OUKA
The University of Osaka
Institutional Knowledge Archive

Title	SX4のモニター活動報告書 : MPIの性能評価
Author(s)	日置, 慎治
Citation	大阪大学大型計算機センターニュース. 1998, 108, p. 125-134
Version Type	VoR
URL	https://hdl.handle.net/11094/66275
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

SX4のモニター活動報告書
MPIの性能評価

帝塚山大学　日置慎治

私はハイパフォーマンスコンピューティングに興味がありますので、

このモニターになり、以下の点を中心に調べて見ましたので順を追って

報告します。

1）SX4 単体（つまり 1PE）の性能を調べる

2）単体の性能をあげるための方法を調べる。

3）並列計算をさせる方法を調べる。

特に、メッセージ通信をあらわに利用したMPI/SX を使った場合について。

4）並列計算における性能評価を行なう。

5）おわりに

1）SX4 単体（つまり 1PE）の性能を調べる

単体での行列積の性能評価を行なった。（単精度、３行３列複素行列）

コンパイルは f77 を使い、オプションは指定せずに行ないました。

結果は以下の通りとなりました。

SU(3) Matrix (3x3 Unitary Matrix) multiplication benchmark

ベクトル長	100	250	1000	10000
MFLOPS 値	1148	1531	1590	1587

考察：

a)ベクトル長が 250 程度でも 10000 の時の 90%以上、カタログ性能(2000MFlops)の

75%程度出ている。

これはある程度満足のいく性能である。

b) 参考までに f90 でコンパイル（オプションなし）してみると、何故か f77 に比べて

１割程度遅かった。

2）単体の性能をあげるための方法を調べる。

報告）IEEE 単精度は遅い！

上記性能はオプションなしの f77 を使ったのですが

例えばこれに、-ew をつけて、倍精度にしてやると

vec. length	100	250	1000	10000
MFLOPS 値	1280	1600	1708	1719

になります。倍精度にするだけで10%以上の性能向上が見られます。

これは、SX4 の内部では常に倍精度で計算するため、単精度にこだわると

常に単精度<->倍精度の変換が必要となり余分なオーバーヘッドとなることが

原因と考えられます。

教訓）単精度にこだわるな！

（えっ？こだわっていないって？、でも僕のように

何も知らずに単精度のソースを使っている人って結構いるんじゃない？）

報告）-float1 オプション

単精度にこだわるが IEEE にはこだわらないとすると、

-float1 で IBM 形式単精度にしたら

vec. length	100	250	1000	10000
MFLOPS 値	1290	1662	1730	1733

となり、先の倍精度と同じ程度の性能が得られました。

教訓) IEEE 単精度にこだわるな！(くどいかな？)

報告) オプションを選べば f90 は遅くない！

上で、f90 は f77 に比べて、遅い！と書いたのですが、これはオプションなしの

場合でした。extendreorder というオプションをつけるとにより、

f77 と同程度以上に速くなる事がわかりました。

ベクトル長を２５６に固定して比べてみたのが以下の結果です。

	f77	f90
コンパイル時間(秒)	1.5	11.0
MFlops 値		
no option	1568	1374
-ew	1638	1485
-float1	1686	1568
-0 extendreorder	1744	

教訓) f90 は遅くない！(でもこのオプション知ってました？)

