コモディティ分散共有メモリ IBM x440 の性能評価

西田 晃[†,††] 額田 彰[†] 小柳義夫[†]

本研究では, コモディティハードウェアによって構成された分散共有メモリ型アーキテクチャを採用することにより, 容易に資源の拡張が可能な計算機環境を実現することを目標とするとともに, ユーザが共有メモリアーキテクチャの下位構造を意識することなく高性能な科学技術演算を行なうことのできる計算環境の構築を目指している. 本稿では, 高速ネットワークを用いた廉価な分散共有メモリである IBM 社の PC サーバ x440 に実装された Linux システムを利用し, 疎行列アルゴリズムの実機上での評価結果について報告するとともに, コモディティ分散共有メモリの可能性と実装上の問題点について検討する.

Performance Evaluation of Commodity Distributed Shared Memory IBM x440

Akira Nishida,[†] Akira Nukada [†] and Yoshio Oyanagi[†]

In this study, we target a commodity NUMA computing environment which enables users to realize high performance scalable scientific computing without thinking the details of its architecture. We report the preliminary results of the performance evaluation of sparse matrix algorithms on IBM's ccNUMA server x440, and discuss the merits and the subjects of the ccNUMA architecures based on commodity components.

1. はじめに

近年, 並列アーキテクチャ技術の発展により, 高性能な共有メモリ型並列計算機が比較的容易に利用できるようになってきた. 共有メモリ型アーキテクチャは, プロセッサの接続形態によってバス結合型とネットワーク結合型に大別することができる. バス結合型アーキテクチャでは, バスの帯域幅による制約から, 構築可能な並列環境の規模が限られている. これに対して, 主記憶を分散配置するネットワーク結合型の分散共有メモリアーキテクチャでは, 主記憶に対するアクセス時間は不均等になるものの, プロセッサの拡張性に関する物理的制約はほとんどなく, 高い拡張性が必要となる大規模科学技術計算に適した形態であると考えられる.

一方で, コモディティプロセッサの高速化に伴い, 高速の PC をネットワークで結合したクラスタ技術が, 大規模科学技術計算においても実用的な選択肢のひとつとなってきている. しかしながら, クラスタ上でのメッセージ通信による並列処理には, 明示的なデータ分割の必要性や通信遅延などの制約がある. 現在, このような安価なノードを組み合わせて, 大規模な分散共有メモリ環境を構築するためのチップセット等の開発が Intel, IBM 等の主要ベンダにより進められている[11].

本研究では, コモディティハードウェアによって構築された分散共有メモリアーキテクチャを採用することにより, 容易に資源の拡張が可能な計算機環境を実現することを目標とするとともに, スケーラブルな性能向上を得る上で障害となるオペレーティングシステム上のボトルネックを解消し, 高レベルな最適化を実現することにより, ユーザがアーキテクチャの下位構造を意識することなく高性能な科学技術演算を行なうことのできる計算環境の構築を目指している. 本稿では, Intel Xeon プロセッサを用いた分散共有メモリ型並列計算機として昨年開発された IBM 社製 PC サーバ x440 を使用して, 実機上での評価結果をもとに考察を行った.

2. 背景

大規模疎行列を扱う反復解法において, 疎な成分を持つ行列とベクトルの間の演算は計算量の大きな割合を占める. これらは BLAS[10] 等を用いて実装することが可能であるが, 疎行列 - ベクトル間演算において, 十分にキャッシュメモリを活用するのは難しい. したがって, 疎行列アルゴリズムの並列化を行う場合, BLAS ルーチンレベルでの並列化が最適な選択肢であるかどうかを決定するためには, 十分な評価が必要である.

BLAS の並列実装に関しては, 1) ScaLAPACK[6] のように, MPI による BLAS と等価な並列化ライブラリを構築するものと, 2) ATLAS[12] のようにスレッドを

† 東京大学大学院情報理工学系研究科コンピュータ科学専攻
Department of Computer Science, the University of Tokyo
†† 科学技術振興事業団 JST

用いた共有メモリアーキテクチャ向けの並列化を行うものが提案されており, 1), 2) とも行列間演算のようなキャッシュメモリの活用の容易な BLAS ルーチンについては, 効率的な並列実装を実現している. PBLAS は分散メモリアーキテクチャを主な対象としており, また ATLAS における並列化は, 本研究で想定しているような疎行列解法への適用を想定したものではないが, メモリアクセス遅延の小さい共有メモリアーキテクチャ上においては, ループレベルでの並列化によって, ベクトル間演算や行列 - ベクトル間演算についても効率的に性能向上を達成できるものと期待される.

