は，VipCQWait を実行し Completion Queue にデータが入るまで待つ．
(5) データが入ってきたらその VI に対して受信確認を行い，受信を完了させる．(完了)

以上で_pmttl_brecany が実現できる．
また_pmttl_brecnode，_pmttl_probe でも同様の処理を行う必要がある．

4.2 ユーザスレッド

MTTL-VIA のスレッドは MTTL-MPI と共通であり，setjmp() とlongjmp() を用いた方法でユーザスレッドとして実装されている．MTTL のスレッドはノンプリエンティブである．つまりプログラマが明示的に suspend を行うか，GlobalPtr や Sync 構造体のデータリード操作の際にブロックするなどしない限りスレッドの切り替えは起こらない．

スレッドの管理は各プロセッサ上で Thread クラスの static 変数を用いて行われる．動作できる状態にあるスレッドの数・サスペンドしているスレッドの数・消滅予定のスレッドへのポインタ・現在動作中のスレッドへのポインタなどが管理される．

各スレッドのインスタンスは queue として管理され，それぞれ queue 上の前・次のスレッドへのポインタ，送信・受信バッファ，スタック，そのスレッド上で実行する関数へのポインタ，その関数への引数へのポインタなどを持つ．

リモートでの関数の起動の際には，スレッドが生成され受信バッファ内に含まれる実行すべき関数のアドレスを取り出し，その関数を実行する．

5. 性能評価

MTTL-VIA の性能評価を行った．MTTL-VIA の通信レイヤの VIA としては，NERSC による Linux 上での VIA の実装である M-VIA Version 1.0[18] を用いた．また性能比較対象として，MPC++-MTTL (SCore3.0, PM-Ether) および MTTL-MPI (LAM 6.3.2[19]) を用いた．

5.1 評価環境

性能評価には我々の研究室の PC クラスタ（presto クラスタ）を用いた．各ノードの構成を表 2 に示す．presto クラスタは 32 ノード構成であり，各ネットワークアダプタごとに 32 ポートの 100Base Switch で接続されている．また 2 つのネットワークアダプタのうち，M-VIA は DEC 21140 を，LAM および SCore は EthernetExpress Pro を用いている．

表 2 presto クラスタの仕様

CPU	Pentium II 350MHz
Cache	512KB
Chipset	440BX
Memory	SDRAM 256MB
NIC	Intel EthernetExpress Pro 10/100 (used by LAM, PM-Ether)
NIC	DEC 21140 (tulip) (used by M-VIA)
OS	RedHat Linux 6.1

5.2 通信レイヤの基本性能

まず MTTL-VIA の通信レイヤに用いている M-VIA の通信性能を示す．具体的には PingPong ベンチマークによって 2 ノード間 Send/Receive 型通信のレイテンシおよびスループットを計測した．また性能比較対象である MTTL-MPI の通信レイヤとして用いる MPI についても同様に通信性能を計測した．その結果を図 2 および図 3 に示す．

計測結果から，通信データサイズが小さい時に特に顕著に M-VIA のレイテンシが小さいことが分かる．したがって，単純に予想すれば，M-VIA を MTTL の通信レイヤとして用いることで，小さなデータの通信や同期の効率化が期待できる．

5.3 MTTL-VIA の基本性能

MTTL-VIA のデータ転送性能を測定するために，リモートメモリリード (GlobalPtr の nread 命令) を用いてスループットおよびレイテンシを計測した．その結果を図 4，図 5 に示す．

M-VIA と LAM との比較と同様に，MTTL-VIA は MTTL-MPI を上回る性能を示すことが分かる．特にメッセージのサイズが 32bytes の時にはスループットが 190%ほど改善されている．また MTTL-VIA と MPC++-MTTL を比較した場合は，MTTL-VIA は MPC++-MTTL に近い性能を達成しているが，メッセージのサイズが小さい場合は MPC++-MTTL のほうがよりレイテンシが小さくなっている．

5.4 CG Kernel benchmark

アプリケーションレベルのベンチマークとして，Nas Parallel Benchmarks より CG を用いた．行列サイズは 14000x14000(class A) を用いた．ただし MTTL-VIA にバグがあるため，現在は 2 ノードまでしか計測できていない．

