ハイパフォーマンス
コンピューティング 63－6
（1996. 10. 4）

超並列計算機 CP-PACS の CG ベンチマークによる性能評価

板倉憲一† 朴 泰祐†
中村 宏†† 中澤喜三郎†††

本研究では超並列計算機 CP-PACS において NAS Paralle Benchmarks の kernel CG の実行時間の解析を行った。解析には CPU に組込まれている clock に同期した極めて正確な時計を用いることで、複雑な処理を分解して測定することができ、その殆どの部分において理論的な解析と一致することを示した。特に無衝突なデータ転送に関しては、そのデータ転送パターンの性質（回数と転送量）とシステムの基本性能（立ち上げオーバヘッドとスループット）から処理時間の正確な見積りができることが分った。本研究で用いたプログラムでは十分な台数効果は得られなかったが、このプログラムのボトルネックを解析結果から示すことができた。

Performance evaluation CP-PACS on CG benchmark

KEN'ICHI ITAKURA,† TAISUKE BOKU,† HIROSHI NAKAMURA†† and KISABURO NAKAZAWA †††

In this research, we evaluate NAS Parallel Benchmarks Kernel CG on massively parallel processor CP-PACS, and analyze the result. CP-PACS's CPU has a special register which is auto-incremented by clock cycle, and we can instrument time spent for any function routine with very high accuracy. As a result of performance analysis, especially for data transfer time, our desk-top estimation fits to the instrumented result almost perfectly. From this analysis, we could show the bottleneck of the program when executing with this a large number of PU's.

1. はじめに

超並列計算機は現在最も計算能力の高いアーキテクチャとして研究が盛んに行われており、筑波大学では計算物理学を目的とした CP-PACS プロジェクト[1]が進められている。このプロジェクトでは千台規模の超並列計算機 CP-PACS[2]を設計・製作し、本年4月から実際に稼働している。

CP-PACS は設計段階において、構成要素および全体性能に関する机上およびシミュレーションによる性能評価がなされているが[3]、本報告では疎行列データに対して様々なデータ転送パターンを持つ CG 法のベンチマークアプリケーションによって、性能の予測値と実測値の比較を行う。これによって CP-PACS の基本性能を明きらかにすると共に、性能予測の妥当性について考察する。

† 筑波大学 電子・情報工学系
Institute of Imformation Sciences and Electronics, University of Tsukuba
†† 東京大学 先端科学技術研究センター
Research Center for Advanced Science and Technology, University of Tokyo
††† 電気通信大学 情報工学科
Department of Computer Science, University of Electro-Communications

2. 超並列計算機 CP-PACS

CP-PACS[2]は 1024 台の PU (Processing Unit) を疎結合した超並列計算機で、理論最高性能は 307.2 GFLOPS である。CP-PACS の構成図を図1に示す。

CP-PACS の PU はプロセッサ、主記憶、キャッシュ、ネットワークインターフェイスアダプタ (NIA)、記憶制御装置 (Storage Control Unit) から構成される。CP-PACS は MIMD 動作方式を用い、プログラム及びデータは各 PU の主記憶に独立して置かれる。PU 間でのデータの受渡しは、ユーザプログラム中に明示的に記述されるメッセージパッシング命令によって実現する。CP-PACS のアーキテクチャの大きな特徴はプロセッサに PVP-SW[4][5]を用い、PU 間を HXB 相互結合網[6][7]によって結合していることである。

PVP-SW はキャッシュを用いずにメモリレイテンシを隠蔽する方法を提供し、擬似的なベクトル処理を可能にする。メモリレイテンシの隠蔽は命令の完了を待たずに後続の命令を処理することが可能な、レジスタへの preload 命令によって行なう。さらに、デー