本稿では, 大規模疎行列を対象とする固有値解法の一種である Jacobi-Davidson 法[13] を例にとり, 計算量の大部分を占める要素演算について, 分散共有メモリシステム上で高速に処理するための条件について考察する. Jacobi-Davidson 法の実装としては Fokkema, van Gijzen ら[9] による Templates[5], BLAS 及び LAPACK[6] を用いたものが知られており, $y \leftarrow \alpha Ax + \beta y$ 型の複素行列 - ベクトル間演算である Level 2 BLAS ルーチンの zgemv をはじめとして, 複素ベクトル間演算 zdotc ($\bar{x}^T y$), zaxpy ($y \leftarrow \alpha x + y$), dznrm2 ($\| x \|_2$) などの Level 1 BLAS ルーチンが演算量に占める割合が大きく[13], ベクトル計算機上での処理に適したアルゴリズムである. 本研究ではこれらの関数を OpenMP API を用いて並列化し, 分散共有メモリシステム上での評価を行った.

3. 評価環境

3.1 アーキテクチャ

本研究では, Intel アーキテクチャベースの分散共有メモリ型計算機として IBM 社で開発された PC サーバ x440 を導入し, コモディティ分散共有メモリ上での計算性能について評価を行った. x440 は Intel Xeon プロセッサ向けの IBM 社製チップセットである Summit を用い, 4 個の Xeon プロセッサを内蔵可能なノードを専用高速ネットワークで相互に接続することにより, 最大で 16 個☆の Xeon プロセッサを単一インスタンスのオペレーティングシステムで管理することができる.[3] 本研究では筐体内の 2 個のノードに 2.4 GHz の Xeon DP プロセッサを 2 個ずつ搭載し, 4 プロセッサ構成で評価を行った. 図 1 に x440 のシステム構成, また図 2, 図 3 に専用ネットワークによるノード間の接続形態を示す[3),4)]. ノード内のプロセッサ - チップセット及びメモリ - チップセット間, ノードを結ぶ通信ケーブルの帯域幅はいずれも 3.2GB/s であり, ノード間は複数のケーブルによるインタリーブが可能である.

☆ ただし本稿の執筆時点では, Xeon DP プロセッサの場合は 1 ノードに 2 個まで, 最大で 2 ノード 4 個までの構成となっている.

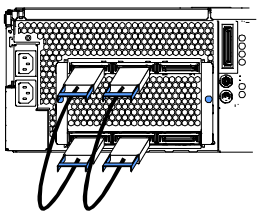

図 2 SMP expansion ports with two SMP expansion modules installed.

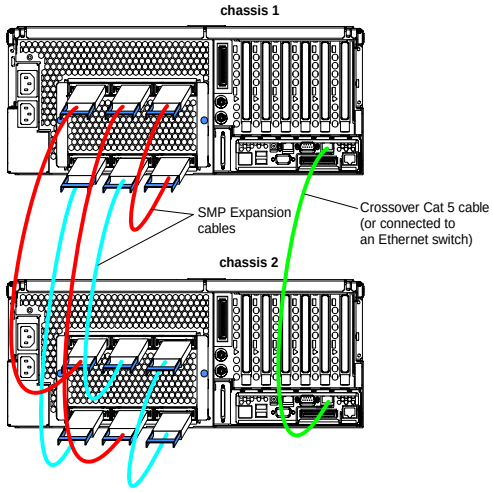

図 3 Connecting the two x440s together in a 16-way configuration.

3.2 オペレーティングシステム

複数のノードから構成される分散共有メモリアーキテクチャ上において, オペレーティングシステムレベルでスケーラビリティを確保するための手法としては, 以下のものを挙げることができる.