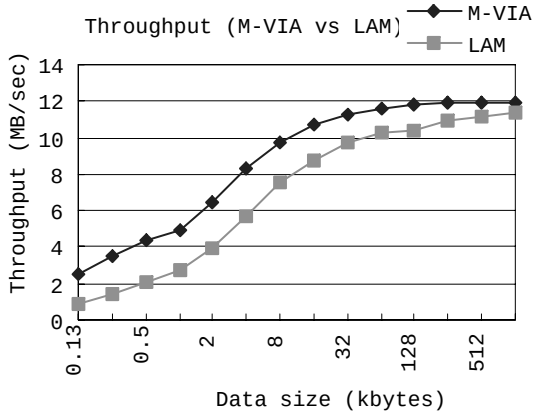

図 2 Throughput: M-VIA vs LAM

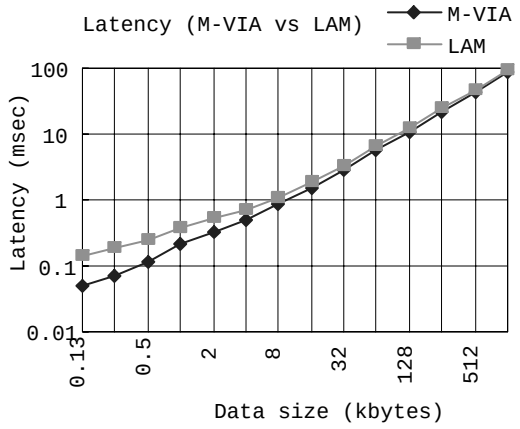

図 3 Latency: M-VIA vs LAM

計測結果を表 3，図 6 に示す．MTTL-VIA は少なくとも 2 ノードまでの性能については他とほぼ同等の性能を示している．しかし，スケーラビリティの問題が現れる可能性がある．この点に関しては引き続きデバッグを行い調査する必要がある．

表 3 NPB CG - Class A

ノード数	MTTL-VIA	MTTL-MPI	MPC++-MTTL
1	57.7401	57.0910	59.7648
2	31.5307	32.1564	32.3026
4	-	20.8920	19.9705
8	-	14.4607	16.8954
16	-	11.6596	14.3579
32	-	12.4950	14.1001

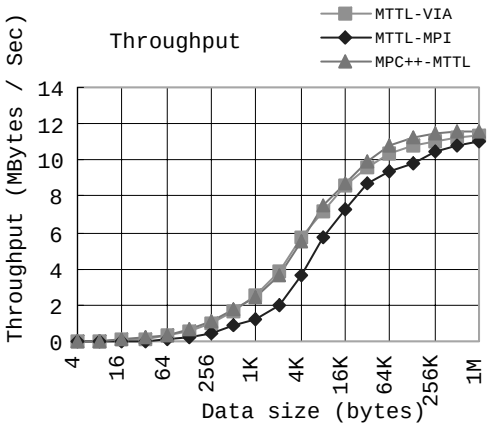

図 4 Throughput: Remote Memory Read

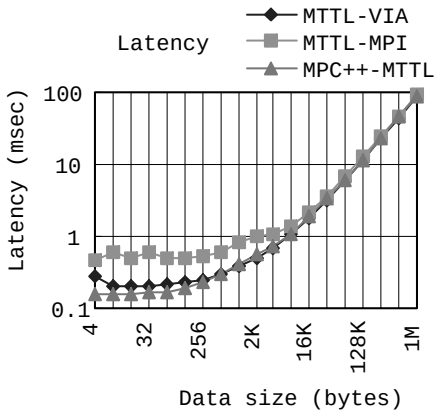

図 5 Latency: Remote Memory Read

6. 関 連 研 究

1 章でも述べたとおり，MPC++の汎用性を得るための方法には，1) MPC++の実行環境である SCore を Myrinet 以外のネットワーク上へ移植する，2) MPC++の通信部分を変更することで他の汎用的な通信ライブラリ上へ MPC++を実装する，の二つの方法が考えられる．前者の方法の研究として，SCore が利用している通信ライブラリである PM を Ethernet 上へ移植した PM-Ether，UDP 上へ移植した PM/UDP[9]，および Gigabit Ethernet 上へ移植した GigaE PM[20] がある．また，後者の方法としては MPC++を通信ライブラリとして MPI を用いて実