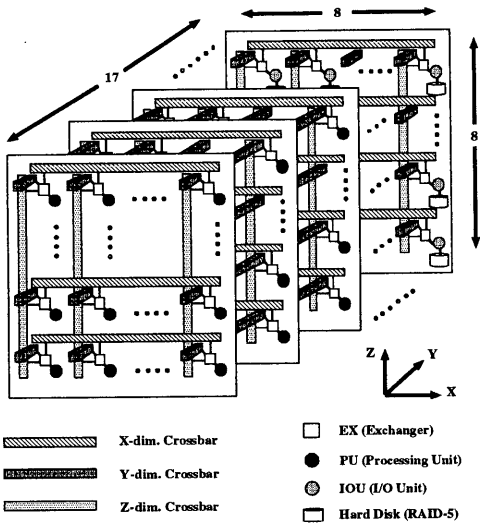

図1　CP-PACS の構成図

タがメモリからレジスタに届くまでの時間に十分な演算を行なうためにレジスタ数を拡張する。そして、PVP-SW はベクトルプロセッサではなく、既存 RISC プロセッサをベースとしてするので、命令アーキテクチャレベルの上位互換性を保ちたい。そこで、拡張したレジスタにはレジスタ・ウィンドウを用いてアクセスし、通常の命令に対しては拡張前のレジスタ数と同等になるようにする。しかし、preload 命令などの拡張した命令は全てのレジスタにアクセス可能にし、実際の演算に先行して preload 命令を発行可能にする。

擬似ベクトル処理はスカラプロセッサのループ処理として実現し、ループの 1 イテレーション内では実際の演算の他に preload 命令による先のウィンドウのレジスタへの preload とウィンドウを前方にスライドさせる命令を含む。また、スーパスカラ方式により preload 命令と演算命令の同時処理を行うことにより、ベクトル処理の効率的に行う。

次に、CP-PACS で用いる 3 次元 HXB について述べる。3 次元 HXB は図 1 に示したように 3 次元直行空間上に PU を並べ、その間をクロスバスイッチ (XB) で結合した間接網である。各次元方向 の PU は直接 XB によって完全結合され、それ以外の PU 間のデータ転送はエクスチェンジャー (EX) と呼ばれる小型のクロスバスイッチによって複数次元方向の XB を経由して行なう。CP-PACS ではメッセージ転送に wormhole ルーティング を用い、経路の選択はデッドロックを防ぐために固定ルーティングを用いる。

さらに、CP-PACS のデータ転送にはリモート DMA と呼ばれる高速な転送方式がある。これは、PU の NIA がユーザメモリ空間のデータを直接 DMA アクセスして送受信を行う。特に受信側では受信命令を発行する必要は無く、メッセージ到着によって自動的に NIA が動作し、受信側のユーザメモリ空間にデータが書き込まれる。このように、システム領域のバッファを使用しないので、ハードウェアの高速なスループットをそのまま活かすことができる。また、転送の立ち上げオーバヘッドを軽減するために、NIA の起動を OS を介さずにユーザが直接行なえるようにもなっている。

現在の CP-PACS は 1024 PU を 512PU, 256PU, 128PU × 2 の 4 つのパーティションに分けて分割運転しており、今回の性能測定には 128 PU ($2 \times 8 \times 8$) のパーティションを使用した。

3. CG ベンチマークプログラム

本報告ではベンチマークプログラムとして NAS Parallel Benchmarcks (NPB1)[8] の Kernel CG (問題サイズ Class A) を用いる。Kernel CG は正値対称な大規模疎行列の最小固有値を CG (Conjugate Gradient) 法によって求める問題である。逐次版 Kernel CG のメインループを以下に示す。

```
do k=1,KERNITR
   call matvec(a, rowidx, colstr, p, q)
   alpha=rho/dotpro(p, q)
   z(1:N)=z(1:N)+alpha*p(1:N)
   rho0=rho
   r(1:N)=r(1:N)-alpha*q(1:N)
   rho=dotpro(r, r)
   beta=rho/rho0
   p(1:N)=r(1:N)+beta*p(1:N)
enddo
```

ここで、p, q, z, r は N (=14000) 要素のベクトル、matvec と dotpro はそれぞれ行列・ベクトル積とベクトルの内積を計算するサブルーチンである。NPB1 では並列化に関してデータのマッピングやアルゴリズムを自由に決めてよい。そこで、各ベクトルはブロック分割し PU にマッピングする。そして、内積計算 (dotpro) といわゆる DAXPY に類する計算はデータパラレル処理によって行う。内積計算では PU 内の計算が終了した後にデータ転送と受信データに対する加算をを実行し、全 PU が同じ計算結果を保持する (Butterfly Summation：図 3)。