ノードアフィニティ
プロセスの移動を指定したノード内に限定する機能[8].

メモリアフィニティ
物理的なメモリ配置情報に基づいて, プロセスのメモリ要求時にローカルノードのメモリを優先的に割り当てる機能.

プロセッサアフィニティ
プロセスを指定したプロセッサに固定する機能.

ページマイグレーション
あるメモリページに対して特定のリモートノードからのアクセスが集中する場合, OS がこれを検知してページをリモートノードにコピーする機能. オーバヘッドに注意する必要がある.

これらの機能により, リモートノードへのメモリアクセスが減少し, データの局所性が向上するとともに, CPU 間のプロセス移動によるキャッシュミスの発生を抑えることができる.

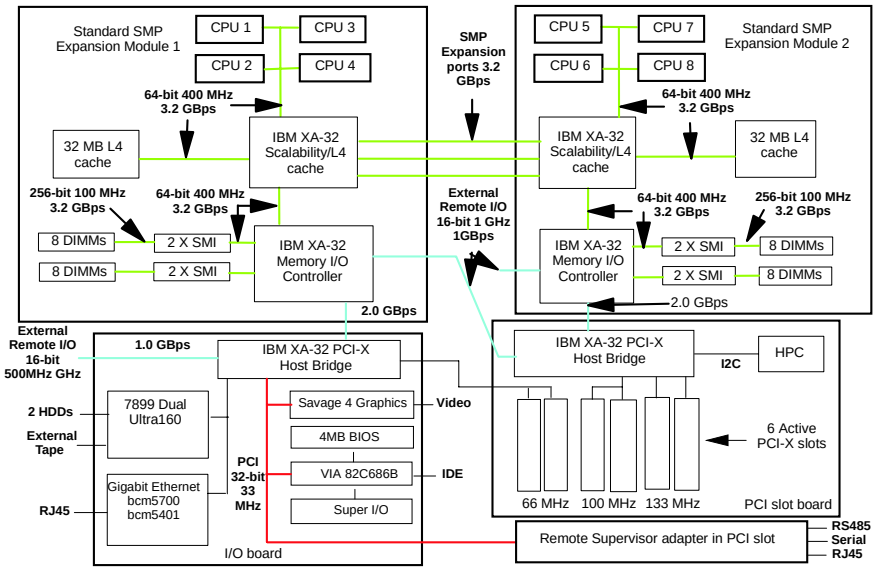

図 1 x440 architecture block diagram.

分散共有メモリアーキテクチャ上で高い計算性能を得るためには, 計算アルゴリズムにおいてもさまざまなレベルでの最適化が必要である. このようなシステムを研究する上で, 仕様が完全に公開され, 無償でソースコードの入手及び自由な改変が可能なオペレーティングシステム及び各種ライブラリの開発が, ネットワークの普及に伴って一般的になりつつある. そのようなオペレーティングシステムのひとつとして, Linux を挙げることができるが, Linux において実現されているスケジューリング機能は、現時点では比較的小規模なシステム向けに設計されており, 本研究で対象とする大規模な共有メモリアーキテクチャ上で使用するには, 性能上問題がある. 以上の背景をもとに, Linux においてスケーラビリティを向上させるための研究が進められている[1]. 本研究では, x440 上に実装された Linux オペレーティングシステムを用いて, 分散共有メモリ上での計算性能を評価した.

4. 評　　価

評価には IBM x440 (2.4GHz Intel Xeon DP Processor ×4, 32KB L1 cache (16KB data), 512KB L2 cache, 64MB node cache, 4GB main memory), 及び比較対象として Itanium プロセッサを用いた分散共有メモリ型並列計算機である NEC AzusA (733MHz Intel Itanium Processor ×8, 32KB L1 cache (16KB data), 96KB L2 cache, 2MB L3 cache, 2GB main memory) を用いた. x440 の 2 台のノードにはそれぞれ 2 個のプロセッサ, 32MB の L4 cache (2-way interleaved PC200 DDR-SDRAM) 及び 2GB の PC133 SDRAM が, また AzusA の 4 台のノードにはそれぞれ 2 個のプロセッサと 512MB の PC100 SDRAM が搭載されている.

