B-021

PC クラスタにおける並列言語 MpC と OpenMP の性能比較

Performance Comparison of MpC and OpenMP on PC clusters

長尾知幸† 渡辺義人† 緑川博子† 飯塚肇†
Tomoyuki Nagao Yoshihito Watanabe Hiroko Midorikawa Hajime Iizuka

1．はじめに

MPI が並列プログラミングの標準 API として多くのシステムで利用可能になり，さらに安価な計算機クラスタの構築が容易になり，並列処理は一般化してきた．さらに最近では，煩雑なメッセージパッシング記述が不要で，逐次プログラムとの継続性もよい共有メモリプログラミングモデルの API として OpenMP が標準化され，共有メモリ型並列マシンのみならず分散メモリ型である計算機クラスタでの利用のための研究が行われるようになってきている．

しかし，計算機クラスタのように，ノード内外のメモリアクセス速度に大きな差のあるようなシステムに対し，本来，共有メモリ型マシンを想定して設計された共有メモリモデルをそのまま適応してしまうと，性能上，いろいろなデメリットが生じる．特に計算機クラスタで OpenMP を稼働させるために，ソフトウエア分散共有メモリ(SDSM)を下層に用いた場合，無駄な一貫性同期が自動挿入されたり，データの分散機能が不充分であったりと，必ずしも良い性能が得られていない．

筆者らは，計算機クラスタにおける SDSM の開発の経験から，データの階層化を意識したメタプロセスモデルという新しい分散共有メモリモデルを提案し[1]，その API として MpC という並列言語を開発した[6]．

本報告では，PC クラスタにおいて，この MpC と OpenMP でプログラムを記述した場合の比較を行った．OpenMP には SCore システム[5]の Omni を用いた．まず 2 節においてメタプロセスモデルと MpC の概要について説明する．3 節において MpC と OpenMP の使い勝手として言語面，実行環境面について比較する．4 節において速度性能について比較する．5 節において総括する．

2．メタプロセスモデルと MpC

2.1 メタプロセスモデル

通常，クラスタでの並列処理は，図 1 のように，各計算ノードが共同で 1 つの処理を行う．メタプロセスモデルでは，これらの共同で処理を行うプロセス群全体のことをメタプロセスと名づけている．メタプロセスモデルでは，メタプロセス内にあるすべてのプロセスがアクセス可能な shared という概念をデータ階層に加える．この共有データは全プロセスにおいて共通の単一アドレスを持つ．そしてその共有データをメタプロセス内のプロセスに関連付けることができる．あるプロセスが特定の共有データに頻繁にアクセスすることが分かっているようなとき，それらを関連付けることで効率的なデータ配置をすることができる．

† 成蹊大学大学院工学研究科 情報処理専攻

2.2 MpC

MpC はメタプロセスモデルを実現するための，ANSI C を拡張した言語である．

shared を C の storage class specifier として組み込み，そのスコープをメタプロセス全体とした．スコープは図 2 のような階層構造となる．異なるスコープの変数に同じ名前が使用されると最も内側のデータを示すものとして扱われる．

また，共有データを任意のプロセスに割り付けることができる．特に，図 3 に示すように配列データの割付に関しては柔軟性のある割付が可能となっている．

MpC プログラムの実行には，クラスタではユーザレベルソフトウェア DSM である SMS[2]，TreadMarks[3]，JIAJIA[4]を用いており，共有メモリ型マシンでは pthread を用いている．