行列・ベクトルの積の処理 (matvec) は 2 次元 PU 平面 ($P = P_x \times P_y$, P は PU 台数) にブロック-ブロック分割された行列に対して行う[9]。各 PU では部分疎行列を値 (a) と 桁位置 (rowidx) と各行のスタートポイント (colstr) によって表現する。そして matvec の処理手順は以下の 4 段階に分けて行う (図 2)。

(1) Collection: 被乗数ベクトルは保持する部分行列の列に対応する N/P_x 要素を持つベクトルを必要とする。このために P_y 個の PU のグループ内

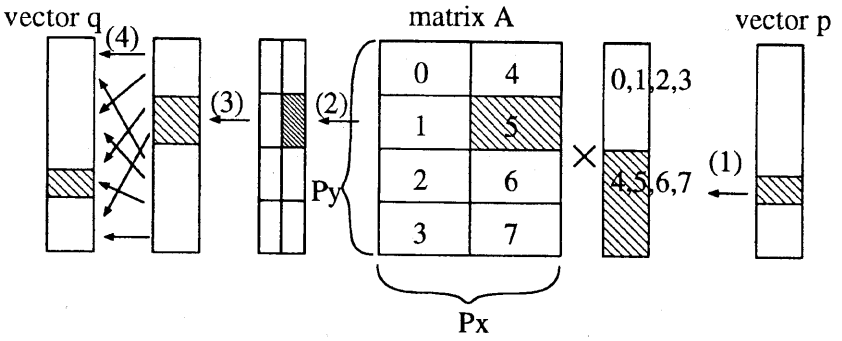

図2 matvec のデータ転送パターン

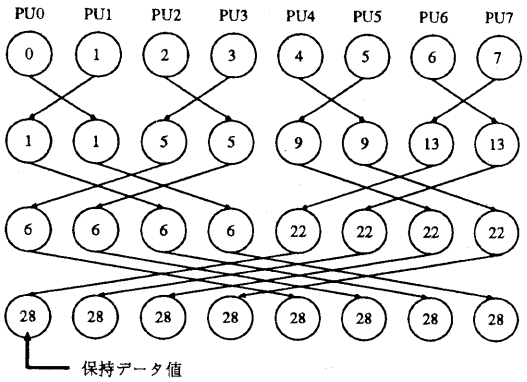

図3 Butterfly Summation

でベクトルの Collection を行う。ベクトルの Collection の方法としては P_y 回のマルチキャスト、$P_y - 1$ 回の「横流し」的な転送などがあるが、global な転送が得意な HXB では転送回数が少ない Butterfly Summation と同じように butterfly 転送によって行う。Summation はリダクション演算の為に毎回のデータ転送量が一定であるが、Collection ではデータ転送量が2倍づつ増加する点が異なる。

(2) Multiply: 部分行列と部分ベクトルの乗算を行う。これは以下のようなリストベクトル処理になるが、

$$q = q + a[i] \times p[\mathrm{rowidx}[i]]$$

浮動小数点データ (a, p) のロードはレジスタへの preload 命令を使うコードがコンパイラによって生成される。これに対して rowidx は通常のロード命令によって行われるが、4PU 以上の並列処理を考えた時には 512KB の 2次キャッシュには入るデータ量である。さらに、整数配列の連続領域に対するアクセスなので1次キャッシュの高ヒット率も期待できる。

(3) Summation: Multiply で求めた結果は部分和なので、真の乗算結果を得るために P_x 個の PU グループ内で Vector Summation を行う。この処理は dotpro の Butterfly Summation と同等な事をベクトルに対して行う。

(4) Shuffle: Summation で求まった q はこの後で dotpro や DAXPY の計算に用いられるが、各 PU が保持している部分が他のベクトルのマッピングと異なってしまっている。そのため保持する部分ベクトルの位置を合せるために Shuffle 転送を1回行う。