オペレーティングにはそれぞれ Red Hat Advanced Server 2.1 及び Red Hat 7.1 ベースの NEC Linux R1.2 を, またカーネルにはそれぞれ 2.4.21-pre4 及び 2.4.7-nec1.2 を使用した. x440 には標準の Red Hat Advanced Server が使用されており, 分散共有メモリ対応については開発版カーネルの 2.5 系列で作業が進められている☆. 一方 NEC Linux には既にメモリアフィニティ機能, プロセッサアフィニティ機能が実装されており, 以下の評価ではメモリアフィニティを有効にした状態で測定した. コンパイラにはいずれも Intel Compiler for Linux 6.0 の最新版を用いた.

4.1 EPCC OpenMP Microbenchmark

OpenMP の実行モデルでは, プログラムの実行はマスタスレッドと呼ばれる単一プロセスとして開始される. マスタスレッドは通常のステートメントを逐次実行し, 指示文の対で構成される並列構造が現れると, 1 つ以上のスレッドからなるチームを生成し, チームのメンバのそれぞれについてデータ環境の設定を行なう. 並列構造内のステートメントは, チーム内の各スレッドによって並列に実行され, 並列構造の終了時点でチーム内のスレッドは同期し, マスタースレッドは更新されたデータを用いて計算を続ける.

ここでは OpenMP API を用いた並列計算に伴う通信遅延を調べるため, EPCC OpenMP Microbenchmark[7] を用いて評価を行った. Microbenchmark は,

☆ なお, 最新の開発版カーネルを用いた評価では一部の機能について安定した性能が得られていないため, 今回のデータとしては採用しなかった.

エジンバラ大学で開発された OpenMP の同期及びスケジューリングオーバヘッド測定のためのベンチマークであり, 内部に dummy 関数を含むループについて, OpenMP 指示文を追加した場合に生じる遅延を測定する. 時間測定にはナノ秒オーダの実行時間を計測することのできる関数 `clock_gettime()` を用いた. 縦軸は実行時間をクロックサイクル数に換算した値である. Intel Compiler にはコンパイラオプションとして `-O3 -zero -ip -nodps -w -openmp` を用い, x440 上ではこれに `-tpp7 -axW` を追加した.

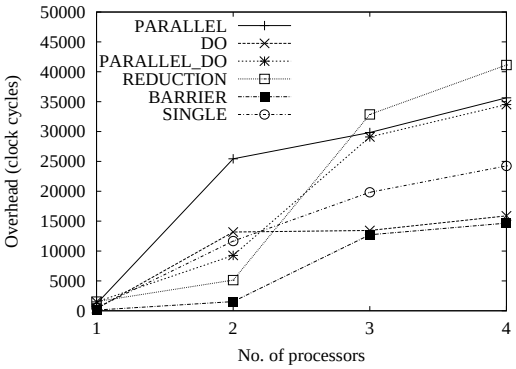

図 4 Synchronization overheads on IBM x440.

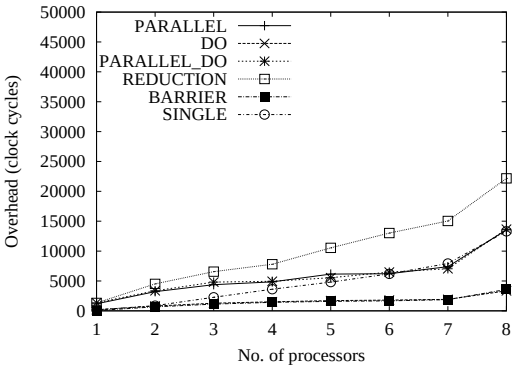

図 5 Synchronization overheads on NEC AzusA.

図 4, 図 5 に x440, AzusA 上での同期オーバヘッドを示す. この結果から分かるように, 同期指示文に関して, x440 では AzusA と比較して相対的に非常に大きなオーバヘッドが生じている. これは Xeon プロセッサの周波数が 2.4GHz であり, 733MHz の Itanium プロセッサの 3 倍以上であるにもかかわらず, ネットワークを経由することで通信遅延が AzusA の場合よりも増大していることが主な原因であると考えられるが, 一方スケジューリングオーバヘッドに関しては, 図 6, 図 7 に示す 4 スレッドでの実行結果から, chunk サイズにもよるが AzusA と比較して小さな値となっていることが分かる.