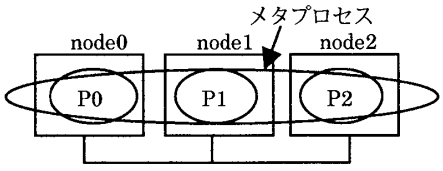

図1 メタプロセスとプロセス

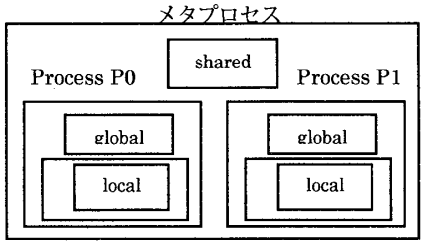

図2 メタプロセスの階層構造

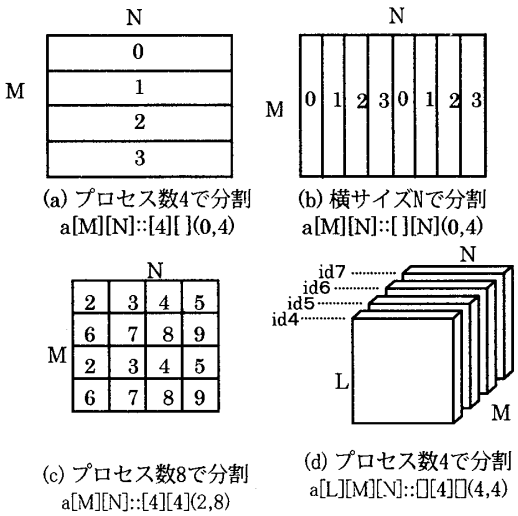

図3 共有変数のプロセス分割例

3. 使い勝手の比較

3.1 プログラミングレベル

3.1.1 動的メモリ割り当て

MpC は shared を用いた静的な宣言方式に加えて，動的な共有データ割り当てができる．一方，通常の OpenMP では PC クラスタ上で動的に割り当てたメモリを共有するための命令は提供されていない．そのため，Omni OpenMP ではそのための命令を拡張として用意している．

3.1.2 暗黙の shared 宣言

OpenMP では，何も指定がなかった場合，すべてのデータは共有される．これは共有メモリ型マシンでの実行では当然のことと考えられるが，そのままクラスタに適用させた場合，不必要な一貫性保持作業が発生する可能性がある．MpC では，共有するデータを明示的に宣言するため，そのようなことが起こらない．MpC は shared 宣言されていない変数はプロセスローカルの変数として扱われる．

また OpenMP では，プログラマはパラレルリージョン内で扱うすべての変数について private 指定をするかどうかを確認する必要があるため，その内部を検討しなければならない．これはパラレルリージョンより前に宣言されていたローカル変数も共有変数となってしまうため注意が必要である． これは，OpenMP がスレッド指向であり，MpC がプロセス指向であるために生じる差である．

3.1.3 分散マッピング

2.2 で述べたように，MpC は柔軟性の高い分散割付をすることができる．一方，OpenMP の標準仕様では分散割付はできない．Omni OpenMP ではデータの分散マッピングとその分散に応じた処理分散をさせるために拡張命令を用意しているが，これらの拡張命令は 1 次元に限った分散しかサポートしていない．そのため，図 3 (c) のような 2 次元での分散をさせることができない．

3.2 実行環境

今回用いた SCore システムにおいて Omni OpenMP をクラスタ上で扱う場合には，カーネルを変更する必要があり，OS インストール作業が発生する．一方，MpC が用いている実装はユーザレベルソフトウェアである．そのため，インストールはユーザ権限で行え，高い移植性があり，様々な OS やアーキテクチャにおいて実行が可能である．

4. 性能比較

測定は表 1, 2 のような環境のもとで行った．測定したのは実際の計算部分のみで，初期化等のオーバーヘッドは除いてある．コンパイルに関して，MpC，Omni OpenMP の各コンパイラはどちらも下層には gcc を用いているので，表 1 には gcc のバージョンも示した．各プログラムの共有データサイズを表 3 に，測定結果を表 4 に示す．比較は，MpC をギガビットイーサネット(以下 Giga)で実行した場合と OpenMP コードを Giga と Myrinet のそれぞれで実行した場合，そして参考として OpenMP コードを逐次コンパイラでコンパイルした場合と，4 種の場合について行った．表 4 で network の行が seq と書かれている列が逐次コンパイルの実行結果である．なお，逐次コンパイルは OpenMP のコンパイル環境での gcc を用いた．