メインループにおいて最も処理時間のかかる部分は matvec であり、matvec の処理時間は 2次元 PU 平面 ($P_x \times P_y$) の構成によって決まる。まず、matvec で行われるデータ転送の回数とデータ転送量を表1に示す。定性的には P_x の増加は Summation 時間の増加を招き、P_y の増加は Collection 時間の増加を招く。簡単なピンポン転送の実験からデータ転送の立ち上げオーバヘッドは 約 5500 clock☆, 転送スループットは 2B/clock という基本通信性能が分っているので全転送時間は以下のようになる。

表1 matvec のデータ転送

処理パターン	転送回数	総転送量
Collection	$log_2 P_y$	$N/P \times (P_y - 1)$
Summation	$log_2 P_x$	$N/P_x \times log_2 P_x$
Shuffle	1	N/P

$$(log_2 P_x + log_2 P_y + 1) \times 5500$$
$$+ N/P \times (P_y + P_y log_2 P_x)/2 \text{ [clock]}$$

次に、CPU の計算時間を求める。計算時間は Multiply の時間と Summation のベクトル和の時間からなる。疎行列の non-zero 要素の分布がほぼ一様なので2次元 PU 平面の構成に関係なく Multiply の演算

☆ プログラミングを容易にするために最速の送信命令は使っていない。

表2　Kernel CG の処理時間 [sec]

		Px							
		1	2	4	8	16	32	64	128
#PU	16	3.68	5.60	6.35	6.41	7.82	-	-	-
	32	2.37	2.66	3.54	3.44	4.05	5.67	-	-
	64	2.09	2.12	2.47	2.67	2.88	4.09	5.89	-
	128	2.76	2.81	3.18	3.02	3.03	3.44	3.30	4.66

表3　メインループの処理時間の内訳 [clock] (128PU)

		Px					
		1	2	4	8	16	32
処理	matvec	240316	259487	310919	313710	377555	532014
	dotpro	1173	1269	1196	1302	1329	1280
	DAXPY	1697	1596	1600	1596	1763	1582

数は一定と考えられる。Summation 時のベクトル和の演算量は N/P_y でなので P_x の増加 (= P_y の減少) は演算時間の増加を招く。

ベクトル和の部分のみを実測した結果、7clock/1add で演算できることがわかったので、Multiply の処理時間を固定とすると matvec の残りの処理時間は図4のように予想され、128 PU の時は $P_x \times P_y = 4 \times 32$ 構成が最も高速に処理できると考えられる。

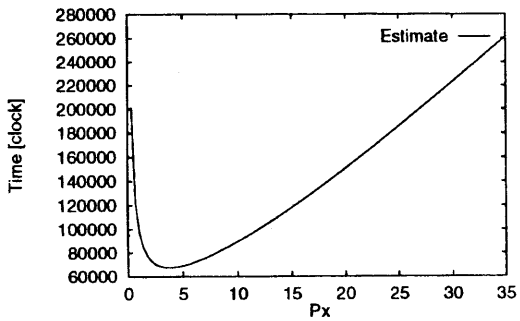

図4　matvec の予想処理時間 (128PU)

4. 性能評価

PU 数を 16 から 128 まで変えて Kernel CG のプログラムを実行した結果を表2に示す。この表から分るように、全ての PU 台数で単純な行ブロック分割 ($P_x = 1$) が最も短い処理時間となった。また、128PU の時には 64PU の時よりも処理時間が長くなってしまっている。さらに、処理速度向上率のグラフ (図5) より PU 台数に見当った速度向上が得られていないことも分る。

このような性能低下の原因を調べるために、128PU の時の細かい処理時間の内訳を測定した。この測定には CPU に内蔵されている、clock に同期してインクリメントされる特別なコントロールレジスタ (すなわちクロックカウンター) を用いた。これにより clock に同期したきわめて正確な処理時間の測定が可能である。メインループの各処理の処理時間を表3に示す。表3より matvec でほとんどの時間が消費されていることが分る。