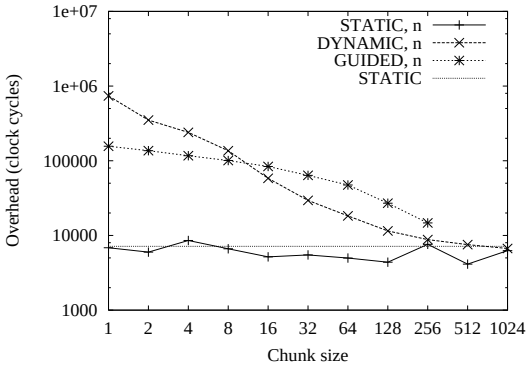

図 6 Scheduling overheads on IBM x440.

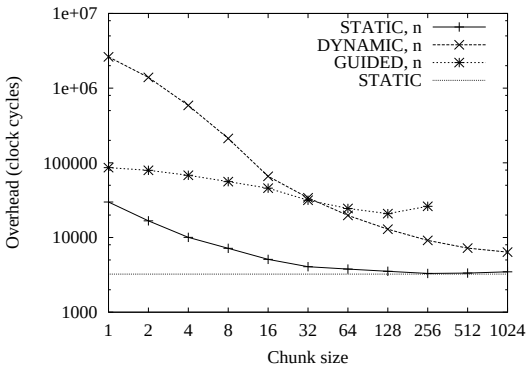

図 7 Scheduling overheads on NEC AzusA.

4.2 STREAM benchmark

計算機の特性についてより詳しく調べるため, バージニア大学で開発されている STREAM benchmark[2] を用いてそれぞれのシステムのメモリ帯域幅を測定した. このベンチマークプログラムでは, 倍精度浮動小数配列に対して以下の演算を行い, 実測値をもとに計算機の実効帯域幅を評価する.

表 1 STREAM benchmark types

Benchmark	Operation	Bytes per iteration
Copy	a[i] = b[i]	16
Scale	a[i] = q * b[i]	16
Add	a[i] = b[i] + c[i]	24
Triad	a[i] = b[i] + q * c[i]	24

ここで扱っているルーチンは, 冒頭で述べた我々の対象とする計算に極めて近く, メモリ帯域幅の大きな計算機に適した内容である. ここでは OpenMP 版の並列プログラムである `stream_d_omp.c` を用い, それぞれのアーキテクチャ上で評価した. 図 8 は問題サイズを

4,000,000 とした場合の x440 上での性能値であるが, 最大で 1GB/s 弱となっており, 図 10 に示すメモリアフィニティ機能を使用しない状態で測定した AzusA 上での値と同程度である. これに対して問題サイズ 2,000,000 の場合には, 図 9に示すように高い値が得られる. 問題サイズ 2,000,000 の場合に必要となるメモリ量は約 44MB であるので, このようなクラスの問題に対しては, x440 に搭載されている 64MB の node cache が効果を発揮するものと考えられる. 一方, 図 10, 11から, NEC AzusA 上においては, メモリアフィニティ機能を利用することにより大幅な性能の向上が得られていることが分かる. x440 ではまだ十分な性能が得られていないが, オペレーティングシステムが対応することで, より高い性能が得られるようになるものと思われる.

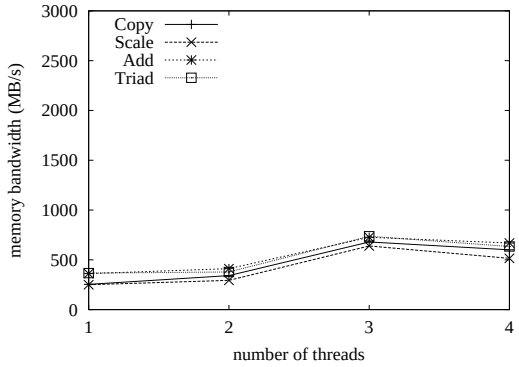

図 8 STREAM benchmark performance (MB/s) with array size 4,000,000 on IBM x440 (without memory affinity).