MpC の実行は SMS0.4.18 を利用し，OpenMP の実行は SCASH(SCore5.6.1)を利用した．

表 1 コンパイル環境

	MpC	OpenMP
compiler	mpcc	omcc 1.6
gcc version	3.2.2	2.96
最適化	-O3	-O3
OS	Red Hat Linux 9.0	Red Hat Linux 7.3
kernel	2.4.20	2.4.18
SDSM	SMS	SCASH

表 2 実行環境

	MpC	OpenMP
CPU	Intel PentiumIII 1.13GHz	
CPU 数	1	
メモリ量	128MB	
OS	Red Hat Linux 7.2	
kernel	2.4.21-1SCORE	
network	Giga	Giga, Myrinet
通信プロトコル	TCP/IP	PM

表 3 共有データサイズ

	パラメータ	共有データサイズ
floyd	1024*1024	12MB
laplace	1024*1024	16MB
mandelbrot	1024*1024	1MB
mm	1024*1024	24MB
galaxy	1000	72KB
ep	S	80B

4.1 floyd

Floyd の最短経路探索アルゴリズムを行うものである．大小比較だけと計算量は少なく，共有データへのアクセス頻度は高い．しかしパラレルリージョン内部でのループの回数が多いため，並列効果が得られている．

MpC での測定結果は，OpenMP を Myrinet 上で実行したものと比べて約 25%の速度向上が見られる．それぞれの 1 台の時の測定時間がほぼ同じことから，コンパイラの差によってもたらされたものではないということができる．

OpenMP を Giga 上で実行した場合，速度向上率は低い．

4.2 laplace

Laplace は 4 近傍の値の平均を自身の値にするアルゴリズムである．共有データへのアクセス頻度は高く，計算量は少ない．

MpC の測定結果は，OpenMP を Myrinet 上で実行した場合と同程度であった．また，floyd と同様，OpenMP において Giga で実行した場合は速度向上率は低い．

4.3 mandelbrot

Mandelbrot はマンデルブロー集合を作図するアルゴリズムで，座標に対応する値を用いて収束するまで繰り返し計算するものである．共有データへのアクセスはほとんどなく他プロセスで使われない上に計算量が多いので，高い並列効果が期待できる．しかし，プロセスあたりの計算量にばらつきができる可能性があるため，ここでは処理の終わったプロセスに動的に一定区間を割り当てる場合と 2 次元座標を等分してそれぞれを静的にプロセスに割り当てる場合の 2 通りで測定した．

結果は，どちらの場合も OpenMP での実行時間が MpC での実行時間よりも短くなった．Giga で実行した場合で

も MpC よりも速くなっている．これは，共有データ量が少ないので，通信の負担が軽いためだと思われる．

4.4 mm

mm は行列積計算プログラムである．共有データへのアクセス頻度が高く，計算量は少ない．データアクセスの際キャッシュを活用する為にブロッキングしたものと，標準のものの 2 種類について測定した．

ブロッキングを用いた場合，MpC の結果は OpenMP の Myrinet での結果とほぼ同じものとなった．1 台での実行で差があるが，seq の結果と比べて MpC の結果が同等なことから，gcc のバージョンの差ではないと思われる．

ブロッキングを用いない場合では，MpC は明らかに OpenMP よりも高い結果を得ている．MpC ではほぼ台数分の効果が出ているのに対し，OpenMP では Myrinet でも低い速度向上率で，Giga に関しては実行時間が増えている．

4.5 galaxy

galaxy は他の星との引力を計算する多体問題である．共有データへのアクセス頻度は高いが，計算量が多いので並列効果が得られやすい．星の総数が 1000 個の場合で測定した．なお，シミュレート時間は 100 でステップ時間は 10 である．

結果は，MpC はほぼ台数分の効果が得られた．一方，OpenMP は台数分以上の並列効果が出ている．理由は現在調査中だが，共有データのサイズが非常に小さいことから，1 台のときは CPU キャッシュに入りきらずにメモリアクセスを起こしていたものが，分散させることでキャッシュに収まり，通信にも何らかの効果があったためではないかと考えられる．