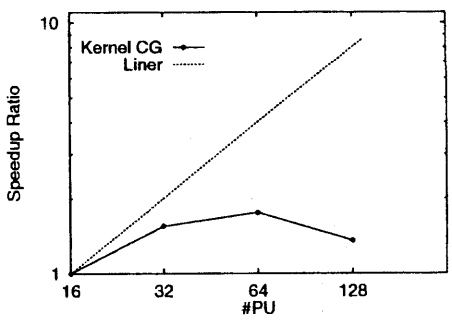

図5　処理速度向上率

次に、matvec の処理時間の内訳を表4に示す。さらに、データ転送パターンごとの予測値と実測値を図6,7,8に示す。これによると、予測値と実測値が良い精度で合っている。そして、行列のマッピングによって Collection と Summation の転送時間がトレードオフしていることが分る。ここで、Shuffle 転送に関しては多次元の HXB ネットワークを用いた場合にネットワーク中で衝突が起こることが分っているが、予測値は無衝突を仮定しているので若干の誤差が生じている。

しかし、PU 内での純粋な計算時間のみである Multiply は、演算量が一定なので計算時間が2次元 PU 構成によらず一定である、と仮定したはのが間違いで

表4　matvec の処理時間の内訳 [clock] (128PU)

		P_X					
		1	2	4	8	16	32
処理	Collection	119577	68260	45248	31354	20830	12512
	Multiply	111787	167844	237147	211768	228613	287601
	Summation	63	10719	21474	44778	102824	208168
	Shuffle	7772	6307	5949	5978	6169	5845

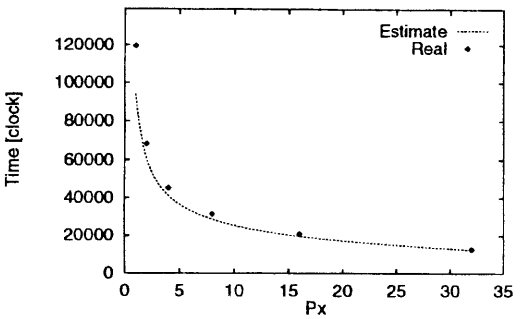

図6　Collection の処理時間 (128PU)

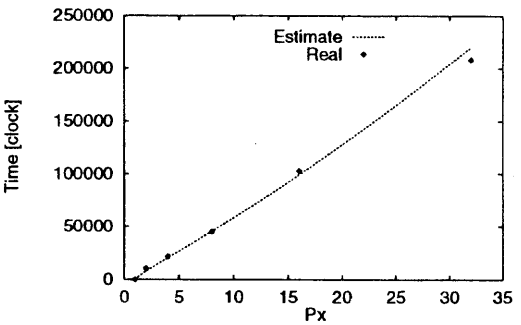

図7　Summation の処理時間 (128PU)

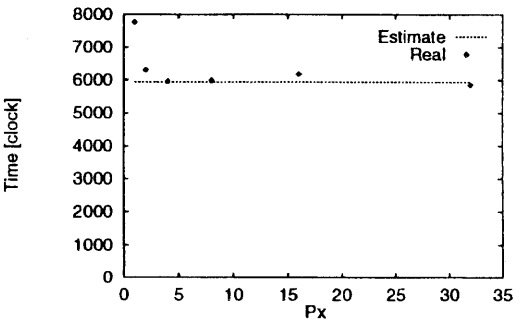

図8　Shuffle の処理時間 (128PU)

あることが分る。これは、CP-PACS の PU は PVP-SW 機構によりベクトルプロセッサと同じような特性を持ち、ループ長が短いループ処理では性能が低下するためである。今回の測定に用いたプログラムでは部分行列の一行ごとに積を計算するので、ループ長は一行当りの non-zero 要素数になる。Kernel CG (Class A) では sparse 率が約 1% なので、平均ループ長は

$$N \times 0.01/P_X = 140/P_X$$

となる。この最内ループのコードにおいて、ループ長が 140 未満ではメモリレイテンシを隠蔽するのに長さが足りず、P_X の増加がループ処理のオーバヘッドによる計算時間の増加を招いたと考えられる。

全体としてベクトル長の減少による Multiply の時間の増加が非常に大きいために、結果として P_X と P_Y のトレードオフが存在せず、常に $P_X = 1$ が最適である結果となったと考えられる。ここまでの精度の良い実測と処理時間の予測から、メモリレイテンシを隠蔽できる程の問題サイズの十分大きな問題 (Class B または Class C など) に対しては 2 次元ブロックマッピングによって P_X と P_Y のトレードオフが存在し、128 台を越えるの PU 構成においても速度向上が期待できる。