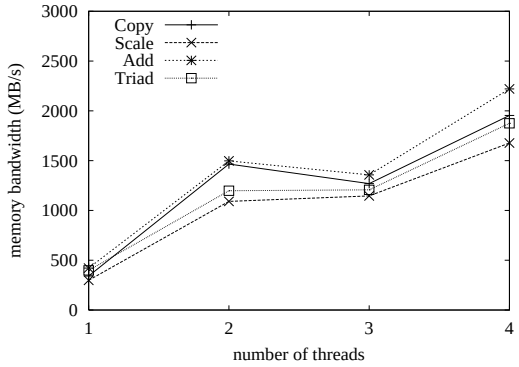

図 9 STREAM benchmark performance (MB/s) with array size 2,000,000 on IBM x440 (without memory affinity).

なお, メモリアフィニティ機能使用時の AzusA において, 図 11 に示すような起動直後の高いスケーラビリティが, 時間が経過するにしたがって次第に失われることが分かっている. これはオペレーティングシステムによって生成されるファイルキャッシュが起動ノード上の主記憶から順に配置されていくことによると考えられるが, この問題は, SGI 社の分散共有メモリ型計算機 Origin で既に実現されているように, ファイルキャッシュをラウンドロビン方式で分散配置することによって解決できるものと思われる.

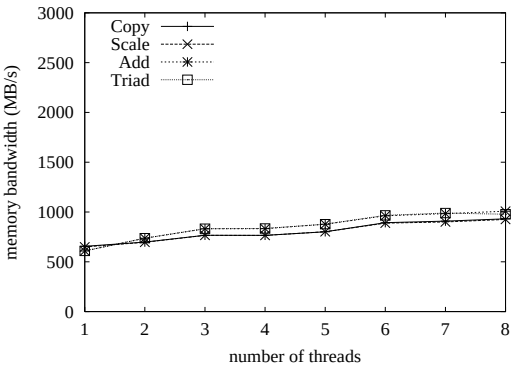

図 10 STREAM benchmark performance (MB/s) with array size 20,000,000 on NEC AzusA (without memory affinity).

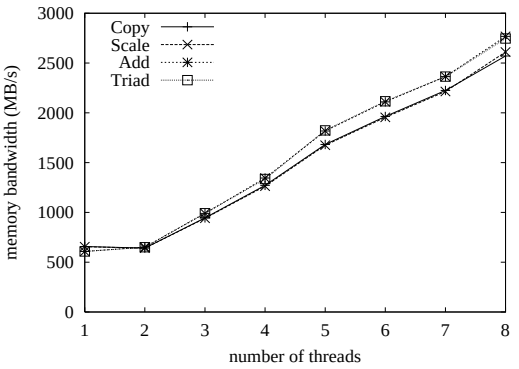

図 11 STREAM benchmark initial performance (MB/s) with array size 20,000,000 on NEC AzusA (with memory affinity).

4.3 並列化 BLAS ルーチン

ここでは, より実際に近い問題として, Jacobi-Davidson 法において計算量の大きな部分を占める複素演算ルーチン zgemv, zdotc, zaxpy, dznrm2 を選び, 評価を行った. それぞれの関数は最外側ループを OpenMP API を用いて並列化している. ベクトルサイズは 256^2, 行列サイズは $256^2 \times 4$ として計測した.

x440, AzusA 上での性能を図 12, 図 13 に示す. x440 が高い性能を示しているが, これは前節で示したキャッシュ階層の効果によるものと思われ, プロセッサ内蔵

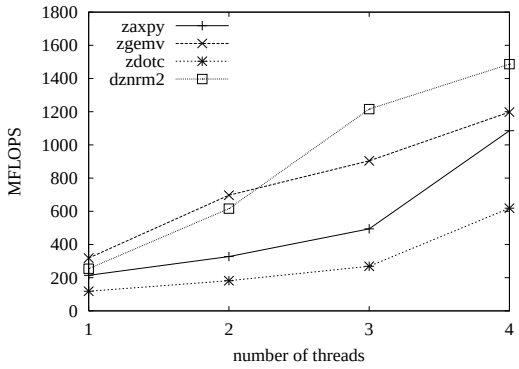

図 12 Parallel BLAS performance on IBM x440.