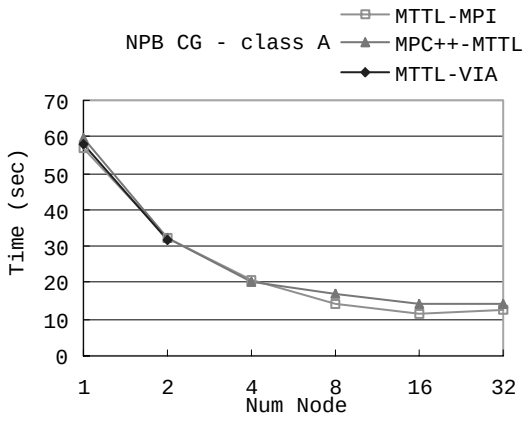

図 6 NPB CG - class A

装した MTTL-MPI[10] がある．

7. まとめと今後の課題

我々は並列言語 MPC++のパフォーマンスとポータビリティを両立するために，MPC++を低レベルでかつ汎用的な通信ライブラリである VIA を用いて実装した．通信ライブラリとして VIA を用いることによって，MTTL-MPI のオーバーヘッドを補うと共に MPC++の可搬性も実現できることが出来ると期待される．

マイクロベンチマークを用いた性能評価の結果から，MTTL-VIA は基本的な通信性能については通信ライブラリの性能に応じた性能が得られている．またアプリケーションベンチマークとして Nas Parallel Benchmarks より CG Kernel Benchmark を行った結果，2 ノードまでについては MPC++他の実装と比較してほぼ同等の性能を示した．しかしまだ実装が不完全なため 4 ノード以上の計測は行うことが出来ず，スケーラビリティに関しては明らかにすることは出来なかった．

今後の課題としては，MTTL-VIA の実装を完全にし，多ノード時でのアプリケーションベンチマークを実行することによって，MTTL-VIA のスケーラビリティについて調査することが挙げられる．また，Berkeley VIA[21] などの，M-VIA 以外の VIA の実装を用いての評価も行う必要がある．

参 考 文 献

1) 石川裕. コモディティハードウェアを用いた並列処理技術. 情報処理, Vol. 39, No. 8, pp. 784–791, August 1998.
2) http://www.fibrechannel.com/.
3) http://www.myri.com/.
4) Thorsten von Eicken, David E. Culler, SethCopen Goldstein, and KlausErik Schauser. Active Messages: a Mechanism for Integrated Communication and Computation. In *19th Int. Symp. on Computer Architecture*. ACM Press, May 1992.
5) http://www-csag.cs.uiuc.edu/projects/comm/fm.html.
6) Anindya Basu, Vineet Buch, Werner Vogels, and Thorsten von Eicken. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. In *SOSP (the 15th ACM Symposium on Operating Systems Principles)*, December 1995.
7) http://www.rwcp.or.jp/.
8) Hiroshi Tezuka, Atsushi Hori, Yutaka Ishikawa, and Mitsuhisa Sato. PM: An Operating System Coordinated High Performance Communication Library. In Peter Sloot and Bob Hertzberger, editors, *High-Performance Computing and Networking '97*, Vol. 1225, pp. 708–717. Lecture Notes in Computer Science, April 1997.
9) 曽田哲之, 手塚宏史, 住元真司, 堀敦司, 石川裕. 通信ライブラリ PM の UDP 上への移植と評価. In *HOKKE '99*, pp. 127 – 132. 情報処理学会, March 1999.
10) 栄純明, 石川裕, 松岡聡, 小川宏高. MPC++ Multi-Thread Template Library の MPI による実装と性能評価. In *SWoPP'99*, pp. 41 – 46, August 1999.
11) Intel Corporation. Intel Virtual Interface (VI) Architecture Developer's Guide Revision 1.0. September 1998.
12) Yutaka Ishikawa. The MPC++ Multi-Thread Template Library on MPI. Technical Report 005, Tsukuba Research Center, Real World Computing Partnership, October 1997.
13) 石川裕. 並列オブジェクト指向プログラミングの動向. 湯淺太一, 安村通晃, 中田登志之（編）, はじめての並列プログラミング, pp. 115–128. 共立出版, June 1998.
14) http://www.intel.com/.
15) http://www.microsoft.com/.
16) http://www.compaq.com/.
17) Soichiro Araki, Angelos Bilas, Cezary Dubnicki, Jan Edler, Koichi Konishi, and James Philbin. User-Space Communication: A Quantitative Study. In *SC98*. ACM/IEEE, November 1998.
18) http://www.nersc.gov/research/FTG/via/.
19) http://www.mpi.nd.edu/lam/.
20) 住元真司, 手塚宏史, 堀敦史, 原田浩, 高橋俊行, 石川裕. Gigabit Ethernet を用いた高速通信ライブラリの設計と評価. 並列処理シンポジウム JSPP'99, 第 99 巻, pp. 63 – 70. 情報処理学会, June 1999.
21) http://www.millennium.berkeley.edu/via.php3.