5. おわりに

超並列計算機 CP-PACS において NPB の kernel CG の性能評価を実機によって行い、その解析を行った。CPU に組込まれている clock に同期した極めて正確な時計を用いることで、複雑な処理を分解して測定することができ、その殆どの部分において理論的な解析と一致することを示した。特に無衝突なデータ転送に関しては、そのデータ転送パターンの性質（回数と転送量）とシステムの基本性能（立ち上げオーバヘッドとスループット）から処理時間の正確な見積りができることが分った。

Kernel CG は様々な転送パターンを必要とし、その多くが細かいデータ転送となるために並列化効率が悪い。この問題を解決するためにマトリックスを 2 次元 PU 平面にブロック-ブロック分割しデータ転送を最適化する手法を考えた。しかし、このマッピングが PVP-SW による擬似ベクトル処理のベクトル長を短くしてしまい、内部処理時間の増大を招くことが分っ

た。結果として Kernel CG (Class A) の処理時間は64PU の時に最短となり、128PU では逆に処理時間が増大してしまった。

今後の課題として、Kernel CG で現れたような二重ループの処理について内側のループ回数が減ってもその分外側のループ回数が増え十分な演算がある場合にはこのループを一重化し効率良くベクトル処理できるような事を考えたい。

謝辞 CP-PACS を利用する機会を頂いた計算物理学センター関係各位に感謝します。また、筑波大学西川博昭助教授ならびにアーキテクチャ研究室の諸氏には貴重な御意見を頂いたことに感謝します. なお, 本研究の一部は創成的基礎研究費（08NP0401）の補助によるものである.

参 考 文 献

1) 岩崎 洋一他:「専用並列計算機による「場の物理」の研究」, 技術報告, 筑波大学計算物理学研究センター (1994). 新プログラムによる研究 研究進捗状況報告書.
2) 中澤喜三郎, 朴泰祐, 中村宏, 中田育男, 山下義行, 岩崎洋一:「CP-PACS のアーキテクチャの概要」, 情処研報 ARC-108-9 (1994).
3) Itakura, K., Hattori, M., Boku, T., Nakamura, H. and Nakazawa, K.: "Preliminary Evaluation of NAS Parallel Benchmarks on CP-PACS", *Proc. of the Int. Workshop on Performance Evaluation and Analysis (PERMEAN'95)*, pp. 68–77 (1995).
4) 位守弘充, 中村宏, 朴泰祐, 中澤喜三郎:「スライドウィンドウ方式による擬似ベクトルプロセッサ」, 情報処理学会論文誌, Vol. 34, No. 12, pp. 2612–2613 (1993).
5) Nakamura, H., Imori, H., Nakazawa, K., Boku, T., Nakata, I., Yamashita, Y., Wada, H. and Inagami, Y.: "A Scalar Architecture for Pseudo Vector Processing based on Slide-Windowed Registers", *Proc. of ACM International Conference on Supercomputing '93*, pp. 298–307 (1993).
6) 田中 輝雄他:「識別子を用いたデータ転送方式を基本とする MIMD 型並列計算機アーキテクチャ」, 並列処理シンポジウム JSPP'89 (1989).
7) 朴泰祐, 斉藤哲也, 板倉憲一, 中澤喜三郎, 中村宏:「ハイパクロスバ・ネットワークの性能評価」, 信学技報 CPSY-93-40 (1993).
8) Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Fatoohi, R., Fineberg, S., Frederickson, P., Lasinski, T., Schreiber, R. and Simon, H.: "The NAS Parallel Benchmarks", Technical report, NAS (1994). RNR-94-007.
9) Agarwal, R. C., Alpern, B., Carter, L., Gustavson, F.G., Klepacki, D. J., Lawrence, R. and Zubair, M.: "High-performance parallel implementations of the NAS kernel benchmarks on the IBM SP2", *IBM Systems Journal*, Vol. 34, No. 2, pp. 263–272 (1995).