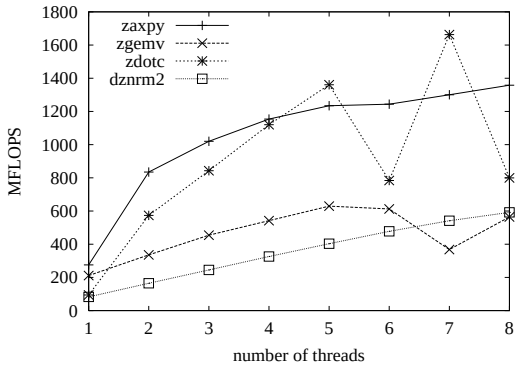

図 13 Parallel BLAS performance on NEC AzusA.

キャッシュサイズの差や, ネットワークレイテンシを補う十分な性能が得られている.

5. ま と め

本稿では, コモディティハードウェアによる分散共有メモリ型アーキテクチャとして, IBM の Intel Xeon 搭載サーバである x440 を用いて, 疎行列アルゴリズムの要素演算の分散共有メモリアーキテクチャ上への実装方式について検討を行った.

疎行列アルゴリズムにおいて必要となる低レベルな BLAS 演算の並列化を考える場合, レイテンシを小さくするとともに, ノード間でのメモリ帯域幅を十分に確保することが必要となる. このためには, x440 で採用されているノードキャッシュは有効な手法であると思われる. また, 開発途上のため今回の実験では十分に評価できていないが, オペレーティングシステムについても引き続きより効率的な実装手法を明らかにしていく必要がある. プロセッサレベルでの最適化と合わせ, 今後より厳密な評価を行っていきたい.

謝辞 本研究を進めるに当たり, IBM 社の Martin J. Bligh 氏, 日本電気株式会社の菅沼公夫氏, 河内隆仁氏をはじめ, 多くの方々より様々なサポート及びアドバイスを頂きました. 感謝致します. なお, 本研究の一部は, 特定領域研究 (2) 14019030, 科学研究費補助金基盤研究 (B) 13480080, 及び科学技術振興事業団戦略的創造研究推進事業によるものである.

参 考 文 献

1) *Linux Scalability Effort Homepage*, http://lse.sourceforge.net/.
2) *STREAM: Sustainable Memory Bandwidth in High Performance Computers*, http://www.cs.virginia.edu/stream/.
3) *IBM eserver Xseries 440 Planning and Installation Guide*, 2002.
4) P. Baldwin, *EXA a new class of Intel based Servers*, in IBM Vision 2002, 2002.
5) R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, 1994.
6) L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley, *ScaLAPACK Users' Guide*, Society for Industrial and Applied Mathematics, 1997.
7) J. M. Bull, *Measuring Synchronisation and Scheduling Overheads in OpenMP*, in Proceedings of the First European Workshop on OpenMP, 1999, pp. 99–105.
8) E. Focht, *Node affine NUMA scheduler*, http://home.arcor.de/efocht/sched/.
9) D. R. Fokkema, G. L. G. Sleijpen, and H. A. van der Vorst, *Jacobi-Davidson style QR and QZ algorithms for the partial reduction of matrix pencils*, Tech. Rep. 941, Department of Mathematics, Utrecht University, 1996.
10) L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh, *Basic Linear Algebra Subprograms for FORTRAN usage*, ACM Trans. Math. Soft., 5, pp. 308–323.
11) G. F. Pfister, *In Search of Clusters*, Prentice-Hall, second ed., 1998.
12) B. C. Whaley and J. Dongarra, *Automatically Tuned Linear Algebra Software*, in Proceedings of SC98, 1998.
13) 西田晃, 小柳義夫, *OpenMP* を用いた *Jacobi-Davidson* 法の並列実装とその性能評価, 2002 年並列処理シンポジウム論文集, (2002), pp. 79–86.