3）並列計算をさせる方法を調べる。

特に、メッセージ通信をあらわに利用したMPI/SX を使った場合について。

私は自分の研究分野が素粒子原子核物理の

「格子色力学モンテカルロシミュレーションによるハドロン物理の研究」

なものですから、このシミュレーションを大規模かつ高速に行なう必要が

あり、このために様々なコンピュータを使って来ました。

そのような時に、いつも気にかかっていたのが

「使う計算機毎にソースを変更しないと動かないあるいは性能が出ない」

という事でした。これではちっとも効率的な計算とは言えないと思っていました。

そこで、利用する計算機つまりプラットフォームに依存しないコードを作ろうと

思い立ち、並列計算機も想定した上で、QCDMPI というパッケージを作りました。

この経緯については、

「大阪大学大型計算機センターニュース Vol.26 No.4 1997-1 p106」

にありますので参考にしてください。また、QCDMPI については

URL http://insam.sci.hiroshima-u.ac.jp/QCDMPI/

に説明と、各種マシン」でのベンチマークがありますので見てください。

さて、ここではQCDMPI を用いて実際の計算を行なうに当たっての方法を調べて

見ました。現在の所、ソースは依存性がなくても、実行するための方法は

やはりマシン毎（というかＯＳあるいは環境毎）に違っているのが現状です。

長くなると読みづらくなるので、要点をまとめることにします。

とりあえず、「速報 1997./.22 No.267 p7」を参考に ~guest/mpi/mpi-sample/

の下のMakefile を参考にして、やりはじめました。

つまづき）-G local オプションを付け忘れていたため、各PE で独立だと

思っていた変数が実は共有されていた。

共有メモリ型計算機の場合には変数は共通なのが当たり前のようですが、

MPI を分散メモリで使って来たために全然こういった事には注意を払いませんでした。

教訓）MPI 利用の時には -G local に注意せよ！

報告）MPI/SXについて「速報 1997.7.22 No.267 p7」を見ると、

「実行方法は(2)のthread方式で行なってください」

とわざわざ書いてあるので、素人の僕はそれを信用してthreadで

行ないました。すると、思ったような性能が出ませんでした。

それには以下の3つの要因が関係している事が分かりました。

あ）簡単のためSXの組み込み乱数を使った。

実はthread同士で乱数の種を共有するため、乱数の部分はどうしても

同時実行はできなかった。

い）測定をインタラクティブに行なっていた

インタラクティブ実行の場合、物理タスク数が1に制限されているので

どうあがいても並列計算にはならないのだった。

う）通信性能が悪いので、フリーのMPICHをコンパイルして通信部分にこれを

使った場合とMPI/SXを比較すると断然MPICHの方が速かった。

教訓）thread実行はあまり嬉しくない！

プロセス実行では課金がうまくできないというか損をするということで

thread実行をすすめてあったわけですが、聞く所によると現在のOSでは

プロセスしかないそうなので、(本当ですか？)

この問題は今や昔のものとなってしまいました。

上記のうち、う）にかんしてはMPICHのデフォルトが

プロセスだったため様々なthreadの制約から逃れたのが性能の差となって

見えていたようです。実際は（同じプロセス実行すると）

MPI/SX > MPICH となりました。

4）並列計算における性能評価を行なう。

上記パッケージ QCDMPI を用いて並列計算ベンチマークを行なった。

QCDMPI はそのまま実行するだけで、全体の計算性能と通信性能を別々に

出力するよう設計されているので、初心者でも並列計算機の性能を見るために

簡単に使えるものとなっている。

PE数		1	2	4	8
計算性能(micro sec/link)	3.89	1.98	1.03	0.55	
通信性能(MB/sec)	-	1158	1176	515	
見積性能(MFlops)	1465	2879	5534	10364	
1PE当たりの性能(MFlops)		1465	1440	1384	1296

コンパイルオプションは

```
-P multi -G local -float1 -I/usr/include -lmpi
```

実行は

```
mpisx -p 8 -e ./qcd
```

などとして行なった。

ここでの計算性能とはもっとも基本となる自由度１つに対する計算を行なうために

必要な時間をマイクロ秒で表したものであり、これが小さい程高速計算ということ

に

なる。通信性能は読んで字のごとくである。

なお、上の全体性能をMFlopsに換算すると、大雑把に見積もって、
上記「見積性能」になり、これを1PE当たりに換算したのが一番下の数字である。

結果を見ると明らかであるが、8PEまではほぼスケーラブルな性能向上が見られた。

（ここでの「ほぼ」は我々のアプリのレベルでは十分使いものになると言う意味である）

8PEの時には何故か通信性能が半分になっているが、全体の時間からすると
1割にも満たない時間なので、全体として見ればスケーラビリティには
それほど影響していない。

この通信性能の低下原因について、また、これ以上のPE数におけるベンチマークに付いては今後行なって行きたいと考えている所である。

したがって、今年度もモニターとして活動させて頂きたいと考えております。

5）おわりに

一年間SX4のベンチマークを細々とやって来たのですが、一番困ったのは
やはりマニュアル類とHOWTOの不足でした。

最初何を調べていいのか？

どこに聞いていいのか？

どうやってコンパイルするのか？

実行はどうやって？

有効なオプションは？

などがまったく手探りの状態でした。もちろん、速報やニュースあるいは直接出向いてマニュアルを隅から隅まで読めばいいのでしょうけど、なかなかできないことなのです。そう思ってメーリングリストに何度が質問を出したのですが、助けられたこともありましたが、大抵の場合は誰からも返事がなく、ちょっと淋しい思いもしました。

ようやく１年を経て、いろんな情報を集めた結果、ここまでの報告をする事ができました。せっかく集めた情報ですから、今後SX4を使い始める時に少しでも参考になればと思い、自分のWebPageに載せることにしました。
現在はまだこの報告の内容だけが載っている状態ですが、今後いろんな事を発見次第、随時更新していく予定ですので参考にしてください。

URL http://tupc3472.tezukayama-u.ac.jp/sx4/

です。