4.6 ep

ep は NAS Parallel Benchmarks 2.3 の OpenMP 版のものとそれを MpC で書き直したものを用いた．共有データはほとんどなく，計算量が多いため高い並列効果が得られるプログラムである．

結果は，MpC が OpenMP よりも高い速度を出しているが，seq の結果と比べると明らかな差があるため，gcc のバージョンの違いによる差があると思われ，厳密な比較は難しい．

これら 6 種類のプログラムで MpC と Omni OpenMP の速度性能について比較した．多くに共通しているのは，OpenMP を Giga で実行させた場合には充分な速度向上が得られないということである．また，共有データのサイズが大きく，アクセス頻度が高い問題に関しては，MpC は OpenMP よりも高速に動作することを示した．

5. 終わりに

PC クラスタ上における MpC と OpenMP の比較を行った．今回計測したプログラムは，OpenMP が得意とする，ループを基本とした規則的構造を持つものである．MpC は tsp などの非定型な構造のプログラムも書くことができるが，対応する OpenMP プログラムがないため比較を行っていない．速度性能に関して，このような規則性を持つ多くのプログラムにおいても，MpC は OpenMP と同等かそれ以上の性能を出すことができた．特に安価に準備できるギガビットイーサネットを用いた実行では，多くのプログラムにおいて明らかに高速に動作することを示した．

MpC はユーザレベルでインストールでき，特別な通信デバイスや OS の変更などの前提なしにある程度の並列処理性能を得られることを示した．

表 4　測定結果

		MpC	OpenMP		
	procs	Giga	Giga	myri	seq
floyd (1024)	1	66.65	65.21	65.12	66.80
	2	35.01	69.80	46.57	
	4	19.42	44.94	25.30	
	8	11.38	41.62	14.71	
laplace (1024)	1	3.36	3.18	3.24	3.26
	2	2.06	2.73	2.18	
	4	1.10	1.61	1.21	
	8	0.75	1.40	0.63	
mandelbrot (1024) static	1	1.38	1.44	1.44	1.36
	2	0.94	0.88	0.85	
	4	0.67	0.51	0.50	
	8	0.49	0.28	0.25	
mandelbrot (1024) dynamic	1	1.39	1.44	1.44	1.37
	2	0.87	0.84	0.79	
	4	0.54	0.50	0.42	
	8	0.43	0.35	0.23	
mm (1024) blocking	1	12.18	13.50	14.41	12.32
	2	5.16	5.75	5.22	
	4	3.22	3.87	2.92	
	8	2.16	3.60	2.20	
mm (1024) nonblocking	1	14.09	13.80	13.80	13.61
	2	7.35	20.40	12.30	
	4	4.23	19.60	7.45	
	8	2.77	29.40	5.30	
galaxy (1000)	1	8.49	8.68	8.68	8.75
	2	4.44	2.97	2.49	
	4	2.37	2.13	1.27	
	8	1.41	0.97	0.67	
ep size S	1	11.57	14.45	14.45	13.99
	2	5.79	7.69	7.28	
	4	2.90	4.57	3.71	
	8	1.45	2.76	1.95	

参考文献

[1] Midorikawa,H ： "Meta Process Model: A New Distributed Shared Memeory programming Model", Proc. of the 15th IASTED International Conference on PDCS 2003

[2] 緑川，飯塚："ユーザレベル・ソフトウエア分散共有メモリ SMS の設計と実装", 情報処理学会論文誌ハイパフォーマンスコンピューティングシステム,Vol.42, No.SIG9(HPS 3), pp.170-190 2001

[3] Peter Keleher, et al.: TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems, Proceedings of the Winter 94 Usenix Conference, pp.115-131, (1994).

[4] M.R.Eskicioglu, et al.: Evaluation of the JIAJIA Software DSM System on High Performance Computer Architectures, Proc.of the 32nd Hawaii Int. Conf. on System Sciences, (1999)

[5] http://www.pccluster.org/

[6] 緑川："クラスタおよび共有メモリ型並列マシンのためのポータブル並列プログラミング I/F MpC" , SWOPP2004 CPSY